CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Secure coding in organisations: practice, culture, motivations and tensions
Authors	Ryan, Ita
Publication date	2024
Original Citation	Ryan, I. 2024. Secure coding in organisations: practice, culture, motivations and tensions. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2024, Ita Ryan. - https://creativecommons.org/licenses/by/4.0/
Download date	2026-11-15 13:34:46
Item downloaded from	https://hdl.handle.net/10468/16374



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Ollscoil na hÉireann, Corcaigh
National University of Ireland, Cork



Secure Coding in Organisations: Practice, Culture, Motivations and Tensions

Thesis presented by

Ita Ryan, MSc

for the degree of

Doctor of Philosophy

University College Cork
School of Computer Science and Information Technology

Head of School: Prof Utz Roedig

Under supervision of:

Dr Klaas-Jan Stol and Prof Utz Roedig

2024

Contents

Abstract . . . x
Acknowledgements . . . xi
List of Tables . . . xi
List of Figures . . . xii
List of Acronyms . . . xiv
My Contributions . . . 1
Note on the text . . . 3

1 Introduction **4**
1.1 Motivation . . . 4
1.2 Problem Outline . . . 5
1.3 Research Design . . . 6
1.3.1 Positionality . . . 7
1.3.2 Study A: Reviewing the literature . . . 8
1.3.2.1 P1: Insecure Software on a Fragmenting Internet . . . 8
1.3.2.2 P2: Understanding Developer Security Archetypes . . . 9
1.3.2.3 P3: Unhelpful Assumptions in Software Security Research . 9
1.3.3 Study B: Assessing organisations: Developer survey . . . 10
1.3.3.1 P4: Measuring Secure Coding Practice and Culture: A Finger Pointing at the Moon is not the Moon . . . 11
1.3.3.2 P5: The State of Secure Coding Practice: Small Organisations and 'Lone, Rogue Coders' . . . 12
1.3.3.3 P6: Training Developers to Code Securely: Theory and Practice 13
1.3.4 Study C: Motivating organisations: Interview study . . . 13
1.3.4.1 P7: 'Money. Or Jail Time.' Improving Software Security by Motivating the C-Suite . . . 14
1.4 Thesis Contributions and Structure . . . 16

2 Insecure Software on a Fragmenting Internet **18**
2.1 Abstract . . . 18
2.2 Introduction . . . 18
2.3 Internet Nationalism . . . 20
2.3.1 OSI Model Layer 1: Physical . . . 20
2.3.2 OSI Model Layer 2: Data Link . . . 21
2.3.3 OSI Model Layer 3: Network . . . 21
2.3.4 OSI Model Layer 4: Transport . . . 21
2.3.5 Data, Applications and Access . . . 22
2.3.6 Multi-Level Divergence . . . 23
2.4 Security Issues for a Global Internet . . . 24
2.4.1 Threat actors in the ENISA report . . . 25
2.4.1.1 State-Sponsored Actors . . . 25
2.4.1.2 Cybercriminals . . . 25

2.4.1.3 Hacker-For-Hire Actors . . . 26
2.4.1.4 Hacktivists . . . 26
2.4.2 Cybercrime threats in 2020-2021 . . . 26
2.4.2.1 Ransomware . . . 26
2.4.2.2 Cryptojacking . . . 27
2.4.2.3 Other Cybercrime . . . 28
2.4.2.4 Cyberespionage . . . 28
2.4.2.5 Cyber Patriotism . . . 28
2.4.2.6 Cyberwarfare . . . 29
2.5 The Role of Software Vulnerabilities . . . 31
2.6 Attempts to Remediate . . . 33
2.6.1 Industry . . . 33
2.6.2 Academia . . . 34
2.6.3 Legislation . . . 35
2.6.4 Exacerbating Factors . . . 36
2.6.5 A 'Zero Tolerance' Approach to Software Security . . . 37
2.7 Conclusion . . . 38

3 Understanding Developer Security Archetypes 39
3.1 Abstract . . . 39
3.2 Introduction . . . 39
3.3 The Role of Environment and Personal Motivation . . . 41
3.4 The Developer Security Archetypes . . . 42
3.4.1 Pragmatists . . . 42
3.4.2 Champions . . . 43
3.4.3 Optimists . . . 43
3.4.4 Heroes . . . 44
3.5 Discussion . . . 44
3.5.1 Developer Motivation . . . 45
3.5.2 Organisational Motivation . . . 46
3.5.3 Supporting the Optimists . . . 46
3.6 Conclusion . . . 46

4 Unhelpful Assumptions in Software Security Research 48
4.1 Abstract . . . 48
4.2 Introduction . . . 48
4.3 Methodology . . . 51
4.3.1 Venues and Search Terms . . . 51
4.3.2 Selection procedure . . . 51
4.3.3 Snowballing . . . 52
4.3.4 Data Extraction and Analysis . . . 52
4.3.5 Positionality statement . . . 53
4.4 Results . . . 53
4.4.1 Technical Assumptions . . . 54
4.4.2 Developer Assumptions . . . 68
4.4.3 Organisational Assumptions . . . 74
4.4.4 Research Assumptions . . . 76
4.5 Discussion and Conclusion . . . 82

5 Measuring Secure Coding Practice and Culture: A Finger Pointing at the Moon is not the Moon **84**
5.1 Abstract . . . 84
5.2 Introduction . . . 85
5.3 Background and Related Work . . . 86
5.3.1 Measuring Secure Development . . . 86
5.3.2 Security Culture . . . 87
5.3.3 Organisational Climate Theory . . . 90
5.4 Method . . . 90
5.4.1 Survey Questions . . . 91
5.4.1.1 Screening Questions . . . 91
5.4.1.2 Measuring Security Practice . . . 92
5.4.1.3 Measuring Security Culture . . . 92
5.4.1.4 Pilot Testing . . . 92
5.4.2 Data Collection . . . 92
5.4.3 Screening Process . . . 93
5.4.4 Ethics . . . 94
5.4.5 Data Analysis . . . 94
5.5 Sample Description . . . 94
5.5.1 Demographics . . . 94
5.5.2 Programming Languages and Tools . . . 95
5.5.3 Adoption of Secure Coding Standards . . . 96
5.5.4 Adoption of Development Methods . . . 96
5.5.5 Secure Coding Policy . . . 97
5.5.6 Security Priorities . . . 97
5.5.7 Training . . . 97
5.6 Results . . . 98
5.6.1 Research Question 1: What are the secure-coding characteristics of our sample group? . . . 98
5.6.2 Research Question 2: What are the security culture characteristics of our sample group? . . . 100
5.6.2.1 Support for Secure Coding (SCQ1) . . . 101
5.6.2.2 Raising Security Concerns (SCQ2) . . . 101
5.6.2.3 Whether there are Security Tools in the Working Environment (SCQ3) . . . 101
5.6.2.4 Introducing Free Tools (SCQ4) . . . 104
5.6.2.5 Asking for Tool Permissions (SCQ5-SCQ7) . . . 105
5.6.2.6 Perceptions of Security Culture (SCQ8-SCQ9) . . . 105
5.6.2.7 How Highly the Team Prioritises Security (SCQ10) . . . 106
5.6.2.8 How Often Security is Mentioned in Team Communications (SCQ11) . . . 107
5.6.2.9 Time per Week Spent on Secure Coding (SCQ12) . . . 107
5.6.3 Research Question 3: Do secure coding practice and culture correlate, and if not, what lessons can we learn to help support development of secure coding? . . . 109
5.7 Discussion and Conclusion . . . 110
5.7.1 Threats to Validity . . . 111
5.7.2 Conclusion . . . 113

6 The State of Secure Coding Practice: Small Organisations and 'Lone, Rogue Coders' 115
6.1 Abstract . . . 115
6.2 Introduction . . . 116
6.3 Related Work . . . 119
6.3.1 State of Practice . . . 119
6.3.2 Secure Coding Guidelines . . . 119
6.3.3 Secure Coding Assessment . . . 120
6.4 Method . . . 121
6.4.1 Data . . . 122
6.5 Results . . . 122
6.5.1 Usage of CAs by Developer Category . . . 124
6.5.1.1 Penetration Testing . . . 124
6.5.1.2 Review . . . 125
6.5.1.3 Quality Assurance . . . 127
6.5.1.4 Operations Feedback . . . 127
6.5.1.5 Security Processes . . . 127
6.5.1.6 Security Tools . . . 130
6.6 Discussion . . . 131
6.6.1 Threats to Validity . . . 131
6.6.2 Discussion . . . 132
6.7 Conclusion . . . 133

7 Training Developers to Code Securely: Theory and Practice 135
7.1 Abstract . . . 135
7.2 Introduction . . . 136
7.3 Background and Related Work . . . 137
7.3.1 Training Availability . . . 137
7.3.2 Training Effect . . . 138
7.3.3 Software Security Learning . . . 139
7.3.4 Training in the Workplace . . . 140
7.4 Methodology . . . 141
7.4.1 Generation of Positive Secure-Coding Training Features . . . 141
7.4.2 Observation of Secure-Coding Training Features in Industry . . . 141
7.5 Secure-Coding Training in Theory . . . 142
7.5.1 Relevance . . . 143
7.5.2 Time Allocated . . . 144
7.5.3 Engagement . . . 144
7.5.4 Additional Training Features . . . 144
7.5.5 Indicators for a Poor Training Environment . . . 146
7.6 Secure-Coding Training in Practice . . . 146
7.6.1 Demographics . . . 147
7.6.2 Breakdown of Training Answers . . . 147
7.6.3 Research Question 2: Are developers offered secure coding training, and if so, does it have the positive features we have identified? . . . 148
7.6.3.1 Relevance . . . 148
7.6.3.2 Time allocated . . . 149
7.6.3.3 Engagement . . . 150

7.6.4 Features Added from Survey Data . . . 150
7.6.4.1 Developer Tone . . . 151
7.7 Discussion . . . 151
7.7.1 Limitations . . . 153
7.8 Conclusion . . . 153

8 'Money. Or Jail Time.' Improving Software Security by Motivating the C-Suite 155
8.1 Abstract . . . 155
8.2 Introduction . . . 156
8.3 Background and Related Work . . . 157
8.3.1 Prioritisation of secure development within organisations . . . 157
8.3.2 Sudden changes in secure development prioritisation . . . 159
8.4 Methodology . . . 160
8.4.1 Interview participants . . . 160
8.4.2 Interview analysis . . . 162
8.4.3 Ethics . . . 163
8.5 Results . . . 164
8.5.1 RQ1: How do management priorities affect software security within organisations? . . . 164
8.5.1.1 If software security is not clearly prioritised, functionality will take precedence . . . 164
8.5.1.2 Software development is fast-moving and needs clear management guidance . . . 165
8.5.1.3 Even when software security is prioritised, compliance may be prioritised over security culture . . . 167
8.5.1.4 It is difficult to impose security as a priority when outsourcing software . . . 168
8.5.1.5 Prioritising software security can have an adverse impact on usability . . . 168
8.5.1.6 Where security is not a management priority, developers are lost, with no guidance . . . 169
8.5.1.7 Where they have not been told to prioritise security, developers often defer to others . . . 171
8.5.1.8 Where software security is prioritised, developers have the resources to code securely . . . 171
8.5.1.9 Some engineering sections cultivate software security expertise without apparent management direction . . . 172
8.5.1.10 Some developers are self-motivated to code securely regardless of management priorities . . . 173
8.5.1.11 Developer security wish lists reveal a desire for greater security prioritisation . . . 174
8.5.2 RQ2: How well do C-suites understand software security? . . . 174
8.5.2.1 It is often assumed by the C-suite that software security is in place . . . 174
8.5.2.2 The C-suite's understanding of software security depends on their background . . . 175
8.5.2.3 C-suites often do not distinguish between cybersecurity and software security . . . 176

8.5.2.4 Top management may ignore software security issues, and may even subvert security practice . . . 176
8.5.3 RQ3: What factors are likely to motivate C-suites to pay attention to code security in organisations? . . . 177
8.5.3.1 Increased visibility and understanding . . . 177
8.5.3.2 Internal and external security breaches . . . 178
8.5.3.3 Meeting client expectations . . . 178
8.5.3.4 Standards and regulatory compliance . . . 178
8.5.3.5 Greater legal obligations . . . 180
8.5.4 RQ4: What factors have participants encountered that had a significant effect on software security in an organisation? . . . 181
8.5.4.1 Good software security practice is expensive in terms of time and money . . . 181
8.5.4.2 Smaller organisations may have fewer security practices in place 181
8.5.4.3 Financial constraints, downsizings, acquisitions and divestitures may affect secure coding practice . . . 182
8.6 Discussion . . . 183
8.6.1 Limitations of this study . . . 187
8.7 Conclusion . . . 187

9 Conclusion **189**
9.1 The Escalating Price of Insecure Software . . . 190
9.2 Overview of Studies . . . 191
9.3 Results of Study A: Reviewing the Literature . . . 193
9.4 Results of Study B: Assessing Organisations . . . 194
9.5 Results of Study C: Motivating Organisations . . . 197
9.6 Implications for Stakeholders . . . 199

Appendices **201**

A Protocol for Literature Review **202**
A.1 Background . . . 202
A.1.1 Previous Reviews . . . 203
A.1.2 Assumptions . . . 203
A.2 Research Questions . . . 204
A.3 Search Strategy . . . 205
A.4 Study Selection Criteria . . . 208
A.4.1 Inclusion criteria . . . 208
A.4.2 Exclusion criteria . . . 208
A.5 Study Selection Procedures . . . 209
A.5.1 Step 1: Search in Digital Libraries . . . 210
A.5.2 Step 2: First Round Exclusions . . . 210
A.5.3 Step 3: Second Round Exclusions . . . 210
A.5.4 Step 4: Snowballing . . . 210
A.5.5 Step 5: First Round Exclusions after Snowballing . . . 212
A.5.6 Step 6: Second Round Exclusions after Snowballing . . . 212
A.6 Study Quality Assessment . . . 213
A.7 Data Extraction Strategy . . . 214

A.8 Synthesis of Extracted Data 216
A.8.1 Synthesizing Assumptions 216
A.8.2 Categorising Assumptions 219
A.9 Dissemination Strategy 220

B Interview Questions **221**
B.1 All Participants: Opening Question 221
B.2 Questions for Developers 221
B.2.1 Demographic information 221
B.2.2 Working environment information 222
B.2.3 Information on specific practices 223
B.3 Questions for consultants with industry knowledge 225
B.4 Questions for senior managers 227
B.5 All Participants: Broader industry 228
B.5.1 Concluding Questions 229

References **230**

Declaration

This is to certify that the work I am submitting is my own and has not been submitted for another degree, either at University College Cork or elsewhere. All external references and sources are clearly acknowledged and identified within the contents. I have read and understood the regulations of University College Cork concerning plagiarism and intellectual property.

Abstract

This thesis considers how to measure and improve secure software development in organisations. The thesis comprises three studies; a literature review, a large-scale survey of software developers, and a study comprising interviews with software professionals.

The work is motivated by the continuing high prevalence of vulnerabilities in software. The proliferation of cybercrime, cyber espionage and other online issues, and their relationship to insecure software, are examined during the literature review study. The literature review also uncovered two main secure-coding influences on software developers; personal attributes such as knowledge and motivation, and environmental factors like secure coding pressure, resources and support. These observations led to the development of the Software Developer Security Archetypes; a two-dimensional framework designed to provide a vocabulary for thinking about software developers and their software security context. Also in this first study, 25 unhelpful assumptions in software security research were identified and documented. These include, that secure-coding activities will be reflected in artefacts, and that findings from a single study are final.

The literature review suggested that some organisations pay lip service to code security without providing the requisite time and leadership support, a phenomenon sometimes called a '*checkbox*' attitude to secure coding. The second study was designed to investigate this contradiction and other aspects of secure development. It entailed a secure coding development survey (n=962). Industry-based research was leveraged to construct a lightweight, empirically-based set of questions to measure practice. A further set of questions grounded in the literature review was included to investigate security culture.

Survey respondents worked in environments with a broad range of secure-coding approaches. Comparison of secure coding practice and culture measurements showed indicators of a checkbox attitude to software security in some organisations. Small organisations, isolated and solo developers and freelance workers used fewer secure development practices, and their secure-coding tool use was limited.

Secure coding requires specific technical knowledge. The answers to secure software training questions indicated that only 39.6% of respondents had been offered secure coding training. When offered, training did not always have the qualities required to make it effective, such as relevance and frequency.

The third study comprised a series of interviews with software developers and senior managers, that sought their views on how software-security prioritisation by senior management affects secure development. The factors that motivate senior management in organisations to prioritise software security were investigated. Interview analysis showed that awareness and knowledge of security, breaches in other organisations, and regulatory and legal obligations were considered organisational software security motivators. This research indicates that increasing the software security obligations of organisations and other entities producing software is essential to increased software security. However, such measures may have unintended consequences, such as the stifling of innovation.

Acknowledgements

Thanks to Ben, Dennis and Molly. You are the best.

Undertaking a PhD was a very positive and enjoyable experience for me. For this I have to thank my supervisors Klaas-Jan Stol and Utz Roedig. Quite apart from their huge erudition and academic input and guidance, they were always accessible, enthusiastic, supportive and good-humoured. I will miss our meetings.

I would like to thank the secure software development research community, whose papers I have pored over, who have produced a wonderful body of knowledge and interesting research, and who are always helpful.

I would like to thank the anonymous reviewers of my published papers for their invaluable insights, criticism and advice, which added greatly to the finished papers.

Finding participants for online surveys and for interviews is notoriously difficult. In this I was helped enormously by other software professionals, to whom I'm very grateful. Many thanks also to the anonymous survey and interview participants themselves. Their insights and humour added richness and depth to the research.

Hugh Kearns has spoken at UCC and to ADVANCE students with practical advice for postgraduate researchers, and his excellent suggestions helped a lot.

I started my PhD in University College Cork as part of a cohort of 30 students, funded by SFI via the ADVANCE program and located at five different institutions around Ireland. Six months into our journey, Covid lockdowns began. Both UCC and ADVANCE worked hard to keep us connected. I would like to thank in particular Ruth Ramsay, Dean of Graduate Studies in UCC, and Stephen Hammel and Claire Collins from ADVANCE.

ADVANCE has been lucky in attracting fantastic administrators. Big thanks to Luis Gómez De Membrillera and James Duggan for their hard work and constructive approach. Undertaking a PhD can be challenging, and they provide wonderful support. They are responsive and helpful to all of us, even though there are over one hundred of us at present.

The other students in my first-year ADVANCE cohort have added so much interest and fun to my research life since we met in January 2020. We were joined by a new group each year. Every time I thought there couldn't possibly be any more equally amazing people out there, another thirty arrived. It's brilliant to have regular contact with peers who are also enduring the inevitable ups and downs of the four-year journey. Ashley and Joan kept me sane through some tough moments. Hugs and thanks to you both, and to so many others. Thanks to Seamus for proposing an alumni network to keep us all in touch.

It all started with Dirk Pesch's vision for the ADVANCE program. Thanks Dirk!

My parents, siblings, in-laws, nieces and cousins were wonderful. With varying levels of understanding of my subject, they provided great encouragement. Thanks also to my wider family and friends. Yes, I've nearly finished that HDip thing.

Diaidh ar ndiaidh a ndéantar caisleán.

List of Tables

2.1 ENISA Cyber Crime Actors and Most Common Activities 2020-2021 . 24

4.1 Unhelpful Assumptions 57

5.1 Gender distribution of survey respondents 95
5.2 Respondent has been Offered Security or Privacy Training 97
5.3 Questions on Twelve Common Activities 99
5.4 Security Culture Questions 102

6.1 Questions on Twelve Common Activities (CAs) 118
6.2 Categories of Developers Considered in this Study 123

7.1 Desirable Features for Secure Coding Training. 145
7.2 Whether Training was Offered in the Work Environment 147
7.3 Types of Comment on Training 148
7.4 Whether training was relevant and interactive. 149
7.5 Training frequency. 149
7.6 Whether training included mentoring, whether it was mandatory. 150
7.7 Developer tone discussing secure-coding training. 151

8.1 Interview Participants 161
8.2 Answers to research questions 184

A.1 List of venues 207
A.2 Details of exclusions in Step 2 211
A.3 Details of exclusions in Step 3 211
A.4 Details of exclusions in Step 5 212
A.5 Details of Step 6 213
A.6 Selected papers per venue 214
A.7 Examples of analyses 217

List of Figures

1.1 Research studies and output 6
1.2 My reflexive thematic analysis (RTA) workflow 15

3.1 The four developer security archetypes 42

5.1 Age distribution of survey respondents 95
5.2 Number of security Common Activities undertaken 99
5.3 Secure coding support from employer or open source team 101
5.4 Questions on being proactive about security 103
5.5 Are there security tools available in your environment? 103
5.6 Is there a security culture in your working environment? 106
5.7 How highly do you think your team prioritises software security? 107
5.8 How often is software security mentioned in team communications? . . 108
5.9 How much time do you spend on software security in an average week? 109
5.10 Grouped Likert of how highly team prioritises software security 110
5.11 Grouped Likert of security mention frequency in team communications 111
5.12 Grouped Likert of time spent on security in an average week 112

6.1 Org size for those who code alone within organisations 124
6.2 Results for the three penetration testing CAs 125
6.3 Results for the two review CAs 126
6.4 Results for the two quality assurance CAs 128
6.5 Results for the two operations CAs 129
6.6 Results for the three security processes CAs 130
6.7 Are there security tools available in your environment? 131
6.8 Heat map of CA answers 132

A.1 Study selection process 209

List of Acronyms

AaaS	Access-as-a-Service
ACA	Arms Control Association
AWS	Amazon Web Services
BIBIFI	Build It, Break It, Fix it
BGP	Border Gateway Protocol
CI	Continuous Integration
CISA	the Cybersecurity & Infrastructure Security Agency
CS	Computer Science
CVSS	Common Vulnerability Scoring System
DCS	Developer Centred Security
DDoS	Distributed Denial of Service
DoJ	Department of Justice
EnCyCriS	the International Workshop on Engineering and Cybersecurity of Critical Systems
ENISA	The European Union Agency for Cybersecurity
EU	European Union
FCC	Federal Communications Commission
GDPR	General Data Protection Regulation
IoT	Internet of Things
MIIT	Ministry of Industry and Information Technology
MOOC	Massive Open Online Course
MS-SDL	Microsoft Security Development Lifecycle
NFR	Non-Functional Requirement
NIST	U.S. National Institute of Standards and Technology

OSI	Open Systems Interconnection
OSS	Open Source Software
QA	Quality Assurance
RTA	Reflexive Thematic Analysis
SAFECode	Software Assurance Forum for Excellence in Code
SAMM	Software Assurance Maturity Model
SO	Stack Overflow
SSD-SES	Secure Software Development Self-Efficacy Scale

My Contributions

This thesis is based on a series of six papers and a paper manuscript. This section lists the papers with their publication details, and describes my contribution to each paper. For clarity, the papers are labelled P1 to P7.

P1 **Ita Ryan**, Utz Roedig, Klaas-Jan Stol. *Insecure Software on a Fragmenting Internet*. 2022 Cyber Research Conference - Ireland (Cyber-RCI). DOI: 10.1109/Cyber-RCI55324.2022.10032675.

My Contribution: As lead author, I researched the paper, wrote the first draft, and contributed to subsequent edits. My co-authors were closely involved at all stages.

P2 **Ita Ryan**, Utz Roedig, Klaas-Jan Stol. *Understanding Developer Security Archetypes*. 2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS). DOI: 10.1109/EnCyCriS52570.2021.00013.

My Contribution: As lead author, I devised the archetypes, wrote the first draft, and contributed to subsequent edits. My co-authors were closely involved at all stages.

P3 **Ita Ryan**, Utz Roedig, Klaas-Jan Stol. *Unhelpful Assumptions in Software Security Research*. Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. DOI: 10.1145/3576915.3623122.

My Contribution: As lead author, I undertook the systematic literature review and identified the assumptions discussed in the paper. I wrote the first draft, and contributed to subsequent edits. My co-authors were closely involved at all stages.

P4 **Ita Ryan**, Utz Roedig, Klaas-Jan Stol. *Measuring Secure Coding Practice and Culture: A Finger Pointing at the Moon is not the Moon*. 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). DOI: 10.1109/ICSE48619.2023.00140.

My Contribution: As lead author, I designed and ran the developer survey, recruited participants, and undertook quantitative analysis on the results. I wrote

the first draft of the paper, and contributed to subsequent edits. My co-authors were closely involved at all stages.

P5 **Ita Ryan**, Klaas-Jan Stol, Utz Roedig. *The State of Secure Coding Practice: Small Organisations and 'Lone, Rogue Coders'*. 2023 IEEE/ACM 4th International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS). DOI: 10.1109/EnCyCriS59249.2023.00010.

My Contribution: As lead author, I undertook the quantitative analysis detailed in the results. I wrote the first draft of the paper, and contributed to subsequent edits. My co-authors were closely involved at all stages.

P6 **Ita Ryan**, Utz Roedig, Klaas-Jan Stol. *Training Developers to Code Securely: Theory and Practice*. 2024 ACM/IEEE 5th International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS) and 2024 IEEE/ACM Second International Workshop on Software Vulnerability (EnCyCriS/SVM '24). DOI: 10.1145/3643662.3643956.

My Contribution: As lead author, I undertook the qualitative analysis outlined in the results. I wrote the first draft of the paper, and contributed to subsequent edits. My co-authors were closely involved at all stages.

P7 **Ita Ryan**, Utz Roedig, Klaas-Jan Stol. *'Money. Or Jail Time.' Improving Software Security by Motivating the C-Suite*. This manuscript has not yet been published.

My Contribution: As lead author, I devised the interview protocol. I conducted and transcribed the interviews. I undertook qualitative analysis on the results. I wrote the first draft of the paper, and contributed to subsequent edits. My co-authors were closely involved at all stages.

Note: the International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS) 2023 also published an additional paper based on my research: '*Studying Secure Coding in the Laboratory: Why, What, Where, How, and Who?*'. This was a side project and is not included in this thesis.

Note on the text

Throughout this thesis, shorter quotes, and quotes within block quotes, are in italics, '*like this*'.

> Longer quotes are in block quotes, like this.

Chapters 1 and 9 of this thesis are not published and will not be submitted for publication. In those chapters, I use the first person singular 'I' to describe the work that I undertook during my PhD program.

The published papers in Chapters 2 to 7 use 'we', as does the manuscript in Chapter 8. Appendix A is available online and also uses 'we'.

1 Introduction

The societal consequences of insecure software are extensive. Over the past few years ransomware attacks have caused financial and operational damage to health services [161] and other critical infrastructure all over the world [300]. Ransomware has crippled businesses, in some cases forcing them to shut down [229]. Democratic parliaments have been subject to cyberespionage [122]. Individuals, including civil rights campaigners, politicians, and journalists have been subjected to intrusive spying campaigns [137].

Many such attacks leverage exploitation of software vulnerabilities [294]. The unifying theme of the publications in this thesis is how the incidence of such vulnerabilities can be reduced. The lens used is a focus on organisations' software security practice and motivations.

1.1 Motivation

When I began my career as a software developer in the late 1990s, software security was not a topic of discussion in the environments I worked in. Critical business processes did not run online. Dedicated private networks were used for back-end data transfer.

From the late 2000s security began to emerge as a concern. However, like colleagues and other developers I encountered, I had no secure-coding training and was only peripherally aware of possible attacks. Becoming interested, I began to pursue software security training and implement security safeguards in my own work. The WannaCry global ransomware episode in 2017 triggered my intense interest in the global implications of poor software security [166]. This increased as I became aware of the extent to which '*digital transformation*' entailed moving everything online, including critical infrastructure. My concern was that it would be people like me, and the other developers I knew, who would be implementing these changes. I could not feel confident that the resulting software would be secure.

Cybersecurity can sometimes be seen as a risk-register concern that should be considered internally by each organisation, allowing poor cybersecurity to be accepted as a risk for profit-maximisation purposes [200]. This analysis does not hold for *software* security. When an organisation distributes insecure software it has adverse implications for all of the organisation's customers. If some of those customers are involved in health, government, critical infrastructure or other activities important to civil society, there are adverse implications for everyone.

I do not approach software security as a problem that can be solved, or signed off as an acceptable risk, individually by each organisation. I see insecure software as a systemic threat that must be urgently addressed by the software industry and by society as a whole for purposes of coherence, privacy and stability.

After two decades of work as a software developer, I am conscious of the ephemeral nature of specific programming tools and languages. External forces such as newer, more efficient approaches eventually overwhelm every technology. Therefore, although tools are important, my philosophy when analysing software security issues is to consider the larger forces at work. My focus is on organisations that produce software, and their larger institutional, societal, and legal context.

1.2 Problem Outline

My initial research question was: *'Why are we programmers so terrible at security, and how can we improve?'*

Before devising a research plan, I reviewed the state of the art. One aspect of the answer quickly became apparent. The major impact that the work environment has on developers' knowledge and priorities is clear from the literature. Secure-coding studies that interviewed, surveyed or observed developers concluded that the organisational or coding environment is a major contributor to how easy or difficult it is for developers to code securely [13, 14, 320, 111, 216, 61]. The results of research on secure-coding aids such as tools and developer interventions cannot be implemented in a vacuum. Software security interventions will not occur if organisations refuse to facilitate them. Developers may wish to use security tools, but may not have access to them. Furthermore, my personal experience suggests that developers are more likely to engage in activities that are prioritised within their work environment.

Therefore, the aim of my research is to investigate the question of how software security is currently addressed in software development organisations, and how it can be improved.

1.3 Research Design

For this thesis I conducted three studies. They can be seen in Figure 1.1, which gives an overview of how the studies fit together. Each study yielded one or more papers, which have been labelled P1 to P7. All have been published except the last paper, P7.

Study A involved desk research; learning from the literature. My initial, informal literature review produced P1, on the consequences of insecure software, and P2, which identified the developer themselves and their work environment as the main influences on secure coding. A systematic literature review identified unhelpful assumptions in software security research in P3. Study B was a large-scale developer survey on contemporary practice, with questions informed by Study A. From the survey data, I reached conclusions on software security practice (P4, P5) and culture (P4). I analysed qualitative training information in the survey for P6. P3 had touched on culture and practice issues that I investigated in P4 and P6. Study C explored these in greater depth via an interview study documented in P7.

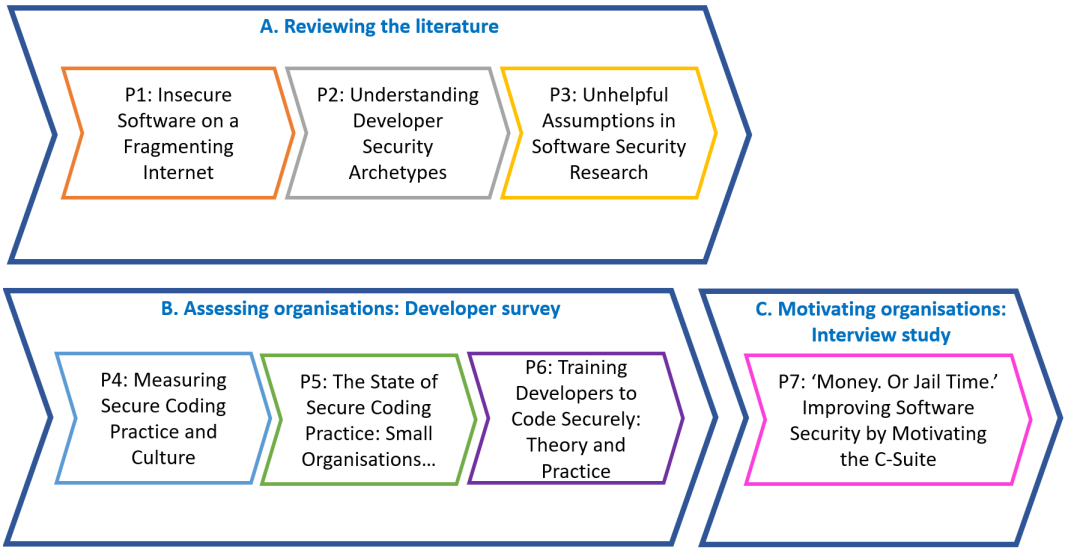


Figure 1.1: **Research studies and output**.

1.3.1 Positionality

With a degree in Philosophy and an MSc in Computing, I have been trained in logic, including the study of logical fallacies. I have a strong technical background, having worked as a professional software developer for two decades, across multiple industries and in local government. I have worked with numerous different languages, platforms, databases and other tools. I have held both permanent and contract positions. Projects have ranged from R&D and creation of new products to the maintenance of legacy software. I have operated within a wide range of formal development methods, including none. I therefore consider myself very much an insider when researching software development.

My coding background includes experience of some of the practical barriers that developers face when it comes to secure coding. This has informed my research direction. I was eager to study the effect on secure-coding practice of poor organisational support, which is discussed in Chapters 3, 5, 6, and 8. Having personally experienced difficulty in accessing secure-coding training, I explored organisational provision of such training in Chapter 7.

Working in organisations where software is developed, I have encountered compliance cultures where secure coding practices are simply a box-ticking exercise. To give one example, I have heard of design documentation being generated immediately before a compliance audit, when it should have been produced months earlier as part of routine development. Chapter 5 examines questions that can be used to detect such a compliance culture in organisations.

As a software development insider immersed in development culture, I have been able to probe a little deeper in interviews than might be possible otherwise. For example, I am interested and engaged when interviewees describe highly technical development details. This helps to forge a rapport, and may have contributed to interviewees having the confidence to confide in me about suspect or questionable practices they have observed.

My extensive practical experience has thus been an advantage when devising survey questions and interviewing other software developers. It also informs and enriches my interpretation of qualitative data. Reflecting this influence, I used Reflexive Thematic Analysis (RTA) [31] for interview analysis in Chapter 8. RTA is a qualitative research technique that foregrounds the specific background and experience that the researcher

brings to bear on a topic, seeing '*researcher subjectivity*' as '*a **resource** for knowledge production*' [31] (emphasis theirs). RTA researchers acknowledge that the framing of the research is heavily influenced by the theorist, from the choice and wording of questions to the development and exposition of themes. Other investigators would necessarily approach the research with a different background and different expertise from mine. Their results would therefore differ from mine in ways that I cannot predict.

In the research field I began my PhD journey as an outsider, although this is changing as I gain research experience. Given my close familiarity with development environments, when reading research papers I sometimes pick up on issues that were not explicitly identified by the researchers themselves. At other times I notice that an obvious explanation for a finding has not been mentioned. However, it could be argued that my experience biases me towards particular solutions and may sometimes blind me to different realities. Therefore, I continue to explore and look for published research that refutes or confirms my ideas. For example, in Chapter 4 I identified 25 unhelpful assumptions made by software security researchers. Although these assumptions seemed suspect to me based on my practical experience, I investigated further to ensure that in each case there was evidence in the literature that confirmed my assessment. This led to verification that one assumption was actually not an assumption but a fact. It was therefore not discussed in the paper.

My unusual researcher status as a software development insider and a research outsider has informed the research direction, interpretation and conclusions of this thesis.

1.3.2 Study A: Reviewing the literature

1.3.2.1 P1: Insecure Software on a Fragmenting Internet

As a software developer, I have been interested in code security for some time, using podcasts, social media and blog posts by relevant experts to stay up to date in the area. I formed a theory that the Internet would inevitably fragment into a collection of local Intranets due to the fragility of having every nation's government and business exposed online to every other nation. P1 involved researching this idea further.

Because this was a position paper I did not feel that it was appropriate to undertake a systematic literature review. Indeed, this would not have been possible given the range of topics covered. In order to write this paper I researched academic and grey literature

on multiple topics. Subjects included current developments at every layer of the Open Systems Interconnection (OSI) model, current cybercrime, and academic writing on geopolitics in the context of Internet fragmentation. Some topics, such as cryptojacking, had very little academic footprint because they were so new.

The geopolitical world has changed considerably in the two years since I wrote this paper, heightening many of the tensions described in the paper and increasing digital fragility. P1 can be found in Chapter 2.

1.3.2.2 P2: Understanding Developer Security Archetypes

At the commencement of my Doctoral program I began to familiarise myself with the history and state-of-the-art of the academic literature, identifying papers of interest and snowballing back and forth to discover new researchers and studies. This is how I encountered the work that was analysed for P2, the second paper in this thesis. This is a position paper, and was my first attempt to articulate the importance of the work environment in secure coding. It places developers in one of four archetypes based on their personal software security interest and knowledge, and the support they get from their work environment. The four developer archetypes provide a useful shorthand for understanding where a developer is positioned in relation to secure coding. See Chapter 3.

1.3.2.3 P3: Unhelpful Assumptions in Software Security Research

Following this initial research, I undertook a systematic literature review of papers related to software security. I was especially interested in aids to code security and their accessibility. The review involved searching for papers that contained information on current aids to software security or barriers to software security that may be present when writing software. The review protocol can be found online [242] and in Appendix A.

For review purposes I read and analysed 109 papers. My analysis of each paper included comments on how the research was conducted, any methodological issues I detected, and my opinion of the reliability of the findings. I also noted when the researchers made assumptions that appeared to me to be unjustified and/or unhelpful. For example, some assumptions lead researchers to underrate organisational influence on

secure coding, or to overestimate the accuracy of artefacts when exploring the secure-coding history of a project.

Aspects of this systematic review have influenced much of my subsequent work. For example, it informed my growing understanding of the fundamental importance of the work environment to secure coding. The specific component of my inquiry that I believed would be of most significance and value to others was the analysis of unhelpful assumptions. I documented these assumptions in P3. See Chapter 4.

1.3.3 Study B: Assessing organisations: Developer survey

While considerable work has been done on developers' secure-coding motivation, the other major influence on secure coding, the work environment, has not been as thoroughly researched. Assessing existing literature, I could not find current work evaluating the extent to which software security practice is undertaken by organisations. There were no large multinational studies examining this question. I believed that the first step to improving organisations' secure-coding stance was to be able to measure it and report on it in a simple and repeatable way. Reviewing options for achieving this, I decided to run a large-scale online developer survey. My rationale for using an online survey was that such a survey is suitable for obtaining results from a large number of geographically dispersed individuals.

To measure practice, I devised a lightweight set of twelve questions on software security practice that are firmly grounded in empirical evidence. The twelve questions ask about the practices that are established in research by Weir et al. as the twelve practices that are adopted first and most often by organisations in industry [323]. Weir et al.'s research analysed the trove of data gathered for BSIMM, an industry-leading secure-coding annual report [171], between 2008 and 2020.

A further set of twelve questions was designed to measure security culture. These questions were based on details from the analysis I had done during the systematic literature review for P3. The security culture questions are designed to probe more deeply into software security in an organisation, and obtain indicators to how highly the organisation prioritises software security.

I asked some other questions on practice, including two questions about training. The survey also included standard demographic questions.

Details of the survey questions, and of the planned interview study (Study C), were submitted for approval to the Social Research and Ethics Committee (SREC) at UCC. SREC approved the application, which was recorded as '*Log 2021-182*.'

The survey ran for three months. In order to ensure a high response rate, I spent an hour online each day soliciting responses via social media. The survey closed with 1100 responses, of which 962 were valid and were used for analysis. This thesis contains three papers evaluating the results of the survey.

1.3.3.1 P4: Measuring Secure Coding Practice and Culture: A Finger Pointing at the Moon is not the Moon

In this paper I did quantitative analysis on the survey results (using R) to assess and compare organisational security practice and culture. The twelve questions on software security practice all allowed the following answers: '*Not applicable*,' '*True*,' '*False*,' and '*I don't know*.' I was interested primarily in those who had replied '*True*.' To provide an overview of software security practice in industry, I summed the totals for all of these common activities for each participant. I called the total the '*CA score*' for the participant. For each participant, the CA score was a value between zero and twelve. Zero indicated that none of the twelve most common secure-coding practices were undertaken in the participant's environment, while twelve meant that all twelve were undertaken. This was a very simple analysis, yielding a single value for each participant. An obvious criticism would be that I asked about only twelve practices from over 120 possible practices identified in the BSIMM. However, Weir et al. found a marked difference between the levels of use of these twelve practices and others [323]. The lowest usage level of the twelve practices in the BSIMM data was 65%. The highest usage level of other practices discussed in the paper was 47%. Weir et al. emphasised that these twelve activities '*are adopted* ***most often, together, and first*** *[emphasis theirs]*.' The range of CA scores can be found in Figure 5.2 in Chapter 5.

I am interested in how well security practice correlates with indicators of security culture. For all twelve security culture questions, I correlated the responses with the CA score using Pearson correlation. The Likert answers for the security culture questions used several different scales, but the Pearson correlation '*is extremely robust with respect to violations of assumptions*' [192].

I am particularly interested in organisations that appear to perform well with regard to adopting common software security activities, but then perform poorly on measurements of security culture. Such organisations are frequently referred to as having a compliance or checkbox attitude to software security. Where organisations have poor security culture, security practice may suffer. I am interested in finding ways to measure security culture that identify the presence of such a checkbox environment.

When analysing survey data it is not possible to establish causal relationships between responses. The most that can be done is to establish a correlation. Correlations can be informative and thought-provoking, and can provide useful directions for further research. In P4, I found a way to visually illustrate correlations in such a way as to highlight the anomaly between practice and culture. I used the likert library in R to display Likert charts for individual culture questions, grouped by CA score. See figures 5.10 and 5.11 for examples. These grouped Likert charts clearly illustrated that factors associated with culture can be lacking even when security practice appears to be excellent.

I did not attempt to reduce answers to the twelve security culture questions to a shorter list, or to a single figure for culture resembling the practice '*CA score*.' I ran factor analysis on the replies and found that three or four factors could be identified, but this reduction did not seem to me to add value to the discussion. Since this is the first work I am aware of to explore security culture questions at scale, I wanted to provoke further thought and exploration amongst fellow researchers. Pointing out three or four factors could imply that the research was complete, the set of questions was finished, and there was nothing new to learn.

My analysis of the results can be seen in Chapter 5.

1.3.3.2 P5: The State of Secure Coding Practice: Small Organisations and 'Lone, Rogue Coders'

I did further quantitative data analysis to explore secure coding practice for freelance developers, developers in small and medium organisations, non-organisational developers, and isolated developers within organisations. This paper broke down the answers to the twelve questions on security practice for these different cohorts. It also examined the level of secure coding tool use. The results can be seen in Chapter 6.

1.3.3.3 P6: Training Developers to Code Securely: Theory and Practice

Secure coding training for software developers has not yet been systematically evaluated. The data from the survey included two questions on software security training. Respondents were asked whether they had been offered security or privacy training in the workplace. If so, they were asked to describe it in free text. This is an area that particularly interests me, because I myself had difficulty sourcing secure-coding training when I was working as a professional software developer.

There were 306 free-text responses to the question asking for a description of training, of which 220 described secure-coding training. (The others described privacy or general cybersecurity training, or were completely off topic). The systematic literature review described in Chapter 4 provided a solid grounding for analysing this free text. I revisited papers that discussed secure-coding training and extracted a set of desirable secure-coding training features from them. These included such items as that training should be frequent (or continuous), relevant and interactive. I then coded the 220 relevant responses, assessing them in light of this set of desirable features. Some responses introduced new secure-coding features, both desirable and otherwise, and I reassessed all other responses for incidences of them. Some participants communicated strong satisfaction or dissatisfaction with their training. I therefore assigned a value for 'developer tone' to each entry. A detailed description of the analysis and its results can be found in Chapter 7.

1.3.4 Study C: Motivating organisations: Interview study

From Study A I determined that organisational support is very significant for secure-coding practice. The results from Study B made it clear that not all organisations provide this support to their developers. That being so, how can all organisations developing software be encouraged to supply secure products to their customers? Organisations tend to focus on profits, and indeed publicly-owned organisations have an obligation to focus on shareholder value, usually equated with profit. What would motivate organisations to spend a portion of their financial and attention budget on software security, which is costly and does not contribute to the bottom line in any obvious way?

I have been unable to find any studies specifically focusing on this question. Organisations are made up of individuals. Priorities are ultimately set by those in upper

management roles, often called the 'C-suite.' I concluded that an interview study would be the best way to gain increased insight into the decision-making and thoughts of the C-suite, which contribute to the prioritisation or downgrading of software security within organisations.

1.3.4.1 P7: 'Money. Or Jail Time.' Improving Software Security by Motivating the C-Suite

For P7, I interviewed 21 individuals working in areas related to software security. A large amount of qualitative data is obtained during an interview study. I had not previously undertaken qualitative analysis at this scale. I researched approaches to coding qualitative data, to determine which fitted the project best. I considered using grounded theory, but ultimately decided on Reflexive Thematic Analysis (RTA) as described and subsequently modified by Braun and Clarke over several different papers [30, 31]. The interpretivist philosophy behind RTA matches my own research philosophy. RTA recognises that the researcher inevitably influences the results of the research, since they interpret the data and generate themes through the prism of their own background, experience and philosophy. The themes are ultimately creations of the theorist. To quote Braun and Clarke, '*you cannot enter a theoretical vacuum when doing TA [thematic analysis]. Paradigmatic, epistemological and ontological assumptions inescapably inform analysis*' [31]. This seems to me to be self-evidently true for research of any depth. The questions asked, and the form in which they are asked, are already heavily informed by the theorist, and influence the likely results of the study. That influence is present before interviews even begin. Embedding explicit recognition of the theorist's influence into the method used appeals to me as an epistemological position.

Figure 1.2 shows the steps I took during RTA, adopting the language used by Braun and Clarke in their 2020 paper on problematic practices in '*(reflexive) thematic analysis*' [31]. In their guidance, Braun and Clarke suggest conducting six phases when undertaking RTA. I found that writing the report entailed continual revision and refinement of existing analysis, so that phases 4, 5, and 6 comprised an iterative process.

Braun and Clarke explicitly discuss that their suggested steps are not prescriptive and are provided primarily to help researchers to find their way. They are also cognisant that there will be revision and reiteration throughout the process. I consider this one of the strengths of RTA. My understanding of the data increased over the period of

writing due to constant review of the original text along with code and themes. Use of RTA meant that my method could seamlessly incorporate continual revision and refinement. Since RTA values the unique context that the researcher brings with them,

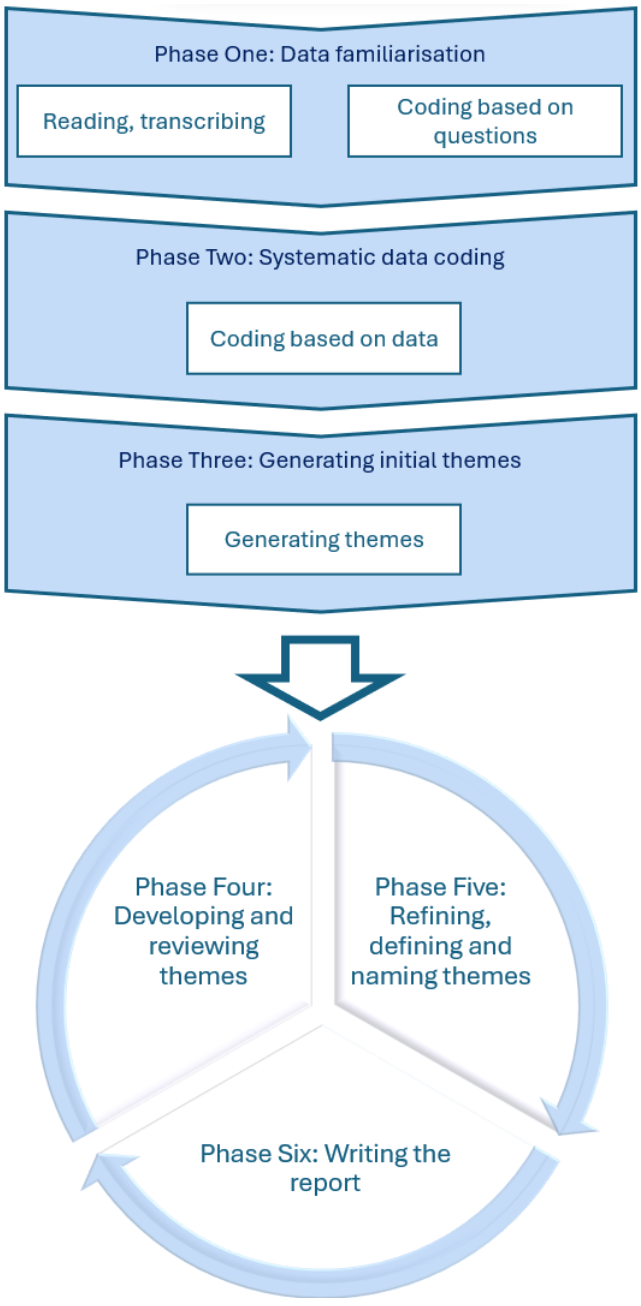


Figure 1.2: **My reflexive thematic analysis (RTA) workflow**.

coding and analysis by a single researcher is preferred. This means that as understanding deepens, it is not necessary to revert to other coders to recalculate coder agreement statistics, reach consensus on rewordings and recategorisations, and so on. This expedites rich understanding of the data, facilitates flashes of insight and allows fast, highly-concentrated work.

I used NVivo as my main tool for data analysis for this study[1]. A worked example by Byrne provided a useful guide to RTA [35].

When the report was complete it was a faithful representation of what I had learned from the interviews. The same data, analysed by a different researcher with their own unique background and experience, would inevitably have resulted in a different report. Nevertheless, based on my extensive experience and research, I believe that the report provides useful, actionable information on the research questions to the research community. It can be found in Chapter 8.

1.4 Thesis Contributions and Structure

This thesis aims to provide a rationale and tools to increase the focus on the organisation in discussions of software security. It describes current issues in software security, including some unhelpful assumptions in software security research. It explores current organisational secure-coding support, and aspects of how organisations support developers to code securely. It considers factors that motivate senior management in organisations to prioritise software security. It evaluates pressures that cause a downgrade of secure coding in organisations. This thesis demonstrates the need to reduce organisations' incentives to prioritise profit over customer security.

The contributions of the papers comprising the thesis are described in this section.

Chapter 2 presents the backdrop of the work, teasing out the original motivation. It contributes a wide-ranging analysis of current cybercrime and other online threats, their relationship to software security, and some of their geopolitical implications.

Chapter 3 begins the search for software security solutions. In this chapter I focus on developers and the factors influencing their secure coding. The resulting contribution is the concept of Developer Security Archetypes which characterise developers by their own interest in security, and the security support they receive in their work environment.

[1] https://lumivero.com/products/nvivo/

These archetypes provide a useful vocabulary for considering individual developers' secure-coding position.

Chapter 4 reviews an extensive body of literature on software security, and contributes analysis of a set of 25 unhelpful assumptions sometimes made by researchers. These assumptions can affect the researchers' interpretation of the phenomena they observe. Awareness of the assumptions can help researchers to take them into account in their own work.

Chapter 5 documents a large-scale developer survey designed to measure software security practice and culture. To measure practice, I devised a lightweight scale based on empirical evidence from industry. To measure security culture, I created a set of questions grounded in prior literature. The contributions of this chapter are the sets of questions on practice and culture, and findings on the state of practice and culture in organisations that develop software.

Chapter 6 explores secure-coding practice data further to obtain information on current software security practice for vulnerable developers such as lone, isolated or freelance developers.

Chapter 7 analyses existing literature to find the desirable features of secure-coding training. It then evaluates the answers to training questions in the developer survey to determine if these features are present in contemporary secure-coding training. This chapter contributes a list of desirable features of software security training, and an assessment of whether these features are currently present in software security training in industry.

Chapter 8 explores the effects and motivations of senior executives' prioritisation of software security within organisations. Such prioritisation is an important factor in secure-coding practice and culture. This chapter contributes a contemporary analysis of the influence of upper-management prioritisation of software security on software-security practice, an analysis of what motivates upper management in organisations (the '*C-suite*') to prioritise software security, and an assessment of the effect of organisational turbulence on software security.

Chapter 9 concludes the thesis. In this chapter I discuss the findings of the papers in this thesis. We conclude with an assessment of the implications of the work.

2 Insecure Software on a Fragmenting Internet

This chapter was published as: Ita Ryan, Utz Roedig, Klaas-Jan Stol. 'Insecure Software on a Fragmenting Internet.' *2022 Cyber Research Conference - Ireland (Cyber-RCI)*. DOI: 10.1109/Cyber-RCI55324.2022.10032675.

The original submission was accidentally sent to IEEE instead of the camera-ready version. The camera-ready version is available on UCC's Cork Open Research Archive at `https://cora.ucc.ie/items/0032e7b0-f6be-4920-9f6f-9d88edc06e11`. That is the version that is included here.

2.1 Abstract

Global geopolitical forces are pushing much of the world towards Internet nationalism, threatening to turn the Internet into a 'Splinternet.' In this paper we argue that the crisis in software security will exacerbate this trend. We examine existing moves towards Internet fragmentation on multiple levels. We discuss current trends in online crime, espionage, and warfare. We look at the role of software vulnerabilities, discussing how the prevalence of software security issues could propel nations further apart. We argue that there is an urgent need for a 'zero tolerance' attitude to software security issues, and discuss what is needed to create this.

2.2 Introduction

With its elegant protocols and built-in redundancy, the Internet is inherently global in nature. Nevertheless, it is not immune to geopolitical forces. Inter-country fragmentation is happening on several different levels, and has been referred to as the 'Splinternet' [118].

Ubiquitous access to the Internet means that software flaws can be exploited remotely from anywhere, with local law enforcement having no jurisdiction in the country from which a crime was committed. Thus, the Internet facilitates previously unimaginable scenarios like the May 2021 ransomware attack on the Irish Health Service Executive [114]. Similarly, espionage, sabotage, and cyberwarfare can be conducted remotely, providing hostile forces with unprecedented access.

Secure software is a core cybersecurity concern. While firewalls, anti-virus tools, network segmentation, and other tools and strategies are deployed to protect digital assets, software defects and design flaws can provide attackers with a back door. It is impossible to prove the security of non-trivial software. Indeed, severe implementation flaws have been found in firewalls [144], anti-virus tools [68] and network segmentation tools [140] themselves.

The number of newly reported software vulnerabilities increases each year [193]. Efforts to tackle software security issues are haphazard. Until very recently there was little government guidance, and organisational software security drives in unregulated industries are entirely voluntary. While critical domains use regulations often based on U.S. National Institute of Standards and Technology (NIST) guidelines, these guidelines are rather heavy-weight, and thus unsuitable for most organisations. We argue that a rapid escalation of effort in eliminating software vulnerabilities is needed. Otherwise, exploitation will continue to increase, exacerbating the trend towards Internet nationalism. The vulnerability of military and critical infrastructure and of nuclear control software to remote exploitation will be seen as too much of a risk.

Previously, Claessen [52] discussed how the understanding of cyberspace as a military as well as civilian domain has led to increasing attempts to impose state sovereignty on the Internet, with particular reference to the different approaches adopted by Russia and the European Union (EU). Hoffman looked at how the new technical standards proposed by China could lead to Internet fragmentation [118]. In this paper, we contribute to this line of work by examining contemporary pressures on a cohesive Internet, explore the forces that are driving the Internet to fragment, and consider how untamed software security risk adds to those pressures. We advocate for a new culture of software insecurity intolerance.

In Section 2.3 we look at drivers towards Internet nationalism and ways in which countries are currently uncoupling from a cohesive global Internet. In Section 2.4

we discuss large scale security issues that the Internet facilitates. In Section 2.5 we examine how software vulnerabilities impact on cybersecurity. In Section 2.6 we discuss approaches to reducing software vulnerabilities, and some exacerbating factors. The conclusion in Section 2.7 discusses possible global consequences of a failure to improve software security.

2.3 Internet Nationalism

Internet fragmentation is an existing phenomenon driven by perceived national interest and facilitated by design choices on different Internet layers. We first discuss the different layers in which changes are happening. We then briefly discuss how some countries are diverging on multiple levels.

2.3.1 OSI Model Layer 1: Physical

Approximately 95% of global Internet traffic travels through undersea fibre-optic cables, which comprise the Internet's backbone [70]. Cables are increasingly perceived as relevant to geopolitical tensions [33]. Russian naval exercises off the Irish coast in January 2022 focused minds on the vulnerability of transatlantic communications cables, damage to which would severely impair Irish and European Internet connectivity [96]. Underlining the fragility of the world's Internet connectivity, in January 2022 Tonga's external communications were almost completely cut off after a volcanic explosion severed the single undersea cable connecting it to Fiji [70].

Russia recently decreed that its transnational cables must be registered with a central authority [79]. Data on transnational cables is already collated and made public in the U.S. [86]. U.S. researchers recently mapped crucial internal cables in a project funded by the Dept. of Homeland Security [259].

There is concern about espionage via physical cable access [271], with China-funded cables increasingly regarded with suspicion [93]. The U.S. Department of Justice (DoJ), in 2020, objected on national security grounds to a new undersea cable connecting the U.S. to Hong Kong [257].

2.3.2 OSI Model Layer 2: Data Link

The U.S. have banned use of Chinese company Huawei's technology in 5G networks, citing security concerns [37]. Four other Chinese tech companies have also been deemed security threats by the U.S. Federal Communications Commission (FCC) [74]. Russia mandates use of local technology for key Internet controls [79]. As each country moves to using only local suppliers, commonality declines and the feasibility of standards and protocols diverging increases.

2.3.3 OSI Model Layer 3: Network

Communication between networks is often done via Internet Exchange Points (IXPs) where multiple network endpoints are located in close proximity, using Border Gateway Protocol (BGP) records to move traffic directly between networks. This may be done for example to avoid transit fees [54]. BGP can be used to prevent a nation's Internet traffic from travelling through another nation's territory. Russia has directed that Internet traffic should only be directed through approved IXPs registered with Roskomnadzor [52]. This policy is likely to keep Russian Internet traffic within the country.

Because BGP is the protocol that allows networks to find destinations, it can also be used for censorship. Pakistan accidentally propagated an incorrect YouTube destination to the global Internet when it banned YouTube in 2008 [163]. Ververis et al. [310] found that BGP configuration is one of the most widely-used tools for Internet censorship. Limonier et al. found that, over the six years prior to 2020, Internet traffic from disputed Donbas in Ukraine shifted to being routed almost entirely through Russia [148]. They concluded that routing can reflect geopolitical concerns.

2.3.4 OSI Model Layer 4: Transport

All Internet traffic currently uses TCP/IP. While the transition from IPV4 to IPV6 brings its own fragmentation concerns [71], China has proposed a 'decentralised Internet' model and associated entirely new protocol named New IP. It argues that the 50-year-old IP protocol is creaking under today's massive Internet use and new communication needs for technologies like virtual and augmented reality [40]. New IP facilitates centralised surveillance and control of the Internet, and is seen by some as entailing the loss of individual freedom to the state [118]. It is suggested that the protocol will not be

adopted by the U.S. or its allies and that this could lead to a fragmentation into at least two separate versions of the Internet, with different countries or blocs using their own protocols. Hoffman et al. [118] note that, although involvement in Internet protocol standards committees is resource-intensive and expensive, nations should participate in order to ensure that their values are reflected.

2.3.5 Data, Applications and Access

China, Russia, and other states require data pertaining to their citizens to be stored within their borders [270]. The EU only allows data to be held overseas if certain privacy and protection guarantees are followed. Data localisation allows states to ensure that their data remains within their jurisdiction, but it also contributes to fragmentation.

Many countries have banned or restricted other countries' websites and applications for reasons of censorship, privacy or national security. For example, China's 'Great Firewall' prevents the use of Twitter, Facebook, Google, Signal and numerous other applications [258]. In 2020, India banned over 200 Chinese apps including Baidu, WeChat and Alipay, citing national security and surveillance concerns amid escalating border tensions [113]. Russia's February 2022 invasion of Ukraine was swiftly followed by a ban on Instagram, Facebook and other sites due to 'extremist activities.' In March 2022, the FCC added Russian anti-virus organisation Kaspersky, already banned from U.S. government networks, to its list of firms posing a security threat [98].

Governments may use strategies to control the flow of information over the Internet [94], often to limit foreign content. In a 2020 global, longitudinal study of Internet censorship, Niaki et al. [190] found the most censorship overall in Iran, South Korea, Saudi Arabia, Kenya, and India. India is the world's largest democracy, a reminder that censorship is not the sole preserve of authoritarian regimes. Conservative countries may resist open access to pornographic or gambling sites, seeing these as conflicting with national values.

Removal of Internet access is a favourite tool of oppressive regimes in times of turmoil. For example, most Internet access was lost for three days during the 2016 general election in Uganda [73]. In Belarus, where all Internet access is government controlled, there was a 61-hour Internet blackout during protests against a disputed Presidential election result in August 2020 [123]. In Myanmar in February 2021, new cybersecurity laws were

introduced allowing mass censorship and surveillance after a military coup. In January 2022, Kazakhstan was subject to an Internet blackout amid anti-government protests about fuel charge hikes [23]. Those are a few examples among many; the Access Now activist group estimated that there were at least 155 Internet shutdowns in 29 countries in 2020 alone (https://www.accessnow.org/).

2.3.6 Multi-Level Divergence

China's Internet is a model for all nations, like Russia, that want to be able to disconnect from the global Internet at will. It has no foreign telecommunications companies within its borders. External connections are made via cables that pass through the 'Great Firewall,' leave China, and connect with external IXPs on foreign soil [44].

Having banned most U.S. apps, China has very successful social media apps of its own. Chinese government organisations were ordered to remove foreign hardware and software from their offices by the end of 2022 in a 2019 edict [45]. This move away from reliance on computers and software developed by the U.S. and its allies, along with the Intranet-like nature of China's Internet, its use of the 'Great Firewall' and its drive to replace IP with New IP show that China is splitting from the global Internet on multiple levels.

Russia has been attempting to emulate China and modify its Internet (the 'RuNet') to remove dependence on external connections at every level. For example, in 2017 the Russian Security Council launched a process for developing a parallel DNS service [235]. A 2019 law mandated installation of local apps on devices sold in Russia [21]. Successful tests of RuNet independence were reported in 2019 and 2021 [132].

Subsequent to the Russian invasion of Ukraine in February 2022, many foreign service providers withdrew from the Russian market [120]. Others were banned by Russia. The Ukrainian representative at ICANN requested that top-level Russian domains and certificates be revoked by ICANN [85]. ICANN refused this unprecedented request. In early March 2022, the rumour that Russia would disconnect itself entirely from the global Internet on March 11, apparently based on a Kremlin document on preparing for separation, was widespread [206]. Some commentators suggested that this would presage an all-out cyberattack on the U.S., or the cutting of transatlantic cables by Russia.

Table 2.1: ENISA Cyber Crime Actors and Most Common Activities 2020-2021

Level	Description
State-sponsored actors	Malware Espionage Supply chain compromise Disinformation\misinformation Cybercrime for monetary gain Sabotage (targeting of industrial control systems) Cyber arms race
Cybercriminals	Ransomware Cryptojacking Malware Cybercrime-as-a-service DDoS, Web Attacks
Hacker-for-hire actors	Access-as-a-Service
Hacktivists	DDoS Sensitive data release Account takeovers

Calls for Russia to be disconnected from the Internet, and rumours that it will disconnect itself, are still circulating at time of writing in April 2022.

2.4 Security Issues for a Global Internet

Metcalfe's law states that the value of a communications network is proportional to the square of the number of connected users of the system [50]. However, it has been shown that an increase in the number of connected users also increases risk, which in turn diminishes value [38]. In this paper we argue that the uncertainty and fragility caused by widespread insecure software is likely to add further pressure to a global Internet infrastructure that is already fragmenting. We base this argument on the fact that insecure software facilitates crime, espionage and sabotage across borders. In this section we discuss the top crimes and threats from the Threat Landscape report issued by The European Union Agency for Cybersecurity (ENISA) [84], which covers the year prior to July 2021. Published in October 2021, the report lists the main threats encountered

and defines four categories of threat actor. Like the ENISA report, we do not consider localised issues such as those related to intimate partner abuse and cyberbullying, because those very real risks are not primarily international. Having discussed threats defined by ENISA, we add cyber patriotism and cyberwarfare.

2.4.1 Threat actors in the ENISA report

The ENISA report defines four different threat actors.

2.4.1.1 State-Sponsored Actors

The report (see Table 2.1) describes a rise in cyberespionage related to Covid-19, with state actors observed searching for information on national Covid-19 responses and treatment. Healthcare and medical research sources were targeted. Supply-chain compromises were significant, in particular the highly sophisticated SolarWinds SunBurst breach [325]. State actors were observed engaging in money-making activities such as cryptojacking, perhaps partially to disguise breaches as cybercrime.

Both defenders and state actors raised their game in the reporting period, with numerous joint declarations and legal stratagems. State actors showed increasing levels of sophistication. 'False flags' were sometimes used to muddy attribution, and hack-and-leak campaigns were used for strategic gains.

2.4.1.2 Cybercriminals

Covid-19 was used by cybercriminals in multiple phishing campaigns preying on concern about the virus. The report notes increased collaboration and professionalism, a move to the cloud and an increasing tendency to attack critical infrastructure. The report mentions the 'Cybercrime-as-a-Service' trend, wherein services for cybercrime are commoditised and broken down; it is possible to purchase access to victim servers from one dark web supplier and run ransomware on them which has been purchased from a different supplier. Many other services are offered in this ecosystem. Since it is global, hackers in one country can sell their services to cybercriminals in another.

2.4.1.3 Hacker-For-Hire Actors

The ENISA report described the Access-as-a-Service (AaaS) market. Commonly known as spyware, AaaS allows the user to access the contents of a victim's phone, potentially including the microphone and camera. The report predicted that this sector will be subject to increasing regulation on human rights as well as national security grounds. This prediction has been borne out by events. In November 2021, the U.S. blacklisted well-known AaaS firm NSO group [22], and Israel drastically reduced the number of countries to which cyber-weapons could be exported [48]. The technology continued to cause controversy in 2022, with a stream of revelations including the discovery in February that Israel had used NSO spyware against some of its own public figures [230]. In April, use of NSO spyware for surveillance of Jordanian human rights defenders was revealed [90].

2.4.1.4 Hacktivists

Early hacktivism was generally associated with idealistic left-wing anti-corporate ideology. Hacktivists use cyberspace for activities related to political activism in the real world, aiming to increase awareness or to cause reputational damage to organisations. Hacktivism is typically not done for financial or material gain [212]. The ENISA report finds low current levels of hacktivism, but anticipates a possible rise in the future as environmental issues come to the fore. It notes that hacktivism can be faked by nation state actors to confuse attribution for subversive activities.

In October 2021, protests in Belarus over the disputed re-election of Alexander Lukashenko were accompanied by hacktivist activity, including the theft and release of information revealing the identities of Belarussian security agents [36].

2.4.2 Cybercrime threats in 2020-2021

2.4.2.1 Ransomware

Ransomware is the practice of encrypting the files on an organisation's devices and demanding a ransom for the decryption key. Since CryptoLocker first appeared in September 2013 [104], ransomware has become increasingly sophisticated. Recent escalation tactics include using Distributed Denial of Service (DDoS) [269], and threatening

to expose sensitive data, including embarrassing data from the devices of organisational decision-makers [49]. Some hackers search networks for details of cybersecurity insurance coverage amounts, tailoring their ransom requests accordingly [104]. With an estimated $590 million of ransomware payments made in the first six months of 2021 [299], by November an insurance backlash had begun, with rises in premiums of up to 300% and steep falls in amounts covered [55].

Ransomware crews make their expertise available to franchisees in what is known as a Ransomware-as-a-Service model [168]. They take precautions to ensure that franchisees do not launch attacks in their home countries, often automating a check of the installed language on a system before file encryption [191]. Security journalist Brian Krebs suggested that installing certain Eastern European languages on a computer could provide protection against some ransomware strains [141].

The ENISA report describes how zero-day vulnerabilities, generally bought by nation-state actors, were in 2021 often used in sophisticated attacks on small numbers of very high-value ransomware targets, a practice known as big game hunting.

In June 2021, the U.S. government raised the priority of ransomware to the same level as terrorism [255]. Subsequent initiatives such as the international 'Counter Ransomware Initiative' [275] sought to improve international ransomware prevention and response. Priorities were increasing resilience, disrupting illicit finance and jurisdictional arbitrage, and improving international cooperation and diplomacy to encourage states to address ransomware operations within their own territories [283]. A series of arrests and forum shutdowns by Russian authorities in January and February 2022 was considered a change in Russian policy towards ransomware and other cybercrime [311]. Whether a conciliatory gesture towards the U.S. [254], or an attempt to keep China, also experiencing severe ransomware incursions [292], onside, enforcement diminished after Russia invaded Ukraine.

Industry commentators in early 2022 observed increased use of ransomware by nation state actors, such as a January fake ransomware attack on Ukrainian government sites, concluding that the ransomware cover provides deniability to an attacking state [39].

2.4.2.2 Cryptojacking

Often seen as a relatively victimless crime, cryptojacking is the practice of surreptitiously mining cryptocurrency on a user's device. When done at scale it can be lucrative

[208]. ENISA reports that cryptojacking incidence was at its highest ever in the first quarter of 2021. It suggests that the rapid increase of cryptojacking and ransomware is facilitated by the ease with which they translate to financial gain, facilitated by the use of cryptocurrencies.

2.4.2.3 Other Cybercrime

While cryptojacking and ransomware require large-scale networks to function, there is also plenty of traditional crime on the Internet. In an analysis of the 'Digital Goods' or 'Services' dark web sales categories, Meland et al. [168] report that credit card fraud ('carding') is the most popular crime. Carding involves the bulk selling of credit card data, sometimes with card holders' personal details [72]. Stealing and selling credit card information at scale is easier online.

2.4.2.4 Cyberespionage

Cyberespionage is now an accepted part of geopolitics. Between December 2020 and February 2021, national infrastructure cyber-intrusions were reported by Finland (parliamentary email) [46], Japan (military contractor) [273], Malaysia (Armed Forces website) [274] and Ukraine (government document sharing) [186], to take just a few examples. In early 2021, intrusions on U.S. and other government networks via Sunburst (SolarWinds) and other supply chain attacks caused concern about the risk of cyberespionage. However, experts in the field expressed the view that this was merely traditional international jostling [325].

2.4.2.5 Cyber Patriotism

Not mentioned in the ENISA report, which concluded observations in mid-2021, there has been an outbreak of activity from what Recorded Future's Allan Liska calls 'cyber patriots' as a result of Russia's invasion of Ukraine. We distinguish cyber patriotism from hacktivism on the basis of its nationalist origins. Sharp divisions have occurred within cybercriminal groups that contained members from both Russia and Ukraine [287]. Many hacker groups have taken sides, vowing to leverage their skills to further their country's cause [304]. Others have also acted. After the notorious Conti ransomware group

announced its support for Russia, a Ukrainian researcher, who had lurked on Conti servers for years, leaked thousands of documents containing their internal communications [157].

Cyber patriotism has had a direct impact on software security. Some software component projects on GitHub have been modified to become 'protestware,' displaying banners like 'Stand with Ukraine,' or facts about the invasion. In one case, the popular 'vue-cli' framework had a component added that deleted all files on its host computer if it detected that it was running in Russia or Belarus. Brian Krebs reports concerns that such activities would '*erode public trust in open-source software*' [142]. Since blind trust in open source components is not conducive to software security, we argue that this might be a good thing.

2.4.2.6 Cyberwarfare

As it moves online, infrastructure is increasingly vulnerable to cyber outages. These can be caused by natural phenomena such as hurricanes. They can be collateral damage from criminal cyber activity, as the Colonial pipeline outage in the U.S. in May 2021 was. They can also be the result of actions by a hostile state. Cybersecurity organisation Recorded Future documented a large increase in suspected intrusion activity in India by Chinese state-sponsored groups during border tensions in 2020. Recorded Future stated that India's power sector and two seaports were targeted in a *'concerted campaign against India's critical infrastructure.'* Severe power outages in Mumbai on October 12 2020 were attributed to Chinese sabotage by Anil Deshmukh, a minister for Maharashtra state. China disputes the claim, but the fact that it was made at all reflects the uncertainty engendered by the mere possibility of cyberattack.

In 2010 the Stuxnet worm, widely attributed to Israel and the U.S., attacked industrial control systems in Iran. The Natanz uranium enrichment site was badly damaged, even though the Natanz network was supposedly air-gapped from the Internet. Stuxnet is considered to be the world's first cyber-weapon [19].

Prior to the February 2022 invasion of Ukraine, Russia-Ukraine history showed a gradual escalation from cyber-warfare to kinetic warfare. The electric grid in Ukraine was attacked on December 23rd 2015. In an incursion attributed by the DoJ to Russia's GRU [298], 30 substations were taken offline and power to 230,000 people in freezing temperatures was lost for up to 6 hours. There were related outages the following year. In 2017, an accounting tool used by approximately half of the businesses in Ukraine was

infiltrated with fake ransomware in what became known as the NotPetya attack. There were huge financial costs to business. The Merck pharmaceutical company lost $1.4bn [332]. This incident was attributed to the Russian state by the UK government [288], but the apparently criminal method of attack allowed for plausible deniability. It was hugely destabilising in Ukraine, and served as a warning to international organisations considering doing business there, signalling that perhaps it would not be worth the trouble [252]. In 2020, the DoJ indicted six Russian nationals for the Ukrainian power cuts and the NotPetya attack, among other alleged crimes [298]. An unintended victim was the insurance industry, forced to contend with geopolitical questions around attribution and 'act of war' definitions in its attempts to avoid payouts [332].

In the build-up to the Russian invasion, cyberattacks on Ukraine increased, with data wipers disguised as ransomware [50], DDoS, bot farms spreading misinformation [51] and widespread infrastructure attacks [336]. A cyberattack on the day of the invasion on Viasat KA-SAT routers used in Ukrainian military communications had an impact on other European countries, with monitoring and control of wind turbines in Germany rendered unavailable [107]. Cyberattacks continued after the invasion [336]. Meanwhile, western officials warned amateur hackers against joining the voluntary 'IT Army of Ukraine,' organising on Telegram.

In a discussion on cybersecurity threat escalation on the website of the Arms Control Association (ACA), Michael T. Klare describes the inherent danger that a cyberattack on Nuclear Command, Control, and Communications (NC3) facilities would justify a nuclear response. The ACA views this as an unacceptable risk, suggesting that even the fear that NC3 facilities were under attack could trigger an escalation to the use of nuclear weapons. If tensions were high enough, even a simple power outage could cause a national leader to feel that their nuclear capability was in imminent danger. This could propel them into striking first [139]. The advent of cyber patriot vigilantes, some of them expert hackers, increases the risk of such an unanticipated outcome.

The danger that cyber incidents could cause escalation to kinetic warfare was raised by U.S. President Biden in 2021 [29]. It is likely to be considered by every nation when assessing the pros and cons of unfettered access to a global Internet.

2.5 The Role of Software Vulnerabilities

We have discussed some of the forces pushing nations to separate from the global Internet, and outlined some threats likely to accelerate that process. We now turn to the role of software vulnerabilities in exacerbating those threats. A perusal of material from hacker training sites such as Bugcrowd University [1] indicates that the discovery and use of software vulnerabilities is at the core of hacking techniques.

Vulnerabilities are mitigated by software patches. In a survey by *BAE Systems Applied Intelligence*, reported in May 2021 [179], 52% of recent security incidents were caused by missing patches. The mean time to patch was 205 days [105]. Patching strategies are complicated by the fact that software normally contains multiple Open Source Software (OSS) components which may themselves include other libraries. One third of studied vulnerabilities in OSS were present for over three years before remediation [147]. December 2021 brought this issue to the fore with the publicising of the Log4j bug, in which a little-known feature of a ubiquitous Java logging component was discovered to be vulnerable to remote code execution [293].

Unfortunately, vulnerable systems are easily discoverable online. Actors wishing to exploit the latest defects can run tailored searches via sites such as Shodan [16], which will find and list Internet-facing systems with specified characteristics. Failure to patch is discoverable.

Not all software vulnerabilities are equal. In the U.S., NIST maintains the National Vulnerability Database, which collates reported software vulnerabilities and assigns a Common Vulnerability Scoring System (CVSS) score to them, with a 'Critical' 10.0 being the highest score available. High CVSS scores indicate that a vulnerability is simple to exploit, remotely available, and likely to result in a severe impact on the vulnerable system. The term 'zero-day' is used to describe a critical software vulnerability that has not yet been patched, may not be generally known about, and possibly has not even been reported to the software manufacturer. Zero-days for popular software are much in demand and can be bought on the dark web [234]. National security agencies are known to stockpile zero-days for use in cyberespionage [327].

In April 2017, the 'Shadow Brokers' published a number of hacking tools widely reputed to originate with the U.S. National Security Agency (NSA) [187]. These leveraged

[1] `https://www.bugcrowd.com/hackers/bugcrowd-university/`

serious bugs such as the Eternal Blue exploit (CVE-2017-0144), which the NSA had reputedly used for several years [327]. Microsoft had been notified of the theft of the exploit, and released a patch for Eternal Blue a month before it was published [253]. Nevertheless, sufficient machines remained unpatched for the WannaCry ransomware cryptoworm attack of May 12 2017 and the NotPetya attack of June 27 2017 to cause worldwide havoc. The Eternal Blue SMB exploit allowed WannaCry and NotPetya to spread and self-propagate without any user intervention [166, 28]. Until it was patched, Eternal Blue was present on all versions of Windows from at least Windows 2000.

Another long-lived critical Microsoft defect was the 'Zerologon' elevation-of-privilege bug. Quietly patched in August 2020 and made public the following month, it infected Windows Server 2008 and all newer versions of Windows Server up to 2019 [272]. The persistence of Eternal Blue and Zerologon for over a decade after Microsoft mandated internal use of Microsoft Security Development Lifecycle (MS-SDL) is a reminder that there are no software security silver bullets.

The relatively collegiate international atmosphere that had surrounded defect discovery and notification began to change in 2018, when the Chinese government banned Chinese security researchers from participating in vulnerability discovery competitions such as CanSecWest's Pwn2Own [27], in which they had previously been highly successful. In 2021, the Cyberspace Administration of China introduced rules forbidding the sale of vulnerabilities or the notification of vulnerabilities to overseas entities other than the manufacturers. Organisations discovering vulnerabilities in their own code must notify them to the Chinese government within two days [47]. For entities trading within China, this could put them in a position of having to notify the Chinese government about vulnerabilities before a patch is in place. Organisations are 'encouraged,' though not obliged, to notify the government first about vulnerabilities discovered in other organisations' code. In December 2021, AliBaba Cloud was suspended from an information-sharing partnership with China's Ministry of Industry and Information Technology (MIIT) for failure to notify it about the Log4j vulnerability. AliBaba staff notified Apache on November 24, while the MIIT was not notified until December 9 [256].

Considered in light of the move by some countries to use only homegrown software internally, these developments could presage a time when foreign adversaries are familiar with the software used by the U.S. and the EU, and its vulnerabilities, while the reverse is no longer true.

2.6 Attempts to Remediate

Having seen the impact of software vulnerabilities on software quality, we now look at approaches in industry and academia to reducing software vulnerabilities. We consider some of the shortcomings of existing approaches and suggest some reasons why they are not effective. We also discuss recent legislation relating to software security in Europe and the U.S.

2.6.1 Industry

Focus on software security in industry varies depending on the industry involved. In the U.S., NIST publishes comprehensive cybersecurity guidelines. Revision 5 of NIST Special Publication 800-53, 'Security and Privacy Controls for Information Systems and Organizations' was published in September 2020 [301]. The guide is used by safety-critical industries; for example, U.S. Nuclear Regulatory Commission Regulatory Guide 5.71, on cybersecurity programmes for nuclear facilities, used NIST 800-53 version 3 to provide a comprehensive cybersecurity approach [302]. Weighing in at a hefty 465 pages, version 5 provides descriptions of numerous cybersecurity controls but includes a mere four pages on 'Developer Testing and Evaluation.' This software development section outlines nine activities that would be familiar to most software security advocates.

Software security methodologies in general use in industry include MS-SDL, Software Assurance Maturity Model, Building Security In Maturity Model, Common Criteria and various ISO standards. All coalesce around a number of activities which are regularly synthesised in academic papers [13][175], such as threat modelling, use of analysis tools and penetration testing. However, the software development security industry is currently convulsed by software developers' move away from regular, relatively infrequent releases, to which blocking 'security gates' can be applied, to automated continuous releases. This move is facilitated by the DevOps emphasis on comprehensive automated tests. DevSecOps attempts to bring security into the DevOps approach, adding security tests to the automated test suite and automating security gates. Advocates of DevSecOps suggest that it 'shifts security to the left,' making it an issue that architects and developers must consider instead of something that is assessed just before release. Done well, DevSecOps can add predictability and credibility to a development team's security stance. However, practitioners express concerns about whether comprehensive security checks can ever

be fully automated [279]. Done badly, DevSecOps adds little value and can even have a negative impact on a development process [174].

2.6.2 Academia

Much work has been done in academia on how software can be made more secure. Wurster and von Oorschot [333] argued that software developers, though seen as warriors in the forefront of the battle for secure software, are in fact part of the problem since they have multiple, often conflicting, priorities and are rarely security experts. They suggested that developer tools should be created with usability in mind, and should make it difficult for developers to code insecurely. They pointed out that developer security training was often still advocated as the solution for developers. They identified an issue with developers who are either unaware of, or who ignore, new security technologies, and noted that security technologies which must be independently run by developers (i.e. they are not embedded in standard tools) will not be run by all developers. They advocated 'security mechanisms which are invisible to the application developer.' This theme was developed by Xie et al. [335], who looked at why programmers make security errors, concluding that developers often feel that someone else is responsible for security and it is not their concern. Acar et al. [4] discussed how 20 years of lessons learned from usable security work can be applied in security research with software developers, and derived a research agenda on these lines. Green and Smith [102] discussed simplifying security APIs to make them less impenetrable to programmers, proposing ten principles for creating secure and usable crypto APIs.

A recurring research theme is that developers, who have other priorities, lack the training and expertise necessary for security proficiency. Weir et al. [320], the Motivating Jenny team (https://motivatingjenny.org/) and others have looked at interventions and tools to help teams to code securely. However, in a 2020 review of top U.S. Computer Science (CS) undergraduate courses, Almansoori et al. [8] found that security-unaware use of insecure C++ functions was passed from teachers to students. Moreover, in all cases there was no mandatory formal secure coding component to the CS course. We argue that while *ad hoc* on-the-job training efforts have value, software security is so critical that it should be automatically embedded in all software development training.

The academic record includes valuable accounts of actual industry practice. Sadowski et al. [245] described the development of a static analysis tool at Google, a model for what can be done in a cohesive environment, even a very large one. By contrast, Morales et al.'s [174] account of a dysfunctional multi-year development by a main contractor using multiple subcontractors with a DevSecOps pipeline gives excellent insight into the ease with which organisational dynamics can damage security outcomes. Previous work [237] highlighted that management security buy-in is vital. Swift software security progress is tied to upper management concern, which may be enhanced by increased regulatory and legal incentives.

2.6.3 Legislation

Both Europe and the U.S. appear to be moving towards regulations for secure software, a welcome recognition of the increasing importance of the field. Here, we give a brief overview of relevant developments.

GDPR The EU's General Data Protection Regulation (GDPR), which governs the protection of personal data in the European Economic Area, came into force in 2018. It mandates obtaining subjects' permission for data storage, sets time limits on data retention, and allows for penalties where data is inappropriately shared. Although the GDPR was not created with the primary aim of enhancing software security it has had this effect, since a data breach could expose the organisation to financial penalties. It does not list explicit cybersecurity requirements, instead using broad phrases such as 'state of the art.' This deters a 'checkbox' security mentality, encouraging awareness of ongoing software security developments [65].

The NIS Directive and ENISA The EU's 2016 NIS Directive dealt with cybersecurity but did not discuss secure software development [82], though this should change with NIS2. A thoughtful and well-researched preparatory paper from ENISA outlines the current EU work on introducing security certification for software, with consideration of existing standards and certifications, and likely pitfalls [307]. This welcome move towards EU-level certification should be expedited.

The UK The UK's 'Government Cyber Security Strategy 2022-2030,' released in February 2022, lists aspirations around a 'secure by design' framework to be adopted by the UK [117]. No specific advice for software development is available yet.

The U.S. In a week in May 2021 in which the ransomware shutdown of an essential U.S. oil pipeline dominated the news, the U.S. President released an 'Executive order on Improving the Nation's Cybersecurity,' which provides specific software-related measures. Part a) asserts that *'the Federal Government must take action to rapidly improve the security and integrity of the software supply chain, with a priority on addressing critical software.'* NIST is required to identify guidelines to evaluate software security and the security practices of developers and suppliers, and to identify *'tools or methods to demonstrate conformance with secure practices.'* Enhancing supply chain security, securing build environments, automating supply chain assurance and providing evidence for these activities are discussed. Identifying 'critical' software is also addressed, as is Internet of Things (IoT) security and a suggested security labelling system for software and IoT devices. U.S. government agencies will be obliged to consider software security when engaging in or renewing critical software contracts. Legacy code that cannot comply with the new requirements will have to be replaced. Steps to secure the software supply chain will be kept under review, with a progress report required within a year of the signing of the order.

2.6.4 Exacerbating Factors

There is an essential imbalance in the software security world. Most software developers prioritise functionality [183], followed by efficiency [279], elegant design, or maintainability. Unless they are working for an organisation that emphasises security, they will probably not put security first. In fact, even if they work for a security organisation, their security practice could be suspect [103].

While software developers struggle with time-to-market and tight deadlines, hackers, security researchers and red-teamers can focus solely on finding the security defects inadvertently left by developers, and exploiting them. When it comes to training, they have multiple resources at their disposal such as the free Bugcrowd University and the

many dozens of courses annually at Black Hat and elsewhere geared towards 'penetration testers.'

This imbalance of time and resources is difficult to tackle. Many software developers have not received training in secure coding. Organisations often have relatively few, if any, software security staff; a single software security expert supporting one or two hundred developers is not uncommon [276]. Outside of regulated industries, there is currently little incentive for organisations to prioritise security over time-to-market, and some organisations have no security process at all [13].

2.6.5 A 'Zero Tolerance' Approach to Software Security

We argue that the software industry now needs to step up and adopt a 'zero tolerance' approach to software security issues. Haney et al. [111] described how organisations that successfully deliver secure software have a 'security culture.' The entire software industry needs to develop a 'security culture,' with a comprehensive upgrade of education, tools, and documentation.

The building blocks to achieving this are not novel. They involve steps that are both widely acknowledged as necessary, and widely ignored. Cultural change is needed in education, from universities to boot camps. It should be unacceptable to teach computer skills without including mandatory security awareness and associated training.

At the corporate level, too often security risk assessments end with a decision to increase insurance provision against cyberattack. The balance of risk must be changed. This can be done by making organisations liable for costs incurred due to secure coding negligence on their part. Some experts argue that mandating secure coding would impose the type of procedural rigidity that leads to an obsession with passing tests, as can happen in the payments industry [222]. However, the flexible wording of the GDPR gives an insight into how secure coding can be mandated without leading to a checkbox mentality. In any case, even a checkbox mentality would be a vast improvement on the current security posture of many organisations [303].

2.7 Conclusion

As we have seen, crime and other aggressive behaviour on the Internet thrives on the existence of software vulnerabilities. A steady supply of zero-days, combined with delays in patching known vulnerabilities, ensures that bad actors can continue to exploit weaknesses for financial or other gain. This state of affairs causes an unsustainable level of uncertainty and risk. The threat of industrial sabotage or breach of military command and control structures from foreign actors adds to the mounting pressures on the global Internet, pushing it towards further fragmentation.

We have also seen how the opportunities for incursion provided by a global Internet can have a destabilising effect on existing power balances. If software security is not taken sufficiently seriously, there is a danger that national administrations will increasingly judge that the price of participating in a global network is too high. National executives may even decide that retreating to a national or regional Intranet would enhance their national security and reduce the danger of cyberattack on NC3 facilities and other critical infrastructure. As incidents of damage from cyber activities increase globally, assessments of this type may not be confined to authoritarian regimes. Though some states might welcome a fragmentation of the Internet, it seems like a failure of human imagination and potential.

The complete elimination of software vulnerabilities may be impossible, but a drastic reduction is not. It is time for a less accommodating approach. A ‘zero tolerance’ attitude to software security issues should be adopted, and it should include cultural and legislative change. This would reduce the perceived vulnerability of vital systems and help to maintain confidence in a networked world.

3 Understanding Developer Security Archetypes

This chapter was published as: Ita Ryan, Utz Roedig, Klaas-Jan Stol. 'Understanding Developer Security Archetypes.' *2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*. DOI: 10.1109/EnCyCriS52570.2021.00013.

3.1 Abstract

As software systems penetrate our everyday lives, security has risen to be a key concern. Despite decades of research leading to new tools and practices for writing secure code, achieving security as a key attribute remains highly challenging. We observe that much of the literature considers developers to be homogeneous and interchangeable. The differing circumstances of developers that might play a role in the writing of secure code have not been clearly defined. In this position paper we introduce the concept of developer security archetypes. Specifically, we suggest two key factors: developers' personal interest in security, and the support that developers receive from their environment. Together, these two dimensions define four archetypes which can be uniquely characterized. By distinguishing developer archetypes, we seek to better understand how developers perceive security-related issues in systems development, as well as how to better support them.

3.2 Introduction

In recent years there has been a focus on how the developer can be centred in discussions of software security. Acar et al. [4] argued that developers cannot be expected to be experts in secure coding. They suggested applying the lessons learned from twenty years

of research on usable security to the context in which developers write code. Developer Centred Security (DCS) has emerged as a term for the study of how developers can be encouraged and facilitated to write secure code [267]. For the purposes of this paper we use the classic definition of "software security" from McGraw: "the idea of engineering software so that it continues to function correctly under malicious attack" [165]. We define a developer as a writer of software code for use by others.

We argue that developers are not homogenous and interchangeable, and operate at different levels of enthusiasm and expertise when it comes to security. Moreover, they operate in multiple different environments, from security-aware organisations and open source teams to very small single-developer enterprises. We posit that developers can thus be thought of as existing along a spectrum in two dimensions: their personal interest in security and secure coding, and the support that is available for secure coding in their environment. To truly put developers at centre-stage, we must examine the role of these two factors on their security stance. We maintain that this interplay between developers' interest in security and developers' environment affects what interventions are of use to individual developers.

In this position paper, we introduce a two-dimensional typology that defines four developer security archetypes. By defining these archetypes, we seek to extend our understanding of developer centred security, the interventions that are available to developers to write secure code, and which activities might support developers in this task. When considering such support activities, we can reflect on which developer archetype they primarily assist, and whether we can broaden or tweak them to assist others. We argue that the archetype most in need of support, which we call Optimists, should be specifically considered when new interventions are devised. We maintain that centring the Optimists will enhance security for all developers.

This paper proceeds as follows. In the next section we introduce the two factors which we argue play a key role in how developers approach security and secure coding. These form the two dimensions, which define the four developer security archetypes, which we discuss in Sec. 3.4. We discuss the implications of these archetypes in Sec. 3.5.

3.3 The Role of Environment and Personal Motivation

There is a considerable body of research on developers and software security that discusses the role that the developer's environment plays in their secure coding practices. Haney et al. [111] conducted an interview study with individuals representing companies who use cryptography in their products. They found that these security-mature organisations had a security mindset, with a strong security culture and deep security expertise. Assal and Chiasson [14] conducted a survey study focusing on the human factors of software security. Using factor analysis, they found that "company-wide engagement" and "personal strategies" were the two main factors influencing participants' software security strategies. Pieczul et al. [216] discussed the developer demographics that should be considered when designing developer studies. They included "developer skill and experience" and "team, structure, culture and policies" as key demographic factors. Danilova et al. [61] considered how a company's emphasis on security influences or dictates its developers' security activities. Morales et al. [174] looked at how the cohesiveness of the development team is vital for security. Rauf et al. [228] discussed developers as social beings and the resulting impact on their security choices. How they relate to other developers and to their context are seen as crucial influences on their security behaviour. The authors advocated further study of how context influences the security posture of solo developers such as app developers.

Developer motivation to code securely is also widely studied. In a qualitative usability study, Naiakshina et al. [184] primed ten of the 20 participating students to code securely. They found that the ten participants who were not primed made no attempt to write secure code. Primed participants did better, especially when mandated to use an API with security helper classes. In a follow-up study using freelance developers, Naiakshina et al. [183] again found that specifying security as a requirement was the main influence on the security of the resulting code. Assal and Chiasson [14] found that identifying with security importance is the most motivating factor for software security, followed by workplace environment and perceived negative consequences. Xiao et al. [334] looked at why security tools spread. They found that the largest influence is co-worker recommendations. Although they did not focus on developer motivation, they did find that an attack on a developer's software can raise the developer's security awareness; this factor is also mentioned by Assal and Chiasson [13]. Haney et al. [111] found that

maturity and experience are common characteristics of developers in companies with a strong security culture.

3.4 The Developer Security Archetypes

Developer interest in and environmental support for secure coding, when placed on abscissa and ordinate axes, can be used to create a two-dimensional typology that defines four developer archetypes (see Fig. 3.1). The archetypes bring different developer circumstances into sharp focus. It is easier to assess the potential role of a proposed technique or tool when the target cohort or cohorts can be clearly identified and described.

We now position and describe the four developer archetypes. We emphasise that these are archetypes, and in reality the extent to which a developer is positioned along the two dimensions will vary. The purpose of these archetypes is not to perfectly characterize each developer, but rather to extend our understanding of different types of developers in relation to attitudes to security and secure coding.

3.4.1 Pragmatists

We adopt the term Pragmatists to describe developers who have no personal interest in security, but are provided with security-related training and tools in a collaborative

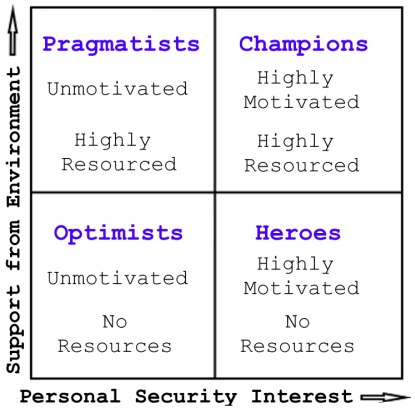


Figure 3.1: **The four developer security archetypes**. The archetypes bring different developer influences with regard to secure coding into focus. The developer's tendency to use secure coding practices increases as we travel upwards and right along the axes.

environment. Although their focus is on other aspects of the work, they will follow secure coding processes if these are mandated and not too inconvenient. Xiao et al. [334], in a study of secure development tool adoption, noted that all developers in their study who were mandated to use security tools did use them. However, since Pragmatists have no personal interest in security and secure coding, they may attempt to bypass inconvenient secure coding guidelines. For example, Ashenden et al. [12] described a developer telling their company's security team that they *"are like a rock in a stream, we just flow around you."* Tomas et al. [279] also found occasional conflict, with one developer laughing about security culture "propaganda".

3.4.2 Champions

Champions are developers with an interest in security who are working in environments in which they are supported and resourced. The term "Champions" is widely used both in industry [171] and in academia [267] to describe developers with a special interest in security in the corporate environment. Security teams, which are notoriously short-staffed, often liaise with Champions and leverage their security expertise [276]. Haney et al. [111], in an interview study of experienced developers working in organisations that develop cryptographic products, encountered a population of developers many of whom were Champions. The researchers described individuals who were highly committed to security and communicated strong security values to the rest of the organisation.

3.4.3 Optimists

The "optimistic bias," which causes people to believe that misfortune will not happen to them, was used by Assal and Chiasson [13] as a possible explanation for cavalier developer attitudes to code security. In this spirit, we adopt the term Optimists to describe developers who have little or no interest in coding securely and are not supported or encouraged to do so within their environment. They are optimists because they believe that their code is probably fine, or someone else is taking care of security [280], or their software is too insignificant to be attacked, or there is nothing worth stealing in the system, or they will be gone by the time there is a breach [267, 13, 335, 202]. Optimists abound in secure development literature. While investigating the propagation of insecure code snippets from Stack Overflow, Bai et al. [17] interviewed 15 software developers

about insecurities they had found in the developers' GitHub projects. Post-interview, only two of the developers fixed their projects. Of the remaining 13, one deleted their project, one deleted their GitHub account, and the remaining developers ignored the vulnerabilities. Palombo et al. [202] were surprised by developers' reaction to a serious security flaw that potentially enabled remote code execution. Developers downplayed the issue, deciding not to fix it to avoid causing problems elsewhere in the software. Their organisation did not appear to assign any time or resources to security, or to prioritise security in any practical way.

3.4.4 Heroes

Assal and Chiasson [13] identified developer "heroes" who personally prioritise security even though they are working in environments where security is not prioritised. If working in a collaborative environment, they may provide the only security checks within a company or team. Our Hero archetype also encompasses solo developers such as app developers who take the time to familiarise themselves with the security requirements for the environments they are coding in. A developer who is highly motivated to code securely is more likely to research, discover, and implement security techniques and tools, even when not supported to do so in their environment [334]. Heroes may find themselves almost in conflict with their employers when trying to improve the security of a code base. Security researchers Palombo et al. [202] experienced this dynamic when embedded with a development team that received no security support. Their suggested security fixes were rejected until they carefully calibrated them to the code base to ensure minimal impact on the time and attention of the development team.

3.5 Discussion

We now discuss implications of our typology, which helps to differentiate among different types of developers in relation to security-related attitudes.

Champions are widely regarded in industry as essential to the smooth working of the Software Security Groups that oversee secure coding in large organisations. These groups liaise with Champions on development teams to promote secure coding techniques within the teams [267] [276]. An initial analysis of the four archetypes might suggest that the

solution to the problem of insecure code is to turn all developers into Champions. To achieve this, it would be necessary not only to educate all developers in secure coding techniques, but to engage their interest so that they use the techniques. Furthermore, all organisations and open source teams would have to embrace and prioritise the need for secure software development. Otherwise, Optimists would merely be converted to Heroes, who still lack much of the support needed to code securely.

Achieving these changes is a worthy objective, but until it is realised we must accept that all four developer archetypes exist and they need different types of support to help them to code securely. The existence of the four archetypes leads us to the following research questions.

3.5.1 Developer Motivation

What causes developers to move from left to right, indicating an increase in security motivation? This area comprises developer motivators such as training and experience. Witschey et al. [329] have found that information from peers can have an impact. Assal and Chiasson [14] found that the top six motivations towards software security are self-driven motivations such as caring about the user, with financial rewards considered least motivating. While drive (arguably) cannot be taught, understanding the implications of poor security scored highly. This suggests that activities such as Weir et al.'s security interventions [319] should have a motivating effect. Weir et al. devised a lightweight series of interventions to motivate teams to develop securely. Activities include incentivisation meetings with teams to introduce core security practices. Other security practices are introduced on an ad-hoc basis if appropriate to the discussion. Follow-up workshops help to copper-fasten the ideas discussed within the teams, assisting with motivation and keeping security firmly in sight.

The "Motivating Jenny" (`www.motivatingjenny.org`) project explores the reasons why developers do not consistently and routinely use security methods and tools. The project seeks to find ways to engage and motivate developers to code securely. It uses several different media [153] and ethnographic studies [152] to examine how developers discuss and engage with security. "Motivating Jenny" researchers have devised a workshop format to help developers to grapple with security topics [150]. Their work focuses on leveraging developers' values and professional pride to attract

them to secure coding. The "Motivating Jenny" project, Weir's work and similar projects aim to interest and educate developers in security, moving them from the left to the right of our typology.

A complementary question is whether there are circumstances that cause developers to regress, moving from right to left? If so, how can these be avoided?

3.5.2 Organisational Motivation

What causes organisations, and indeed open source teams, to move upwards, indicating an increase in security motivation? Customer engagement, industry standards and legislation are frequently cited [111] [174]. Are there other motivators? Conversely, are there circumstances which cause organisations or teams to move downwards, lessening their focus on security? If so, how can these be avoided? Organisations will rarely declare their lack of interest in security, making unmotivated organisations difficult to study.

3.5.3 Supporting the Optimists

DCS research often focuses on interventions and tools for developers who are highly resourced in a security-aware environment, highly motivated to code securely, or both. We argue that special thought must be given to how to support the Optimist, who has not (yet) developed an interest in security and does not benefit from a collaborative security environment. Although it is tempting to plan to train them all, the high numbers of new entrants to software development [64] and the low barriers to entry [261] make it a Sisyphean task. Furthermore, Optimists with an interest in security become Heroes, still under-resourced for secure coding. Given that there will always be Optimists, how can they best be supported?

3.6 Conclusion

In this position paper we introduced four developer security archetypes, organized along the two axes of personal interest in software security and environmental support for software security. Defining these four archetypes sheds light on a blind spot in the current literature on developer centred security, and can help broaden our understanding

of issues surrounding software developers and security. Whereas prior literature has started to recognize human factors such as developer motivation, this paper contributes a systematic conceptualisation of developers in relation to secure software development. Summarising, the Pragmatist is uninterested in security, but will often use security tools provided in their environment. The Champion is a security enthusiast operating in an environment where security is emphasised. Heroes work in a low-security environment, but their own interest in and drive towards security should help them to secure their code. The Optimist is uninterested in security and codes in an environment where security is not considered. Optimists are entirely dependent on the security behaviour of the tools they use to make their code secure.

We argue that our typology helps to clarify issues around secure software development. Individual developers may move between these four archetypes, whether via training, other motivators or a change of role or employer. However, the four archetypes will continue to be populated for the foreseeable future. Interventions and tools aiming to enhance secure coding should be tailored to the appropriate archetypes. Solutions that treat developers as a homogeneous group are likely to lose the opportunity to address as many developers as possible.

4 Unhelpful Assumptions in Software Security Research

This chapter was published as: Ita Ryan, Utz Roedig, Klaas-Jan Stol. 'Unhelpful Assumptions in Software Security Research.' *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. DOI: 10.1145/3576915.3623122.

4.1 Abstract

In the study of software security many factors must be considered. Once venturing beyond the simplest of laboratory experiments, the researcher is obliged to contend with exponentially complex conditions. Software security has been shown to be affected by priming, tool usability, library documentation, organisational security culture, the content and format of internet resources, IT team and developer interaction, Internet search engine ordering, developer personality, security warning placement, mentoring, developer experience and more. In a systematic review of software security papers published since 2016, we have identified a number of unhelpful assumptions that are commonly made by software security researchers. In this paper we list these assumptions, describe why they sometimes do not reflect reality, and suggest implications for researchers.

4.2 Introduction

In an increasingly-connected world, online security is ever more important. Many hacking techniques depend on the existence of vulnerabilities in software. Their presence facilitates cybercrime, from ransomware to cyber espionage [238]. The field of DCS explores reasons why developers write vulnerable code [267], seeking to understand developers, how they can be helped, and what problems they face in practice. Seeking

to gain greater understanding of DCS challenges, we conducted a systematic literature review focusing on papers published in known DCS venues since 2016. The objective was to identify reasons why, despite over two decades of software security research, the discovery rate of exploitable vulnerabilities continues to increase [215].

In the course of the review we observed a number of implicit assumptions within the field of software security research. Some are common, some less so. All are unhelpful when attempting to understand developer behaviour. Implicit assumptions may lead researchers to miss relevant information, or to jump to conclusions.

One example of an unhelpful assumption occurs in an excellent ethnographic study by Palombo et al. [202]. The researchers were embedded as coders in a software organisation selling network access software for a combined total of 18 months. Their observations indicated that the development team did not prioritise software security. Working alongside the team, they conducted penetration tests on the software. To their surprise, developers did not immediately fix revealed issues and did not even appear to take them all that seriously. This held true even for a defect involving remote code execution. Defects involving broken authentication looked like a potential systemic threat to the company's existence, yet they were not prioritised for fixing. The researchers discussed the developers balancing their primary aim to develop successful features for the software with their understanding that security is important.

The assumption made here was that the organisation prioritised software security, and it was the development team who were forced to balance that priority with other priorities. However, to send a genuine signal that security is important, management would need not only to state it explicitly, but also, for example, invest in testing including the commissioning of penetration tests, invest in security training and request a prioritisation of security issues in backlog meetings [111]. In this organisation, the researchers undertook penetration testing and found a defect on the first day, suggesting that penetration tests had not been done before. The researchers had to smuggle secure coding techniques into the code base, minimising the resulting impact on development and testing time. The length of time the researchers spent on site allowed them to get highly informative insights on the development process. The assumption that security was an organisational priority, however, meant that, although they observed a lack of security concern on the development team, the researchers did not focus on the secure-coding direction that the team received.

In this paper we list 25 such assumptions in 4 categories. For each assumption we outline the reality that is hidden by the assumption, and the implications for future work in the study of software security. The aim of this paper is not to point fingers or reveal flaws in work by others—much of which we admire and value greatly. Rather, we seek to initiate a discussion and reflection on the line of work that is done in the area of software security research. The goal of this paper is to draw the attention of the software security research community to unhelpful research assumptions. We hope this will trigger discussion and broaden the search for the meaning of observed phenomena.

This paper is, clearly, not the first to review the challenges of developing secure software. Several papers are noteworthy in this context. Recently, Mokhberi and Beznosov identified '*Human, Organizational, and Technological*' dimensions of secure coding challenges, containing 17 challenge categories [173]. They viewed their work as the beginning of a systematic way of thinking about the challenges in software security. Some of the challenges have wide-ranging effects; for example, managers set and enforce priorities, contributing to culture, and therefore have a substantial effect on an organisation's security stance. To develop a security culture, both internal motivations such as valuing security and external motivations such as a belief that security will increase market share are needed. Software security is a Non-Functional Requirement (NFR), and Mokhberi and Beznosov reflected on the additional challenges entailed by its lack of visibility and the constantly changing nature of the threat landscape. Wide-ranging recommendations from this study included using organisational strategies to improve security; for example, making software security visible, ensuring developers know that they have responsibility for it, and designing tool-friendly policies. Providing training and support to developers was also advocated.

Das Chowdhury et al. analysed developer-centred studies, identifying 72 papers for their analysis that were written over 13 years [62]. They sought to categorise challenges, behaviours and interventions in the literature, and how challenges and behaviours interact. They also analysed five '*tropes*,' or '*misplaced beliefs about how developers behave*.' They argued that these five misplaced beliefs strongly influence the design of libraries and tools, and the secure coding interventions that developers endure. These tropes are, that '*developers are experts*,' that '*everyone has the same security and privacy needs*,' that '*security is easy*,' '*security is a universal priority*,' and '*maintenance is fun*.' We

too identified the common assumptions that all developers are security experts and that security is easy, although our naming is different.

This paper is organised as follows: Section 4.3 discusses our research method, Section 4.4 gives our results. We discuss and conclude the paper in Section 4.5.

4.3 Methodology

4.3.1 Venues and Search Terms

We selected a number of venues to search for related papers, based on conferences and journals that are known to publish DCS-related material. We also included workshop papers where available. We included papers published from 2016 on. We used the following core search term, allowing autostemming.

(software OR code) AND (developer OR development) AND security

Since no one search engine provided papers from all of our venues, we used the ACM Digital Library, IEEE Xplore, and Scopus. The exact syntax used depended on the search engine. The search resulted in an initial total of 852 papers. The full protocol including paper breakdown is available at `https://osf.io/7a2dr/?view_only=e39e7822618f47f0a9bca7e5ec46d86c` and in Appendix A.

4.3.2 Selection procedure

The first author undertook the selection procedure, with consultation with the other authors where necessary. The titles and abstracts of all 852 papers were reviewed for first-round exclusions, which excluded duplicates and incomplete papers, such as abstracts, posters, and two-page outlines. In second-round exclusions, our rules were that we would include papers:

- that identify one or more factors that reduce the incidence of security vulnerabilities in code while it is being written;
- that identify one or more factors that increase the incidence of security vulnerabilities in code while it is being written.

The second-round exclusions therefore excluded papers that did not identify factors that reduce or increase the incidence of security issues in code. We included studies that described aids that help developers to code more securely. Some were existing or desired aids that were described in interviews with security experts. Others were new or proposed aids that were devised and assessed in a specific domain with a view to general application in the future. Papers describing tools that help to write secure code in specific problem domains with no application for the broader developer community were excluded. An example would be a paper on a tool to verify contract rules on the Ethereum blockchain. We also excluded papers describing specific implementations of tool types that are already well-known. An example would be a paper describing a new static analysis tool for a specific language. Finally, we excluded proposals for new processes or tools that have not been empirically assessed either in a laboratory setting or in industry. An example would be a proposed method for integrating security into Agile that has not been put into practice anywhere. When in doubt about a paper, it was reviewed by the research team. This process resulted in 90 papers selected for review.

4.3.3 Snowballing

We identified authors who had worked on three or more of the papers reviewed. Other papers by these authors were assessed for inclusion in the study, yielding 16 additional papers. The references sections of all our selected papers were evaluated and papers of interest with multiple references were selected for review. Since many of these papers were already included in the review, this resulted in only three additional papers.

To ensure that we had not missed important relevant venues, we analysed the venues where our 19 snowballed papers were published. We looked for venues that had published two or more snowballed papers and that we had not already searched. We did not find any such venues amongst the snowballed papers.

Our cutoff date for papers was September 30, 2022.

4.3.4 Data Extraction and Analysis

Data identification and extraction was done by the first author, with frequent consultation and discussion with the second and third authors. Each paper was analysed and factors affecting the security of code at coding time were extracted and categorised. Such

factors could affect the code either positively or adversely. During this process, a number of apparent contradictions between papers came to light. In other cases, an obvious interpretation of the data collected by the researchers was not present. On analysing these contradictions and missed opportunities, it became apparent that in some cases the root cause was an unconscious assumption made by researchers.

We identified these assumptions and categorised them based on the three dimensions identified by Mokhberi and Beznosov, which we found useful in our own work. Thus we have '*Technical*', '*Organisational*' and '*Developer*' (Mokhberi and Beznosov called these '*Human*') assumptions. Given that in this paper we are exploring and discussing the research process, we added a fourth dimension: '*Research assumptions*.'

4.3.5 Positionality statement

Given the nature of this study, it is worth noting the background of the authors of this work. The first author has a strong background in logic, having both a degree in Philosophy and a Masters in Computing. They also have 20 years of practical software development experience as a contractor, including a focus on secure software development. The second author has over 25 years of experience in security research. The third author has 15 years of experience in software engineering research, has a strong interest in (and has published about) research methodology, and has focused on 'assumptions' in other works.

4.4 Results

In the course of our research a number of assumptions emerged from the literature. They appeared to affect researchers' observations and what they paid attention to. In this section, we list these assumptions. For each one we discuss studies where it was made, and studies showing why it is unhelpful or untrue. Table 4.1 presents a summary of these assumptions. In the table, each assumption is immediately followed by references to the papers that made that assumption. An example of how the assumption affected a paper is included in the 'Implications and Examples' column.

4.4.1 Technical Assumptions

Assumption: code that obviously should be secure will be written securely.

Reality: security is seen as a cost, not a feature, and is not always prioritised.

Some code is so clearly security-related that it seems self-evident that it should be written securely. Cryptography is a good example. Haney et al. interviewed 21 individuals from organisations with cryptography-related products [111]. The dedication to security of the interview subjects seems impressive. The importance of security in their field is self-evident, and the observations made by Haney et al. appear to confirm that security features highly in the considerations of the organisations under study. Nevertheless, in this excellent study Haney et al. did not attempt to evaluate the actual security of the code produced by the organisations whose employees they interviewed, or to evaluate the secure-coding stance of each organisation. For example, there is no mention of questions regarding code breaches or successful hacks. The limitations acknowledged that the paper is based on perceptions and self-reporting.

Evidence that even cryptography developers do not always use the security tools at their disposal comes from Jancar et al., who interviewed 44 well-qualified developers of 27 open source cryptographic libraries to determine whether they used tools to assess their libraries' vulnerability to constant-time attacks, and if not, why not [131]. They found that, although all participants were aware of timing attacks, tool use was not universal. 38.6% of the developers interviewed used timing attack detection tools, while 25% did not even know about them.

While it would seem self-evident that building security into tools and frameworks is desirable, Pieczul et al. noted that tool vendors may feel that extra security inhibits adoption [216]. They gave the example of enforcing HTTPS for cloud-service endpoints, pointing out that this was not usually done because it would entail significant overhead on the developer side. Tool vendors worried that developers would reject this complexity, choosing a less secure but simpler service instead.

Even where security is a stated focus of a project, factors such as tight deadlines, poor organisation, bad management and running out of time can result in insecure implementations [95]. In their examination of a sub-optimal move to Agile and DevSecOps in a very large development program, Morales et al. found that security concerns were ignored or downplayed, even while the adoption of DevSecOps seemed to indicate that they

would be prioritised [174]. Subcontractors faced tight deadlines and inadequate testing resources. The project leadership of a subcontractor explicitly stated that secure coding was not a priority, and speculated that the prime contractor '*probably does security testing*;' not caring enough to know for sure. The absence of secure coding leadership or attention was pervasive, present in the prime contractor and subcontractors, and was reflected in the absence of a security mindset amongst subcontractor developers. It was likely to result in vulnerabilities being released to production and remaining undiscovered for long periods of time [174].

Implications: researchers should carefully investigate the security posture of organisations and developers, even when the need for security appears self-evident.

Assumption: developers can introduce secure coding practices and tools to organisations easily.

Reality: developers are constrained by organisational rules, team norms, faulty tools and peer friction.

An example of this assumption arose in a study of security activities in Finland. Rindell et al. noted that although external penetration tests were perceived by survey respondents to have a high impact on security, they were not widely undertaken [232]. They speculated that perceived impact might not correlate with actual impact. A simpler explanation would be that the employees and engineers who responded to the survey were not in charge of the security budget. Penetration tests are expensive [320].

Developers may not have control over software security and how it is dealt with in their environment [154]. This causes difficulty with the introduction of security interventions and tools [260]. While security tools are essential to secure coding, they can be costly in a number of different ways, making them inaccessible to under-resourced developers. Expensive tools [155] will typically only be available to large or well-resourced organisations. Even ostensibly cheap or free tools will not be accessible to small teams or to isolated developers if they entail invisible overheads. These may include setup time, training and learning time, and maintenance time [201]. Selecting the wrong tool is costly [172].

Another important factor when developers attempt to introduce new tools or practices is peer buy-in. Embedded in an organisation, Palombo et al. found that their attempts to introduce secure coding practices into their team were resisted until they found a way

to make adoption by their peers easy and frictionless [202], a finding echoed in another ethnographic study by Tuladhar et al. [284]. Tools with a high false-positive ratio waste developer time [276, 42, 69]. Other developers may resist their introduction, and the false alerts will create alert fatigue [324]. Tools introduced to existing legacy code may flag enormous amounts of technical debt, causing developers to feel overwhelmed [201]. Security tools can be slow to run and difficult to configure. This causes friction within the team [276, 156]. Tool response times must be prompt. Distefano et al. found that static analysis issues detected overnight and assigned to developers had a response rate close to zero [69]. By contrast, issues detected at commit time and added via bot to code review notes had a fix rate of approximately 70%.

Exploring the reasons for poor use of timing-issue detection tools by cryptography developers, Jancar et al. identified barriers including a lack of developer awareness, a lack of tool usability, the need to annotate code, resource overheads and a fear that tools would not be available or maintained as the researchers who originally developed them moved on [131].

Implications: an absence or low take-up of software security tools and practices may not reflect a lack of developer interest. Environmental constraints should be investigated.

Assumption: if a tool exists, it will be used.

Reality: developers must find and evaluate the tool, and have faith that it will be maintained.

Useful and interesting security tools for developers are constantly under development [7, 189, 194, 89, 97]. There seems to be an implicit assumption that these security tools will be used; we could not find discussions of how the tools would be publicised and disseminated. Achieving tool use is a challenge [42]. Developers may not be aware of the most useful tool for a task, even if it is available in the IDE [260]. Information on program analysis tools is not widely disseminated [6]. Tool efficacy varies [201, 26], and tool evaluation is '*cumbersome and demotivating*' for developers [20]. The pace of development is swift, as is the pace of adoption for new languages and frameworks. Perfect, but top-heavy, languages and IDEs, and the painstakingly-learned lessons they exemplify, can be quickly abandoned [205]. The relatively recent move to a microservice architecture is an example of a new technology causing new pain points, with limited support for microservice-based security found in a search of academic and grey literature [214].

Table 4.1: Unhelpful Assumptions

Assumption	Reality	Implications and Examples
	Technical Assumptions	
Code that obviously should be secure will be written securely [111].	Security is seen as a cost, not a feature, and is not always prioritised.	Researchers should carefully investigate the security posture of organisations and developers, even when the need for security appears self-evident. For Haney et al. [111], this could have revealed a disconnect between stated and actual security.
Developers can introduce secure coding practices and tools to organisations easily [232].	Developers are constrained by organisational rules, team norms, faulty tools and peer friction.	An absence or low take-up of software security tools and practices may not reflect a lack of developer interest. Environmental constraints should be investigated. For Rindell et al. [232], not considering this resulted in a missed opportunity for deeper understanding of the data.
If a tool exists, it will be used [7, 89, 189, 194].	Developers must find and evaluate the tool, and have faith that it will be maintained.	Researchers should consider whether a proposed tool will be used. Ohm et al. devised a framework to detect malicious dependency updates [194]. Their contribution would be greatly enhanced by a plan to promulgate the tool.
Secure coding is an individual undertaking [4].	Most developers work in complex environments with multiple stakeholders.	Researchers should evaluate developers' work environment. Acar et al. [4] suggested that researchers should measure actual developer behaviour, but did not propose gaining a holistic understanding of the developer's context.

Table 4.1: Unhelpful Assumptions (continued)

Agile is unsuited to secure development [200, 220].	Issues attributed to use of Agile methods may have causes not related to Agile, for example, they may be due to not prioritising security.	Researchers working in an Agile environment should carefully consider whether the use of Agile methods is relevant to uncovered issues. Although attention to good design is an Agile principle, Oyetoyan et al. [200] attributed a desire for secure design training to the Agile nature of the work environment they were investigating, missing that this training may be a universal need.
Keeping software updated is always good [67, 121].	Patches can cause new issues or failures in software.	Researchers should bear in mind that developers may rationally delay patching their code. Derr et al. suggested automated library updates as a solution to terrible update rates, noting that if the updates turn out to cause defects they can be rolled back [67]. If the defect is a vulnerability, the rollback may not occur on time to prevent exploitation; this concern is not discussed.
Safe defaults are always good [249, 209, 102, 169, 314].	Defaults presumed safe can cause unexpected consequences.	Before advocating for safe defaults, the relevant software's usage characteristics should be assessed. Advice such as Patnaik et al.'s based on 'deny access' [209] may not be appropriate for cyberphysical systems.
Secure coding regulations and compliance constraints improve software security [80].	Compliance requirements can have unintended consequences.	Careful consideration should be given to the wording and structure of legislation and regulations. Gasiba et al. expressed surprise that developers ignore secure coding instructions they consider unreasonable; that could be due to their making this assumption [80].

Table 4.1: Unhelpful Assumptions (continued)

Developer Assumptions		
All developers want to code securely [227, 320, 224].	Some developers are uninterested in secure coding or feel that it is not relevant to them.	This assumption can cause researchers to misdiagnose reasons for insecure coding behaviour. Rahman et al. did not consider laziness or indifference as reasons for bypassing security tool warnings, although 85% of bypasses originated from 20% of developers [224]. This may have affected their analysis of the phenomena they observed.
All developers know how to code securely [321, 319, 320].	There is a very broad spectrum of code security knowledge and experience amongst software developers.	Researchers should not assume that developers know how to code securely. Weir et al. [320] did not investigate the secure coding stance or training needs of workshop participants, and may have missed important nuances.
Developers' perceptions on whether security is needed in their work environment are accurate [152].	Developers who are not well informed about software security may be poor judges of when it is required.	Researchers should not rely exclusively on developers' own assessments when judging whether software is vulnerable to attack. Lopez et al. appeared to take developers' assessments of security needs at face value [152]. They did not explore further as to whether the software produced was actually secure, or consider how that could be measured.
Developers are enabled decision-makers not subject to psychological pressures like fear of downsizing [14].	Experience of downsizing events may impact developers' confidence and motivation.	Researchers should pay attention to developers' job security and the turbulence of their work environment when assessing software security motivation and practice. Assal and Chiasson's survey study might have detected disillusionment and fear as security blockers if relevant questions had been asked [14].

Table 4.1: Unhelpful Assumptions (continued)

API and library developers are security experts [41, 67].	API and library developers have the same range of security expertise as other developers.	Researchers should not assume that API developers have any special security expertise. Derr et al. studied the difficulties inherent in keeping applications updated; their discussion on API developers' use of semantic versioning and bundling security updates did not consider that API developers might not understand the security implications of update policies [67].
All coders have access to the same resources [17, 232].	Constraints include background, training, isolation, the organisational secure-coding environment, and budget.	When analysing developers', teams' and organisations' security decisions, researchers should consider their budget and other resource constraints. Bai et al. did not assess budget or language skills when studying propagation of Stack Overflow security errors, which may have affected their analysis of issues [17].
Organisational Assumptions		
Managers are software security champions [321, 319, 320, 227].	Many managers are ill-informed about software security, fail to grasp the risk involved, and/or perceive it as unimportant.	When evaluating software security within an organisation, researchers should explicitly assess the stance of upper, middle and line management. For the software security interventions devised by Weir et al., upper management permission for the interventions is assumed [320]. This work would benefit from discussion of what would motivate upper management to permit security interventions in organisations that are inattentive to code security.

Table 4.1: Unhelpful Assumptions (continued)

Organisations prioritise software security [202, 320].	Not all organisations prioritise software security.	Implications: researchers should pay attention to the priorities developers receive from their managers, both stated and unstated. Palombo et al., although noting that security fixes were sometimes resisted by developers and were often not implemented at all, did not explore the secure coding priorities of the organisation they were embedded in, or whether software security was in fact an organisational priority [202].
The organisational environment is cohesive and therefore can control the complexity of secure software development [11, 227].	In large organisations, different departments and teams may have different priorities, causing conflict over software security goals. Subcontractors may complicate the issue.	Researchers should pay attention to inter-departmental dynamics when studying large organisations. Arizon-Peretz et al.'s insightful paper on software security climate theory assumes that the organisation is cohesive [11]. It does not investigate issues of inter-departmental dynamics and power-plays, outsourcing, or fragmentation caused by frequent acquisitions, which could affect attempts to build a positive software security climate.
The presence of secure coding activities implies that security is a high priority within an organisation [177, 152].	Secure coding activities may be compliance driven, with little concern for actual security.	Researchers should assess the secure coding culture of an organisation as well as its secure coding activities. Morrison et al. observed secure coding practice in an IBM team but did not ask whether practices were ever skipped for deadline reasons [177].

Table 4.1: Unhelpful Assumptions (continued)

Research Assumptions		
Developers and/or organisations will admit to a lack of understanding of, or enthusiasm for, security [219, 322].	Security is seen as a positive quality, and therefore both developers and organisations are unlikely to admit to a lack of understanding of it or interest in it.	Individuals' and organisations' assertions on prioritising software security should not be taken at face value. Poller et al. reported survey results that were positive or neutral about penetration testing [219]. Had they considered their own close relationship with management, they might have been more sceptical about these results.
Activities that can lead to insecure code cannot also lead to secure code [88, 3, 17].	Some common coding activities are widely used, and do not reflect the security of the resulting code.	Researchers should assess context of research findings, including whether control groups were used. Bai et al. found reuse of insecure code snippets from Stack Overflow in GitHub, resulting in insecure code [17]. However, they did not check for reuse of *secure* Stack Overflow code snippets, which could have revealed a positive effect.
Findings in a single paper are final and can be cited confidently, with no need for replication [62, 318, 267, 20, 13, 89].	Scientific evidence requires confirmation, preferably from different types of study.	Researchers should not perceive research questions as definitively answered unless there is a cumulative body of research. Das Chowdhury et al. [62] uncritically cited a Van der Linden et al. study [306] finding that developers go by answer length when choosing a Stack Overflow answer, even though participants for that study were not vetted.

Table 4.1: Unhelpful Assumptions (continued)

Measuring secure coding activities is equivalent to measuring code security [13, 200, 232].	Code security for non-trivial software cannot be definitively measured.	Researchers should bear in mind that the level of secure coding activities may not reflect actual code security. Oyetoyan et al. used a list of secure coding practices to assess secure coding in organisations [200]. They did not note that these practices may not reflect code security, and did not attempt to assess issues of culture such as deadline constraints, which may impact on whether the activities are thoroughly carried out [239].
If secure coding activities take place, they will be reflected in artefacts [211, 277, 126].	Artefacts do not always reflect work done.	Definitive conclusions should not be reached based on artefacts alone, although they can be used to triangulate. Imtiaz et al. studied use of the Coverity tool in open source projects [126]. One finding was that developers are slow to fix alerts and do not fix them based on tool-assigned priority level. It is possible that a coherent strategy, such as matching team member expertise to defect assignment, exists. It would not be available in Coverity history if so.
Artefact analysis done at scale is error-free [277, 88, 126].	Artefact analysis when done at scale may be flawed and results should be viewed with caution.	Researchers should carefully evaluate artefact analysis steps when considering results. Fischer et al. found a large percentage of insecure snippets in Android applications, using a complex pipeline to compare snippets to bytecode [88]. The paper did not discuss manual verification of the comparison process, and it is difficult to determine how this could be done, so the results may be open to question.

Table 4.1: Unhelpful Assumptions (continued)

Coders, product managers, and testers can all be studied as a homogeneous group [320, 151].	Different disciplines need different types of support.	Researchers should consider the differing support needs of the different roles that they are studying. Lopez et al. assessed how developers responded to security episodes [151]. The paper veered between using 'developer' for all roles and coding roles, and may have missed an opportunity to distinguish responses and behaviour by role.

There is also a problem with tool abandonment by academics. We encountered several frameworks [177] and tools [188, 76, 63] that have not been updated by their creators for several years. This lends substance to the problem mentioned in Jancar et al., that the fear that researchers may abandon tools inhibits tool adoption by developers [131].

Implications: researchers should consider whether a proposed tool will be used.

Assumption: secure coding is an individual undertaking.

Reality: most developers work in complex environments with multiple stakeholders.

Research on software security has converged around a number of themes. One is an analogy with usable security proposed by Acar et al. [4]. Just as ordinary users should have usable authentication and other security tools, developers should have usable tools in their environment to facilitate secure coding . There has been considerable research into how to optimise developer security tools for usability. Researchers have focused on issues such as speed and scope of IDE warnings [61], and optimising library error messages [100, 99]. Implicit in the usability analogy is the concept of the developer as a tool-using individual, separate from the social environment in which they write code. Absent pair programming, only one person at a time can write a line of insecure code or fail to include an automated security test.

Issues with the usable security analogy include the fact that the developer's environment is complex, with multiple stakeholders all having potential conflicts of interest [125], and is difficult to replicate in the laboratory in a realistic way [216].

Laboratory studies that focus on the results of a single developer's work have validity, but by their nature [264] they ignore the complexity with which developers must usually contend. This should be borne in mind by researchers. We did not encounter any proposals for or evaluations of tools with a team focus; for example, tools to enhance team secure coding communication.

Implications: researchers should evaluate developers' work environment.

Assumption: Agile is unsuited to secure development.

Reality: issues attributed to use of Agile methods may have causes not related to Agile, for example, they may be due to not prioritising security.

Some studies show a tendency to attribute software security characteristics to the use of the Agile method [200, 220]. For example, Oyetoyan et al. concluded, amongst other things, that effective software security in an Agile setting needs a driver [200]. This implies that software security drivers are not needed in non-Agile settings. Other papers in the review suggest that drivers are needed in all settings. Rajba gave an account of strategies used in a large automotive organisation to improve security in both new and legacy systems [225], vividly illustrating the advantages of having an enthusiastic software security driver. The advantage of availing of software security '*champions*' to spread security enthusiasm in the organisation is discussed throughout the literature [276, 111, 284], with no emphasis on Agile.

This assumption was also made by Poller et al., who observed several development teams during different stages of reaction to an external penetration test [220]. They assessed whether the organisation made changes to prevent the same types of vulnerabilities from reappearing. They were disappointed to see no long-term changes, follow-up, or extra security tasks in the Agile pipeline. They concluded that this was partly due to the Agile development methodology, and that, in order to give developers some autonomy, managers had avoided giving precise security direction. However, where management genuinely prioritises security, it is made an overt rather than a non-functional requirement. For example, Whitmore and Tobin described work in IBM that was inspired by the need to facilitate developers' move towards rapid deployment via Agile and DevOps, demonstrating that where there is software security consciousness there can be a corresponding software security drive [326]. Tuladhar et al. showed that in an organisation that desires security, security stories may be added to the scrum [284]. Extra time may be allocated for security concerns [151].

This assumption should be avoided, unless it is possible to compare software security results to other teams in the same organisation that are not using Agile.

Implications: researchers working in an Agile environment should carefully consider whether the use of Agile methods is relevant to uncovered issues.

Assumption: keeping software updated is always good.

Reality: patches can cause new issues or failures in software.

Researchers have observed and criticised a tendency to delay updating and patching third-party libraries [67, 121]. The assumption is that keeping patches up-to-date max-

imises code security [121]. This is usually true, but sometimes new code or patches contain vulnerabilities or even malicious code [77]. Das Chowdhury et al. noted that the '*maintenance is fun*' trope they identified can lead researchers to expect unrealistically rigorous update behaviour. However, even security-aware professionals may prefer to wait a period after patches have been released before upgrading, to ensure that they are not exposed by vulnerabilities introduced in the patch [77]. This is rational behaviour and can only really be averted by ensuring that all upgrades are vulnerability free; not easy to do. One partial solution to this is the correct use of semantic versioning [41] to isolate security patches, so that developers can be sure that the patches will fix vulnerabilities and have the minimum possible additional effect on their code.

Implications: researchers should bear in mind that developers may rationally delay patching their code.

Assumption: safe defaults are always good.

Reality: defaults presumed safe can cause unexpected consequences.

Several studies in the review advocated provision of secure defaults [102, 169, 314]. While it seems safe to assume that safe, secure defaults are always beneficial to security, there can be exceptions. It is not possible to anticipate every use of a piece of software in advance of releasing it [216]. Meng [169] advocated deprecating outdated encryption algorithms, but Gorski [99] noted that deprecation can entail locking API users to the version of the library that still contains the deprecated function, preventing them from receiving new updates. Naiakshina et al. [184] suggested that insecure options could be eradicated entirely, for example by removing deprecated hashing functions from encryption libraries. However, they noted that this could raise backward compatibility issues, potentially making applications even more insecure.

One of the recommendations often seen for secure development, dating at least from Saltzer and Schroeder's 1975 design principles, is that access decisions should be based '*on permission rather than exclusion*' [249], defaulting to denial of access. Massacci raised the issue of '*deny access*' being a potentially dangerous default response to authentication failure in cyberphysical systems [162]. He described a DDOS attack on the heating system of an apartment complex in Finland which caused freezing temperatures in the block for a week. A more deadly failure was the refusal of the software on a

Boeing 737 Max plane to allow the pilots to override a dive triggered by a faulty sensor [162].

Implications: before advocating for safe defaults, the relevant software's usage characteristics should be assessed.

Assumption: secure coding regulations and compliance constraints improve software security.

Reality: compliance requirements can have unintended consequences.

One approach to improving software security is to introduce legislation and regulations to impose security requirements. While this might improve overall software security, especially for those organisations that currently do not consider software security at all [13], it can have unintended consequences. Vendors may in some cases choose to misclassify issues to avoid notification requirements, reducing the likelihood that the issues will be properly addressed [216]. In other situations, certification may be lost if a major change is made. For example, with FIPS 140-2 certification, fixing a security vulnerability may be classed as a major change and entail loss of certification. Organisations may sometimes be forced to choose between the secure option of fixing the vulnerability and the insecure option of retaining the certification [111].

Implications: careful consideration should be given to the wording and structure of legislation and regulations.

4.4.2 Developer Assumptions

Assumption: all developers want to code securely.

Reality: some developers are uninterested in secure coding or feel that it is not relevant to them.

Rauf et al. [227] explicitly made '*the assumption that developers have internalised these security goals, i.e. they have become their own goals, regardless of their origin*'. However, some developers in the studies they analysed explicitly expressed a lack of interest in security goals [13]. In their literature review, the authors catalogued the reasons why developers, despite having this assumed interest in secure coding, failed to code securely. A general potential for security enthusiasm was also assumed by Weir et al. in their secure coding interventions [320].

This assumption is one of the '*tropes*' identified by Das Chowdhury et al. [62]; they called it '*security is a universal priority*' and cited it as an indicator of '*a mismatch between what security experts think developers value and what they do in practice.*' Many of the studies in this review indicate that this assumption is false. Ashenden and Lawrence reported a developer saying '*that security practitioners "are like a rock in a stream, we just flow around you"'* [12]. In a study of the use of a tool for detecting checked-in secrets, Rahman et al. found that 51.4% of the time the commit proceeded with a one-time or a permanent bypass of the warning [224]. They found that 20% of developers generated 85% of bypasses, and speculated that these developers were unusually busy or security unaware. However, it is equally possible that the developers were lazy, or indifferent to security. Discussing software security training for engineers, Crowther and Rust said that '*our first experiments...failed because the training was boring and optional*' [58]. When embedded with a software team, Palombo et al. were taken aback to find that some vulnerabilities were considered by their fellow developers to be features, because they reduced support demands [202]. Patnaik et al. identified a Stack Overflow behaviour they called '*passing-the-buck*', where developers '*don't care about learning and just want to be told what to do*' [210].

Implications: this assumption can cause researchers to misdiagnose reasons for insecure coding behaviour.

Assumption: all developers know how to code securely.

Reality: there is a very broad spectrum of code security knowledge and experience amongst software developers.

Developers vary widely in their level of basic programming ability [1] and also in their degree of software security knowledge and training [102, 81]. Nevertheless, some studies implicitly perceive all developers as having an ability to understand code security, and to code securely, and do not delve into this question. This may be because developers can be reluctant to admit ignorance on any topic, particularly in group environments [143]. One example is the series of studies on security interventions in organisations by Weir et al. [321, 319, 320]. These studies included workshops where code security was discussed in a group setting. Within the workshops, there did not appear to be space to admit to ignorance of secure coding practice, or a desire for secure coding training. Given that other studies have revealed a high degree of ignorance amongst developers

on software security [5, 183], this seems surprising. It is possible that developers are reluctant to reveal a feeling of ignorance when surrounded by their peers; this possibility was not explored, and exit interviews did not appear to ask about developers' feelings of competence. The '*blockers and motivators*' section in a paper on the long-term effects of the interventions [320] does not include any mention of developer ability, knowledge or need for training. This may be due to these omissions.

Contrasting with this apparent interchangeability of developers in a secure coding context is the widespread perception that secure coding in the work environment needs reinforcement from '*champions*:' people who are enthusiastic and knowledgeable about code security [276, 130, 95].

Implications: researchers should not assume that developers know how to code securely.

Assumption: developers' perceptions on whether security is needed in their work environment are accurate.

Reality: developers who are not well informed about software security may be poor judges of when it is required.

Developers are expert workers and are usually respected for their expertise. This can lead researchers to assume the correctness of developers' assessments of when security is needed. This appeared to be the case for Lopez et al. [152]. In an ethnographic study, they largely took developers' secure coding self-assessment at face value, missing an opportunity to dig deeper.

This assumption is related to the *'security is easy'* trope identified by Das Chowdhury et al. Developers can often suffer from the '*optimistic bias*,' which encourages them to believe that their software is too isolated, insignificant or obscure to be attacked [13]. The reality is that all software running on an online device can be targeted and exploited. Software that itself does not store significant data or perform significant activities may allow hostile actors to gain a foothold in a target's network or system [224]. Even software that does not run online may be leveraged by attackers if the host device is connected, using a technique known as '*living off the land*' [57]. Many developers are not familiar with hacking techniques and are poor judges of whether the software they are working on may be vulnerable to attack [247, 224]. Studies have found that developers can be over-confident in their secure coding abilities [184]. For example, Naiakshina et

al. found that several of their participants encoded passwords with Base64, thinking that this made them secure [183]. Base64 changes text's appearance, but it is designed to aid binary data transfer, has no security properties, and is trivial to decode.

Implications: researchers should not rely exclusively on developers' own assessments when judging whether software is vulnerable to attack.

Assumption: developers are enabled decision-makers not subject to psychological pressures like fear of downsizing.

Reality: experience of downsizing events may impact developers' confidence and motivation.

The Agile principles assume that developers are enabled decision-makers, and it has been argued that true Agile depends on psychological safety [34]. Job insecurity can affect organisational citizenship behaviours [160]. Current studies of software security behaviour do not explicitly take developer vulnerability or organisational turbulence into account. Haney et al.'s study of organisations with good security culture found such indicators of job security and employer respect as mentoring, training, and long-term, committed staff [111]. By contrast, when Weir et al. introduced security interventions into organisations, one of the three teams they worked with ceased to exist during the three-month lifetime of their initial interventions [320]. One year later, only one of the initial interviewees still worked for the company. Institutional memory cannot survive the disappearance of an entire team.

As discussed, numerous studies cite the value of software security '*champions*'; individuals who embrace the software security message and spread it to their peers [276, 111, 284]. Most papers advise that there is no need to specifically recompense champions for their software security role, and recommend sending them to security conferences and organising activities for them as motivators. It is difficult to see how such champions can sustain organisational commitment in an adverse environment. Security culture is widely recognised as an important ingredient of a successful secure coding environment [111, 174, 284, 239]; such a culture depends on mutual trust and is unlikely to thrive in a hostile corporate environment.

Yet current studies of software security behaviour do not explicitly take developer vulnerability or organisational turbulence into account. Assal and Chiasson's excellent survey on secure coding motivation [14] asked numerous questions on motivators

and blockers, but none concerned the effect of organisational turbulence or rounds of 'downsizing' on motivation. Anecdotally, and in the first author's extensive professional experience, such events affect employees' enthusiasm, loyalty and commitment to an organisation.

At time of writing in early 2023, the tech industry is in the throes of widespread layoffs of technical staff, including security staff. In this environment, the potential for good institutional secure-coding memory and sustained security practice and culture is not good. If developers work in institutions with a history of large-scale downsizing, or have experienced a mass-downsizing event, this may impact their belief that they can act autonomously. It may also affect their motivation to prioritise the organisation's best interests.

Implications: researchers should pay attention to developers' job security and the turbulence of their work environment when assessing software security motivation and practice.

Assumption: API and library developers are security experts.

Reality: API and library developers have the same range of security expertise as other developers.

Although API and library developers are, by definition, developers [62], some researchers seem to assume that they have more security expertise than developers. This may be because secure coding has been found to be associated with development experience in some studies [200, 5] (though not in others [196, 185, 183]). Many of the papers we reviewed dealt with API and library concerns such as documentation, abstraction level, level of support for developer testing and updates [181, 100, 169]. The implicit belief was that library developers understood, or should understand, the implications of their design decisions, and their impact on API users. Yet many libraries are developed by the open source community, where security guidance and policies may be more ad hoc [324].

Multiple studies show that library developers are not always better informed than other developers. Acar et al. noted that system developers' understanding of permissions on the Android ecosystem is not significantly better than that of other developers [1]. The issue of package updates in third-party libraries illustrates a similar failure by library developers to understand important security-related concepts; in this case, semantic

versioning. Semantic versioning indicates whether a fix is major, minor, or a patch, and can be used by library developers to indicate the likelihood of an update breaking backwards compatibility [121]. Since some updates are security updates, it is important to encourage developers to update libraries as frequently as possible. Developers are reluctant to update libraries because dependency updates can break code [207]. They would appreciate well-indicated security fixes that avoid breaking changes [207], but these are often not available. Chinthanet et al. found that over 90% of npm fixes studied were bundled with unrelated commits [41]. Library developers often fail to use semantic versioning correctly; this is a likely contributor to the reluctance of developers to update libraries [67].

Implications: researchers should not assume that API developers have any special security expertise.

Assumption: all coders have access to the same resources.

Reality: constraints include background, training, isolation, the organisational secure-coding environment, and budget.

Some studies implicitly assume that all developers have access to the same resources, and could code securely if they wanted to [17]. In fact, developers' environments vary widely and they may be subject to constraints that are not immediately apparent. The relatively isolated security environment of app developers was examined by van der Linden et al. [305] in a study of security decision-making. Working for an organisation is not necessarily better, as some organisations lack the positive security climate that facilitates developers who wish to code securely [10]. Developers constrained to use languages with characteristics that tend to cause high levels of vulnerabilities, such as C and C++ [204], are at a secure-coding disadvantage.

When it comes to issues of budgetary constraint, there is not much work available in the literature. Indeed, researchers themselves may be constrained; a study analysing and comparing static analysis tools only included free tools [268]. At a time when premium charges for basic security features like cloud-hosted logging is in the news [116], the effect of budgetary constraints should be considered by researchers.

Language comprehension may be another issue that affects some researchers. Votipka et al. found that 78% of the coding mistakes in their '*Build It, Break It, Fix It*' series of secure coding challenges were due to participants misunderstanding requirements

[314]. Gorski et al., studying the embedding of security information in documentation for non-security APIs, noted that their recruits were German speakers, who might struggle with English documentation [101]. This has implications for language use and simplicity, and would be an interesting topic for further research.

Implications: when analysing developers', teams' and organisations' security decisions, researchers should consider their budget and other resource constraints.

4.4.3 Organisational Assumptions

Assumption: managers are software security champions.

Reality: many managers are ill-informed about software security, fail to grasp the risk involved, and/or perceive it as unimportant.

Implicit in discussions of how to introduce interventions into organisations is the assumption that management in the organisation will be in favour of increased software security [321, 319, 320, 227]. In reality, managers are not always aware of the importance of software security [219]. Senior managers, though paying lip service to software security, may perceive a diversion of resources to software security as impinging on more important goals [13, 151]. Project managers and product owners sometimes have little interest in security, and this can be exacerbated if they perceive that security is taken care of by someone else such as a security officer [285].

Implications: when evaluating software security within an organisation, researchers should explicitly assess the stance of upper, middle and line management.

Assumption: organisations prioritise software security.

Reality: not all organisations prioritise software security.

Throughout the literature, we observe an unstated assumption that organisations prioritise software security [202, 320]. Like maintainability and efficiency, software security is an NFR. While managers assume that NFRs will be present in software, they do not always explicitly require them [267]. This has led to an assumption that managers would require and prioritise software security if they thought of doing so. This assumption is not borne out by the evidence. It is rare, though not unknown [13], for managers to explicitly state that software security is not a priority. Actions signal priorities more clearly than words though, and many studies have found that security is

not prioritised in practice [174, 13, 202]. Functionality usually takes top priority. Tight deadlines and inadequate staffing and training may result in decisions to release software that is likely to be insecure [280].

As discussed in Section 4.2, Palombo et al.'s ethnographic study found numerous secure development issues [202]. The researchers seemed to assume that the organisation prioritised security, but that it slipped through the cracks due to the development team being subject to conflicting priorities [202]. However, a close reading of the paper reveals a wholesale absence of management concern about software security. An absence of management support and instruction affects secure development emphasis [81]. Teams receive direction on priorities from the organisation they work in, and are unlikely to prioritise security if not asked to [184, 185, 183, 182, 11].

Implications: researchers should pay attention to the priorities developers receive from their managers, both stated and unstated.

Assumption: the organisational environment is cohesive and therefore can control the complexity of secure software development.

Reality: in large organisations, different departments and teams may have different priorities, causing conflict over software security goals. Subcontractors may complicate the issue.

It is easy to assume that within an organisational environment, there is a coherent and cohesive software security policy. All aspects of the organisation move in concert. An example of such an environment appears in a study by Wang et al. describing an API hardening approach to XSS elimination at Google [317]. This was carefully designed and engineered to run on the enormous Google codebase, which is largely contained in a single repository.

However, in many large organisations software security policy may be fragmented. Security checks required by one department may conflict with the goals of another department or unit [276]. Subcontractors complicate the issue [280]. Their goals are unlikely to be aligned with the long-term success of the organisation. They will generally focus on meeting tight deadlines within budget [174].

Implications: researchers should pay attention to inter-departmental dynamics when studying large organisations.

Assumption: the presence of secure coding activities implies that security is a high priority within an organisation.

Reality: secure coding activities may be compliance driven, with little concern for actual security.

Assal and Chiasson [13] evaluated 13 teams on whether they undertook security activities at different stages of the software development life cycle. From their results, they identified '*security inattentive*' and '*security adopter*' organisations. All of the organisations they studied fell firmly into either one group or the other. However, the existence of the '*checkbox*' phenomenon is well documented [285, 111, 12, 222]. This is a tendency of an organisation to undertake software security activities for compliance reasons, without an inherent security concern. Studies have shown that some organisations will do the minimum work necessary to pass known checks; for example, one study found that organisations evaluating PCI compliance did not test widely and did not vary their tests, allowing client organisations to limit security work [222]. This mindset is not universal. Enck and Williams identified considerable security energy amongst organisations debating supply chain security, despite compliance weariness [77].

The absence of secure coding activities is an indicator that an organisation is security inattentive, but the inverse is not necessarily true. In the absence of a security culture, organisations may undertake a minimal amount of code assurance. This can lead to trouble as the software security problem will not have the drivers it needs [239]. One suggested solution to this issue is to use organisational climate theory to improve the climate for software security within the work environment [11].

Implications: researchers should assess the secure coding culture of an organisation as well as its secure coding activities.

4.4.4 Research Assumptions

Assumption: developers and/or organisations will admit to a lack of understanding of, or enthusiasm for, security.

Reality: security is seen as a positive quality, and therefore both developers and organisations are unlikely to admit to a lack of understanding of it or interest in it.

Developers may be reluctant, particularly in group interviews that include colleagues, to admit to a lack of knowledge about software security [143]. There may be power dynamics and group history at play. The same is true of admitting to a lack of interest. Most organisations are reluctant to admit to corporate failings [316]. The chances of their acknowledging that they are ignoring software security are slim. Thus it is important to probe and explore further, rather than taking statements on software security at face value.

In one study where the researchers had close relationships with management, negative feelings towards security may have been suppressed by the development team in survey entries for fear of management finding out [219]. Several people chose '*neutral*' on security matters, which seemed equivocal. Weir et al. found a poor correlation between survey respondents' claim to use assurance techniques and the actual security of apps they had written [322]. They did not appear to consider that the self-reported use of assurance techniques could have been exaggerated.

Implications: individuals' and organisations' assertions on prioritising software security should not be taken at face value.

Assumption: activities that can lead to insecure code cannot also lead to secure code.

Reality: some common coding activities are widely used, and do not reflect the security of the resulting code.

This assumption is made in a study of the use of copy-and-paste from Stack Overflow and other online forums [17]. Acar et al., in a study comparing different information sources during coding, found that use of copy-and-paste from Stack Overflow resulted in insecure code [3]. However, Naiakshina et al. did a qualitative evaluation of participants' development process during their studies on secure password storage [185]. They found that all participants who produced secure solutions used copy-and-paste from an online location. In a study on the usability of five different cryptographic APIs, Acar et al. found that there was a statistically significant correlation between the use of copy-and-paste and the functional correctness of a task, but there was no such correlation between copy-and-paste and the security of the solution [2]. This shows that use of copy-and-paste helps developers to achieve functional correctness. If they are sufficiently competent to produce functionally correct code they are likely to be using copy-and-paste. They will

continue to use it while trying to secure the code; this does not affect the security of the code.

Bai et al. searched for instances of known cryptographic errors on Stack Overflow and matched them to code in GitHub projects [17]. For projects where they found a match, they invited the developers for a survey and interview. One of the limitations listed in the paper was that they did not include a control group. Such a group would be difficult to identify; it is impossible to prove that software is secure; a new defect could be found tomorrow. But a group could have been approximated by identifying secure code snippets in Stack Overflow, matching them to GitHub projects in the same way as was done for insecure snippets. This might have revealed that a good deal of secure code in GitHub originated with Stack Overflow.

Implications: researchers should assess context of research findings, including whether control groups were used.

Assumption: findings in a single paper are final and can be cited confidently, with no need for replication.

Reality: scientific evidence requires confirmation, preferably from different types of study.

The findings of a single study are not conclusive. Findings should be replicated, or even better, reached in multiple ways by triangulation [178]. However, several well-designed studies in the DCS field are routinely referenced with no reservations as to whether their results have been reproduced elsewhere. For example, the several papers based on a 2014 study of security tool adoption [329, 334] appear to be considered definitive. While not wishing to detract from that original work, we suggest that further investigation into this phenomenon could shed new light on security tool adoption and use. Similarly, work by Oliveira et al. on API blindspots is frequently cited [196]. This study found a correlation between the personality trait of openness and an ability to fix security issues in the presence of blindspots, suggesting novel recruitment strategies could be developed based on this finding. Given the potential impact on candidates of filtering recruits by their level of openness, an attempt at confirming the finding appears to be indicated.

In a good example of confirming findings, Naiakshina et al. and Danilova et al. conducted a series of studies on secure password storage with different developer populations

including students, freelancers and professional developers [184, 185, 183, 182, 59]. The studies consistently and markedly found that explicitly asking for security has a strong impact on whether developers attempt to code securely.

Implications: researchers should not perceive research questions as definitively answered unless there is a cumulative body of research.

Assumption: measuring secure coding activities is equivalent to measuring code security.

Reality: code security for non-trivial software cannot be definitively measured.

It is relatively easy to confirm that a piece of software is **in**secure; one need only find a vulnerability. By contrast, it is impossible to confirm that complex software is secure. The best that can be done is to confirm that to date, it has no known vulnerabilities. There was only one study in the review that attempted to assess the actual security of software [322]; its results were inconclusive. Absent a widely-embraced standard way to assess secure coding in an organisation, many researchers have devised measures of their own [175, 200, 13, 232]. These are all based on the performance of widely-recognised software security assurance activities. Such measures are usually recreated by each group of researchers and tend not be reused across multiple studies. The Ryan metric, a lightweight measure based on empirical data, was proposed by Ryan et al. [239]. Whatever measure is used, researchers should avoid conflating use of security assurance techniques with code security.

Implications: researchers should bear in mind that the level of secure coding activities may not reflect actual code security.

Assumption: if secure coding activities take place, they will be reflected in artefacts.

Reality: artefacts do not always reflect work done.

Several studies that examined artefacts such as source control comments for evidence of whether activities were performed [211, 277, 126] were included in the review. This approach will not always find evidence of activities even if they are in fact performed; there is not necessarily a correspondence between how frequently something is done and how frequently it appears in artefacts. Morrison et al. [177] noted that their artefact analysis did not reliably reflect the use of techniques that they had confirmed via a

survey and in qualitative analysis. Palombo et al. found that developers rarely used security-related terms in defect ticket descriptions. They had to manually correlate tickets with implementation code [202].

Implications: definitive conclusions should not be reached based on artefacts alone, although they can be used to triangulate.

Assumption: artefact analysis done at scale is error-free.

Reality: artefact analysis when done at scale may be flawed and results should be viewed with caution.

Large-scale artefact analysis was used by a number of interesting studies in the review. Since it uses automated analysis tools, usually created by the researchers, this method can be subject to large-scale analysis failure that remains undetected. Good research in this area [223] will include manual checking of the results of automated tools, and will acknowledge threats to validity [121].

Thompson and Wagner attempted to assess the correlation between code review and security vulnerabilities in 3,126 GitHub projects using 143 languages [277]. They used metadata analysis techniques to determine code review status and machine learning to identify security vulnerabilities. They found a weak negative correlation between code review and numbers of both '*all issues*' and '*security issues*.' The techniques used to identify issues did not ascertain issue severity, limiting the usefulness of the results for helping to decide on code review strategies. The authors were forced to make a number of approximations and assumptions to determine what is a code review and what is a security defect when analysing source code at large scale. They made a good attempt at both objectives and gave detailed explanations of their approach. This makes the paper a valuable learning tool, but a degree of scepticism should be applied to the results.

Fischer et al. developed a pipeline to match security-related code snippets in Stack Overflow (identified using machine learning) to applications from Google Play [88]. They stated that 15.4% of the Android applications contained security-related code snippets from Stack Overflow. 97.9% of these applications contained insecure snippets. However, from reading the paper, it was difficult to determine how much verification of the Android snippet detection they had done or been able to do.

Paul et al. looked for factors that impacted on the success of code reviews in finding security vulnerabilities in the Chromium project [211]. They compared attributes of

code reviews that successfully detected vulnerabilities to attributes of code reviews that failed to detect vulnerabilities that were subsequently discovered. The study made use of automation to find relevant commits, with extensive checking by the authors to verify results. The result was a convincing paper finding that the time taken to complete a review, whether it was a bug fix review, and the number of mutual reviews between two developers, had a positive impact on the probability that a vulnerability would be found.

Imtiaz et al. studied five open source projects that had used static analysis tool Coverity for at least five years, attempting to answer six research questions such as how developers use the tool and what their fix rate is [126]. They found that the developers fixed between 27.4% and 49.5% of alerts. They also found that developers may take a long time to fix the alerts. Since all their analysis was done via automated processing of commits and commit messages, they could not provide any insights into their findings beyond conjecture.

Implications: researchers should carefully evaluate artefact analysis steps when considering results.

Assumption: coders, product managers, and testers can all be studied as a homogeneous group.

Reality: different disciplines need different types of support.

This assumption was made by Weir et al. [320] when introducing security interventions. Interventions were introduced to teams, with team members not differentiated by function. Similarly, in an ethnographic study examining how organisations respond to security events, Lopez et al. included programmers, testers, designers and product managers in their opening definition of '*software developers*' [151]. While conflating all these roles may make sense in some contexts, in others this assumption is unhelpful.

Project managers and product owners may have little interest in security, and may need training or support to engage with software security [285]. Studies that overtly look at all of these roles but implicitly focus on coding without distinguishing other functions are likely to miss important nuances. Coders and their managers do not have the same challenges and demands [130]. They need different training and support [219].

Implications: researchers should consider the differing support needs of the different roles that they are studying.

4.5 Discussion and Conclusion

This paper revealed a considerable number of assumptions in prior literature focusing on developer centred security, and identified by the first author who has extensive practical software security experience. We note that such assumptions can only be identified using the reader's knowledge, experience, grasp of the field and familiarity with other research, as described in the positionality statement in Section 4.3.5, and that additional assumptions could be present which we did not identify. We now discuss the implications of our findings organised by category of assumptions. Researchers that study secure coding could use the list of assumptions as a '*checklist*' during study design, to ensure that the study does not inadvertently rely on any of these assumptions.

Technical Assumptions. Technical assumptions are those that make generalising claims regarding the nature of code, processes, and tools. These assumed generalised truisms are problematic because they oversimplify reality, which means that studies may be based on assumptions that do not generally hold, and that findings cannot be generalised to other settings. In other words, context matters, and 'it depends.' What factors study findings may depend on is something to consider in designing future studies.

Developer Assumptions. Developer assumptions are those that make generalising claims regarding the developers who are expected to produce secure software. For example, studies assume that developers *want* to produce secure software (i.e., they care about it), or that they *know how to*. Other assumptions include those that developers have considerable decision power in organisations, or that they are well resourced to develop secure code. Clearly, these assumptions do not hold across organisations, and so future studies should consider these aspects, for example, by investigating developers' training, context and resources. It may very well be that a major obstacle to secure code is uninterested or under-resourced developers, rather than a missing process.

Organisational Assumptions. Organisational assumptions are those that make generalising claims regarding the nature of organisations and their management. From the secure software researcher's perspective, secure code is a key topic of interest. It is clear from our findings that this is not always the case in organisations. However, no organisation will ever acknowledge that secure code is not a top priority, therefore we must scratch below the surface. Having secure coding policies in place is not an indicator of an organisation's true commitment to delivering secure code. Future studies, for example

ethnographic studies, may be able to observe the contrast between what organisations and managers say, and what they do on a day-to-day basis.

Research Assumptions. The fourth category, Research Assumptions, are assumptions that tend to result in generalising statements regarding organisations, study findings, observations, artefacts, and stakeholders. Some of the assumptions are not specific per se to secure coding research. For example, the assumption that a study's findings are '*final*,' in that they do not need to be replicated or put into context, is a general issue within software development research. Other assumptions relate to how researchers operationalise concepts under study. For example, studies that seek to measure code security might consider secure coding activities, but clearly they are not the same; the proof is in the pudding, not the recipe.

Conclusion In this paper we discussed a number of unhelpful implicit assumptions that can adversely affect software security research. We outlined cases where the assumptions appeared to hold and discussed the reality that they obscured. We considered the implications for researchers when considering the production of secure software.

5 Measuring Secure Coding Practice and Culture: A Finger Pointing at the Moon is not the Moon

This chapter was published as: Ita Ryan, Utz Roedig, Klaas-Jan Stol. 'Measuring Secure Coding Practice and Culture: A Finger Pointing at the Moon is not the Moon.' *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. DOI: 10.1109/ICSE48619.2023.00140.

5.1 Abstract

Software security research has a core problem: it is impossible to prove the security of complex software. A low number of known defects may simply indicate that the software has not been attacked yet, or that successful attacks have not been detected. A high defect count may be the result of white-hat hacker targeting, or of a successful bug bounty program which prevented insecurities from persisting in the wild. This makes it difficult to measure the security of non-trivial software. Researchers instead usually measure effort directed towards ensuring software security. However, different researchers use their own tailored measures, usually devised from industry secure coding guidelines. Not only is there no agreed way to measure effort, there is also no agreement on what effort entails. Qualitative studies emphasise the importance of security culture in an organisation. Where software security practices are introduced solely to ensure compliance with legislative or industry standards, a box-ticking attitude to security may result. The security culture may be weak or non-existent, making it likely that precautions not explicitly mentioned in the standards will be missed. Thus, researchers need both a way to assess software security practice and a way to measure software security culture. To assess security practice, we converted the empirically-established 12 most common

software security activities into questions. To assess security culture, we devised a number of questions grounded in prior literature. We ran a secure development survey with both sets of questions, obtaining organic responses from 1,100 software coders in 59 countries. We used proven common activities to assess security practice, and made a first attempt to quantitatively assess aspects of security culture in the broad developer population. Our results show that some coders still work in environments where there is little to no attempt to ensure code security. Security practice and culture do not always correlate, and some organisations with strong secure coding practice have weak secure coding culture. This may lead to problems in defect prevention and sustained software security effort.

5.2 Introduction

Software security was defined in 2004 as *'the idea of engineering software so that it continues to function correctly under malicious attack'* [165]. Since then, our world has become increasingly connected, and the hacker community has morphed into a global criminal industry. Insecure software is now routinely exploited for ransomware, cybercrime and cyber-espionage purposes, so that software security is an increasingly urgent requirement [238].

Academia has not reached consensus on a way to measure the level of secure coding in an organisation or open source team. Ethnographic studies provide rich qualitative data, but by their nature [264] do not produce quantitative measurements or generalisable conclusions. Most researchers attempting quantitative secure coding research synthesise a number of practices from industry methodologies and assess participants' use of them. These synthesised secure coding practice lists tend to be used only by their authors. They are not empirically evaluated, and if fine-grained may be too rigid to be re-used over time, as software security is a quickly-evolving field.

To explore these issues, we conducted a large-scale survey of software developers. In a recent analysis of a large trove of industry secure-coding data, twelve security activities were identified by Weir et al. [323] as being those most commonly adopted in secure coding initiatives by large organisations. Given their empirical backing, we adopted these 12 Common Activities (CAs) as our core measurements for secure software coding practice.

To measure security culture we devised a list of questions grounded firmly in prior literature.

In this paper we seek to address the following research questions:

Research Question 1: *What are the secure-coding characteristics of our sample group?*

Research Question 2: *What are the security culture characteristics of our sample group?*

Research Question 3: *Do secure coding practice and culture correlate, and if not, what lessons can we learn to help support the development of secure coding?*

This paper makes the following contributions: first, we present the '*CA Score*', a lightweight instrument for assessment of the level of software security practice in a work environment. Second, we provide an overview of the current state of security practice based on our survey results. Third, we introduce a group of questions designed to probe security culture. Fourth, we assess security culture in our participants' working environments. Finally, we explore the interplay between security practice and security culture.

The paper is organised as follows: background and related work are discussed in Section 5.3, method is discussed in Section 5.4, an overview of the sample is given in Section 5.5, results are given in Section 5.6. Section 5.7 contains the discussion and threats to validity, and concludes the paper.

5.3 Background and Related Work

5.3.1 Measuring Secure Development

While assessing the security of software developed in controlled studies is feasible and repeatable [184, 183, 182], measuring secure development in organisations and open source environments is an ongoing issue in academia. As far back as 2012, Jaatun [128] discussed whether software security can be measured at all. Counts of known security defects for publicly available applications could be used; however, some applications are subject to considerably more analysis and attack than others. An absence of known vulnerabilities could simply indicate that the application has not been closely studied. An example of an attempt to demonstrate a correlation between use of assurance techniques and decreased levels of security issues in code comes from Weir et al. [322]. They

surveyed 335 Android app developers on their use of security assurance techniques, downloaded a free app from each developer, and used automated analysis to detect instances of three categories of security issue in the apps. Surprisingly, app analysis showed no increased security for apps whose creators claimed to use assurance techniques, and more cryptographic security issues for apps whose creators had access to security experts. The authors concluded that issues with analysis tools and missing cryptography may have skewed the analysis. This study illustrates the complexity and difficulty inherent in determining whether secure coding efforts are correctly reported, and whether they result in secure code outcomes.

BSIMM reports [171] on contemporary software security activities are compiled semi-annually by Synopsys, who interview contributing organisations about their security practices. For example, in 2021 BSIMM 12 collated data from 128 organisations. The contributing organisations are usually large and have active software security strategies. Jaatun [128] discussed adopting a BSIMM-like analysis method for academic studies. Variations on this approach of measuring activities from BSIMM and other industry practices and guidelines such as OWASP's Software Assurance Maturity Model (SAMM) [198], MS-SDL [170] and Software Assurance Forum for Excellence in Code (SAFECode) [246] have been widely used by the research community [129, 175, 200, 232, 13, 313]. BSIMM currently includes a total of 122 activities, with correspondingly large numbers of similar activities in the other methods. This is cumbersome, therefore a synthesised subset is usually generated. Each research team measures different activities. New groups of activities to measure are continually defined. None are empirically evaluated. Research teams rarely reuse the groups defined by previous teams, making results difficult to compare.

Similarly, Morrison et al. [176] found that 85% of 324 unique security metrics had been proposed and evaluated solely by their authors. They concluded that there is no convergence as yet on an accepted set of metrics in the software life cycle security metrics field.

5.3.2 Security Culture

It might be argued that engaging in security practices would naturally entail a good security culture. However, many observers have noted that organisations may follow

recommended practices (perhaps for compliance) but have a box-ticking attitude to security. Although mentioned in several papers [285, 111], and implied in others [12, 174, 222], this phenomenon and its implications are rarely explored. It may lead to the implementation of security activities purely to demonstrate compliance with regulations or standards such as the payment card industry's DSS [213]. Such standards may have a narrow focus [222], and security investment which focuses solely on compliance is not sufficient to ensure secure software [115]. The significance of the organisation's security posture in the area of secure coding is widely acknowledged in academia [227, 207, 286]. Assal et al. [14] found that most deterrents to developers coding securely relate to the absence of clear secure-coding plans, resources and priorities in the developers' working environments. Concluding that it is important for companies to build a security culture and offer learning opportunities for secure coding, they suggested that a research focus on problems in the organisational approach to security would increase understanding of software security deterrents.

Recent ethnographic studies have provided useful insights into the software security posture in organisations. Lopez et al. [152] immersed themselves in the software development unit of a large company, examining how non-experts treat security during their daily tasks and focusing primarily on developer security motivation. They concluded that the organisation had put procedures and activities in place to ensure software security, and that the developers accepted these and attempted to implement them in good faith. This resulted in *'mostly secure'* software. By contrast, Morales et al. [174] published a devastating paper on a well-funded project using Agile and implementing DevSecOps without rigorous adherence to DevSecOps principles, and the resulting security implications. It is a valuable reminder that no development method can compensate for bad management. Broken pipelines, adversarial subcontractor relationships, inadequate definitions of done, poor test resources, a failure to emphasise security and more all combined to produce a product with poor security assurance. On paper, this dysfunctional project adopted appropriate security practices. So where did it go wrong?

The differentiating factor between these two organisations is security culture, defined by Haney et al. [111] as *'a subculture of an organization in which security becomes a natural aspect in the daily activities of every employee.'* Haney et al. found that all of the cryptography-focused organisations they studied placed a high value on a strong security culture. Tuladhar et al. [284], having worked embedded in an organisation

adopting secure coding practices, concluded that key security culture enablers were upper management setting security as a goal, and the team applying security knowledge in context. In a longitudinal study, Tøndel et al. [285] noted the adverse effect of managerial lack of interest in security. They also observed that developer security work was often invisible to others and therefore did not help build security culture.

Managerial attitude affects the question of how individual developers' security enthusiasm is regarded within their work environment, an important aspect of security culture that has not been much explored by the research community. Jaatun et al. [129] noted organisations' dependence on individual developers' enthusiasm. Tahaei et al. [265] examined the experience of privacy *'champions'* in software teams, finding that they play an important advocacy role. Ryan et al. [237] identified a *'hero'* software security archetype; a coder struggling to introduce secure coding practices in a security-hostile environment. Such environments are difficult to study, since most security experts work in roles where security is already a focus [112]. Security interventions, such as those introduced to organisations by Weir et al. to attempt to empower developers to code securely [318], require prior organisational agreement. Thus a level of management support is assured.

Developers' tool adoption behaviour may give useful insights into the work environment with which they must contend. Papers on tool adoption have focused on social influences [334]. Although Witschey et al. [329] observed that many developers seek out information on security tools when needing to write secure code, they did not examine how developers can introduce such tools into their working environment, and what constraints they encounter when attempting to do so. An ethnographic study by Palombo et al. [202] shed some light on how heroes can succeed in improving security in a *'security inattentive'* [13] environment. Two researchers used and advocated a *'co-creation'* model for secure development in which they added security checks seamlessly, with no developer friction for other team members. While this required considerable investigation and work, other team members then adopted the checks, possibly because of their frictionless nature.

The interplay between developer and work environment is the key to security culture. Therefore, ways to measure it are of interest. One suggested measurement tool is the Secure Software Development Self-Efficacy Scale (SSD-SES) [313], although a

developer's self-efficacy can arise from previous employment or personal motivation rather than their current work environment.

5.3.3 Organisational Climate Theory

Academic work on organisational culture originated from anthropology and considers shared myths, beliefs and ritual [251]. The term '*security culture*' as used in software security literature is adopted from industry [111], and may have more in common with organisational climate theory, which provides a way to consider and evaluate the organisational '*climate*' for differing organisational '*facets*.' For example, an organisation may demonstrate a consistent commitment to the software security facet by halting releases when there is a security concern. Arizon-Peretz et al. [11] applied this theory to software security, asking developers about privacy and security activities in their organisation and interpreting the answers via organisational climate theory. They categorised activities into seven climate themes which provide cues to developers as to the relative importance of security and privacy within the organisation, and what is expected of them in these areas. They found that the cues that developers get can be confusing and do not clearly indicate that security and privacy should be prioritised in their daily work. They proposed that climate theory can be used to change this and to promote security and privacy by design within organisations. In a follow-up paper [10], they measured organisational climate for the software security facet using questions derived from the seven climate themes and tailored to a multinational organisation. They found an interplay between security self-efficacy, proactive security behaviour and a positive organisational climate for software security within the organisation.

5.4 Method

We conducted a large-scale cross-sectional survey. In this section we discuss the design of the questionnaire, data collection procedures, data screening procedures, and data analysis procedures. Data and scripts are available online [240].

Surveys that constrain respondents' answers to specific values can impose forced choice bias, which is inappropriate in a context where increased understanding of context is sought [91]. Therefore we allowed free text in many questions, and asked a question

towards the end looking for comments on any aspect of the survey or of the respondent's secure software development experience. The free text aspect of the survey provided us with many interesting insights. As the use of *'(sic)'* can be seen as condescending, we do not use it when quoting respondents. Any quote from respondents is presented exactly as entered in the questionnaire.

5.4.1 Survey Questions

Initial questions concerned demographics and participants' personal relationship with secure coding. These questions are not the focus of this paper and most are not discussed. Some standard secure coding questions were asked; answers are discussed in Section 5.5.

5.4.1.1 Screening Questions

One concern in sample research is that the sample is representative. We were only interested in responses from people who were coding frequently at the time of answering the questionnaire, so we began by asking respondents *'Do you write computer code frequently, either professionally or open source?'* Those who answered *'No'* were excluded from further participation.

Researchers running secure software development surveys have encountered issues with respondent quality when offering payment for completion. Witschey et al. [330] cleaned 313 suspiciously-speedy participants from their survey, leaving only 61 usable responses. Danilova et al. [61] found that of 129 respondents sourced from Qualtrics, 96 did not pass programming tests. As a result, Danilova et al. [60] assessed a number of screening questions to filter out non-developers from paid studies. The Danilova paper recommended two programming comprehension questions as the best screening choices in surveys where a time penalty is acceptable. These two questions have since been used successfully, for example by Kaur et al. [135]. We adopted the two questions, adapting the wording slightly since the answers documented by Danilova et al. are in the public domain. We had some interesting findings on them, discussed in Section 5.4.3.

Since our questionnaire was relatively long with 62 questions, we wanted to ensure that participants were paying attention all the way through. Rather than ask complex attention questions with opposing answers, we borrowed a simple strategy from Murphy-Hill et al. [180], simply stating for example *'This is an attention question. Please select a little.'*

We had two such questions, designed to blend with adjacent questions so that participants who were simply clicking blindly would not notice them. This strategy caused some amusement; for example, one participant noted as a comment at the end of the survey *'The attention questions are funny ^^.'*

5.4.1.2 Measuring Security Practice

We asked our participants whether they were aware of each of the 12 CAs in their working environments. These questions can be found in Table 5.3. More detail on the rationale for these questions can be found in Section 5.6.1.

5.4.1.3 Measuring Security Culture

Security culture questions were based on an extensive literature review of software security papers since 2016. These questions can be found in Table 5.4. More detail on how these questions were devised can be found in Section 5.6.2.

5.4.1.4 Pilot Testing

The questionnaire was pilot tested in two phases. First, we asked for criticism and feedback from a developer with over two decades of experience. After making recommended changes, we pilot tested again with the same developer and two other highly experienced developers. Small adjustments suggested by these testers were made before launching the questionnaire. Their responses were not included in reported results.

5.4.2 Data Collection

The survey was publicised via personal contacts, a conference talk by the first author, and social media platforms such as LinkedIn, Facebook, and Twitter. We found that it was possible to generate significant interest by asking for retweets on Twitter, posting to Facebook development groups, and promoting the survey on LinkedIn. The survey was open for just over 3 months, to the end of January 2022. This long window provided time to publicise the survey, and helped build momentum. At closing time we had received 1,100 responses.

Given the considerable interest we succeeded in generating in our survey, we did not need to offer financial incentives or use platforms such as Prolific to source participants.

This removed the danger of non-developers participating for financial gain. It remained important to carefully screen responses to ensure high quality analysis; we discuss the screening process next.

5.4.3 Screening Process

As only the questions on programming expertise were mandatory, and some people think more quickly than others, we did not remove responses that were completed quickly.

Fifty-two respondents answered *'No'* to the initial screening question on frequent coding and were brought directly to the end of the survey, leaving 1,048 respondents. We then proceeded to the two mandatory programming expertise questions. We had intended to reject all participants who got either of the programming questions wrong. However, a tranche of six or seven survey entries was obtained immediately after the first author gave a software security talk to an audience of SQL programmers. These respondents included SQL in their list of programming languages, and some got the second programming question wrong. Upon reviewing the questions, we realised that these included pseudo-code that would be typical of C++ or Java interview questions but is meaningless in SQL. The questions' limitations were confirmed later, when a survey participant communicated with the first author, commenting that Python programmers could have difficulty with these questions. Fifty-five respondents answered one or both of the programming screening questions incorrectly. Similarly, we had planned to remove any respondent from the data set who got an attention question wrong. This seemed like an uncontroversial position, but one participant, answering Question 60 which gave an opportunity to comment on the survey, responded *'Ididint understand q56. Attention? No idea what you are wanting from this poor question.'* Thirty-seven respondents failed one or both of the attention questions.

Having considered the feedback on our screening questions, we decided to consider two groups of respondents during analysis. Group 1 ('*all valid participants*') includes all respondents who confirmed that they write computer code frequently (n=1,048). Group 2 ('*all correct participants*') additionally omits respondents who got programming and/or attention questions wrong (but retains the respondent who explicitly told us that they didn't understand the attention questions), giving a total of 962. Statistical results

reported in the paper are based on Group 2; however, we also ran tests on Group 1. We found no significant differences between the two groups.

5.4.4 Ethics

Our questionnaire involved research on human subjects, and therefore we obtained approval for the questionnaire from our institution's Social Research Ethics Committee. Participants were advised that taking the questionnaire was not obligatory, and were asked to consent to taking the questionnaire. All submissions were anonymous.

5.4.5 Data Analysis

Statistical analysis was done using R [221]. Positive answers to the 12 CA questions were summed to create a CA score between 0 and 12, representing the secure coding level in an environment. The Pearson correlation coefficient was used to assess correlations between this score and security culture indicators, measuring correlation of increasing scores with increasing levels of these indicators [192]. Participants were asked to choose '*Not Applicable*' where relevant; for example, if they worked alone and the practice required co-operation. For the Common Activities, participants were also provided with an '*I don't know*' option. Levels of '*I don't know*' responses varied from 5.93% (CAQ1) to 22.6% (CAQ12). Since levels were high for some questions, we ran correlations excluding NA and '*I don't know*' responses. We did not find any differences in these results that would affect the paper's conclusions. Therefore, the correlations given in the results section do not omit '*Not Applicable*' or '*I don't know*' answers. Qualitative data from open-ended survey questions was used to provide context for our quantitative results.

5.5 Sample Description

5.5.1 Demographics

Respondents ranged in age from 18 to over 65 (see Fig. 5.1). We asked respondents what gender they most identified with. The majority of respondents answered *'Man.'* The numbers seemed to correspond with approximate gender balance numbers in the industry (see Table 5.1). We asked developers what country they live in, allowing them to add

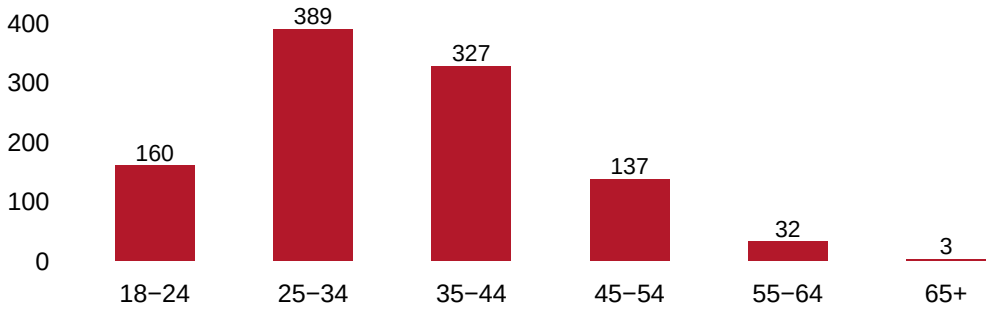


Figure 5.1: Age distribution of survey respondents

Table 5.1: Gender distribution of survey respondents

Gender	Frequency	Percent
Man	852	81
Woman	94	9
Non-binary	63	6
Prefer not to say	34	3
Other	4	0.4

their country if it was missing. We obtained responses from 59 different countries across all continents except Antarctica. The highest numbers of respondents came from the United States (453), United Kingdom (110) and Germany (92).

5.5.2 Programming Languages and Tools

We provided developers with a list of programming languages and other technologies and asked which they used frequently, also allowing free text entries. We ended with 73 programming languages, of which JavaScript (454), Python (447) and HTML/CSS (332) were the most popular. Some single-user languages, such as CHICKENscheme and MoonScript, were entirely new to us. We asked about other technologies; 129 were entered. Git (966) was by far the most popular, followed by Docker (470), PostgreSQL (349) and Amazon Web Services (AWS) (336). Sixty-eight unusual technologies appeared only once, including Cassandra, ScyllaDB, and Godot, which is apparently '*The game engine you waited for.*'

The field of tools used to enhance code security is rapidly evolving. We wanted to get a sense of which are in general use. Asked which security tools they use, developers entered over 100 different tools. SonarQube (196) and Clang Analyzer (175) were by far the most popular, but the large number of tools indicates little convergence in the tools market.

5.5.3 Adoption of Secure Coding Standards

We asked developers what secure coding initiatives or standards they are aware of in their organisations or teams. We provided a list of well-known initiatives, and allowed developers to add their own. By far the most common response was *'None'* (304), followed by *'Not applicable'* (212). PCI-DSS, used in the payments industry, followed at 78, with FIPS 140-2 (43), Common Criteria (37) and US-CERT's Top 10 Secure Coding Practices (33) also fairly common. Although included in the predefined list, industry secure-coding standards referenced in academia fared badly; the BSIMM got only three mentions, OWASP's SAMM two, and SAFECode five, and of the commonly-referenced general approaches only MS-SDL hit double digits at 29.

The free text section provided further insights. MISRA, standards set by the Motor Industry Software Reliability Association, appeared seven times. Mention was also made of government standards from France, Germany, the US and the UK. We had inadvertently omitted an *'I don't know'* option; 18 people added this in free text. We received other thoughtful entries such as *'Standards are artificial, we try to focus on actual security (incidentally we probably apply some of them)'* and some less thoughtful ones: *'Yes there is some I don't care.'*

5.5.4 Adoption of Development Methods

We asked participants about the development methods in their organisation, providing 15 common options (allowing selection of multiple options), and a free text option. We found that Agile (650), DevOps (445), Scrum (358) and Kanban (296) were the most common responses. Some of the 28 participants who used the free text option were positive, with comments such as *'Good practices but no dogmatic application of any of these.'* The majority of free-text respondents delighted in letting off steam, for example:

'...They claim it's scrum / agile. It is not, it's waterfall with extra meetings,' 'Fake cargo cult agile,' and *'RDD (Resume-Driven Development).'*

5.5.5 Secure Coding Policy

Asked whether their organisation or team has a written secure coding policy, less than a fifth of our participants (184) answered *'Yes.'* Other preset options were *'No'* (537), *'I'm not sure'* (238) and *'Not applicable'* (65). Testament to the positive impact of security surveys, one free text respondent wrote: *'I'm going to write one today.'*

5.5.6 Security Priorities

We asked people what aspects of software security are important. Data protection (978), Preventing vulnerabilities (954), Customer privacy (877) and Customer confidentiality (829) were widely chosen. Most other aspects were selected by at least 30% of respondents. Only two people chose '*I do not think that software security is important.*' There were 49 free text entries comprising a wide range of priorities and experience, usefully summed up by the single entry '*It Depends*™.'

5.5.7 Training

We asked participants whether they had ever been offered security or privacy training, and the majority answered *'No'* (see Table 5.2). We then asked for training details, and received over 300 individual free text entries. They ranged from negative comments such as *'Boring'* to more positive feedback: *'Nothing formal. It happens ad hoc (usually directed by me) as a part of code review.'* A reminder of the reluctance of individuals who take security seriously to part with confidential information, several answers were inscrutable, e.g. *'Confidential,' 'Information refused,'* and *'Internal training.'*

Table 5.2: Respondent has been Offered Security or Privacy Training

Response	Frequency	Percent
Yes	416	40
No	525	51
Not applicable	98	9

5.6 Results

One of our survey participants, asked about the importance of security activities, responded that *'External checks are a "finger not the moon" problem, but a useful diagnostic.'* This is a reference to a Buddhist saying on dogma, *'A finger pointing at the moon is not the moon,'* suggesting that it is easy to confuse descriptions of a thing for the thing itself. Given the difficulty in ascertaining whether software is secure, and the comparative simplicity of evaluating lists of software security activities, it is easy to confuse the activities with the goal. Evaluating security culture alongside security activities may get us a little closer to secure coding reality.

5.6.1 Research Question 1: What are the secure-coding characteristics of our sample group?

As discussed in Section 5.3, there is no agreed measurement of software security practice in academia. Different groups of academics roll their own, based on BSIMM, SAMM, MS-SDL and other practices. Measurement tools are thus not empirically validated, are not easily comparable between studies, and tend not to be reused across studies.

Our empirical approach was motivated by Weir et al. [323], who analysed the BSIMM assessments dataset to study how security activities were introduced in organisational settings over a period of 12 years. They found that there were 12 activities that were adopted '*most often, together and first*' by a majority of organisations, with at least half of them found in 92% of the assessments. There was '*a marked jump*' of 18 percentage points between the frequency of use of the other developer activities studied, of which the highest frequency was 47% (i.e. less than half), and these 12 activities, which had frequencies from 65% (CAQ5) to 89% (CAQ10). We judged that determining the presence or absence of these activities in an environment would give a contemporary objective assessment of the environment's software security practice. (Note that although named and categorised by the BSIMM investigators, these activities are common in industry and their adoption does not depend on familiarity with the BSIMM).

We asked respondents about the presence of these 12 known most Common Activities (CAs) in their environment. See Table 5.3 for the exact text of each of the 12 questions. We evaluated the number of the CAs that they identified as in use in their working environments (see Fig. 5.2). We assigned a score of 0-12 to work environments by

counting the number of CAs they were undertaking from the 12 we asked about in the survey. We call this the *'CA Score.'* This score is empirically justifiable, practical, and repeatable, and thus fills a gap in secure coding measurement.

Percentages of respondents choosing '*True*' to the CAs are in Table 5.3. A full breakdown of answers can be found in the online supplemental material. While a small number of participants stated that all 12 CAs were used, most used fewer, and quite a few participants indicated that none or very few of the activities were undertaken in their work context.

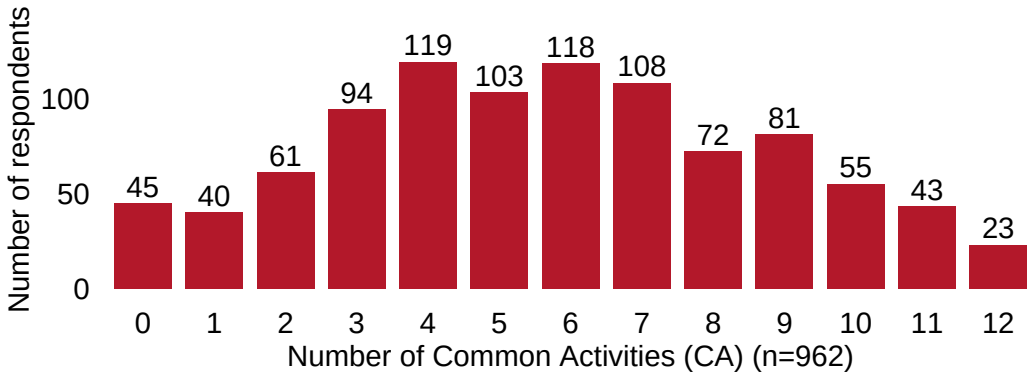


Figure 5.2: Number of security Common Activities (CAs) undertaken in respondents' coding environments.

Table 5.3: Questions on Twelve Common Activities

No.	True	Question
CAQ1	58.5%	Static analysis tools are brought into the code review process to make the review more efficient and consistent.
CAQ2	41.1%	Compliance constraints are translated into software requirements for individual projects and are communicated to the engineering teams.
CAQ3	36.2%	QA efforts go beyond functional testing to perform basic adversarial tests and probe simple edge cases and boundary conditions, with no particular attacker skills required.

Continued on next page

Table 5.3 *Continued from previous page*

No.	True	Question
CAQ4	31.3%	QA targets declarative security mechanisms with tests derived from requirements and security features. A test could try to access administrative functionality as an unprivileged user, for example, or verify that a user account becomes locked after some number of failed authentication attempts.
CAQ5	28.7%	Penetration test tools are used internally.
CAQ6	62.1%	Emergency codebase response can be done. The organisation or team can make quick code and configuration changes when software (e.g., application, API, microservice, CAQ infrastructure) is under attack.
CAQ7	67.8%	Defects found in operations are entered into established defect management systems and tracked through the fix process.
CAQ8	77.5%	Bugs found in operations monitoring are fed back to development, and may change developer behaviour. For example, viewing production logs may reveal a need for increased logging.
CAQ9	36.5%	Penetration testing results are fed back to engineering through established defect management or mitigation channels, with development and operations responding via a defect management and release process.
CAQ10	65.8%	Host and network security basics are in place across any data centers and networks and remain in place during new releases.
CAQ11	34.9%	Security-aware reviewers identify the security features in an application and its deployment configuration (authentication, access control, use of cryptography, etc.), and then inspect the design and runtime parameters for problems that would cause these features to fail at their purpose or otherwise prove insufficient.
CAQ12	31.7%	External penetration testers are used to identify security problems.

5.6.2 Research Question 2: What are the security culture characteristics of our sample group?

We asked a series of questions about security culture which can be found in Table 5.4. The questions were based on themes related to security culture mentioned in multiple papers

in the existing literature but which appeared to be largely unexplored. The rationale for each question is given in the discussion of the questions below.

5.6.2.1 Support for Secure Coding (SCQ1)

Support from the organisation has been found to be a significant aspect of security culture in multiple studies [13, 14, 284, 111]. We asked to what extent participants felt supported to code securely (SCQ1). Fig. 5.3 presents the answers to this question. As can be observed, answers are fairly evenly divided between those who feel supported, those who do not feel supported, and neutral responses. Answers to this question had a weak to moderate correlation of .46 with CA scores ($p<.001$).

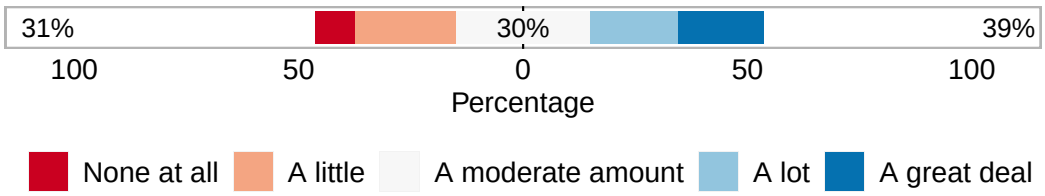


Figure 5.3: SCQ1: How much support do you feel that you get from your employer or open source team to code securely? (n=877).

5.6.2.2 Raising Security Concerns (SCQ2)

We asked participants whether, if they had a security concern at work, they would raise that concern. This question is part of our investigation into how security-motivated developers can influence their working environment. Fig. 5.4 shows that the vast majority of participants would be somewhat or very likely to raise the concern. Those who would not may work in organisations or teams where raising security concerns is actively discouraged, a phenomenon that has previously been observed [13]. This question can be a useful way to detect coding environments that are extremely unfavourable to security. Answers to this question had a correlation of .25 with CA scores, with $p<.001$.

5.6.2.3 Whether there are Security Tools in the Working Environment (SCQ3)

Security tool use is a fundamental aspect of secure coding, mentioned in most of the relevant literature on the subject [335, 216, 276]. While it may also be used to assess

Table 5.4: Security Culture Questions

ID	Question
SCQ1	How much support do you feel that you get from your employer or open source team to code securely?
SCQ2	If you had security concerns about your current project, how likely would you be to raise them with someone?
SCQ3	Security tools are tools that help to ensure source code is free from security vulnerabilities. For example, they may analyse code for known issues or check configurations for problems. Some examples are Coverity, CodeSonar and SonarQube. Some security tools like SonarLint integrate with the IDE. Are there security tools available in your environment?
SCQ4	If you saw a **free** tool that you felt would help you to code more securely, how likely would you be to install and use it **without** asking anyone for permission?
SCQ5	If you needed permission to use the tool, how likely would you be to ask for permission?
SCQ6	If you asked, how likely do you think it is that you would get permission?
SCQ7	If you asked, how likely do you think it is that you would get funding for a **paid** tool?
SCQ8	Have you ever heard people say that there is a software security culture in your working environment?
SCQ9	Would you agree that there is a security culture in your working environment?
SCQ10	How highly do you think your team prioritises software security?
SCQ11	How often is software security mentioned in team communications?
SCQ12	Roughly how much time do you spend on software security in an average week? This would include any tasks aimed at making or keeping a product secure, such as fixing vulnerabilities that could cause a breach, keeping third-party components updated and assessing code for security issues.

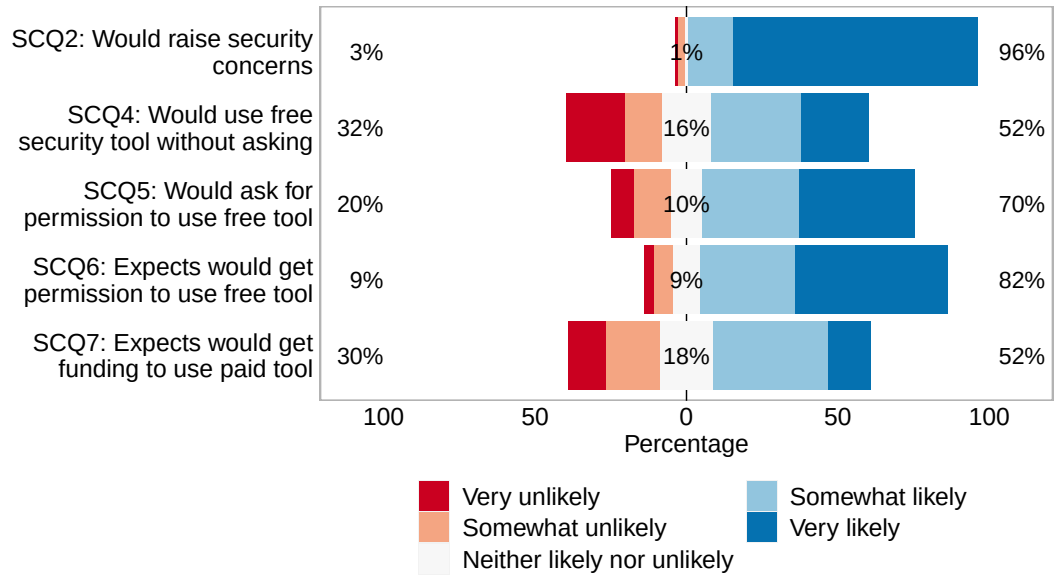


Figure 5.4: Results from questions SCQ2 (n=959), SCQ4 (n=959), SCQ5 (n=858), SCQ6 (n=860) and SCQ7 (n=891). Questions concern the reactions respondents anticipate if they were proactive about security.

security practice, tool use is an effective proxy for the cultural value of expending effort, time and money on security. Therefore, we asked whether security tools are present in the developer's environment. See Fig. 5.5, which shows that 33% of respondents had no security tools present in their working environment. Answers to this question had a low correlation of $r = .36$ with CA scores ($p<.001$).

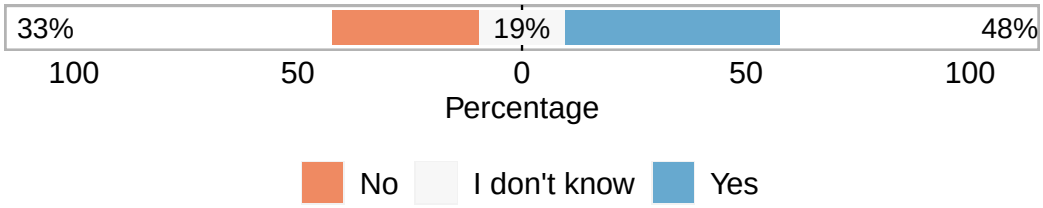


Figure 5.5: SCQ3: Are there security tools available in your environment? (n=932).

5.6.2.4 Introducing Free Tools (SCQ4)

Previous research on reasons why developers adopt security tools focused on their motivations and inspiration to do so. In this survey we wanted to explore the level of support for finding and integrating such tools that developers are likely to find within their environment. Xiao et al. [334] noted that several participants in their widely-cited interview study on how developers adopt security tools were self-motivated, seeking out tools because they needed them, and finding them online. Those developers would then need to introduce the tools to their work environment. This process of introduction was not a focus of the Xiao et al. or related Witschey et al. [329] paper, and remained unexplored. In a follow-up survey by Witschey et al. [330], the authors expressed surprise that although statements expressing *'security concern'* were predictors for security tool use, they had weak effects. We suspected that this could relate to a difficulty for developers in introducing security tools. In security-conscious environments, there could be a prolonged approval process or outright prohibition on new practices and tools. In security-hostile environments, they could be seen as a waste of time. The interaction of security-motivated developers with their environment influences their security activities [61, 228], yet the degree of support or resistance they are likely to encounter has not been investigated. Therefore we asked several questions in our survey about attempting to introduce new security tools. Questions SCQ4-SCQ7 explore how developers would go about introducing such tools, and what reaction they anticipate they would get within their environment. Developers' subjective expectations are important, since a perception that tools would be rejected could result in their not seeking out or proposing them [11]. An organisational climate that is positive for software security has been shown to be positively related to proactive developer behaviour [10].

SCQ4 asked whether the respondent would use a free security tool without asking for permission. See Fig. 5.4 for the results; 52% of those surveyed would use such a tool, while 32% would not. Note that while it might be expected that security-aware organisations would encourage experimentation with security tools, use of a free tool could itself introduce security vulnerabilities. Ironically, many security tools run with elevated privileges and can be sources of risk. Answers to this question did not correlate at all with CA scores ($r = -0.08$, $p = .009$).

5.6.2.5 Asking for Tool Permissions (SCQ5-SCQ7)

In SCQ5, we asked participants how likely it was that they would ask for permission to use a free tool (see Fig. 5.4). Those selecting *'Not Applicable'* were discounted. 70% of respondents would ask, 10% were neutral and 20% were unlikely or very unlikely to ask. Thus, 20% of respondents apparently have a particularly low interest in security, particularly low expectations from their management team, or both. (Answers to this question did not correlate with CA scores (r = .15, p<.001)).

However, SCQ6 indicated that a large 82% of respondents thought that it was somewhat or very likely that, if they did ask, they would get permission for a **free** tool. The figure for funding for a **paid** tool (SCQ7) was much lower at 52%. It can be argued that paid security tools are often better supported than free tools, so that they should be preferred. The only negative difference between a free and paid tool is financial outlay. The contrast between the answers to these two questions may indicate a poor security culture in the relevant participants' organisations. Haney et al. [111] included spending money on security in their list of attributes needed for a positive security culture in a development environment. The signals and cues received by employees influence their understanding of the real priorities in their work environments [11, 14]. In this case, participants have concluded that security is not a management priority.

Answers to SCQ6 did not correlate with CA scores (r = .11, p = .001), but SCQ7 answers showed a weak correlation of .27 (p<.001), indicating that paid tools are slightly more likely to be considered if other security precautions are in place.

5.6.2.6 Perceptions of Security Culture (SCQ8-SCQ9)

Although the importance of security culture is emphasised in numerous papers on organisations and teams with secure coding processes [12, 266, 279, 284, 285], there can be a disconnect between management and developers when it comes to security [12]. Some researchers have found that developers can be cynical about management drives for a security culture [279], perhaps perceiving them as lip service. Therefore we first asked whether the developer is aware of a claim to security culture in the environment, and then asked whether the developer would agree with this claim. Asked whether they had heard it said that there was a security culture in their working environment (SCQ8), only 261 respondents said *'Yes'*; 633 selected *'No,'* with 133 selecting *'Not*

applicable.' We allowed free text contributions; one was: *'no real people say stuff like that,'* a reminder that the term *'security culture'* can seem like just more management speak or propaganda [279] if its use is not accompanied by concrete steps towards prioritising security. Another contribution was *'We're a small firm, shipping working software at all is the priority.'* Answers to this question correlated weakly with CA scores, (r = .40, p<.001).

The following question (SCQ9) was whether participants would agree that there was a security culture in their work environment. As can be seen in Fig. 5.6, 48% of participants agreed or strongly agreed with this statement. 28% disagreed or strongly disagreed, while 24% were neutral. This question correlated with the CA score, with a weak to moderate correlation (r = .41, p<.001).

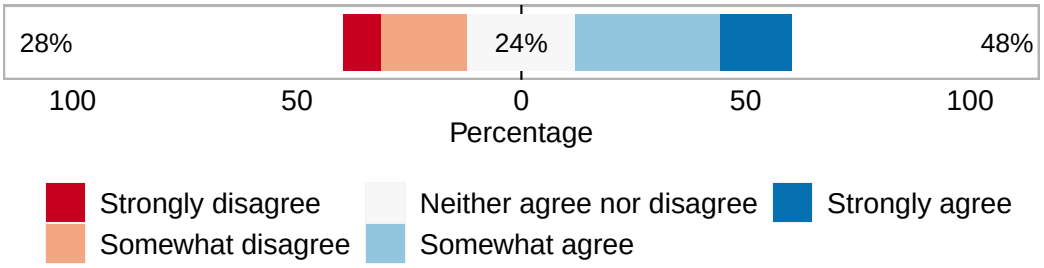


Figure 5.6: SCQ9: Would you agree that there is a security culture in your working environment? (n=946).

Of particular interest is whether there is a strong correlation between how security culture is portrayed in a work environment (SCQ8) and how respondents themselves view the environment's security culture (SCQ9). In fact, there is only a moderate correlation of .52 (p<.001) between these values. Developers may not be buying in to security culture claims.

5.6.2.7 How Highly the Team Prioritises Security (SCQ10)

Security is often treated as an NFR in the software development process, and is not explicitly prioritised by management or teams [267, 335]. Continuing to probe the security culture in the developers' working environment, we asked them how highly they thought their team prioritised software security. See Fig. 5.7; while 39% felt that their team prioritised it a lot or a great deal, 25% felt that they prioritised it only a little, or not

at all. Correlation between answers to this question and the CA score is low to moderate (r = .48, p<.001).

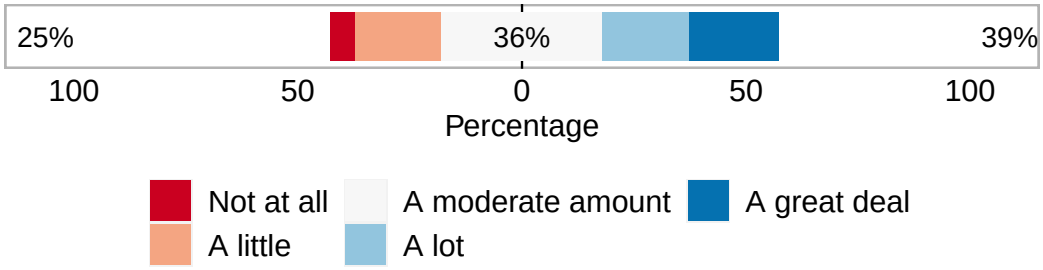


Figure 5.7: SCQ10: How highly do you think your team prioritises software security? (n=896).

5.6.2.8 How Often Security is Mentioned in Team Communications (SCQ11)

A developer may perceive their team's prioritisation of software security inaccurately, or their reporting may be subject to social desirability bias. Haney et al. [111] discussed how a security culture involves a *'commitment to address security'* and the *'perpetuation of a security mindset.'* Communication of priorities is essential within working environments [284, 219]. Furthermore, if indications of concern for secure coding are missing, that in itself allows developers to infer that secure coding is not considered important on the team [11]. In order to explore security prioritisation further, we asked participants approximately how often software security is mentioned in team communications. Fig. 5.8 presents the results, which differ markedly from some of the previous answers. 21% of respondents said that security was mentioned on the team about once a week or more often, and 21% said it was mentioned a few times a month. However, a large 58% said it was mentioned about once a month or less, with 9% stating that it is never mentioned. Correlation between answers to this question and the CA score is low to moderate (r = .44, p<.001).

5.6.2.9 Time per Week Spent on Secure Coding (SCQ12)

Time is a scarce resource, and time spent should give further insight into the real emphasis on security in the work environment [195, 11]. We asked participants how much time they spent on security during the week, supplying options from *'None,' 'Less than*

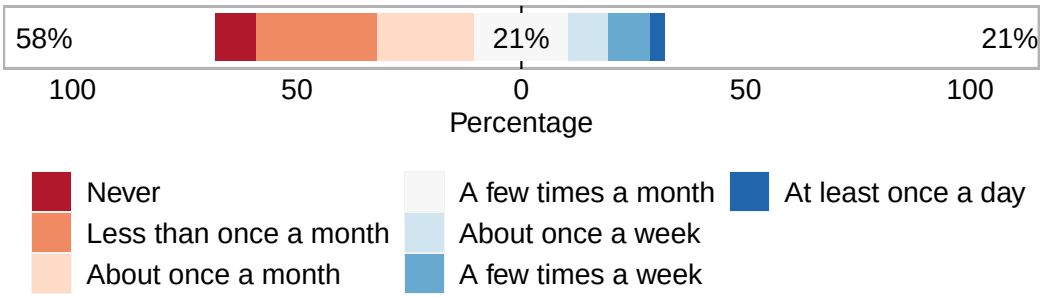


Figure 5.8: SCQ11: How often is software security mentioned in team communications? (n=862).

half an hour,' etc., to *'A week,'* and also allowing free text replies. This question can be criticised on the basis that security should come with quality. For example, one respondent replied: *'impossible to quantify this way; all work that leads to higher quality software also usually leads to more secure software.'* However, developing secure software entails many security-specific activities; for example, threat modelling is considered an essential component of secure coding [6, 320, 326]. Security conscious respondents could be expected to have an approximate mental model of how much time they spend on concerns that are primarily security motivated. Therefore, answers to this question are of interest, giving us additional insight into the security culture in the developer's working environment.

Even bearing in mind the qualification above, the answers to this question are not encouraging; Fig. 5.9 is dominated by red and orange blocks. Less than two hours a week is spent on security by 55% of respondents, with nearly 20% spending none. One free text response was: *'Fluctuates on whether my project is blocked by security review. Usually not at all, sometimes a few hours.'* Another participant added *'less than half an hour a month.'* Correlation between answers to this question and the CA score is low (r = .38, p<.001).

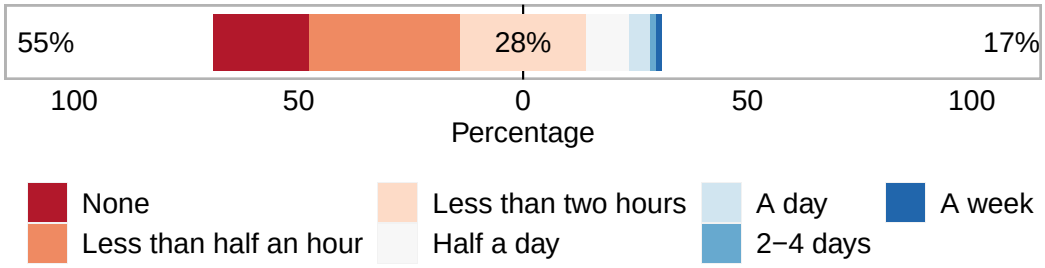


Figure 5.9: SCQ12: Roughly how much time do you spend on software security in an average week? (n=878).

5.6.3 Research Question 3: Do secure coding practice and culture correlate, and if not, what lessons can we learn to help support development of secure coding?

As we examined the answers to our security culture questions we noted the correlation score of each question's answers to the overall CA score for software security practice. In many cases, there was no correlation. Where it did exist it was weak to moderate. By contrast, if good security practice entailed good security culture we would expect to find high correlations between culture answers and practice findings. In an attempt to attain a greater understanding of our results, we broke down some of the answers by CA score. Fig. 5.10 shows the degree to which the participant thinks their team prioritises security, broken down by CA score. There is clearly a correlation, and for high CA scores we are led to believe that security is very highly prioritised by the team.

However, when we look at how frequently the team communicates about security, a prioritisation of security is not at all clear (see Fig. 5.11). Even in the teams with the highest security practices, security is mentioned within the team relatively infrequently. At the very highest level of twelve CAs, security is mentioned less than once a week in 36% of teams. Low levels of communication about a topic is an indicator that the topic is low-priority for a team [11]. We suggest that this question gives some insight into environments where, although security is apparently high-priority, the security culture is unfavourable. The minimum work necessary for security compliance is done.

Similarly, when we look at the amount of time spent on security per week it is surprisingly low at all CA levels. At the zero-CA level, 95% of respondents spend less than half an hour a week on security-related activities, with two thirds of these choosing

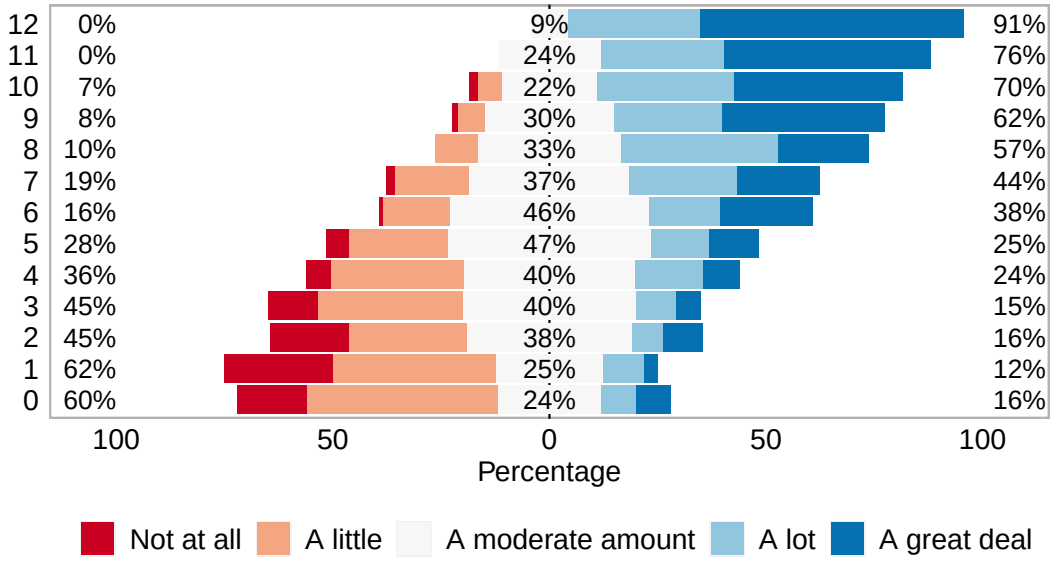


Figure 5.10: SCQ10: How highly do you think your team prioritises software security? (n=896) This graph shows the value broken down by the number of common security activities undertaken in the environment. The correlation is visible.

'None.' Even at the highest level, 59% of participants spend an average of less than two hours per week on security activities. Fig. 5.12 suggests that this question could be useful when attempting to identify a compliance-focused mentality.

5.7 Discussion and Conclusion

Asked to comment on the survey, one participant said of the 12 CAs: *'A lot of the questions about bugs or security issues being tracked are kind of missing the point; the system exists, but often times fails. Bugs and bug fixes are in the correct system, but there is not enough time allocated for reporters to verify or validate fixes or for regression.'*

This is an excellent illustration of the importance of assessing security culture alongside security activities. If the organisation's employees are going through the motions, producing paperwork and audit trails for compliance purposes, but are not allowed time to do the job properly, a good security culture is missing. In this environment it is inevitable that security issues will slip through the cracks, adversely affecting the

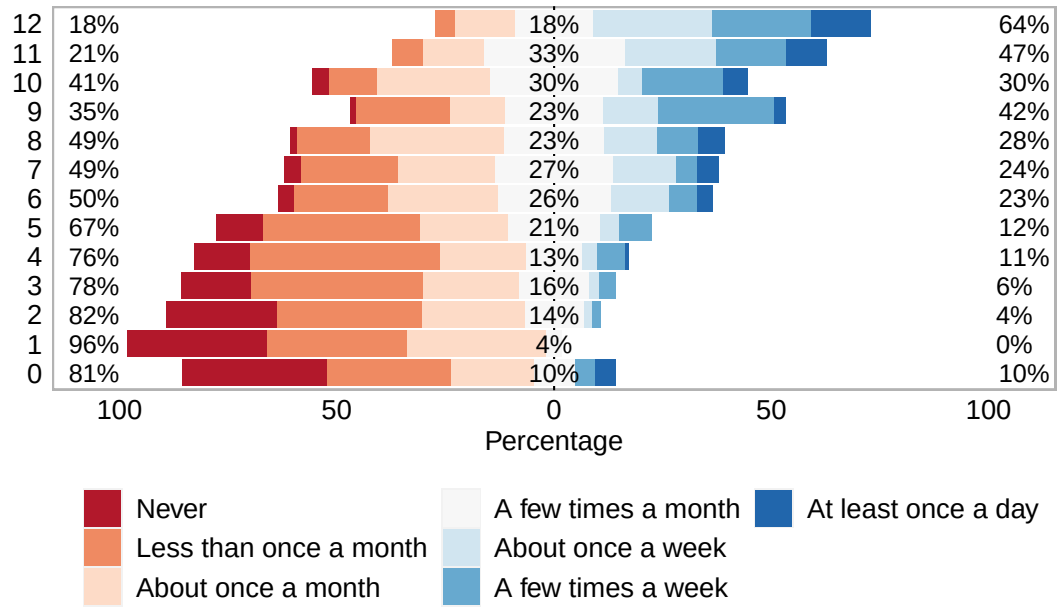


Figure 5.11: SCQ11: How often is software security mentioned in team communications? (n=862). This graph shows the value broken down by the number of common security activities undertaken in the environment. Even at the highest CA scores, team security communication is relatively infrequent.

organisation or open source team, impacting their clients, and sometimes posing a systemic threat to their very existence.

Detailed understanding of organisational culture requires longitudinal ethnographic research. In many contexts (such as surveys) this is not achievable, but relative values are of benefit. Our questions are designed to probe core cultural indicators in an organisation. Where values are low, security culture is lacking.

5.7.1 Threats to Validity

The questions we used to objectively assess environmental secure coding practice are based on the secure coding activities identified by analysis of BSIMM results as those adopted first and most frequently by security-aware organisations. These activities might not be suited or appropriate to all coding environments. However, the BSIMM data is the largest trove of such data available at this time. It is the closest approach we have identified to finding an empirically-established objective measure of secure

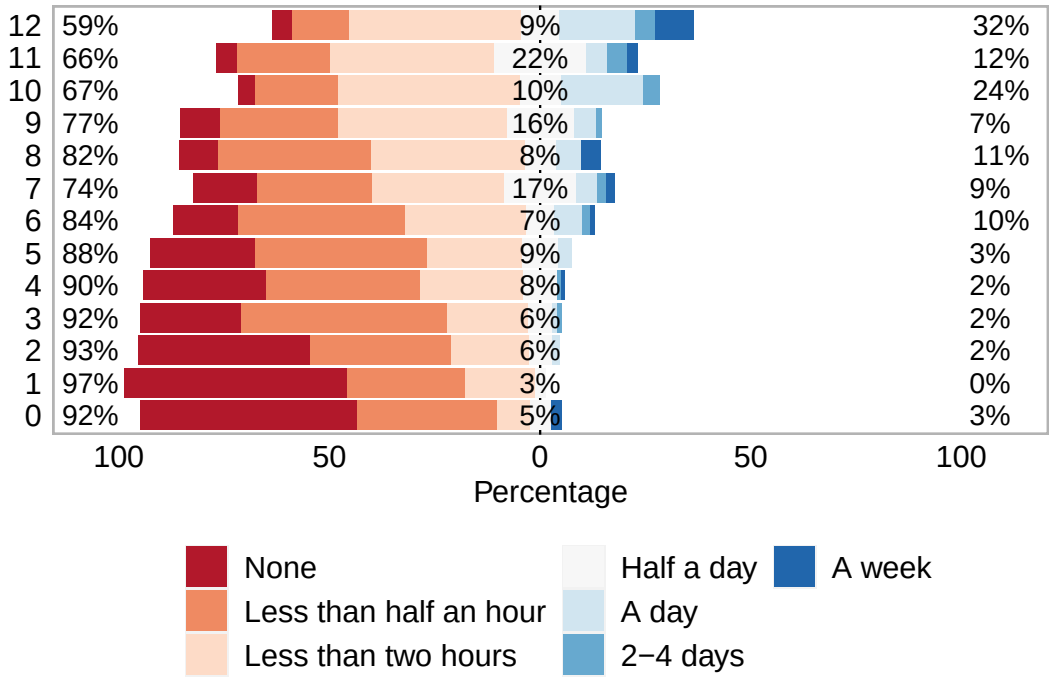


Figure 5.12: SCQ12: Roughly how much time do you spend on software security in an average week? (n=878). Even at the highest level of common security activities, many participants spend less than two hours a week on security.

coding in environments developing complex applications. The CA Score is a simple measure which does not explore how activities are performed or whether they are implemented correctly. In the necessary research trade-off between cost and detail, it is a lightweight measurement suitable for surveys and quantitative comparisons. For a more thorough measurement of secure software development, more detailed analyses [285] are appropriate.

The questions used to provide insight into participants' understanding of the security culture of their working environment were centred around topics that are prevalent in academic software security literature. Some include ideas that can also be found in organisational climate theory. It is possible that other wordings or additional questions exist which would further enhance our understanding of security culture. Finding and evaluating such questions can be the subject of further research.

As with all questionnaire-based studies, another risk to validity is social desirability bias: the danger that developers will tell us what they think we want to hear. Respondents

were self-selecting, and the data cannot be verified. We attempted to mitigate this risk by emphasising in recruitment materials and in the survey consent form that participants did not need to be interested in security and did not need to know anything about it.

Our interpretations of the answers to our questions could be mistaken. For example, the time spent on security by our respondents could in fact be appropriate in an environment with a good security culture. Future research should explore this. In a discussion of assessing software security in the field, it is necessary to bear in mind that the direction of causation cannot be determined. However, correlations and relative emphasis can be observed and can be instructive, which is the approach we have taken here.

5.7.2 Conclusion

In this study we add to the existing body of knowledge on how to identify environments where secure coding is prioritised and interest in code security is valued. We wish to ensure that the correct thing is being measured. In our research, we want to focus on the moon, not on the finger pointing at the moon.

We adopted a list of 12 secure coding activities which have been empirically established as being those adopted first and most often in software security drives. We found that while many of our respondents identified some of these activities in their work environments, in some workplaces none of them were present.

It is well known that security activities can be undertaken for compliance reasons. Researchers into secure coding have established that to achieve secure coding success, a security culture should be established. Based on a thorough literature review, we asked a number of questions designed to evaluate security culture. For example, we asked how much time the respondent spends on secure coding, and how often, on average, secure coding is mentioned in team communications in a week. In this paper we discuss the answers to 12 such questions. We find indications of poor security culture at all levels of security practice. Even where participants state that all 12 common security practices are undertaken in their working environments, communication about security and time spent on security can be very low.

Our results may have an impact in several areas.

Academia: Research on software security practice does not always attempt to measure organisational security [202], and even when this is measured, culture is not quantified.

When studying developer behaviour, for example in ethnographic studies, it is important to also consider the security processes and culture in the environment. If this is not done, the research may miss fundamental influences on the developer or team under study. The CA Score provides a simple but empirically validated measure for practice. Our culture questions, especially questions SCQ11 and SCQ12, introduce a way to do a quick litmus test for security culture.

Organisations: For organisations that are serious about encouraging secure coding, it is not enough to introduce practices and follow processes. In addition, time, budget and space for security must be provided.

Industry: Changes to regulation, law and industry standards must look beyond the checkbox approach [222] and consider the holistic background.

The results of this survey show that when evaluating security posture it is not enough merely to measure activities. Security culture must also be evaluated. Understanding the time, money and support available for secure coding activities is crucial to assessing how thoroughly they will be implemented.

6 The State of Secure Coding Practice: Small Organisations and 'Lone, Rogue Coders'

This chapter was previously published as: Ita Ryan, Klaas-Jan Stol, Utz Roedig. 'The State of Secure Coding Practice: Small Organisations and "Lone, Rogue Coders"' *2023 IEEE/ACM 4th International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*. DOI: 10.1109/EnCyCriS59249.2023.00010.

6.1 Abstract

Software security is a rapidly developing problem. Malware, ransomware and spyware routinely leverage vulnerabilities in software to gain access to systems, escalate privileges and run adversarial code. One approach to solving this issue is to use secure software methods, which attempt to guide organisations in improving their software assurance. However, these methods implicitly assume the presence of substantial resources deployed in a compliance-mandated environment. The distinct and often limited environment in which small organisations, independent teams and lone coders operate is not considered. Advice for software security in small teams is almost absent from the literature, as is a way to measure the levels of secure coding in such teams. In order to address this problem, we must begin by understanding it. As part of the analysis of a large survey on current software security practice, we examined the current software security practices of small and open source organisations, and of lone and non-company developers. We present our results in this paper. We hope that they will facilitate the targeting of security advice to these neglected developer categories.

6.2 Introduction

The world's businesses, economies, governments and critical systems are moving online at ever-increasing pace. This global digital transformation comes at a cost, with nations, businesses and individuals more vulnerable than ever before to cybercrime threats such as ransomware, and to cyber-espionage [238].

An understanding of the issues that facilitate online interference is essential. At the root of many cyber attacks are code vulnerabilities. At the root of code vulnerabilities is software code that, due to poor planning or faulty execution, contains errors that allow malicious exploitation. The question arises, why do developers not code securely? The field known as DCS focuses on reasons why software developers fail to secure their code. Well-explored issues include the poor usability of security tools [131] and APIs [328], inadequate documentation [181] and resources [3], conflicting advice [184], and a lack of knowledge [32], awareness [224], understanding [268] or motivation [267].

There are also well-known constraints in the coding environment such as lack of time [280], security not being a priority [202], and over-speedy deployment [291]. Other issues outside the developer's sphere of influence include lack of clarity on who is responsible for security [285], budget pressure [13], and questionable management decisions, such as choosing to release insecure code for business reasons [267].

Secure software methods in industry attempt to address some of these issues, introducing practices and activities to be undertaken throughout the development lifecycle [171, 198]. Cybersecurity guidelines for critical systems include such practices, although the sections on secure software development can be rather short [238]. Such methods and guidelines tend to implicitly assume the presence of substantial resources deployed in a compliance-mandated environment. But what about software which is not developed in this type of environment? Much of modern software has dependencies on third-party components. These may be created by lone developers, or by small organisations or OSS teams having few resources, few developers and no budget. Small, under-resourced teams can have a disproportionate influence on global software security. An obvious recent example is Log4Shell (CVE-2021-44228), a vulnerability in popular open-source Java logging tool Log4j. Disclosed in late 2021, this popular defect was widely weaponised by threat actors. By late 2022 it was reportedly being used by North Korea to obtain initial entry to US energy company networks [164].

The distinct and often limited environment in which isolated developers, open source developers, freelancers and small organisations operate is not clearly understood. It has not been much studied in secure software development literature, and to date there has been little attempt made to separate out secure software development guidance for such teams. This contrasts with the overall area of cybersecurity, where in the UK, for example, the 'Cyber Essentials' program [289] attempts to provide small organisations with the information and guidelines they need to achieve a base level of security.

The existence, importance and unique software development needs of small coder groups have been acknowledged by the creation of the ISO/IEC 29110 Series of '*Systems and Software Engineering Standards and Guides for Very Small Entities*' [127]. However, these guides do not currently have a secure development component, though the addition of security improvements has been proposed [167]. Advice for software security in small teams is largely absent from the literature. An academic study in 2022 was unable to find any formal secure development guidelines for OSS teams or small organisations [25].

In addition, there is currently no established way to measure the levels of secure coding in such environments. While measurement of secure coding by individuals is well understood [185], researchers have not yet converged on a metric to measure secure coding for organisations and teams. In previous work [239], we proposed a potential lightweight metric derived by evaluating organisations' use of the twelve security activities most commonly used in industry, as determined in a study by Weir et al. [323]. We called these the '*common activities*' (CAs). Questions to evaluate the CAs can be seen in Table 6.1. Weir et al.'s work was based on data accumulated by the BSIMM team working with large or very large organisations. Implementing the CAs ideally requires the deployment of large budgets and the existence of multiple teams within an organisation. Thus, it seems unlikely that the Ryan metric would be suitable for small organisations, lone developers or OSS teams.

This intuition has not yet been verified using data. As part of the analysis of a large survey on current software security practice, we examined use of the CAs by small and medium organisations, lone developers, and developers outside organisations. In this paper we present a summary of use of the CAs and other security measures by these developer categories.

Our contribution is that we provide information about the current state of software security practice in small and medium-sized organisations, amongst coders outside

Table 6.1: Questions on Twelve Common Activities (CAs)

No.	Usable	Question
CAQ1	Yes	Static analysis tools are brought into the code review process to make the review more efficient and consistent.
CAQ2	No	Compliance constraints are translated into software requirements for individual projects and are communicated to the engineering teams.
CAQ3	Yes	QA efforts go beyond functional testing to perform basic adversarial tests and probe simple edge cases and boundary conditions, with no particular attacker skills required.
CAQ4	Yes	QA targets declarative security mechanisms with tests derived from requirements and security features. A test could try to access administrative functionality as an unprivileged user, for example, or verify that a user account becomes locked after some number of failed authentication attempts.
CAQ5	Yes	Penetration test tools are used internally.
CAQ6	Yes	Emergency codebase response can be done. The organisation or team can make quick code and configuration changes when software (e.g., application, API, microservice, infrastructure) is under attack.
CAQ7	Yes	Defects found in operations are entered into established defect management systems and tracked through the fix process.
CAQ8	Yes	Bugs found in operations monitoring are fed back to development, and may change developer behaviour. For example, viewing production logs may reveal a need for increased logging.
CAQ9	Yes	Penetration testing results are fed back to engineering through established defect management or mitigation channels, with development and operations responding via a defect management and release process.
CAQ10	Yes	Host and network security basics are in place across any data centers and networks and remain in place during new releases.
CAQ11	No	Security-aware reviewers identify the security features in an application and its deployment configuration (authentication, access control, use of cryptography, etc.), and then inspect the design and runtime parameters for problems that would cause these features to fail at their purpose or otherwise prove insufficient.
CAQ12	No	External penetration testers are used to identify security problems.

organisations such as open source developers and freelancers, and amongst developers who code alone within organisations.

This is a first attempt to assess the levels of secure coding practice for these cohorts. It will provide much-needed information to the software security community. The paper is organised as follows: Section 6.3 discusses related work, Section 6.4 is on method, Section 6.5 provides the results, Section 6.6 discusses the results and Section 6.7 concludes the work.

6.3 Related Work

6.3.1 State of Practice

An investigation of app developers' rationale by van der Linden et al. [305] looked at how app developers deal with choices that are not overtly security-related, such as choice of advertising library, finding that app developers need support to make the right choices. Wermke et al. [324] examined security and trust in OSS projects by interviewing 27 open source contributors on their practices. They found that there were few engagements with security challenges. Guidance and policies were ad-hoc except for large projects. Most of the contributors thought of improvements in terms of bug bounties, code-review hours, code upgrades and in general more person hours. The authors recommended that security help for OSS projects take project size and resources into account.

6.3.2 Secure Coding Guidelines

It seems that the help and support that these researchers advocate is currently lacking. Research has found that there is a shortage of secure coding guidance online, with significant gaps in coverage in several areas such as program analysis tools and logging [6]. Where guidelines do exist, developers are not always aware of them [80]. This may be somewhat mitigated in large organisations by the availability of secure coding standards such as BSIMM [171] or SAMM [198], but there is a marked shortage of secure coding guidelines designed explicitly for small organisations and teams.

In a recent literature review, Bender et al. looked at usability, security and privacy development processes for app developers, small and medium companies, and the open

source community [25]. They found 15 papers across the range of three development types, but all dealt with usability. They found nothing related to security or privacy.

Although not tailored to small teams and isolated developers, work on security interventions by Weir et al. is of interest because they explicitly considered situations where budgetary and resource support could be low [320]. Their interventions approach should be of benefit in introducing security practices to small organisations and teams. They set out to find the best way to encourage developers to use secure coding practices. From detailed consultation with industry experts, they identified eight interventions of interest. Of these, they considered three to be sufficiently lightweight to introduce in their relatively brief intervention sessions; '*incentivization session*,' '*threat assessment*,' and '*on-the-job training*.' During these sessions they hoped to promote the other five interventions at opportune moments. It is interesting that they did not include automated security analysis as one of their three primary interventions [320]. However, they did identify automated static analysis as an intervention to be encouraged when opportunity arose.

6.3.3 Secure Coding Assessment

Assessing individuals' level of secure coding is well understood, and a level of researcher convergence has emerged around the Naiakshina criteria [184, 185, 110]. There have been multiple attempts to define measurement tools for secure coding at a group or organisational level. Since these are frequently derived from industry guidelines such as BSIMM and SAMM, they often include practices that are not available to or practical for small teams [175]. Assal et al. developed a useful approach; in an interview study, they assessed whether participants used techniques to ensure security in each of six software development lifecycle steps from '*design*' to '*post-dev testing*' [13]. The qualitative analysis was work-intensive, requiring the researchers to understand the development process and security steps and to be able to correctly interpret developer responses. The approach would not easily translate to analysis of survey answers at scale, although it was a useful and flexible way to compare teams' approach to secure coding.

We did not find any studies dealing specifically with the assessment of secure coding for small organisations or teams.

6.4 Method

We undertook a large-scale survey of current security practice for software developers. We publicised the survey to our personal contacts and via social media and developer groups, receiving 1,100 responses over three months, of which 962 were valid responses. Ryan et al. [239] describes the composition of the survey questions, pilot tests, ethics approval and recruitment processes in detail. It also gives demographic data for the survey participants. The focus of Ryan et al. was to devise and use a metric for software security for organisations, to attempt to assess software security culture, and to compare the results of the two measures to discover whether a high level of software security practice necessarily entails a good security culture.

Ryan et al. describes in detail the rationale for using the empirically-determined 12 CAs as a lightweight metric to measure software security in coding environments. In this paper we revisit the survey data, this time focusing on developers who code in small and medium organisations, non-organisational developers, freelancers and developers who code alone. We evaluate the CAs in light of their relevance to small and impoverished teams. We posit that nine of the CAs may be accessible. We have indicated these with a 'Yes' in the 'Usable' column in Table 6.1. We examine survey responses and use the results to assess the truth of our conjecture on which of the common security activities may be suitable for isolated developers and resource-constrained developers.

This paper is a first exploration of the secure development work environment of five developer cohorts that have not previously been investigated. The questions asked were derived from data sourced in much larger organisations. Therefore we have presented the data simply. The reader can easily compare the rates of adherence to the 12 most common security activities between the five developer categories. The corresponding adherence for all valid respondents is displayed for information reasons.

In addition, we briefly examine use of security tools by the developer categories of interest. We theorise that one approach to accessing secure coding that is available to isolated developers is leveraging automation of secure coding aids developed elsewhere. Such automation can be applied, for example, to tools for static analysis, code review, configuration, dynamic scanning, monitoring, licensing and testing. There is overhead to setting up such tools, but they provide an analytic consistency not otherwise achievable on a small team.

6.4.1 Data

Where participants are quoted in the text, the quotes appear exactly as entered into the survey. Data analysis was done in R. Data was plotted using the viridis colourblind-friendly 'plasma' colour map [1].

6.5 Results

As discussed, the CAs are derived from data on large organisations. They are likely to be less useful for secure coding measurement in small teams or open source environments. The resources available in such environments are more constrained. Asked for feedback on the twelve CAs, one survey respondent remarked:

> '*(Some of these don't fit that well for the open source project that I work on)*'.

While it is easy to assume that lone coders will take fewer security measures, this assumption may not make sense to the coders themselves. For example, one of the respondents said:

> '*I suspect, even as a lone, rogue coder, that I generally write more secure code than a lot of companies*.'

In Table 6.1 we made an initial assessment of which CAs we thought it would be difficult for coders working alone or in small teams to undertake. However, this assessment was based on the theoretical accessibility of activities, rather than on data. In our analysis we look at the data for several categories of coder, briefly described in Table 6.2. They are as follows:

Lone Coders in Organisations: When asked whether they code alone or in a team, these respondents stated that they code alone. There are 141 valid respondents who stated that they code alone, but in this category we consider only the 64 of them who stated that they work in an organisation with a size greater than one. See Fig. 6.1. We wanted to see what secure coding practices these isolated coders, working inside organisations, are

[1] `https://ggplot2.tidyverse.org/reference/scale_viridis.html`

Table 6.2: Categories of Developers Considered in this Study

Name	N	Description
Lone Coders in Organisations	64	Codes alone rather than on a team. Works within an organisation with size greater than one.
Coders in small organisations	234	Codes on a team within a small organisation (less than 50 staff members).
Coders in medium-sized organisations	129	Codes on a team within a medium-sized organisation (between 50 and 199 staff members).
Coders not working for organisations	67	Codes in a non-organisational environment, either alone or on a team.
Freelancers	23	Codes in a single-developer organisation, states that they code alone.
All Developers	962	All valid survey participants. Includes developers in large organisations.

using. There are certainly some in this cohort who do not feel supported. One participant stated that he coded alone in an organisation with between 10 and 50 developers and a staff of more than 10,000. Asked for comments at the end of the survey, he had this to say:

> '*TLDR - we got pwn't, we learned nothing, back to old boy's club and the mentality security doesn't contribute to bottom line of revenue so we get Foxtrot-All for budgets/tools/time/attention/consideration. But of course it's our fault when inevitably kaboom happens.*'

Coders in Small Organisations These respondents, when asked whether they code alone or in a team, said that they coded within a team. Asked to estimate the size of the organisation they work in, they specified a value of up to 49 employees, which we consider a small organisation [78].

Coders in Medium-Sized Organisations These respondents stated that they worked on a team in an organisation with between 50 and 199 employees.

Coders not Working for Organisations Some developers chose 'Not applicable' for both organisation size and number of developers in the organisation. These developers

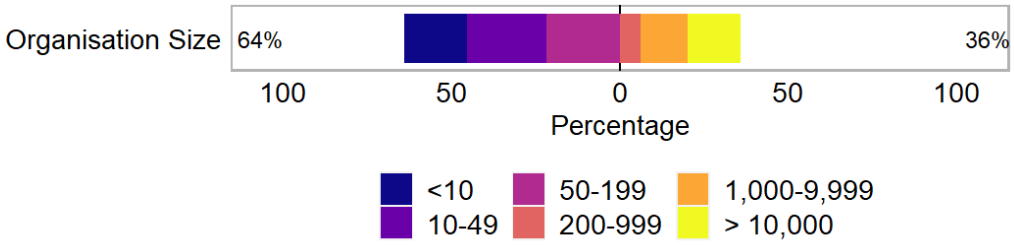


Figure 6.1: **Organisation size for the 64 respondents who stated that they code alone within organisations**. This cohort is of interest because they may lack the support available to those working on teams.

may be open source developers, students or independent developers. Although they do not have organisational support, and some code alone, many may have support from teams and communities.

Freelancers These developers chose an organisation size of 1 and stated that they worked alone. We deduce that they work as freelancers. They may be the most isolated of all the categories, with no organisational, institutional or open source support.

All Developers Finally, in order to give a point of comparison for the data, we included the CA percentages for the category comprising all 962 valid survey participants.

6.5.1 Usage of CAs by Developer Category

In this section we examine the usage of the twelve CAs by each of the five categories of coder. The CAs are collected into five groups, each containing two or three related activities. These are now discussed.

6.5.1.1 Penetration Testing

When considering the twelve CAs and their accessibility to under-resourced coders, we thought that internal penetration testing could be a practical option, since it does not necessarily require additional staff or resources and can theoretically be conducted by a coder on their own code. We were interested to see that actually, there was more external than internal penetration testing for the developer categories we examined. None of the small number of freelance coders in the study undertook any internal penetration testing at all. See Fig. 6.2.

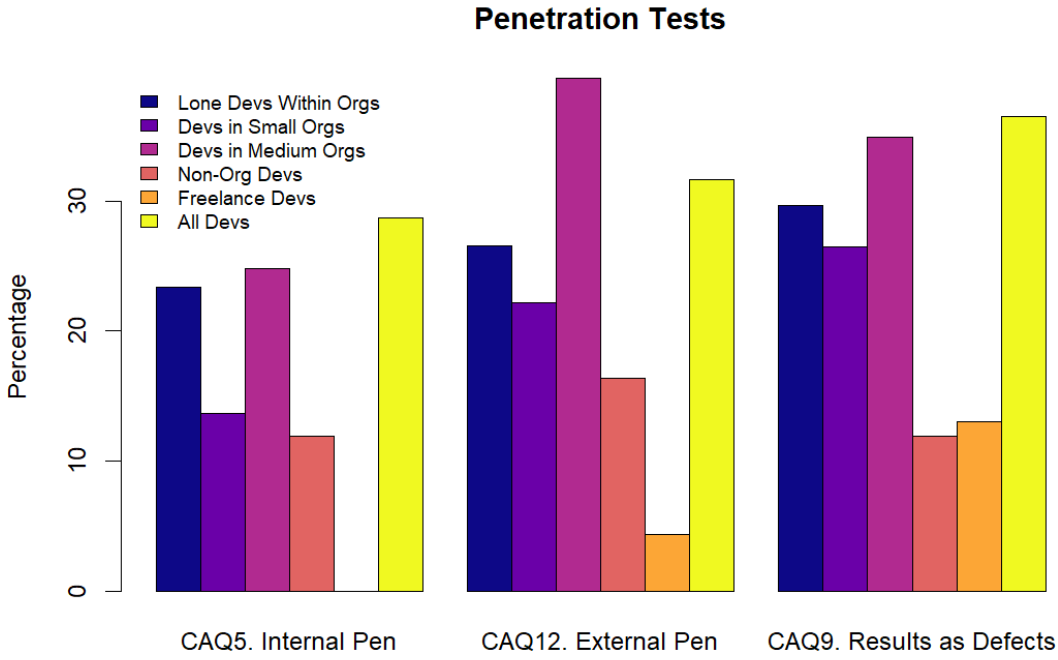


Figure 6.2: **Results for the three penetration testing CAs for all five developer categories and the whole sample.** None of the freelance coders who responded to the survey did any internal penetration testing.

6.5.1.2 Review

Two of the CAs deal with review. The first is incorporating static analysis into code review. This is the most commonly used activity from the BSIMM data [323]. The second is having security aware reviewers inspect the features of the software and determine if there are any possible issues in the deployment configuration or design that would cause security problems. While both would seem to be natural undertakings for a security-aware environment, the first assumes that code reviews are taking place. For isolated developers who code alone within organisations, code reviews will necessarily be problematic. The second activity, having security-aware reviewers look over the code and deployment, could be difficult to provide in any resource-constrained environment. The take-up of these activities can be seen in Fig. 6.3.

There are nuances to all of these activities, as illustrated by one of our respondents who commented:

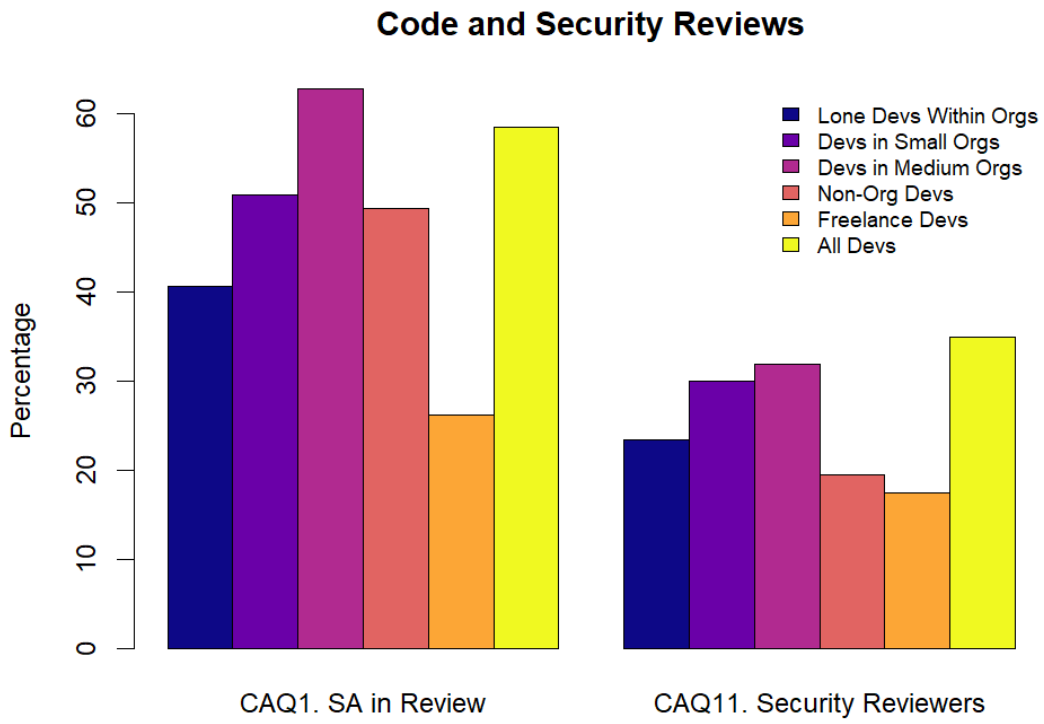


Figure 6.3: **Results for the two review CAs for all five developer categories and the whole sample.** Freelancers are least likely to use review activities, while developers in medium-sized organisations use them the most.

> '*Security-aware reviewers will review code and/or designs when there is an obvious security angle in new work, but there is no formal review process for the security content of changes that aren't obviously related to security.*'

This speaks to the shortage of resources in most environments, as all code changes should ideally be reviewed for security. Defects can be chained by attackers to provide access in ways that are not immediately obvious to the average developer.

Among the categories of developer we consider, freelancers have least access to both review activities. Developers in medium-sized organisations use both review activities the most. Access to security reviewers is higher in '*all developers*' than in any of the five categories.

6.5.1.3 Quality Assurance

The presence of a Quality Assurance (QA) team does not always ensure that security testing will take place. Two of the CAs specifically focus on this. Number three advocates that the QA team should conduct adversarial testing, checking edge and boundary cases. Number four suggests that the QA team attempt to attack the security procedures that are in place. For example, they could attempt to access systems when not logged in, or access admin systems when not logged in as an administrator. No hacking skills are required to conduct either set of tests. The incidence of these quality assurance activities within the five developer categories we consider, and the whole sample, can be found in Fig. 6.4.

Non-organisational developers are least likely to engage in these activities. It can be argued that they rarely have access to QA teams. Developers usually do some basic probing on their own account, however, and could conduct these quality assurance activities themselves.

6.5.1.4 Operations Feedback

Two of the CAs deal with feedback from the operations team. CAQ7 mandates that the operations team should add defects found to the standard defect-tracking system, while CAQ8 concerns giving operations feedback to developers. Since we are dealing with lone developers and those who work in small organisations we would expect that there would be a lot of informal as well as formal feedback. In some cases the coders and the operations team could be one and the same person. See Fig. 6.5 for the results for these questions. As expected, there is a high occurrence of both of these activities in the developer categories we consider. It is interesting that the incidence is low in the non-organisational cohort. This may indicate that there is no formal defect tracking process, or no operations environment, or it may be that these developers simply do not undertake these activities. Predictably, the incidence is highest in medium-sized organisations.

6.5.1.5 Security Processes

Finally, we look at three questions that are concerned with security process. CAQ2 suggests that compliance constraints should be conveyed to software engineering teams

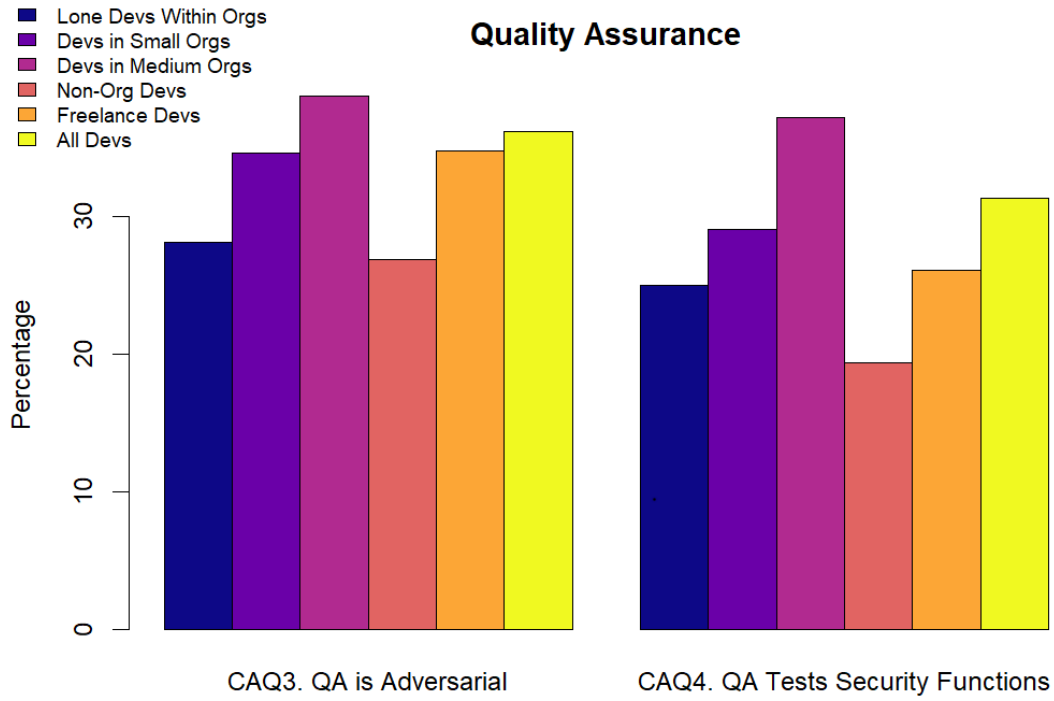


Figure 6.4: **Results for the two quality assurance CAs for all five developer categories and the whole sample.** Non-organisational developers are least likely to engage in these activities. Developers in medium-sized organisations are most likely to engage in them, beating the '*all developers*' sample in both cases.

as requirements. CAQ10 states that network security basics should be kept in place while new code is being deployed. A team must be able to respond to an attack with swift code and configuration changes to satisfy CAQ6.

We can observe from Fig. 6.6 that there is little conversion of compliance constraints to requirements outside the organisational environment. The non-organisational cohort lags behind the other cohorts on almost every measure, but it is particularly far back on this one. Freelance compliance is also low. This suggests that the type of work that is done outside organisations is not typically subject to compliance. With the likelihood of an increased compliance and legislative overhead for all developers in the short to medium-term future [83, 231], this may spell trouble ahead for open source and other non-organisational coders.

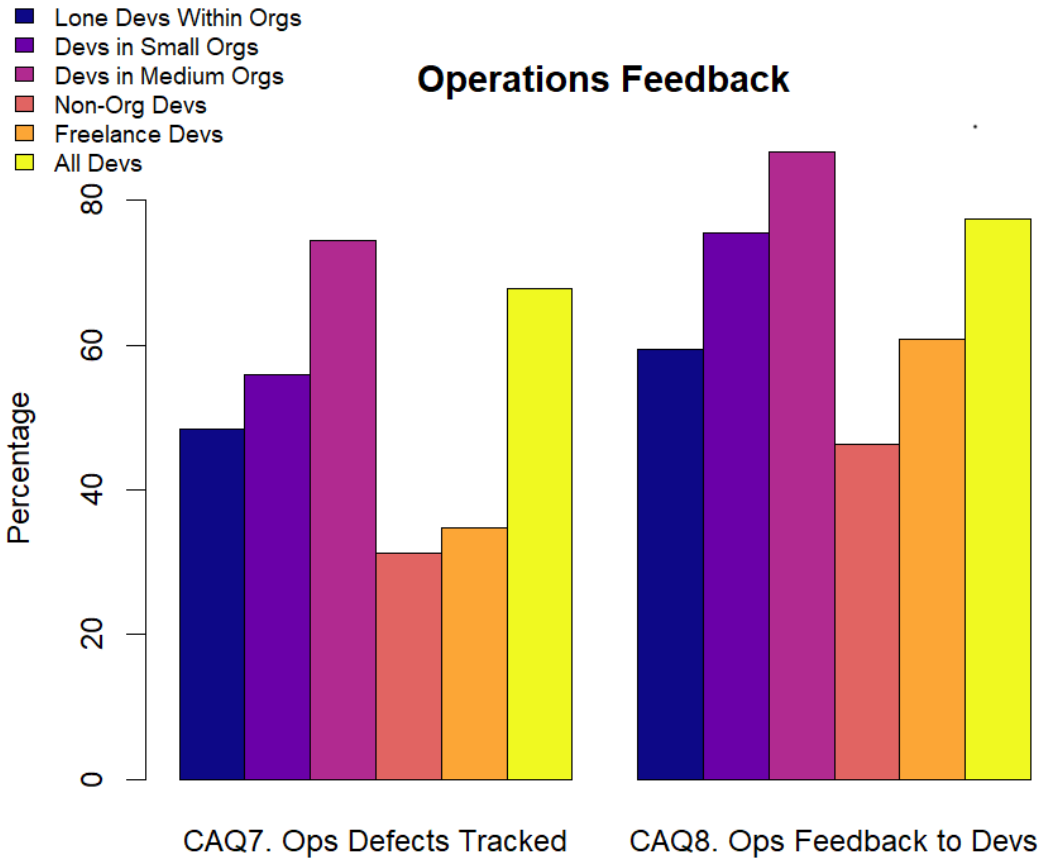


Figure 6.5: **Results for the two operations CAs for all five developer categories and the whole sample.** Non-organisational coders are least likely to engage in these activities. Developers in medium-sized organisations are most likely to engage in them, outdoing the '*all developers*' sample.

Lone developers working within organisations, who lag behind on other questions, shine when it comes to ensuring host and network security basics are in place. This may indicate a degree of self-reliance around their deployment environment which would be a natural consequence of working without a team of peers. Lone developers are likely to work on smaller projects where they are involved in deployment, support and all other aspects of the system.

The ability to be able to respond quickly to attack is the only area in which the non-organisational cohort does not lag behind the others. It is slightly ahead of the lone developers on this metric, which may indicate that lone developers inside organisations lack the autonomy to respond quickly to external attack, requiring authorisation to change

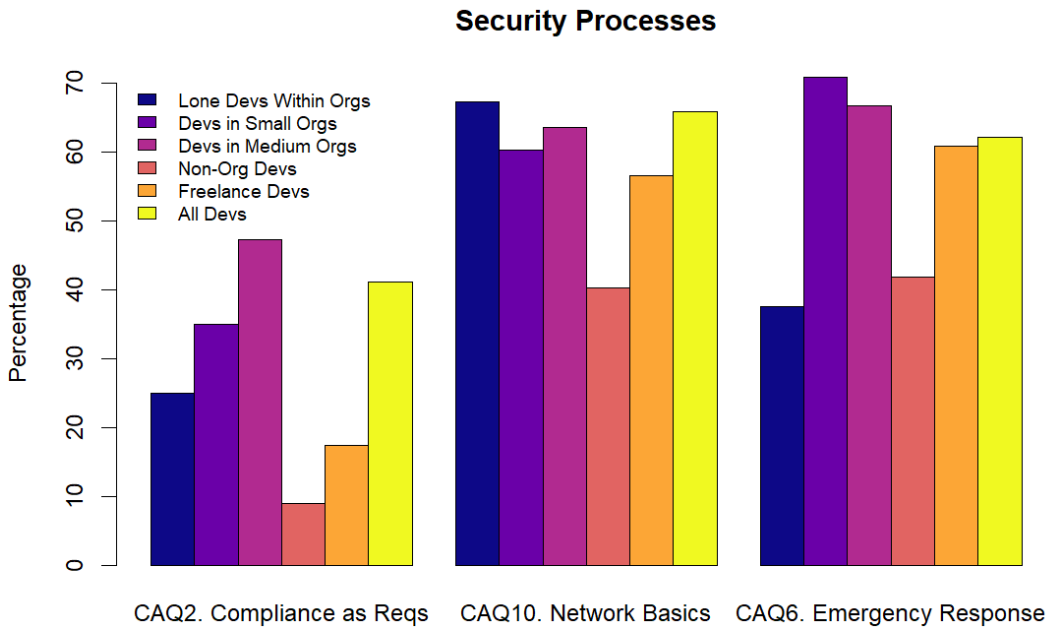


Figure 6.6: **Results for the three security processes CAs for all five developer categories and the whole sample.** Freelance and non-organisational developers engage least with compliance. All categories fare well for network basics, particularly lone developers within organisations. Developers in small organisations are most likely to express confidence in their emergency codebase response.

production code. Coders such as open source teams or app developers are likely to be able to ship without such constraints, though the difference is small. This question gave developers from small organisations their only chance to shine, topping the numbers at 70.9% adherence. Developers within small organisations may be nimble, with short command chains, and able to get needed authorisations quickly. Small organisations have fewer than 50 employees. Their software may be fundamental to the organisation's well-being and could even be its only product.

6.5.1.6 Security Tools

In an extensive literature review of software security aids, in which we considered accessibility for under-resourced or isolated developers, we concluded that automation and security tools hold high promise for developers who do not have much security support.

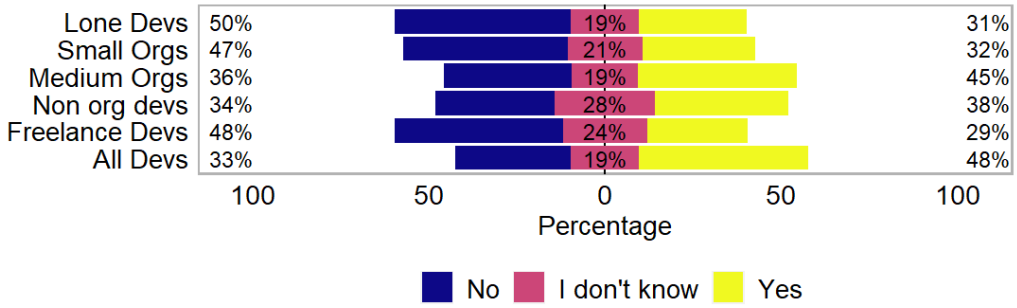


Figure 6.7: **Are there security tools available in your environment?** Answers from all six developer categories. We can see that the percentage '*Yes*' answers for the '*all developers*' sample is higher than for any of the developer categories we examine. Tool use in the individual categories we consider is relatively low, particularly amongst freelance and lone developers.

Therefore, we explored the usage of security tools in the five developer categories. The results can be seen in Fig. 6.7.

We were disappointed to observe that developer categories who are most likely to need the support of automation and security tools used them least, with freelancers at 29%, lone developers at 31% and developers in small organisations at 32%. Developers outside organisations had access to security tools at a rate of 38%. There was a noticeable difference between developers in small organisations at 32% and those in medium-sized organisations at 45%. The number for respondents as a whole was 48%, suggesting that large organisations get the most benefit from security tools.

We would like to see much higher numbers here. Only 29% of freelancers use security tools. There is a long way to go.

6.6 Discussion

6.6.1 Threats to Validity

As with most surveys, our respondents are self-selecting and the results may therefore suffer from self-selection bias or nonresponse bias. We attempted to mitigate this by recruiting organically, by keeping the survey open for three months to collect a large sample, and by publicising the survey through multiple different venues. Nevertheless, our respondents may not accurately reflect the general developer population. However,

we believe that the relative CA compliance between the different developer categories we examine is likely to reflect the broader population.

Survey responses sometimes suffer from social desirability bias, as respondents try to anticipate expected answers. To counter this effect, we emphasised that no security expertise was needed to participate.

Our sample size for freelancers is low at 23 participants. We do not claim that these results are generalisable, but we find them thought-provoking and believe that they may stimulate discussion and further research.

6.6.2 Discussion

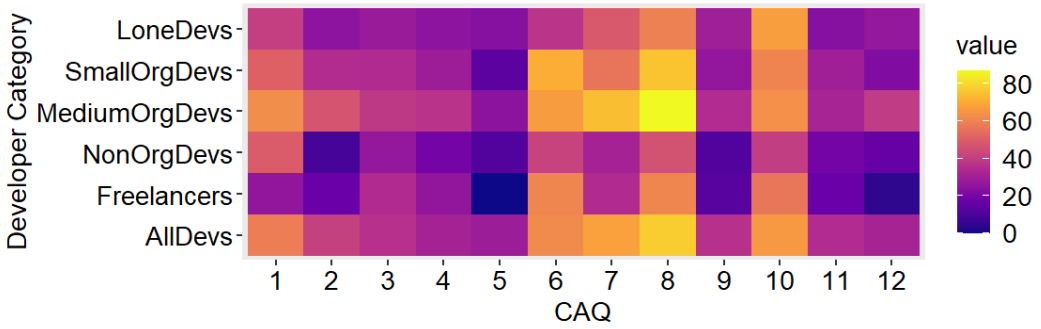


Figure 6.8: **Heat map of CA answers.** This heat map allows us to easily see the most-used common activities in the five developer categories.

When investigating use of the twelve CAs in our target developer categories, we were prepared to see little to no adherence to the activities. It was encouraging to see that these security practices are present in at least some instances. We can use Fig. 6.8, which shows a heat map of common activities use, to evaluate our initial intuition as to the three CAs that would be least used. The levels of compliance constraints translated to requirements (CAQ2) reached 47.3% in medium-sized enterprises, but were low at 8.96% for non-organisational developers. The highest level of security review (CAQ11) was 31.8% for medium-sized organisations. The lowest, 19.4% for non-organisational environments, was higher than we expected given the likely budgetary constraints. External penetration testing (CAQ12), used by 39.5% of medium organisations but only 4.35% of freelancers, performed badly but not as badly as internal penetration testing (CAQ5), used by 24.8% of medium organisations but not used at all by our small sample of freelance developers.

We speculate that internal penetration testing is too time-consuming and may take too much knowledge for developers to undertake it themselves. Although it is notoriously difficult to persuade management to invest in external penetration testing, several of our participants mentioned bug bounty firms in their responses. For example, one of our respondents from a medium-sized enterprise said:

> '*HackerOne is a service our company subscribes to for bug bounties/external penetration testing and it provides a lot of value for us.*'

As to the other activities, CAQ3 and CAQ4 are relatively little used, which is disappointing since these concern QA work which could be done by developers themselves. Our developers are strongest on CAQs 6, 7, 8 and 10 which concern deployment speed and resilience, developer and operations coordination, and network security basics. These activities can be undertaken for quality assurance and general cybersecurity reasons, without a software security focus. Therefore their presence does not imply a software security mindset in the working environment. In general, the level of practice of the CAs is higher than expected. We were disappointed by the level of security tool use, and would like to see this rise for developers who may not receive much other support.

6.7 Conclusion

Security vulnerabilities in software may expose its users to multiple types of cyberattack, including ransomware, cyber terrorism and cyber espionage. This is a particularly acute concern for critical systems. Yet many contemporary software systems depend on components developed externally, often by small organisations or open source teams. We wished to examine aspects of software security practice in five developer categories that are under-represented in the literature. These are lone coders within organisations, coders within small organisations, coders within medium organisations, non-organisational coders, and freelancers (see Table 6.2).

To do this, we analysed a dataset of 962 valid respondents to a survey on the state of software security practice. The survey asked about use of the 12 most common software security activities (CAs) (see Table 6.1). Many of these CAs can be undertaken even by lone developers — for example, ensuring that host and network security basics are in

place. However, we posited that some of them would be too expensive or time consuming for under-resourced developers to undertake.

We extracted the data on adherence to the twelve CAs for the five developer categories of interest. The results were encouraging, with some level of adherence to all of the activities within each developer category. However, there is a large degree of variation between developer categories for many of the activities. We found that most of the activities are undertaken most frequently by medium-sized organisations, the largest developer category we studied. This ties in with our understanding that the activities can be resource-intensive.

We also explored security tool use. Intuitively, automated tool use should provide most benefit to under-resourced developers. Surprisingly, we found that security tool use is quite low in our developer cohorts. This may be because developer sources of advice have gaps when it comes to discussing the benefits of security tools [6].

Based on our results, we advocate greater targeting of small and under-resourced software development communities with software security advice. Secure software development standards tailored to these categories of developer are needed. We also suggest that the benefits of security tool use should be promoted to under-resourced developers whenever possible. The accessibility, usability and price of security tools and other security aids should be considered by the industry in a holistic way. Isolated and under-resourced developers must be able to find and deploy them.

7 Training Developers to Code Securely: Theory and Practice

This chapter is published as: Ita Ryan, Utz Roedig, Klaas-Jan Stol. 'Training Developers to Code Securely: Theory and Practice.' *2024 ACM/IEEE 5th International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS) and 2024 IEEE/ACM Second International Workshop on Software Vulnerability (EnCyCriS/SVM '24)*. DOI: 10.1145/3643662.3643956.

7.1 Abstract

Software security is essential. Flaws in software design and coding produce vulnerabilities that can be exploited by hostile actors, resulting in ransomware, espionage and the hacking of critical infrastructure. Meanwhile, DevOps and continuous integration introduce speed imperatives, often bypassing traditional security gates. Software security responsibility is increasingly shifting to software developers. Industry insiders advise that this extra responsibility should be accompanied by developer training. But is it? Analysis of what constitutes good software security training is sparse, and there is little information on the amount and quality of training actually offered to developers in industry. We analyse recent literature to find the positive features of effective secure development training. We examine training information from a large developer survey (n=962) to assess how training in the field matches key positive features. We find that while some developers experience excellent secure-coding training, others receive inadequate training, and the majority receive none.

7.2 Introduction

Software security as a concern becomes more pressing as the world becomes more connected [238]. At the heart of software development, and therefore of software security, is the developer, who is most directly responsible for vulnerabilities in software design, and mistakes in implementation. Security-conscious workplaces may have software security teams, but their role as gatekeepers has declined with the rise of DevSecOps [172, 276]. Security auditors are often outnumbered by factors of more than a hundred to one [276]. Yet we have little understanding of the secure-coding training that developers receive. What features are important for secure coding training, and are they present in contemporary training courses? Are there security-training barriers that may reduce effectiveness, such as time pressure or lack of focus? What type of training is effective, and is it currently being offered in the real world?

The importance of developer secure-coding training is emphasised in the literature [290, 276, 279, 267, 314, 11, 10]. Tomas et al. discussed developer security training as a metric that can usefully be monitored as part of DevSecOps [279]. By correlating with the number of mistakes in different security categories, this metric could be used to guide training programs. For this measurement to be effective it should distinguish between vastly different training. An annual video does not compare, for example, with an immersive one-week instructor-led training course. More exploration of the nature of secure-coding training is needed [173].

Secure-coding training is used and recommended in industry. For example, the Building Security In Maturity Model, an industry secure-coding model, suggests that organisations provide software security awareness training, including individualised training and training during onboarding [171]. However, availability of training materials does not guarantee that they will be accessed. Potential barriers to security training, such as time pressure and lack of relevance, should be investigated and removed. Management buy-in is essential to having training time allocated for developers.

As part of a systematic literature review [241], we explored the features of secure-coding training. In this paper we analyse successful training and identify the features that appear to affect training quality. As part of a large developer survey [239], we enquired about the secure-coding training the participants received. We analysed the answers and compared them to the features we had identified. Our research questions are as follows:

RQ1: What are the features of effective secure-coding training?

RQ2: Are developers offered secure coding training, and if so, does it have the positive features we have identified?

Our contribution is twofold: we identify the features necessary for successful secure-coding training according to academic literature, and we describe the current state of secure coding practice, including whether those features are present.

The paper is organised as follows: Section 7.3 discusses background and related work. Section 7.4 outlines the methodology. Section 7.5 uncovers the features of successful secure-coding training. Section 7.6 analyses the survey data on current practice, to see if key features are present. In Section 7.7, we discuss our findings, their implications and limitations. Section 7.8 concludes the paper.

7.3 Background and Related Work

7.3.1 Training Availability

Developers may be motivated to code securely but ignorant of how to do it. Researchers have long called for more software security training in computer science curricula [200, 276, 32, 80, 10]. Even the most reputable universities still conduct a surprisingly low amount of secure-coding training [8]. In fact, lecturers routinely use insecure functions in code examples; thus they are actively training future developers to code insecurely [8]. Many teachers would like to adjust training materials to reflect code security concerns, but lack the time to update their content [43].

With little to no secure coding education before employment, developers are dependent on the organisations they work for to provide this training. Arizon-Peretz et al., in interviews with 27 practitioners, found that '*many of the participants' statements indicated that they did not feel that they had received sufficient training with regard to security and privacy*' [11].

There is little literature quantitatively evaluating whether developers are generally offered software security training in the workplace, and whether they are satisfied with the training offered. Pikulin et al. ran a small survey, concluding that developers' training preferences vary with experience and personality [217].

7.3.2 Training Effect

We did not find any previous work that focused primarily on correlation or causation between secure coding training and actual code security. We did identify studies which considered secure training effectiveness as a secondary concern. For example, Witschey et al., studying security tool adoption, found a correlation between secure coding training and adoption of security tools [330].

In studies that considered training and security correlation as a secondary concern, researchers disagreed and sometimes contradicted themselves. A laboratory study by Oliveira et al. examining '*blind spots in developer's heuristic-based decision-making processes*' stated that security training had a significant effect on security outcomes [195]. Developers were asked to modify and comment on vulnerable code snippets. The researchers checked for correlations between total scores and occupation, level of education, and so on. They stated that developers' total score and whether or not they had previously taken security classes were '*highly correlated*.' However, the Pearson's correlation value given in the paper's Table 5 is −0.313. This suggests a weak negative correlation, with the developers who had taken security classes actually producing slightly *less* secure code than others. It is unclear whether it is the positive assertion in the text, or the negative value in the table, that is correct, or whether the apparent discrepancy is due to a misunderstanding. Also, the content, relevance and duration of the security classes are not described. It appears that the evidence from this paper is inconclusive.

A 2016 coding competition and study called *Build It, Break It, Fix it (BIBIFI)* required teams to code software to specifications, and then attempt to break each others' solutions using hacking techniques [236]. The solutions and vulnerabilities were monitored and evaluated. A 2020 paper that quantitatively analysed the BIBIFI study data stated that there was no statistically significant effect on secure coding from having taken the related Massive Open Online Course (MOOC) [204]. This was not the whole story, however; the original 2016 BIBIFI paper stated that '*the effect size is quite large [though not statistically significant]; it is possible that the MOOC security training played a role*' [236]. This original paper's Table 5 reported a p-value of 0.086, so that the effect from MOOC participation may have narrowly escaped conventional statistical significance. If some non-MOOC participants had taken secure-coding classes elsewhere, this might have improved their secure-coding ability, and reduced the statistical significance of the

difference between the two populations. Considering '*logic, background knowledge and experimental design*' alongside statistical significance [75], we argue that the finding that *MOOC participation* had no statistically significant effect on the secure coding outcome does not imply that *secure coding training* does not have an effect on secure coding. This is important because a paper describing qualitative analysis of the same BIBIFI data, published in 2020 and also reporting no statistically significant effect of MOOC completion on secure coding, concluded that '*it is likely that increased development experience and security training have, at most, a small impact*' [314]. Note that MOOC completion was conflated with training here; participants may have taken different training.

This confusion from the BIBIFI papers seems to have muddied the research waters. Mokhberi and Beznosov, citing the qualitative 2020 paper [314] in a recent literature review [173], said:

> 'The appearance of different types of security vulnerabilities did not seem to correlate significantly with [...] completed security training'

Later, they concluded that [173]:

> 'as some research suggests, security training is not a significant factor in preventing vulnerabilities [...] more investigation is necessary to understand why education is not working and how developers' education can be adjusted or even reformed'

We agree that more investigation is necessary. This paper is a first step towards exploring this issue further.

7.3.3 Software Security Learning

Researchers have examined the learning tools available to developers to help them to code securely, identifying security gaps in the resource ecosystem [6]. Qualitative studies have suggested that developers should use Stack Overflow (SO) to learn about software security [154], although laboratory studies have indicated that use of SO can result in insecure code [3, 88, 244]. Imaginative approaches to training such as those based on serious games are described in the literature [80, 108]. These studies have mixed results.

7.3.4 Training in the Workplace

Writing in the first decade of this century, van Wyk and Steven recommended that organisations should adopt interactive, engaging secure-coding training, tailored for different job roles and experience levels [308]. The academic literature on software security training in industry usually discusses a single implementation. For example, at Dropbox, participating in an internal capture-the-flag competition appeared to reduce security overconfidence in developers, made them more aware of a range of potential vulnerabilities, and improved their comfort-levels with documentation and the security team [315]. Considering security in cyber-physical systems, Crowther and Rust [58] advocated continuous training including gamification and other interactive training. The presence of training materials does not guarantee that they will be used, and researchers warn against optional training, or dull training [58, 248]. Annual security compliance training was described by a participant in a Haney et al. study as '*a layer of Dante's Hell*' [112].

Interviews with software security auditors suggest that organisations often tailor software-security training to the defects and challenges that occur in-house [276, 225]. Ethnographic researchers have observed that, within organisations, events such as audits, code reviews and the receipt of static analysis reports can create secure-coding learning opportunities [152]. On-the-job training is recommended by secure-coding experts [319], and is usually done by mentoring, embedding security champions within teams, or having security experts available for ongoing learning. Security champions themselves may benefit from on-the-job training, leveraging expertise from security groups [130]. Tailored and on-the-job training or mentoring occur in studies describing organisations with a positive security culture [111, 284]. Studying the effect of a one-off security consultancy and training event, Poller et al. observed that the security momentum dissipated over time and was not facilitated by the organisational structure [220]. An illuminating contrast was provided by Tuladhar et al., who documented the introduction of a software security drive by upper management in a software company [284]. They observed that lessons learned from formal software security training were initially difficult for software developers to apply to their daily work. Improvements in software security practice came from a positive security culture, with daily software security discussions, and situated learning.

Surprisingly, we could not find any previous work that discussed software security training in industry, identifying and evaluating the training currently available to developers within the workplace. We leveraged the knowledge gained from our literature review, and the information on training in practice gleaned from our developer survey, to begin to fill this gap.

7.4 Methodology

7.4.1 Generation of Positive Secure-Coding Training Features

The first research question asks how secure development training should be structured and delivered. We generated a list of desirable features of secure-coding training from a systematic literature review. This review considered papers related to developer-centred security from a large range of venues, and analysed them for activities that aided in secure development. The review procedures are discussed in more detail in a previous paper [241], and the in-depth review protocol is available online [242] and in Appendix A. In the course of the review we encountered multiple papers that mentioned secure-coding training, as discussed in Section 7.3. However, little exploration of training quality and adequacy was done.

We generated a list of desirable software security training features from relevant literature review papers. Each paper was read carefully, and recommendations related to secure-coding training extracted. See Section 7.5 for further details.

7.4.2 Observation of Secure-Coding Training Features in Industry

The second research question concerns how secure coding training is structured and delivered in practice. While investigating factors that affect secure coding, we created a developer survey with questions about secure coding procedures in the working environment [239, 243]. The questionnaire was pilot tested in two phases. It was then publicised via personal contacts, a conference talk by the first author, and social media platforms such as Twitter, LinkedIn and Facebook. The survey was designed to be engaging. The first author publicised it daily over three months, generating 1,100 organically-sourced responses. Screening removed respondents who did not code frequently, either professionally or in an open source context. We also screened out non-programmers, and

respondents who were answering at random, ending with 962 valid responses. Prior work gives a more detailed description of screening, and of the process of devising, testing and distributing the questionnaire [239].

In prior work, we analysed questionnaire answers to questions on secure coding practice and culture. We found that secure coding practice in the participants' environments ranged from non-existent to thorough [239, 243]. We also found evidence that some organisations with thorough secure coding practice appeared to be focused primarily on compliance, lacking a good secure coding culture [239].

The survey also included two questions on secure-coding training, which are the focus of this paper. Respondents were asked whether they had been offered secure-coding training. If they had, they were asked to describe it. Surveys that constrain respondents' answers to specific values can impose forced choice bias, which is inappropriate in a context where increased understanding of context is sought [91]. Therefore, the description was requested in free text. The free text comments were both informative and enjoyable to read. We have included several of them in this paper, to provide readers with a flavour of the richness of the responses. This qualitative data was coded by the first author to identify the features of secure-coding training offered, and any other themes. A description of the coding process can be found in Section 7.6.

We obtained approval for the questionnaire from our institution's Ethics Review Board. Correlation analyses were done in R using Pearson's correlation. Any quote from respondents is presented exactly as entered in the questionnaire.

7.5 Secure-Coding Training in Theory

In this section we attempt to answer Research Question 1: *What are the features of effective secure-coding training?* We base our answer on the studies encountered in our systematic literature review, and on broader related research. Since we did not find any research explicitly focused on measuring the effectiveness of secure-coding training, we are unable to discuss statistical effect sizes. There is no statistical data available to suggest which training types are most effective, or which training features result in the most secure code. This area of research is in its infancy, and we hope that this paper will serve as a foundation to encourage others to explore these questions. The studies we encountered that discussed secure-coding training were ethnographic and

interview studies. They describe the secure-coding training features that are most valued by developers and organisations. Table 7.1 provides a list of the recommended features and their sources, in four groups. In the following sections we discuss these four feature groups.

7.5.1 Relevance

As illustrated in Table 7.1, every paper we analysed for this work discussed the importance of making training **relevant** to developers and their context. While decontextualised learning can be effective in some situations [109], when developing with specific computer languages and frameworks, secure coding depends on intimate knowledge of the security qualities of the framework or language [216]. As stated by Larios-Vargas et al., '*developers perceive generic security training as too abstract, time-consuming, and ineffective*'.

The importance of training relevance was echoed during coding of respondent feedback, where an explicit criticism found in some free text comments was that training content was off-topic.

Related to relevance is the necessity for training to be **actionable**, and this was also mentioned in all the work we reviewed. Developers sometimes find it difficult to put secure-coding training into practice; this problem is ameliorated where the training is immediately relevant to the developer's work [284]. Ideally, developers work through examples in training which they can then apply directly when coding.

Furthermore, to be relevant, training should be **up-to-date**. Most of the papers that we analysed either suggested or implied that training should be **iteratively improved**, a process likely to keep the content fresh and topical. This is very much the opposite of designating an annual video for developers to watch, and then ticking the 'training' box. For example, a security auditor interviewed by Thomas et al. [276] said that:

> 'What we often do is periodically assess the category of vulnerabilities, what the kinds of volume and mistakes the developers are making and provide a very small part of training just on those things.'

This is an example of iterative improvement that makes the training relevant, actionable and up-to-date. It also illustrates how **errors-based** training, recommended by several of

the papers we reviewed, can be implemented. The auditors interviewed by Thomas et al., conscious of the high proportion of developers to security experts, suggested that secure-coding tools could double as **training tools** if they accompanied error notifications with information about defects, secure coding and learning opportunities [276]. A common suggestion for content was that it should **include risks**.

7.5.2 Time Allocated

As can be seen in Table 7.1, several of the papers we read emphasised the importance of **continuous** training for ongoing learning. Analysing how situated learning contributed to security culture in a software company, Tuladhar et al. found that ongoing discussion of security requirements and application of training knowledge in context is important. **Ongoing mentoring** is a useful way of delivering continuous training, and was mentioned in those papers that focused on having a secure-coding culture in the workplace [111, 284]. Several papers also mentioned that a secure-coding training regime should **include time** for developers to engage in self-directed learning, exploring secure-coding issues in depth. Several papers imply that **mandatory** training is essential, while Crowther and Rust, writing about their industry experience coding cyberphysical systems, emphasise that '*effective training must be mandatory*' [58].

7.5.3 Engagement

Two studies explicitly advise that training should be **engaging** to retain developer interest [276, 58]. This is easier when **varied** training materials and delivery vehicles are available to developers [146]. Training material that is **interactive** engages the user. **Gamified** training takes interactivity up a notch, providing the trainee with game-like rewards as they progress. A well-known general example is Duolingo, which gamifies natural language learning. Crowther and Rust note that '*a gamified delivery has increased the level of engagement, with the average user doing more training than the requirement*' [58].

7.5.4 Additional Training Features

There were four additional recommendations on secure training delivery that are somewhat related. Several qualitative studies of successful secure-coding environments note

	Crowther and Rust [58]	Arizon-Peretz et al. [10]	Larios-Vargas et al. [146]	Tuladhar et al. [284]	Thomas et al. [276]	Haney et al. [111]	Assal and Chiasson [13]
Relevance							
Relevant	●	●	●	●	●	●	●
Actionable	●	●	●	●	●	●	●
Up-to-date	◐	●	●	●	●	●	●
Iteratively improved	●	●	●	●	◐	○	◐
Errors-based	●	○	○	○	●	●	●
Include risks	●	○	●	●	○	○	●
Training tools	○	○	○	○	●	○	○
Time allocated							
Continuous	●	◐	●	●	○	●	○
Ongoing mentoring	○	○	○	●	○	●	○
Include time	●	○	●	○	○	●	○
Mandatory	●	○	○	○	◐	◐	◐
Engagement							
Engaging	●	○	○	○	●	○	○
Varied	●	○	●	●	○	○	●
Interactive	●	○	○	●	○	○	○
Gamified	●	○	○	○	●	○	○
Additional training features							
Available expert	○	○	○	●	●	●	○
Peer review	○	○	○	●	◐	●	○
All hierarchy	○	○	○	○	○	○	●
Non-tech skills	○	○	●	○	○	○	○

Table 7.1: Desirable Features for Secure Coding Training. ● = explicitly mentions the training feature. ◐ = implied only, but not explicitly mentioned. ○ = no discussion of this feature.

that they benefit when there is an **available expert** whom the developers can consult and from whom they can learn. Security **peer review** allows more expert developers to draw their peers' attention to security issues in code, leading to discussion and a learning opportunity. Assal and Chiasson advise that software security affects all of an organisation, and there should therefore be tailored, relevant training available throughout the hierarchy, from management to developers [13], a requirement defined in Table 7.1 as '**All hierarchy**.' This suggestion gains credence from a study comparing two organisations by Oyetoyan et al., who assessed the specific secure code training needs of developers, architects and testers, finding that they differed [200]. Developers may also benefit from some training on **non-tech skills**, more typically targeted at management. Larios-Vargas et al. [146] were the lone voice suggesting this, saying:

> 'most security challenges are indirectly related to conflict management, negotiation skills, communication ability, and empathy [...] we encourage organizations to include topics related to non-technical skills as part of any security training program, which is helpful in the context of security and boosts collaboration and productivity in the company.'

7.5.5 Indicators for a Poor Training Environment

Having absorbed all the advice discussed above, we considered indicators for a poor secure-coding training environment. Indicators for a poor training environment would be that the training is irrelevant, off-topic, dull, or infrequent, is not adjusted over time based on feedback and industry events, or is skipped due to tight deadlines.

7.6 Secure-Coding Training in Practice

In this section we assess the survey respondents' feedback on training in the light of our research questions. Since we only had two questions on secure-coding training, we consider this preliminary work. Future research in this area should consider the list of desirable features from Section 7.5 and include them in investigations.

7.6.1 Demographics

A detailed description of the sample can be found in prior work [239]. Respondents ranged in age from 18 to over 65. When asked what gender they most identified with, 82% of respondents answered *'Man,'* which is in line with other industry surveys [262]. Respondents selected or included 59 different countries as their working location.

7.6.2 Breakdown of Training Answers

To assess secure-coding training in practice, we evaluated answers from two questions in our secure-coding survey. Developers were asked '*Have you ever been offered training relevant to software security or privacy in your environment?*' with possible answers '*Not applicable*,' '*Yes*,' or '*No*.' Table 7.2 reports the results. Only 381 respondents, 39.6% of the total, answered 'Yes.' The next question was: '*If you answered yes to the previous question, please describe the training here*.' Input to this answer was in free text, as we did not want to constrain responses.

Of the possible 381 free-text descriptions of secure-coding training, 75 were left blank. The remaining 306 were coded by the first author. Each entry was initially evaluated and flagged as either relevant to secure coding training or not relevant. If not relevant, it was assigned to one of the codes in Table 7.3. These codes were generated as analysis proceeded, appearing in the survey data. Some respondents did not comment, or listed off-topic training; this included standard IT training such as anti-phishing, with no software development component.

Once all of the irrelevant comments had been flagged, only 220 comments remained that actually described secure-coding training. In the next section, we analyse these comments to attempt to answer Research Question 2.

Table 7.2: Whether the respondent states that they have been offered training relevant to software security or privacy in their environment. Only 39.6% of respondents replied 'Yes' to this question.

Response	Frequency	Percent
Yes	381	39.6
No	495	51.5
Not applicable	86	8.9

Table 7.3: Types of training description by the 381 respondents who stated that they had been offered training. 'Blank' indicates that no comment was made. 'IT Security' means that the respondent described standard IT security training, with no secure coding component. 'Privacy' means the description involved privacy only. 'OT Comment' indicates that the respondent did not actually discuss training when answering this question. 'Description' indicates that text compatible with secure-coding training was supplied. The examples are complete verbatim quotes; many responses were brief.

Category	n	%	Example response text
Blank	75	19.7	*NA*
IT Security	40	10.5	'*Mostly how to avoid phishing attack vectors*'
Privacy	27	7.1	'*Video lessons about the Brazilian GDPR (LGPD)*'
OT comment	19	5	'*If by offered, you mean sales offerings*.'
Description	220	57.7	'*internal, often language or platform specific*'

7.6.3 Research Question 2: Are developers offered secure coding training, and if so, does it have the positive features we have identified?

In this section we explore the descriptions of secure coding training given by participants in our survey, and whether they correspond with theoretical features of good training described in Section 7.5. The descriptions were coded by the first author. Key features identified from the literature review were assessed. Several additional features were found in survey responses. Two respondents' security training was so good that they refused to divulge any details, saying simply *'Confidential.'*

7.6.3.1 Relevance

On analysing the 220 comments that described secure-coding training content, we discovered that 160 indicated that the content was slightly or very relevant (see Table 7.4). Inevitably, some comments were negative, such as *'Boring'* and *'Dry training courses around secure development in conjunction with sales from SAST providers.'* Others were very positive, and these tended to describe relevant training. For example:

> ‘My team offers a security champion program at our workplace :), other than that we have an internal, mandatory security training course for all employees, and optionally free courses on udemy / generous budget if necessary for further learning.’

An example of training that lacked relevance:

> ‘ “Secure Code Warrior” – an online platform that shows sample insecure code, and asks you to identify “vulnerabilities” and pick the correct remediation from multiple choice. Very broad and unfocused, targets niches that are not applicable to our team, mostly a waste of time.’

It is interesting that Secure Code Warrior was mentioned in positive terms by six participants but negatively here. In this case it seems that context was not considered when the training was chosen, which, if it is happening generally, is cause for concern.

Table 7.4: Whether training was relevant and interactive.

Feature	No	Not really	Unknown	Slightly	Very
Relevant	6	8	46	69	91
Interactive	33	5	149	9	24

7.6.3.2 Time allocated

In our list of positive features of good secure-coding training we included several related to time, including ‘Continuous.’ When coding responses, we found that a ‘Frequency’ variable was needed (see Table 7.5), because durations varied widely. There were 35 participants whose training seemed to be continuous, while 31 were given annual or one-off training. Annual, or even less frequent, training suggests a compliance approach that does not add value.

Table 7.5: Training frequency. Frequency for 133 participants was unknown. ‘Old’ means the training took place years ago.

Continuous	Frequent	Monthly	Regular	Annual	One-off	Old
35	8	2	7	19	12	4

Two other features relevant to time are whether training involves ongoing mentoring, and whether training is mandatory. Our findings on these values can be seen in Table 7.6. Ongoing mentoring allows the developer to obtain timely and relevant help. We only identified eleven free-text entries that had explicit or implicit references to mentoring, while in 39 of the entries it was clear that no mentoring was taking place.

When training is not mandatory it can be skipped due to deadline pressures. If this is allowed it conveys the message that secure coding is not important [11]. We identified six instances where training was not mandatory, and 32 where it was. Five participants explicitly told us that they did not or could not attend security training. One participant said *'I can't [describe the training], I ended up not going to it due to project pressures.'*

Table 7.6: Whether training included mentoring, whether it was mandatory.

Feature	No	Unknown	Yes
Ongoing mentoring	39	170	11
Mandatory	6	182	32

7.6.3.3 Engagement

Due to the brevity of most of the answers, the easiest engagement feature to assess was interactivity (see Table 7.4). While 38 participants seemed to receive training that was not interactive, training for 33 others was slightly or very interactive. This shows the large range of effectiveness in the training reported.

7.6.4 Features Added from Survey Data

A number of additional features were found in the survey data. Some were positive, some less so. We describe the top three here.

'**Timely**' indicates that help is available when needed. For example, one participant said '*We can schedule security reviews with a dedicated security team*,' which is an indicator that on-the-job training is available. Seven comments had the 'timely' feature.

'**Compliance focus**' indicates that the training may be a box-ticking exercise. There were 13 comments that appeared to indicate a compliance focus, for example:

> 'None of it was practical – it was all "*Read this document that says we are security focused, and then sign off that you read the document. Your security training is now complete*". '

> 'The training was fairly pro forma security training that, frankly, didn't teach me anything I didn't already know, but let my company tell clients that it is an annual training all of us do.'

'**Certification**' indicates that the training included aspects aimed at certification. There were nine such entries. An example is '*Pentester lab licenses, regular CTF events*.'

Table 7.7: Developer tone discussing secure-coding training.

Very Negative	Negative	Neutral	Positive	Very Positive
5	17	164	31	3

7.6.4.1 Developer Tone

The tone of developers' text entries varied widely, so we analysed it; Table 7.7 reports the results. Some training was considered terrible by the participant: *'It was honestly awful – unbelievably bad. Less than useless.'* Other participants were very positive: '*Many options by internal security teams and external trainers including web technology focused trainings, tool/language best practices, secure coding puzzles, etc*.'

Overall, we found that while most of the participants' descriptions were neutral, 17 were negative and 5 very negative. Positive comments comprised 31 of the responses, with 3 very positive entries. Developer sentiment correlated with the relevance of training, with a correlation of .497 and a p-value less than 0.001.

7.7 Discussion

Training in the workplace has symbolic significance, signalling management priorities, as highlighted by Arizon-Peretz et al. [11]:

> 'A lack of effective training for handling security and privacy concerns may convey two messages to employees: (a) that the employee is not really expected to handle

them, and (b) that privacy and security are not considered main goals, even if they are declared to be important.'

Aside from signalling intent, training should also impart knowledge and improve skills in the targeted area. The question of whether secure coding training improves developers' secure coding performance is a vexed one. We found very little work that considers this question. Where the effect of secure-coding training is discussed, it is as a secondary concern in studies whose main focus is elsewhere (see Section 7.3.2). It is somewhat surprising that secure-coding training has not yet been formally evaluated by the academic community. The explanation may lie partly in the use of an analogy with computer security for general computer users made by Wurster and von Oorschot in 2008 [333], and more recently by Acar et al. [4]. Wurster and von Oorschot [333] argued that:

'While it is generally accepted that "more user training" is not a viable solution to some security problems, apparently many in the security community continue to believe that developer education will solve the [software security] problem...we argue the best approach is to develop and enforce technical solutions instead of assuming that developers can be educated to *do the right thing* [emphasis theirs]'

We agree that it is important to create technical solutions [333] and usable security tools [4]. However, it is also essential to educate software developers about secure coding in general, and about the secure-coding traits of the platforms and development tools they are using in particular [319]. Software that does not itself process sensitive data may be used as a stepping-stone to other parts of a system [24]. Software security should be a central concern for most software developers, in a context where much of the software they write will be under continuous attack [92, 309]. Software security is a broad and far-reaching problem. There is no silver bullet. Like cybersecurity in general, an approach of defence in depth is necessary. Programmers cannot be left in ignorance of what is needed.

Our research questions are addressed in Sections 7.5 and 7.6. To answer RQ1 we analysed several qualitative studies which described the training features valued by developers and security experts. These features may appear to be trivial, or just common sense, but evaluating RQ2 we discovered that most survey respondents were not even offered secure-coding training. When it was offered, basic attributes from RQ1 such as

relevance, frequency and timeliness were often missing. Quality varied widely, as did the tone of training descriptions. Some had a neutral tone. Negative comments tended to describe the training as irrelevant, while positive comments described varied, easily available and targeted training. This work provides a grounding for future research, defining some of the differentiating features which should be assessed when evaluating training. Future work could explore grey literature on this subject, and practitioners' assessments of secure-training needs.

Developers have been observed not to apply their security knowledge to a task that obviously requires security, storing passwords in a database, unless asked [183]. In ethnographic research, developers tolerated known security issues in code because they reduced support calls [202]. Such findings suggest that leadership in organisations should overtly prioritise good security culture, including ongoing learning and mentoring [111, 319, 284]. More research is needed to ensure that training budgets are wisely spent, and well-resourced.

7.7.1 Limitations

Extraction of training features from the literature was done by a single author, which increases the chances of bias. However, the analysis was conducted rigorously, and notes were kept throughout. The current study is exploratory and designed to initiate discussion; further research could replicate and extend this study.

Coding of survey responses was also done by a single author. The responses were short, with the longest being only 583 characters (98 words) and the mean number of characters being 78. We considered this a sufficiently simple sample to be coded by one researcher, with consultation with co-authors when required.

7.8 Conclusion

There is little research available on types and effectiveness of secure-coding training. This paper makes a first attempt to address the issue. We began by analysing the training-relevant papers from a prior literature review, to define the features of good secure-coding training. These features are based on developer, security auditor and researcher observations in relevant qualitative research.

We then analysed the answers to training questions in our secure-coding survey (n=962) to find whether the training being offered to developers has the features we identified from the literature. Unfortunately, only 39.6% of respondents stated that they had been offered training relevant to software security or privacy in their coding environment. Only 220 respondents, or 22.9%, described their secure-coding training. Negative or very negative sentiments on the training were expressed by 22 respondents. At least five could not attend. We found indications that training was sometimes subpar, being compliance-focused, lacking context or relevance, or being insufficiently frequent. Positive developer tone correlated positively with the relevance of training. Research is needed on how to ensure excellent developer secure-coding training in industry.

8 ‘Money. Or Jail Time.’ Improving Software Security by Motivating the C-Suite

This chapter is a manuscript that will be submitted for future publication: Ita Ryan, Utz Roedig, Klaas-Jan Stol. “‘Money. Or Jail Time.” Improving Software Security by Motivating the C-Suite.’

8.1 Abstract

In a world of increasing digitisation, the security of software and systems is more important than ever. Software continues to contain vulnerabilities that can be exploited for ransomware, cybercrime and cyber-espionage purposes. Vulnerabilities are introduced during the development process. Much software security research focuses on software developers; their ability, tools and motivations. However, the work environment in which they operate is a key determinant of the budget, tools and secure-coding support available to developers. Budgets and priorities ultimately derive from an organisation’s senior management. Little work has been done on whether and how senior management understands and prioritises software security. We interviewed 21 professional developers, managers and consultants to obtain a sense of how management priorities shape software security within an organisation. During our interview process we investigated participants’ observations on C-suite software security understanding and motivators. We found that management priorities are important in providing the resources needed for secure software development. Software-security understanding can be lacking in the C-suite, which can affect its prioritisation. While client satisfaction is a consideration, regulatory and compliance requirements are other key determinants of software security prioritisation. We also investigated the software security implications of organisational

turbulence, finding that acquisitions, divestitures, downsizing, and funding issues are all likely to affect efficient implementation of software security policies.

8.2 Introduction

Chapter 2 outlines how cybercrime, and the use of cyberespionage and cyberattacks in geopolitics, are constantly on the rise. With the consequences of these activities becoming ever more prevalent and significant, vulnerabilities in software remain an area of focus. The constant proliferation of vulnerabilities is partly a consequence of the rapid increase of software use cases in recent decades and the increasing professionalisation of cybercrime [119]. It is also due to poor secure development. This issue is investigated further in Chapters 5 and 6, which show that not all developers and workplaces pay attention to software security. In work on security culture, Chapter 5 further outlines how, when organisations do prioritise security, genuine security interest may be absent and security operations may be based on legislative or industry standards. Although better than situations where no software security precautions exist, this can result in inadequate or non-existent training, which is considered in Chapter 7, and in a poor security culture where vital security steps are skipped.

Exploring these issues with a focus on the direction set by top management, we could not find research into how the priorities set by leadership affect secure coding within organisations across industry. We also did not find research on what causes organisational leadership to prioritise code security. This latter question is highly significant, since without leadership support, a trustworthy, ongoing secure-coding regime is very difficult to establish [80]. As a practical example, consider proposed software-security interventions that involve engaging development teams within organisations and teaching them basic security practices [150, 320]. Without goodwill from the organisation, such interventions are unlikely to occur, and very unlikely to succeed. Most organisations already have large numbers of conflicting objectives and priorities, such as tight deadlines [202, 285] and the desire to limit staff numbers [282]. Since secure coding is likely to result in more work and the need to change either deadlines, staffing numbers or both, what would motivate senior management to prioritise it?

In this paper we interview 21 professionals who either work in organisations where software is developed, or consult for a range of such organisations. We investigate the

influence of leadership priorities on software security practice in these organisations. We discuss participants' opinions of what motivates the CEO of an organisation, and the senior executives who form the rest of the C-suite, to prioritise software development. Finally, we explore what eventualities in organisations tend to cause a shift in software security emphasis. Our research questions (RQs) are as follows:

RQ1: How do management priorities affect software security within organisations?

RQ2: How well do C-suites understand software security?

RQ3 What factors are likely to motivate C-suites to pay attention to code security in organisations?

RQ4: What factors have participants encountered that had a significant effect on software security in an organisation?

8.3 Background and Related Work

8.3.1 Prioritisation of secure development within organisations

There is extensive research into software security for developers. Researchers have asked what secure-coding abilities they have [5, 184, 185, 183, 182], what types of interventions would be of use in making them code more securely [320, 227], and what their secure-coding motivations are [14]. One aspect of the developer's secure-coding experience which is repeatedly raised as important is the environment in which they work [216, 14, 61, 228, 174, 237, 243]. This suggests that, within organisations, prioritisation of secure coding by middle and upper management is important. An example of the resources under management's control is time. In a laboratory study on information sources in secure development, 16.7% of developers commented that the tight time limit made it difficult to consider security [3]. Translated to a work environment, a lack of prioritisation of secure coding by upper management causes developers the same time issues, among others [149]. Indeed, investigating software security in two organisations, Lopez et al. noted that '*the security positions of organisations may have more weight in promoting security activity than does championing by individuals*' [151].

Since organisations themselves cannot have priorities, research discussing an organisation prioritising software security is in fact referring to software security being a priority for top management within the organisation. Once upper management backs a

software-security drive, '*it's just a standard change management practice*' [146]. In an ethnographic study of how a security culture was developed in an organisation, Tuladhar et al. concluded that one of the key security enablers was upper management '*setting security as a goal*' [284]. Espinha Gasiba et al. ran a survey to investigate whether developers are aware of and adhere to secure coding guidelines [81]. The first '*actionable item*' they presented for consideration by practitioners was that '*without management understanding and approval, it is not possible to establish secure coding practices in a company*.' A practical illustration of the advantage of prioritisation comes from Tøndel et al., who said that '*one Product Owner compared security to testing and explained that testing was part of the procedures and processes and were part of KPIs, and thus testing was done despite time pressure*' [285].

Given the importance of management software security support, studies that find that it is missing are of interest. Morales et al. described numerous difficulties in a large multi-year DevSecOps project with multiple subcontractors where security (despite the '*Sec*' in DevSecOps) was not prioritised [174]. Amongst the implications of this failure to prioritise secure coding was a complete absence of security testing by the subcontractor featured in the study. Morales et al. noted that a failure by leadership to emphasise software security conveys the message that secure coding is not important; a point also made by Arizon-Peretz et al. [11, 10].

Numerous studies describe techniques that organisations can use to enhance software security. These range from identifying, and acting upon, the reasons for a gap between organisations' declared security goals and their employees' understanding of those goals [10], to designing secure-coding interventions tailored to specific developers [227] and contexts [146], to running regular meetings to surface secure-coding issues [286]. It is difficult to envision such interventions being embraced in places where secure coding is currently ignored.

Interest from senior decision-makers is needed to make budget and resources available. The secure-coding posture of organisations and top management is therefore of interest when considering secure-coding issues [286, 134]. The question that seems most relevant is this: what would motivate organisations that currently ignore code security to make it a priority? Assal and Chiasson advocated research on '*why some teams completely ignore software security, and what factors could encourage them to adopt a security initiative*' [13]. Oyetoyan et al. noted that '*We need to further investigate the drivers for increase*

in software security adoption in an organization' and speculated on what they might be, concluding with management security commitments. Ryan et al. noted that researchers sometimes assume that organisations and managers prioritise software security, without exploring further [241].

Researchers have identified in passing a number of factors that motivate secure coding within organisations. Weir et al. noted that '*compliance checks, certification and recent relevant legislation have all caused an increased interest in security in the organization*' [320]. Haney et al. interviewed individuals from organisations that implemented cryptographic solutions and found a strong security culture pervading the organisations, including secure-coding interest all the way from individual developers to top executives [111]. They noted that external security drivers include '*gaining a larger market share or customer requirements*.' They emphasised that the organisations they studied were not representative of software development as a whole, and tended to have a mature and considered approach to software security which could be emulated by others.

To our surprise, we did not identify a study that specifically examined drivers or motivations for introducing software security to organisations.

8.3.2 Sudden changes in secure development prioritisation

An aspect of the drivers for secure coding within organisations that interested us was sudden change in an organisation's secure coding posture. There is little available literature on this subject. However, two pieces of research give insight into such events. Weir et al. studied the effect of lightweight secure coding interventions [320]. During the 3-month intervention period, one team lost three participants to redundancy. A year after the interventions began, only one of those initially interviewed was still with the company. No final interview could be obtained. It is difficult for institutional memory to survive such comprehensive disruption.

Lopez et al. undertook a year-long ethnographic study of secure development in two organisations [151]. Both had been recently acquired and now had a different security focus, yielding an interesting contrast. In one, developers were newly enthusiastic about secure coding, and were proactive. In the other, developers had lost their previous software security confidence. They were transferring into a compliance nightmare, where

they did not feel confident that their software was secure despite being made to jump through multiple security hoops.

These two papers provide indicators that there is an unexplored and important gap in the literature. In this paper we make an initial attempt to investigate further.

8.4 Methodology

We considered different ways of investigating the research questions and decided that use of semi-structured interviews would best suit our goal. Surveys can impose forced choice bias if answers are predefined [91]. This bias is avoided if answer fields allow free text, providing the respondent with the opportunity to volunteer potentially interesting new information. However, in normal survey conditions the researcher is not present and cannot respond to the interesting information, probing deeper. In addition, survey questions are most useful when grounded firmly in prior literature. There is little literature available on our central topics of organisational secure-coding motivations and organisational turbulence. Qualitative research is more appropriate for research questions where an existing theoretical framework has not been identified.

8.4.1 Interview participants

To evaluate the research questions we interviewed 21 professionals working in software development. We had two sources of interviewees. While conducting a survey we solicited contact details from professionals interested in being interviewed in relation to secure software development. This gave us thirteen participants. The remaining eight interviewees were contacts of the authors, selected for their expertise in the areas of interest.

See Table 8.1 for a list of professionals interviewed. Due to anonymisation requirements we did not assign nationality, organisation size or industry in the table. Most interviewees came from the U.S. (10), Ireland (6) and the U.K. (2), with a single participant each from Belgium, Germany and India. Organisation size ranged from multinationals (5) and large organisations (2) to organisations with 5000 employees or less (10). Two respondents were self-employed, one coded only in open source, and one was a tech journalist who had left active development work. The range of industries was

Table 8.1: **Interview Participants**. Due to anonymisation requirements, industries, countries and organisation size are omitted from this table. See text for those details.

Development Team Members	
P1	Tech journalist, ex-developer
P2	Systems engineer intern
P3	Software engineer
P4	Software engineer
P5	Software engineer
P6	Software engineer
P7	Software engineer
P8	Open source developer
Senior Software Engineers	
P9	Senior software engineer
P10	Senior business analyst/engineer
P11	Senior software engineer
P12	Senior software engineer
P13	Senior software architect
P14	Lead software engineer
Consultant Developers	
P15	Director
P16	Consultant developer
Senior Managers	
P17	Global M&A lead
P18	Cybersecurity IT Lead
P19	Vice President
Management Consultants	
P20	Management consultant
P21	Management consultant

broad, including services (6), critical infrastructure (5), financial (4), security software (3), software (2) and open source (1).

Five different types of professional were interviewed. **Development team members**' view is practical and based on the reality of development in an organisation. While

developers are usually not privy to discussions at board level, they have a keen awareness of the time, tools and training available for software security at the coalface. **Senior developers** have more experience and have often worked in multiple organisations. They have an understanding of resources available to development teams in the organisation, and of the direction received from senior management. **Consultant developers** are independent developers who work with many different organisations and have profitable apps of their own. They have a good grasp of industry-wide challenges. **Senior managers** are aware of the main areas of concern and risk in the organisations they work with. They know what topics are under discussion at board level, and also what topics are not on the radar. Two of those we interviewed had experience in evaluating organisations for acquisition, giving them excellent insight into secure software development in multiple environments. **Management consultants** have a broad view of the industry, and they too have dealt with many different organisations. They are accustomed to quickly sizing them up, and evaluating issues of concern.

8.4.2 Interview analysis

Interviews were conducted by the first author and were semi-structured. We devised a list of questions designed to explore issues of software security and decision-making within organisations. We retained the flexibility to ask additional questions when the interviewee introduced interesting topics. The first author also analysed the interviews, using reflexive thematic analysis (RTA) with careful note-taking. This form of analysis was chosen because it is a flexible technique designed to explore themes in qualitative datasets [53]. Coding in RTA is acknowledged to be partially subjective, indeed this acknowledgement is considered one of RTA's strengths [31]. RTA is therefore best implemented by a single researcher who is thoroughly familiar with the entire dataset.

Byrne's worked example of Braun and Clarke's approach, which incorporates developments in Braun and Clarke's thinking subsequent to their canonical paper [30], was used as a guide during analysis [35]. The coding was done using the NVivo software package, with additional work in Word and Excel. We used a combination of deductive and inductive coding of interview transcripts to identify themes in the interviews. Phase one involved transcribing and rereading the interviews, noting and considering interesting points made by the subjects, and becoming familiar with the data. During phase two, we

conducted two stages. The first involved reading through each transcript again, assigning descriptive codes to blocks of text to categorise them and group related subjects together. Many of the codes at this stage reflected topics, and could have been predicted from the list of questions. In the second stage, the descriptive codes were reexamined and inductive coding was used to identify ideas and common experiences within the data. This involved recoding multiple items, eliminating single-item codes, and beginning to think in terms of groups of codes. These codes were generated from analysis of the data; most could not have been predicted from the list of questions. In the third phase the codes were further reviewed to generate themes. During these phases, and throughout the work, notes of the process were kept in an ongoing journal. The original transcripts were frequently revisited to ensure that the participant's statements were reflected as accurately as possible. Copies of the NVivo project were made after each block of work so that previous thinking could be revisited where necessary.

The fourth phase involved reviewing the themes to assess whether they were coherent, high quality and informative. They were reviewed against the original data and against each other. RTA entails constantly revisiting and considering the data in an iterative way. Some codes were revisited and sub-themes were moved, merged or removed during this phase. During the fifth phase the thematic framework was analysed. Themes and sub-themes were written up. Completing the final report is the sixth phase of RTA, although writing was ongoing during the previous phases.

Throughout the paper we use quotations from interview transcripts to illustrate the themes we discuss. These quotations were lightly edited for clarity. For example, we removed repeated words and phrases, and filler words such as '*um*.' Shorter quotes, and quotes within block quotes, are in italics, '*like this*'.

> Longer quotes are in block quotes, like this.

8.4.3 Ethics

Since the interviews involved research on human subjects, we obtained approval from our institution's ethics committee. Participants were advised that doing the interview was entirely voluntary. They signed a consent form and at the beginning of the interview they were reminded that the interview was being recorded. All interview data was

transcribed and anonymised. Names of participants and organisations were removed from the transcripts, as was any other identifying information.

8.5 Results

Management priorities are an essential component in determining which type of organisation a developer will work in. In the following sections we revisit the four research questions on management priorities that were introduced in Section 8.2, and attempt to answer them based on interview data.

8.5.1 RQ1: How do management priorities affect software security within organisations?

Throughout the interviews, at all participant levels, we found mentions of conflict over software security. For the interviewees, secure coding is in constant tension with other goals. How this tension is resolved depends largely on priorities set by management. In the following sections we discuss different prioritisations and their consequences.

8.5.1.1 If software security is not clearly prioritised, functionality will take precedence

A very experienced IT consultant, P20 speculated that '*maybe 10% of developers would consider secure coding as a practice*'. It should be impossible for developers not to consider secure coding if it is prioritised within an organisation. But often, secure coding is seen as causing missed deadlines, and can be ignored if it impacts the time to market. As P9 told us, '*it needs thought and it needs more time. And time is that commodity that nobody has, and the functionality needs to get out as fast as possible*.' Asked if he felt supported writing secure code, P6 gave an emphatic negative: '*I've got stuff to do and there's just never really any time and even if there isn't a backlog, someone will make one, as soon as it's free. There's just never any time to think about it.*' Several other participants, asked what the biggest barrier is to their coding securely, replied immediately '*time*', and expanded on this in similar terms. One participant, whose pseudonym we will not give here for precautionary reasons, told us: '*So right now we're*

using an application that doesn't actually have authentication. It's not like minimum, it's just not there. And it's very critical. But there's this huge rush to get the work done.'

In a similar vein, participants working in the early stage of a project told us that security would not be addressed until functionality was in place. The immediate objective was to find whether the code would work. Only then would security be considered. This has long been considered bad practice, with interesting confirmatory evidence in a recent study by Fulton et al. [95].

In some cases, insecure software is released based on management risk-reward calculations. Asked what he sees as the biggest barrier to secure coding, Vice President P19 replied '*time to market*', adding later '*Come hell or high water, they'll move that code out. They'll move that release out and then accept risk*'.

The security / time-to-market tension is pervasive in industry. By contrast, P8 had experience coding open source, and remarked that in open source '*deadlines tend to be a bit more flexible so that you don't get the "let's just get something that looks like it works out and we can fix it later".*' Several participants also raised the visibility of open source code as a positive attribute, with defects being picked up quickly (P2) and '*a community of people out there who is willing to test your software* (P5).' However, there is a tension there. Not all those studying the software have the best intentions, as described by P3: '*They either find them and they exploit them or they find them and they fix them or help fix them.*'

8.5.1.2 Software development is fast-moving and needs clear management guidance

> I think the key thing is we just keep changing the tools. Every couple of years we say, '*Oh, this is going to be different. This is gonna be better*'. Jeez I'm maybe working 30 years now. And it's just the same stuff. It's just rebadged and it's slightly different, but the concepts are exactly the same. The tool sets are slightly different and we end up in a situation where we continuously write bad software with new tools.

P20 expressed this pessimistic view, but many participants felt that software security was improving. For example, the Rust programming language is gradually replacing older, issue-prone languages like C and C++. Nevertheless, software development is a fast-moving profession. As P14 said, '*best practice changes every six months. You're*

never going to stay on top of it if you're just a general person, so you need the experts there to dig in.' One reason for the rapid pace of change is that many of developers' most common tools now reside in the cloud, where software providers make constant configuration changes. While on-prem systems are inaccessible unless a port is opened, on-cloud systems have some insecure defaults, and may be exploited unless they are carefully secured [218]. A couple of interviewees felt that the cloud infrastructure set up by their organisations was so secure that '*it's easier to not misstep*.' However, several mentioned the risk of accidentally exposing data. There are multiple configuration and security settings for many cloud systems. P7 is '*concerned about just how much time and effort you have to invest as a customer of these things into learning their nuances and how to configure them securely*,' explaining that when using more than one of them there's '*kind of a combinatorial explosion*':

> GitHub is enormously complicated, right? And Google Cloud is enormously complicated. Now suppose I want to set up my GitHub stuff so that it can automatically deploy code into Google Cloud. Well now I have to reason about the intersection or I would argue the cross product of the surface of complexity of those two things together. And it's difficult.

P15 described configuring AWS and Azure access for new team members: '*you get items 1 to 20 of 674 available security roles, and they're all called things you've never heard of*.' P3 worried about smaller organisations' ability to '*keep up with security pieces*.' Others felt there was a lack of control over cloud infrastructure: '*a lot of it you're just like, "well, we have to trust them"'* (P15).

The advent of large language models (LLMs) such as ChatGPT [197], which happened while we were conducting interviews for this study, is likely to entail a sea change in software development. Subsequent to the release of ChatGPT, we introduced an interview question on the likely impact of recent AI developments on code security. Many interviewees felt that poorly-understood AI-generated code would be insecure. The feeling that such code would rapidly appear in codebases was widespread. P5 was scathing: '*If the issue is that humans cannot write secure enough code, an AI model which does not have any further understanding beyond making it look like what a human would write. I don't yet see the value.*' Issues with developers copying poorly-understood code are certainly not new, and there has been extensive study of developers' use of

copy-and-paste [244]. P15 picked up on this, saying '*tools like GitHub Copilot are making it easier to write code without understanding what the code does, which is just the same copy-and-paste repeated at scale*.' A few interviewees thought that use of AI-generated code by novices would put additional pressure on senior developers during code review. There were concerns about developers using LLMs as a learning tool to gain understanding, without appreciating that the information they received could be wrong. Experienced development lead P14 commented: '*I'd be amazed if it's not happening, to be honest*.' P11 said, '*I worry about it a lot*.' Without a good understanding of the possible issues, senior management could get caught up in AI hype and fail to control for the risks.

8.5.1.3 Even when software security is prioritised, compliance may be prioritised over security culture

Participants working in industries subject to regulatory constraints, such as banking, had detailed software-security processes in place. However, some described aspects of secure coding practice in their organisations as '*a box-ticking exercise*' (P14). For P20, the box-ticking came from development teams, engaging late with security teams to tick a box. This suggests that software security culture is poor. Security should be under constant discussion [284].

For P14, the box-ticking was on the management side, providing Continuous Integration (CI) functionality and security tools without a security team to explain and support the process. He felt that '*teams are getting flooded with [support tickets]*,' automatically generated by the CI process. In the absence of a supporting security team '*a team of 6 or 7 people have to become experts in everything, and nothing to do with what actually they're trying to build*.' He added that '*you don't know how to defend against it*' and attempting to understand issues creates '*cognitive load*'. This illustrates one of the problems with moving from traditional, manually-evaluated security gates to automated security testing. The test results can be time-consuming and incomprehensible to developers if security expertise is not available. Without in-house experts to explain and advise, the opportunity to build a security culture is lost. P5 observed that disputes over third-party CVEs flagged as issues have '*eaten a bit of time in the past*', while P3 said that on first introducing security tools '*you're going to be overwhelmed with observations and things that you*

need to fix.' Other participants expressed appreciation of automated vulnerability checks but specified that the system must be well-maintained.

Another example of a poor security culture was a plea for help from P10. His organisation mandates secure coding and includes it in his annual goals where he gets '*kudos for at least thinking it through*', but when asked whether he used a written security checklist, he replied to the first author: '*I would ask you a question, are there resources to help me get started on that path? That's kind of what I think's lacking, you know?*'

However, even where there is a compliance mentality, secure-coding regulation can have positive outcomes for developers. P7 pointed out that if compliance is required it gives developers a good argument to push for increased software security practice and resources from management. P16 learned best practice for storing user passwords during a mandated code audit conducted by a '*super security nerd [...] and the good thing was that that nerd was very kind of accessible [...] and we could nerd it out then. And he was also willing to explain things that I didn't understand, and didn't just brush me over*.'

8.5.1.4 It is difficult to impose security as a priority when outsourcing software

Where a tight budget is prioritised, there can be a tension between the need for secure code and the desire to outsource coding to save money. This is a real issue for small organisations where, according to P16, '*the third party is kind of a volatile thing because yeah, they don't really care*.' It is also an issue for large organisations; P19 felt that an offshore model:

> certainly has pros on the balance sheet, but has cons [...] there's probably a drop off in the quality of the security that's getting baked into these products, which then leaks more security vulnerabilities and anomalies into the code that gets propagated to our production environments.

8.5.1.5 Prioritising software security can have an adverse impact on usability

There are other tensions that affect secure coding. Security can affect usability, for developers as well as users. Several developers described usability issues with Dependabot, a GitHub tool that prompts users to update third-party libraries, that were caused by a security improvement in the tool. P7 discussed a new security level in a mobile operating system that forced developers to jump through hoops to implement other functionality,

saying, '*opting into that level of data protection is very challenging for the developers of, say, the photos application.*' P16 used the term '*security theatre*' to describe new security bars for mobile devices, which he said had inadequate explanation and support for app developers.

Several developers described situations where rigid rules were unenforceable, did not work and wasted time. Regarding secure coding policies, P15 said that '*it was too easy to write bad code that ticked all the boxes*.' Policies on updating dependencies were widely seen as too all-encompassing, and were ignored by more than one participant. Questions on security tools raised many of the familiar responses [201]; they are too complex, slow and unreliable. P3 remarked that there can be pushback when tools suggest changes because '*there's a lot of ego around code*.' Too many alerts from security tools can cause developers to ignore or abandon them. Or, as P12 told us, '*Those that we knew how to fix, we did. Some others we suppressed [...] Of course, maybe some of them are not really false positive.*'

An app developer told us about a workaround he used to avoid mobile app security restrictions. Concerned that, as a result, his program could be an entryway for hackers, he consoled himself by reflecting that it is not widely used and '*not everybody uses it in root mode*.'

8.5.1.6 Where security is not a management priority, developers are lost, with no guidance

An indicator of the importance of the priorities they are given, where there is a complete absence of support, developers may abandon secure coding. P1 told us that '*in my professional working environment, the extreme lack of security is what led me to on my own, not really attempt anything of the sort.*' Developers do not always understand security-related events and are not always good judges of whether their code should be written securely. Many of them stated that they suffered from '*a lack of knowledge*' (P5, P10, P2), '*[lack of] knowledge of what the vulnerabilities are and really how they manifest in code*' (P11). Asked whether he tried to code securely, P15 replied that '*I think every developer in the world would say yes to that question. And the value in the question is understanding what they believe the question means*.' Some of the interviewees are largely unaware of secure coding practice, but are also unaware of their lack of knowledge. They have latched on to one piece of advice they have encountered

(for example, to use a modern language, or to use exception handling) and believe that by following it they have secured their code.

Participants experienced a lack of training in both universities (P7) and the workplace, where, as P6 said, '*if you don't have that particular guidance from your mentors [...] it's really hard because there's a lot of jargon and it's hard to piece together*.' We asked developers whether they had been offered secure coding training. Several had not. For others, training was theoretically available via online videos from generic suppliers like Pluralsight or LinkedIn Learning, but it was not mandatory and there was no time to take it. Some participants mentioned the availability of on-the-job training and learning opportunities in their workplace. P11 explained that security training in his organisation had previously entailed advice on phishing and other non-coding topics, but was improving and '*developers are finally getting some real explanations about what's vulnerabilities and how to fix them*.' Questions about barriers, and potential aids, to code security, revealed a widespread desire for more training and support.

Guidance outside the workplace, from the broader industry, also came in for criticism. P16 discussed trying to independently research good security information, including via Stack Overflow, saying '*the information is very kind of spotty [...] the security information that needs to be known is not concisely found. It is so spread out and often so hidden...*'

Participants often displayed uneasiness when asked how they checked whether their code was secure. A couple of them obfuscated, reverting to describing security precautions they take during development, such as coding in a modern language like Rust that is less susceptible to vulnerabilities than C or C++. Developers do not always have a good grasp of their own security posture. P7 explained '*in software development, you make trade-offs in order to get to your MVP or to 1.0 or something. And so sometimes defence-in-depth security stuff or that kind of rigorous analysis that would let you have an idea of what your security posture is, will get deferred*.' Others listed possible ways of checking code, but without claiming to have used any of them. P6 won the honesty prize:

> Probably the most straightforward answer that I can give is that most of the time I don't [check whether the code is secure] it's fun to talk about and it's fun to think about [...] but it's not a priority. So I guess if I were to give you a rating like 1 to 10: really the bare minimum, like 3 or 4 in terms of safety.

8.5.1.7 Where they have not been told to prioritise security, developers often defer to others

Developers who had not received software security as a priority tended not to think about it, or to defer to others. P6, for example, said that '*I'll have to be honest with you, during my daily life in my job, that's not really something I ever think about [...] it's just sort of in the background. And I hope that third party libraries take care of it for me*.' P4 said '*we do have within the engineering organisation an enterprise software group [...] I'm guessing that they think about security a lot more than I do*.' Others rely on audits, bug bounties, on CI security checks, or on secure defaults in infrastructure-as-code scripts to check the security of their code. One participant discussed security improvements to the configuration of SQL servers in their organisation, noting that security changes not targeted at software security can nevertheless reduce the risk of threats such as SQL injection.

P4 and P10 had experienced the recent creation of a security team in their organisations, and felt an increased expectation that they would code securely. P20 expressed a familiar [276] cynicism about security teams: '*the reality is when the security team get involved, there is an element of they're coming to stop you as opposed to engage and collaborate*.' The security teams in the interviewees' organisations did not seem to be very helpful. P10 spoke about reaching out to them. He was highly motivated to code securely, but seemed lost, saying '*the IT security team I think has other bigger things at the moment but just giving some direction on, on where to begin*.' P4 complained that the security team in his organisation did not address an issue he raised. P6 found them so slow and red-tape bound that they were effectively a hurdle to secure coding.

8.5.1.8 Where software security is prioritised, developers have the resources to code securely

Developers who worked in organisations where they felt secure coding was appreciated talked about how those values were reflected within the work environment. We asked developers whether secure coding requirements got in the way, and many of them felt that they did not, with P3 saying '*I would say no, because that's part of the job [...] if you think they're getting in the way, you're probably not a great programmer*.'

In environments that are positive for secure-coding, some developers told us that there was enough time available to code securely. This was reflected in their being able to put thought into implementing their own security checks, such as adding extra logging, or implementing static analysis checks and security code reviews. P12 had found multiple issues via manual fuzzing and was working on automating fuzzing for all new code. Others felt that resources such as training and time were likely to be available if requested. Two developers identified this explicitly as being a result of support from upper management, with P4 saying that top management at his organisation were '*very receptive*', and P14 telling us that '*the whole push on software security has come from the very top*.'

P7 was happy with his organisation's security culture, saying '*to brag a bit, I think everyone on my team also cares deeply about this sort of thing, so it's rarely a particularly difficult battle*.' Many participants felt that secure development emphasis in the work environment improves understanding. Reasons given included learning from tools (P12, P17), and from peers. P3 expressed this well: '*in companies that encourage security to be front of mind, there are more people who care about it, so there's more opportunity for discussions about it and people to bounce ideas off of and learn more*.' Working in an atmosphere that values software security improves developers' software security interest.

Developers were positive about work environments where secure-coding training was given, or was explicitly available. P4 discussed how the security team in his organisation were in contact with their third-party training provider to improve training content.

Other positives in the work environment included having good secure-coding values but an absence of blame culture, as P15 described: '*everybody involved, I think, appreciates that there are too many variables and too many moving parts for "relying on a person not to make mistakes" being a reasonable security policy.*'

8.5.1.9 Some engineering sections cultivate software security expertise without apparent management direction

P3 summarised the range of organisational approaches to software security as he saw them:

> So I'd say, there's probably maybe three groups of organisations. There's ones where nobody cares about it at all. There's the ones where it's really

> only a topic within the engineering organisation, and it kind of has to be self-motivated. And then there's the ones where the entire company is aligned.

P7, asked whether a security drive was valued, said that '*a knowledge and caring about security and cryptography is not a requirement, but certainly something that stands out when we're interviewing people [...] it's something we value very much in technical staff.*' Others had observed secure coding being part of the appraisal process, or had seen that '*amongst the technical staff, it's generally considered a positive*' (P3).

The danger here is that if the drive for security is not prioritised and supported, it will not be resilient. Asked how being a security driver manifested, P11 said '*trying to modernise practices, trying to get people more informed, get processes in place that allow for these practices to be in place even when I'm not part of the organisation any more*.' When the motivated developer leaves, good practices may fall by the wayside.

8.5.1.10 Some developers are self-motivated to code securely regardless of management priorities

Asked about their motivations, security-aware developers often equated security and quality, or said that they just want to do a good job. They are motivated to produce code that '*is as good as it can be on all aspects* (P11),' and they do not want to have to deal with the technical or reputational fallout that a breach could cause. P15 gave a graphic description of the consequences of discovering a security issue:

> There's been a few incidents [...] there have been occasions where something went wrong and you suddenly realise "*this could potentially be quite bad*." And we don't know how long it's been bad for and then follows a period of waiting where you fix the defect, you notify the people responsible, and then you wait. And you wait and you wait. And when after 18 months, you haven't seen any articles in The Register about it, you know, like, alright, maybe that one was okay, Maybe we got away with it.

A couple of developers mentioned peers whose security enthusiasm sparked interest. Asked whether they considered themselves software security drivers in their organisation, several responded that they did. P15 had previously encountered an individual about whom he thought '*You're in charge of this now because you know way more about it than*

I do,' but in general felt that he was the one reminding everyone to check security and access. P3 pushed security in code reviews and at architecture meetings, and introduced static analysis tools to the build.

8.5.1.11 Developer security wish lists reveal a desire for greater security prioritisation

When asked what changes would improve secure coding in their environment, developers' answers frequently reflected inadequate security prioritisation and budget. P7 argued that security tools should be free and widely available. Several developers mentioned training-related aids, for example '*better tools and documentation for configuring software as a service*' (P7), '*an awareness of what [code analysers] do and why one would use them*' (P13), '*more time, possibly better training available*' (153), and '*resources, but guided*' (P6).

Several interviewees advocated for having a security team. Asked what improvement would help with secure coding, P13 wished for '*having a security officer attached to the project [...] that would be a very good thing if there was someone to worry about all this for me*.' Working in an environment without secure coding support, P6 was aware of what was missing:

> We have a faint idea of what security means [...] but in order to shift from that to actually having really secure testable, in some sort of environment being able to test the security, I think it would have to come from above. They would have to say '*you need to have better security in your applications, and this is the task that we're giving you rather than some afterthought*'.

Similar ideas came from P9: '*I think security must be enforced through some sort of procedures like periodically or part of the day to day activities [...] it has to be enforced before it becomes part of the developer's self*.'

8.5.2 RQ2: How well do C-suites understand software security?

8.5.2.1 It is often assumed by the C-suite that software security is in place

Software security is frequently described as a non-functional requirement like efficiency and maintainability. There can be an assumption that it is present, without the corresponding clear direction from management [267]. When security fails it often fails invisibly.

P15 said: '*the biggest problem with security is that nothing actually crashes when it's broken*,' adding '*you can check the locks on all the doors that you know about, but you can never be 100% confident that there isn't a hole in the wall that you missed*.'

In many organisations, management is optimistic about code security, and assumes that secure code is the default. P13 attributed this to its invisibility:

> From the perspective of the business, it's a non-functional requirement. It's like when people want a new office building and they talk about the number of desks and the lighting and how much natural light there is and these things. But they don't talk about the toilets. They expect toilets and running water to be there.

P20 used a metaphor that emphasised how management expect software security to not only be there, but to be thoughtfully engineered:

> If you're driving a car, you're going to assume that the engine is sound. You're assuming that it's manufactured to a certain standard. You're going to assume that this is all in place. It's not. We don't engineer software, we develop software and it's not as rigorous a process as people think it is.

8.5.2.2 The C-suite's understanding of software security depends on their background

Given that support from top management would help to facilitate better software security, does top management understand what is needed and how to supply it? Software security expertise in the C-suite depends on their background. P4 felt that leadership in his organisation were only '*very peripherally aware of [software security]*,' being '*more from the insurance domain*.' P3 said that in one organisation he encountered, '*senior leadership were journalists in the food space. It would be unfair to expect them to have any experience whatsoever in software security*,' adding that '*part of being in senior management is not knowing all the things that go on underneath you [...] they can't know everything*.' While this is true, there is undoubtedly a benefit to having a rudimentary understanding of issues which could become a systemic threat to an organisation's existence.

P18 felt that management should '*give a level of importance to it at a higher level and then obviously allocate resources after that, so understand what the risk is*.' P21 felt that

'*it does warrant some focus in and around C-suite tables, to the extent that I have not witnessed it in my last 15, 20 years*.' P19, working for a large multinational, felt that things were improving: '*the C-suite is getting much smarter. They are starting to bring in stakeholders and C-suite execs that have a security mindset*.'

8.5.2.3 C-suites often do not distinguish between cybersecurity and software security

In management teams that P21 has encountered, software security '*just doesn't feature as part of the discussion*.' P20 said that '*I would say a lot of people are focused on monitoring and detection as opposed to prevention and software. I just think it's easier to detect and monitor than it is to build in*.' P6 described an over-reliance on IT security that may have caused management over-confidence resulting in low secure-coding emphasis, saying that '*as soon as anyone gets access to this VPN [...] they have the keys to the castle*.' Even where senior management are aware of software security requirements, they may not prioritise them. A challenge for management is that software security needs people, and software security experts are not only expensive but are also in short supply. P18 was concerned that while there are many security recommendations, to act on them his organisation needs '*champions*'; '*people who are able to fix things, to be able to implement the changes*.' The need for more people with the requisite security expertise was reiterated by others.

8.5.2.4 Top management may ignore software security issues, and may even subvert security practice

Determining the priority given to software security within a specific organisation can be difficult. There can be inconsistency between management's public posture and actual software-security activities in an organisation. P14 has experienced issues with confusing CI loops and automated security tickets, and said '*the problem is of course when the effort is there, then they can say, "Oh, we're so secure. We've implemented X, Y and Z", but on the ground is it really?*' P6 was forthright when asked if their manager cares about software security: '*I think if you asked them that, then they would say yes, but I'd probably have to say no*.' Later, he suggested that managers would '*put on their biggest smile*,' but '*it's just not really anyone's priority*.' P3 felt similarly: '*Obviously no one's*

like, you know, twirling an evil moustache or anything like, "we don't want any security" or anything like that. But the priority is always features.'

We encountered one interviewee who had seen management try to circumvent their own stated policy on secure coding:

> I can think of a couple of incidents where somebody publicly has to say that "*This is our commitment to security*" and then privately afterwards will be like, (whispers) "*Do you need to do that?*" And you're like, "*Well, you just said we were taking this seriously*". And they're like, "*Yeah, but you know what I mean*".

We were entrusted with the description of a security workaround that we cannot describe in detail here lest the participant be identified. The essence was that there were careful, audited precautions taken to protect customer identity for sensitive data. These precautions were, however, silently bypassed at top management's request. A secret spreadsheet of the sensitive information was maintained on the network, to facilitate easy processing by secretarial staff.

8.5.3 RQ3: What factors are likely to motivate C-suites to pay attention to code security in organisations?

8.5.3.1 Increased visibility and understanding

Given the importance of allocating sufficient resources to software security, several participants talked about raising the visibility of software security at board level. P20 said, '*it's not a technology risk, it's a business risk*,' comparing it to natural disasters. P8 proposed having people '*who had experience being developers*' on the board. P1 thought that managers would be more motivated towards secure coding '*if they truly knew the risks, and deeply understood that any open hole will be passed through*.' P21 envisioned an '*executive management dashboard*' to monitor not only software security but also software quality. Similarly, P15 recommended making security metrics more visible and following '*principles of good observability*.' P20 suggested tabletop exercises with leadership to raise awareness, as did P17: '*it's part of an education for them*.' P20 proposed case studies, about which P18 had this to say: '*these are already working for cybersecurity, but I don't think software security stands out so maybe more along the*

lines of, from a business perspective, what companies have lost from not building in security by design into software?'

8.5.3.2 Internal and external security breaches

With experience in senior management, P19 and P21 felt that an internal breach would definitely push software security further up the management agenda. Security breaches have been cited in other research as motivators for increased software security [13]. Interestingly, those of our participants who had experienced a breach such as a ransomware attack within their organisations described it as a trigger for increased *cybersecurity* rather than *software* security (P10, P6, P20). However, breaches in external organisations appeared to initiate a spirit of enquiry amongst management, in the experience of some interviewees. P15 told us: '*I remember my manager, the managing director at the time, we'd been trying to get him to take security seriously. He came over to my desk and he's like this to me: "Could that happen to us?*" P3 had had a similar conversation with management, while P14 believed that the Log4Shell [133] incident '*either - it depends on your perspective - helped or didn't help*.' Others quoted the NotPetya ransomware's effect on Merck [66], increased federal scrutiny of US-based banks and their security controls, and '*how expensive it can be for competitors to get hit with vulnerabilities*' (P11) as motivators for increased software security interest.

8.5.3.3 Meeting client expectations

P19 mentioned meeting client expectations as a software security driver for senior management. P12's organisation works on security software, and needs to show that their own software is secure to earn customers' trust. P18 gave client expectations as the top reason for prioritising software security, followed by legal and reputational concern, which he saw as linked.

8.5.3.4 Standards and regulatory compliance

Compliance was seen as a good motivator. P15 used the payment-card PCI-DSS accreditation process as an example of successful compliance requirements, saying '*these people actually know what they're doing. And this is clearly based on experience of breaches and problems over many years*.' It is sometimes argued that an emphasis on

standards or regulatory compliance can result in a failure to focus on software security itself [239]. This can be an improvement on a state of zero security, however: '*[auditing requirements] are not always the best, but at least it starts a conversation about it*' (P3). P5 had seen this process in action: '*compared to [15-18 months beforehand] there's been more and more interest [in software security], partly because of regulators*.'

P3 warned us that he was going to be quite philosophical, before replying to a question about the best possible improvement to support him in writing secure code:

> People used to build warehouses that didn't have fire escapes. And people started dying because of that. And it became painful enough from government regulation, from insurance companies and such that if you wanted to build a building that didn't have a fire escape, your insurance would be so expensive that you couldn't run a business. So building the fire escape was now cheaper. I think the software industry has to get to that point where building insecure software becomes so expensive that it's worth investing the money to actually make it secure. And unfortunately, I think that's the only way that it's going to get done in the long term.

He elaborated on the problems with the current risk landscape for organisations:

> It's very hard to quantify what the risks are of insecure code. And at least in the modern world, insecure code hasn't been all that expensive. We may look at some of the big breaches, you know, fine, they pay for credit reporting for 100,000 people for a year. It's just that that cost is cheap enough that it's not worth trying to fix the underlying problems.

P20 mentioned that it is easier for organisations to fix software security after a vulnerability is found than to build in defence in depth, also saying '*I think if you're relying on good practice without any validation or audit, it's not going to happen [...] there has to be consequences*.' Several interviewees work in regulated environments, and described how regulatory compliance takes resources, and is therefore in tension with other organisational goals. P17 explained the complex legal requirements in the US and Germany, among other countries, for organisations that are considered part of critical industry. Discussing Germany's BSI-Act, which regulates KRITIS (critical infrastructure) [87], he said that '*you need security tools to evidence, and you need secure software development practices so that you're not showing really bad numbers*.' Over years of dealing with

acquisition candidates, he has found that '*where we've seen really good governance on software and infrastructure security, a big driver is the regulatory requirement*,' singling out GDPR where penalties can amount to 5% of global revenue. This sentiment was echoed by P19, who noted, however, that regulations such as SOX and GDPR do '*slow down time to market*.' P15 introduced a resulting concern:

> Software and the companies that rely on software have this expectation that you can do amazing things very, very quickly and saying to everyone, '*no, no, no, we need to slow this down. We need to start certifying websites the way we certify aircraft or we certify medical devices*.' You know, while you were there doing your paperwork, some competitor from an unregulated territory would run rings around you and steal all your customers.

8.5.3.5 Greater legal obligations

Asked whether legislation around software security would increase the focus on it, P21 responded: '*100%!*' Others, including P6, also felt that changing the financial implications of insecure software would help in the security battle. P12 introduced another real-world simile: '*eventually it will have to appear like it is engineering, like the bridges used to collapse all the time in the 18th century, and now it is quite rare*.'

Organisations are not people, and regulatory consequences or punishments for businesses do not always affect management motivation, as observed by P16:

> If someone finds out that you did it wrong, then you're in trouble [...] but of course, that doesn't work with law, with legal law. If you are big enough, you can always drag it out [...] the CEO or the COO or CTO probably is in the end responsible for it. They usually earn so much money that they say just, "*Oh, I'm already a multimillionaire. F*** it, I'm just quitting*." And then you have to go legal after them and then the damage is already done.

One of our respondents had a solution for this when asked what developments he thought would push software security higher up the agenda for C-suites: '*Follow the money. Follow the money. Or jail time*.'

8.5.4 RQ4: What factors have participants encountered that had a significant effect on software security in an organisation?

8.5.4.1 Good software security practice is expensive in terms of time and money

Good software security is expensive, as P15 reminds us:

> I think there is this Utopian ideal that some armchair keyboard warriors have like, "*well, no, you just don't let the boss decide. You just make it secure.*' Yeah, that's not how real companies work, you know? It's like "*We don't have any money to pay salaries*." "*Why not*?" "*Oh, because the software team was busy doing security instead of building anything we could actually sell for the last six months.*"

Coding securely day-to-day can often be relatively cost free once developers are familiar with good coding practice. But other aspects of security are not cheap. P11 told us that in a previous job '*we were encouraged and even incentivised to try to break our own systems from a security standpoint. It's better if we find and break that then if an attacker does that, but I understand, it can be expensive to run a team that way.*' In addition, most excellent security tools are costly. For example, P17 discussed obliging acquisition candidates to submit code to a security scanner, adding that '*this is a top tier tool and the companies typically don't have access to this level of tooling. We often uncover things for them that they're surprised by, and actually, it's of huge value to them.*'

8.5.4.2 Smaller organisations may have fewer security practices in place

Some participants felt that organisation size had an effect on software security practice. For example, P21 described restrictions on open-source libraries but added the caveat '*I would say that's in the large [organisations]. The small to medium; I doubt very much you'd see it there*.' Also discussing use of open source, P15 said that:

> It comes down to a pragmatic understanding of the probability of a vulnerability, the probability of the vulnerability actually getting exploited, the fallout, if it was to do so, and the cost of mitigating against that [...] "*We'll take the risk*." I've not really heard any organisations say it that succinctly, but I think that's the attitude that a lot of certainly startups and smaller places take, if security is not their core business.

8.5.4.3 Financial constraints, downsizings, acquisitions and divestitures may affect secure coding practice

This belief that smaller organisations are more likely to take software-security risk appears to be partly based on the cost of doing software security well. Perhaps this is why a number of interviewees indicated that a change in organisational status or finance can affect the software security of an organisation. P15 discussed how the difficulty of fixing security in a newly-acquired organisation can mean that it never happens. He also talked a little about organisations going bankrupt: '*I was working at [redacted] when they were placed into administration [...] the impact there was every single person who knew how the system worked was told, "give us your keys and go home*"'.

Asked whether organisational turbulence affects software security in his experience, P3 discussed how, the more institutional knowledge is lost, '*the more likely you are to make mistakes or to break assumptions that might have been part of the security posture of that software from before*,' adding that organisational turbulence '*makes it significantly harder*.'

P6 had experienced a period where a lot of people left his organisation, and he felt that there was an indirect effect on security:

> It sort of reinforces the idea of "*there's so much stuff to do and so little time to do it that I have to focus on getting this working. And if it's not secure, no one is even going to know who to blame. So I'm not going to focus on it*."

P5 described a downturn in software security during a period of organisational turbulence that ended in a downsizing: '*there was a lot of pressure to [...] "we just need this by this date, it has to work and it doesn't matter how you make it work*."' Downsizing can also result in outsourcing, which can have adverse security implications as we saw in Section 8.5.1.4.

P17 had observed a diminution of software security concerns in divested organisations, and an increase in acquired ones. He had considerable experience of assessing software security in organisations that were due to be acquired. In one case, he found that the target organisation had a higher level of software security than the acquirer, but that was very unusual:

> Companies being acquired, in my experience, fall into two categories. One: Financially, they're not doing well, or two: they know exactly what they're doing and they're sizing themselves for acquisition. And in both cases, you're talking about reducing costs, rationalising, only having the people you need to serve your consumers [...] So the organisation that devs are operating in when they're being acquired is often budget restricted. For different reasons, but it's the reality of it. So [...] we know they're not going to be in great shape, typically.

Asked whether he had encountered sudden increases or decreases in security, he zeroed in on financial constraints:

> I think companies that are going through any sort of a financial crunch, or that they're just being shaped for sale, can impact security. I think the financial crunch is more severe [...] because some of the first things that get cut: training budgets, R&D, security is close behind that then [...] just keep the product afloat, just keep the company afloat, just fix what we need to fix so that the customers keep paying us money.
>
> And even regulatory fears start to recede if the company is non-viable, right? Because it's like, what are you protecting? I think the idea of protecting your customers and your information and your PII or PHI, you know, it starts to become secondary. Folks know as well it's coming, so some of your best talent might walk. Your more senior people especially.

8.6 Discussion

The results of our analysis are listed in Table 8.2. Leadership priorities set the priorities for an entire organisation. A failure to visibly prioritise a specific aspect of an organisation sends the implicit message that it is not a priority [11]. Addressing RQ1, we found that management failure to prioritise software security can have adverse implications. Security checks can be skipped where there is pressure to ship. Vulnerable software can be shipped and added to a risk register, or indeed ignored.

Table 8.2: Answers to research questions

Research Question	Findings
RQ1. How do management priorities affect software security within organisations?	• If software security is not clearly prioritised, functionality will take precedence. • Software development is fast-moving and needs clear management guidance. • Even when software security is prioritised, compliance may be prioritised over security culture. • It is difficult to impose security as a priority when outsourcing software. • Prioritising software security can have an adverse impact on usability. • Where security is not a management priority, developers are lost, with no guidance. • Where they have not been told to prioritise security, developers often defer to others. • Where software security is prioritised, developers have the resources to code securely. • Some engineering sections cultivate software security expertise without apparent management direction. • Some developers are self-motivated to code securely regardless of management priorities. • Developer security wish lists reveal a desire for greater security prioritisation.
RQ2. How well do C-suites understand software security?	• It is often assumed by the C-suite that software security is in place. • The C-suite's understanding of software security depends on their background. • C-suites often do not distinguish between cybersecurity and software security. • Top management may ignore software security issues, and may even subvert security practice.

Continued on next page

Table 8.2 *Continued from previous page*

Research Question	Findings
RQ3. What factors are likely to motivate C-suites to pay attention to code security in organisations?	• Increased visibility and understanding. • Internal and external security breaches. • Meeting client expectations. • Standards and regulatory compliance. • Greater legal obligations.
RQ4. What factors have participants encountered that had a significant effect on software security in an organisation?	• Good software security practice is expensive and time-consuming. • Smaller organisations may have fewer security practices in place. • Financial constraints, downsizings, acquisitions and divestitures may affect secure coding practice.

Failure to prioritise software security can have a negative effect on developers. They may feel lost and lack the training and support needed to code securely. Even where software security procedures are adopted for compliance reasons, if security culture is poor developers may remain uncertain and confused about what is required. By contrast, where software security is prioritised and resources are made available, developers report having better learning opportunities and increased understanding of secure coding. Asked what would help them to code securely, developers repeatedly expressed a desire for better training and guidance. They also mentioned that prioritisation by senior leadership would help them to prioritise software security in their work. Some participants were self-motivated to code securely, but an occasional lone developer passionate about secure coding is not enough to change company or corporate culture.

Where software security was prioritised, some participants reported usability barriers to easy development.

When answering RQ2 we observed that senior leadership may lack the background and education to appreciate software security concerns. In some cases, the C-suite is oblivious to software security, assuming that it is engineered into software. More well-informed management may struggle to recruit staff with the necessary security expertise. A few interviewees had encountered management that was happy not only to ignore software security requirements but to actively bypass audited security precautions.

Answers to RQ3 broadly corresponded to received wisdom about software security motivators for senior leadership in organisations. Interviewees felt that increased visibility for software security at senior management and board level would increase its prioritisation. Some had experienced an increase in software security interest from management when external organisations experienced a costly breach. Client expectations were also mentioned as a driver. We see from the results that there is a large component of regulatory compliance in the drive to software security in organisations. In certain industries that are designated '*critical infrastructure*' in the US, of which there are currently 16 including the energy, communications and chemical sectors, there are existing regulatory constraints on software security [295]. Other industries such as the automotive and payment card industries have self-imposed standards with which participants must comply. There was universal agreement that standards and legislation that require software security increase the likelihood that software security resources will be allocated by senior management. There are potential issues with introducing regulations to increase software security, such as the danger of losing ground in the race to develop new software. However, while a speedy release of insecure software may benefit the organisation's bottom line, it may also damage customers and even society as a whole.

One participant distinguished between organisational and personal incentives, pointing out that C-suite members can jump ship if things go wrong. Another expressed that the two highest motivators for senior management are money and jail time. This is particularly interesting since in November 2023, two months after the interviews concluded, the US SEC charged the CISO of SolarWinds with fraud and internal control failures [15]. Although cybersecurity related rather than software-security related, this sets a rather worrying precedent for C-suites and boards. On December 1, Eric Goldstein from the Cybersecurity & Infrastructure Security Agency (CISA) in the US made a widely-reported speech exhorting technology providers to '*"take accountability" for the security of their customers by doing things like [...] making security logs available, using secure development practices and embracing memory safe languages such as Rust*' [309].

While exploring RQ4 we noted some participants felt that organisation size could affect security priority, a view that has not to date been reflected in research [13, 111, 239]. Being acquired can improve software security in an organisation as it is brought up to

the standards of the acquiring business. We also observed that limited funds, downsizing rounds, and a financial crunch can cause a decrease in software security practice within an organisation. Even regulatory constraints become ineffective when an organisation is no longer viable. This is an issue of concern, since the organisation may be shipping insecure software to unsuspecting customers. Software security resilience in times of organisational turbulence could be a fruitful research topic.

One of our participants, who once managed the response to a breach that was globally reported, told us '*I've seen a lot. I'll hopefully never go through anything like that again.*'

8.6.1 Limitations of this study

We are aware of a number of limitations of this study, which we discuss next. First, the findings of this study are not generalisable in the traditional sense that is more common in survey or other quantitative studies; nor is that the goal of qualitative inquiries such as this study. Instead, we sought to shed light on hitherto unexplored questions to develop a better understanding, which may help towards theoretical generalisability. Instead of traditional quality criteria for quantitative studies, we adopt Guba's criteria for qualitative inquires [106]. The sample of informants was purposive, which means we sought to interview informed participant who would be in a position to present relevant insights. This helps to establish transferability of the results. We assessed credibility by triangulating across different informants. Establishing an audit trail through recording and transcribing the interviews helps to establish confirmability of the results.

8.7 Conclusion

The priorities set by senior management within organisations are a large determinant of the priorities at other levels. We interviewed 21 professionals working in or consulting for organisations that develop software, to investigate the influence of C-suite software-security prioritisation on software practice. We found that where software security is prioritised by leadership, developers feel supported to code securely and confident in their abilities. Where it is not, developers seem to be lost and in need of secure-coding training, time and support.

We explored current C-suite attitudes to software security, finding that many senior managers do not have the background to understand it. Some prioritise it, but a few pay lip service to software security while ignoring or undermining it. We examined the motivations that cause organisational leadership to prioritise code security, finding that education, breaches in other organisations, and compliance and regulatory requirements feature highly. We explored what eventualities in organisations tend to cause a shift in software security, finding that downsizing rounds and funding constraints tend to have adverse software-security effects.

9 Conclusion

The introduction to this thesis in Chapter 1 discussed the overarching theme of software security within organisations. It outlined the structure of the research and aspects of the methodology. Chapter 2 was the motivational chapter, providing some reasons why releasing insecure software is bad, not just for organisations and their customers but for society as a whole. Chapter 3 introduced the Developer Security Archetypes, a typology useful for describing developers' software security interest and the software security prioritisation in their environment. While examining and absorbing existing research, I published a description of 25 unhelpful assumptions occasionally made by researchers, that will now be easier for future researchers to avoid. This can be read in Chapter 4. Using a large multinational survey, I established in Chapter 5 that many organisations ignore software security entirely, or tolerate a poor security culture. Chapter 6 showed that small and isolated developers engage in fewer software-security practices and tend not to use secure-coding tools. This should be of concern to both industry and academia. Chapter 7 provided evidence that secure-coding training is not widespread, which means that developers are inadequately prepared for a world where most software is constantly under attack. Chapter 8 provided increased insight for the research community, for industry and for policymakers on what the insides of organisations look like when it comes to software security, and what senior managers' software security drivers are. Finally, this thesis highlights the urgency of assessing organisational turbulence such as downsizing and bankruptcy in a code security light. Overall, I have provided the research community with information and insights that I believe should allow us to rebalance the secure-coding focus from the developer to the organisation.

9.1 The Escalating Price of Insecure Software

The work in this thesis is in step with the ongoing software security concerns of business and civil society. Chapter 2 provided an overview of software security issues in early 2022. Geopolitical tensions have heightened since then. At time of writing, the ongoing war in Ukraine has been of absorbing interest to pundits gripped by the intersection of cyber and kinetic warfare. Cyberattack, facilitated by vulnerable software, is perceived as a way of increasing confusion and reducing governmental authority [281]. Secure-coding failings have real-world consequences. Recently, U.S. government cybersecurity experts raised the alarm on the China-directed 'Volt Typhoon' botnet used to infiltrate U.S. civilian energy and water infrastructure [250]. There is concern that in the event of invasion of Taiwan by China, which is widely considered likely in the medium term [263], sabotage of civilian infrastructure could be used to spread confusion within the U.S., and reduce the inclination to come to Taiwan's defence [250]. The Volt Typhoon operation exploits both known and zero-day vulnerabilities in '*networking appliances such as those from Fortinet, Ivanti Connect Secure (formerly Pulse Secure), NETGEAR, Citrix, and Cisco [T1190]*' [296]. In this context, it is not surprising that CISA's Executive Assistant Director for Cybersecurity, Eric Goldstein, has been calling on software organisations to release resilient, secure software [309], as discussed in Section 8.6.

In the cybercrime arena, the big story of 2023 was that the total of ransomware payouts during the year passed the one billion dollar mark for the first time [158]. Prominent contributors to this figure were zero-day vulnerabilities in the GoAnywhere and MOVEit file transfer software systems, which were exploited by the Clop ransomware gang [56]. Meanwhile, zero-days in Barracuda, VMWare and other software facilitated espionage activity in the U.S., Asia and elsewhere by actors from hostile nations [18, 145]. These and many other instances showed once again the role of software vulnerabilities in digital fragility.

The devastating human impact of ransomware was driven home in January 2024 by a report from the U.K.'s Royal United Services Institute on the effect of ransomware attacks on small-business owners and incident responders [159]. Aside from the obvious financial ill-effects, the report found that personnel involved in ransomware response may suffer physical and psychological harm. Examples included a heart attack, hospitalisation

for dehydration, suicidal thoughts and the reported suicide of an IT staff member. The devastating toll exacted by cybercrime shows no sign of easing off.

A human toll is taken by forms of cyber espionage too. A February 2024 report by the Google Threat Analysis Group [124] detailed what is currently known about the lucrative commercial surveillance industry, discussed in Section 2.4.1.3. This industry is extremely dependent on zero-day vulnerabilities. They are exploited by vendors to infect targets' phones and devices with spyware. Such software has been used to target journalists, human-rights defenders and opposition politicians worldwide. The report mentions the '*fear*' and '*chilling effect on their professional relationships*' that people experience when they discover that spyware tools were used against them.

This small subset of current events illustrates that since I wrote about online bad actors in Chapter 2, poor software security has continued to cause havoc around the world.

9.2 Overview of Studies

The common thread in the studies documented in this thesis is the importance of the organisation and the work environment in issues of software security. As a professional software developer with two decades of experience, I am personally conversant with the difficulties inherent in secure coding. In my PhD research I set out to study the question of why we developers are terrible at software security, and how we can improve. This question is not new. It has been under discussion for over two decades. My research, mediated by my professional experience, led me to appreciate the substantial role that the work environment plays in influencing developer software security prioritisation and learning. Without training and support, developers struggle to code securely. In fact, existing laboratory research shows that most developers will not even attempt to code securely unless security is specifically requested [182]. Therefore, prioritising software security in the work environment is vital. Some developers become interested in software security independently, but it is difficult for them to prioritise it at work if faced with deadline pressures and organisational indifference. The three studies included in this thesis move the conversation forward by addressing aspects of software security in the organisation. In this chapter, we will discuss the content and conclusions of these studies.

My positionality as a software developer with a strong technical background and considerable development experience was described in section 1.3.1. My position as

a software security insider influenced my research direction and the topics I chose to explore in this thesis. I believe that this position is a valuable asset that adds insight and depth to the research undertaken. Readers should be aware that I may be biased in favour of software coders. This is because, while other roles are valuable, only those writing a piece of code can code it securely. My overarching research focus has therefore been on what they need in their work environment to be able to write secure software, and on how organisations can be encouraged to provide it. I also consider recommendations for how government, educators, and the development community as a whole can help to facilitate secure coding.

Study A entailed analysis of the literature. Contributions from Study A began in Chapter 2 with a journey through geopolitical and other reasons why the eradication of software vulnerabilities is increasingly urgent. In Chapter 3, I considered evidence that organisational practice is a major influence on software security. In Chapter 4, I uncovered twenty-five unhelpful assumptions in software security, including several that have sometimes led researchers to underestimate the influence of organisational priorities.

Study B used quantitative analysis to allow me to evaluate current secure-coding practice. I devised a simple but empirically-backed way to evaluate organisational secure-coding practice in Chapter 5. Using a large survey of developers in industry, I examined what this evaluation method revealed about secure coding practice in general (Chapter 5), and coding practice amongst isolated and under-resourced developers in particular (Chapter 6). I found a wide spread of secure-coding practice amongst organisations, with some adopting all of the evaluated practices and others adopting none. Also in Chapter 5, I devised questions that can be used to detect failings in secure-coding culture. Evaluating survey answers to these culture questions, I determined that organisations that have good secure-coding practice on paper may nevertheless have security culture failings.

In Chapter 7, I presented a first study of the essential components of software security training courses for developers, and found that few developers in industry are offered satisfactory software-security courses.

Study C involved interviews with software professionals, qualitatively analysed using reflexive thematic analysis. Chapter 8 investigated the effects of poor software security prioritisation within organisations. I inquired into top management's understanding of the need for secure coding, and what factors could motivate members of the C-

suite to prioritise code security. I did a first scrutiny of the effect of organisational turbulence on software security and resilience. This is one of several areas that need further investigation.

9.3 Results of Study A: Reviewing the Literature

As discussed in Chapter 3, my initial research led me to an understanding of software security as being most influenced by two factors; software developers themselves and the environments in which they work. Putting these into a two-dimensional typology yielded the Developer Security Archetypes, which provide a useful vocabulary for discussing developers in a software security context.

Improvements in secure coding practice can come from enhancing code security focus in either of these two dimensions. For developers, this might mean attempting to improve their motivation and/or knowledge. For work environments, it could entail improving organisations' software security prioritisation and practice.

Considerable research has been done on software developers in the software-security context. Projects like '*Motivating Jenny*' and '*Why Johnny doesn't write secure software? Secure software development by the masses*'[1] focus on topics such as how individual developers relate to software security advice [154], and the personal attributes that may affect their approach [226]. Elsewhere, Assal and Chiasson explored the blockers and motivators to secure coding that developers encounter during their work [14]. Votipka et al. looked at developing a scale to evaluate developer software security self-efficacy [313]. Several researchers attempted to find good ways to intervene in the software development process to assist developers with secure coding, via workshops [319, 150], via targeted messaging within tools [188, 100, 61, 224], and by escalating warnings to the developer's team members [227].

I did not observe the same focus on the organisational dimension of secure coding support. Assal and Chiasson identified the need to '*focus on understanding organizational issues that lead to insecure practices*' [14]. In the course of an extensive literature review I identified twenty-five unhelpful assumptions in software security research, which are the subject of Chapter 4. Assumptions such as '*managers are software security champions*' and '*organisations prioritise software security*' are sometimes made by researchers who

[1] `https://gtr.ukri.org/projects?ref=EP%2FP011799%2F2#/tabOverview`

are focused on developers and teams and do not carefully consider the organisational context in which they are operating. Studies that engaged with organisations to observe software security practice in industrial settings sometimes seemed to expect individuals or teams to prioritise software security, without considering the impact of priorities signalled by senior management [219, 202]. These assumptions are not universal; other studies did identify poor software security prioritisation by management as a factor in poor software security [285, 151]. Useful work by Arizon-Peretz et al. emphasised the importance to secure coding of explicit management support and a good security culture, including time and training devoted to software security [11, 10]. However, perhaps partly due to the assumptions above, missing from the literature was an attempt to quantify software security practice and culture in industry at scale, across multiple organisations and countries. Nor did I encounter any academic literature evaluating the reasons why organisations and other software producers choose to introduce software security practices or support software security culture.

Without a generally accepted way to quantify current software security practice and culture within organisations, this aspect of developers' experience was sometimes overlooked, so that research on software developers' security actions took place in a vacuum. I observed that different measurements of practice were devised by different researchers. Some were too complex for practical reuse [175], particularly in survey environments. Others were difficult to evaluate without manual analysis [13]. All lacked an empirical basis. The research environment needed a simple, repeatable, and empirically grounded way to quantify software security in organisations and other working environments. This would allow evaluation of software security progress over time. Measurements of software security culture were also needed. Furthermore, I believed that it was essential to do with organisations something that several researchers have done with individual developers; assess their secure-coding motivation and how this can be improved.

9.4 Results of Study B: Assessing Organisations

To develop a better understanding of current secure coding practice I devised and ran a software security survey. The survey was extensive, with 1100 participants from 59 countries, of whom 962 passed screening. Survey results for questions on security practice and culture are discussed in Chapter 5. Participants' work environments were

assigned a 'CA Score' for software security practice. Based on survey results, there are very few software security practices undertaken in a quarter of organisations. Only 23 individuals, or 2.4% of all valid respondents, worked in environments where all 12 software security practices comprising the CA Score were present. Qualitative research by Assal and Chiasson [13], Morales et al. [174], Palombo et al. [202] and others had previously shown that software security is not prioritised by all software development organisations. For example, Palombo et al. observed developers ignoring authentication vulnerabilities because they resulted in loose permissions, reducing support calls [202]. The survey confirmed that poor or non-existent practice is widespread.

Chapter 6 drilled down further into the data on practice, breaking down information on secure coding practice for freelance developers, developers in small and medium organisations, non-organisational developers (possibly open-source coders), and isolated developers within organisations. Although many of the twelve common activities comprising the CA Score are in fact performed by these cohorts, their level varies widely between the groups. Most of the activities are performed most often by medium-sized enterprises. I recommended that there should be more targeted advice and standards tailored to small or medium organisations and isolated coders. I also emphasised the importance of secure-coding tools. These are not much used by our cohorts. If automated static and dynamic analysis tools were leveraged by all isolated developers and teams, there could be a large improvement to code security. With this in mind, I also advocated for an industry-wide effort to make such tools more affordable and easier to use.

Across all levels of security practice, some organisations display poor security culture indicators. In Chapter 5 I looked at whether indicators of good secure-coding culture correlate with good secure-coding practice. I found that while many do correlate, closer examination of the data shows that even where secure-coding practice is strong, culture is sometimes poor or missing. Even when organisations have excellent security practice, at least 20% of them appear to have little or no security culture. This creates what is often called a checkbox security environment [111], where security practices are present, often for compliance reasons, but may be side-stepped, poorly-implemented or skipped because they are not a priority.

The chapter concluded with recommendations for various groups. For academia, I suggested that when studying software security in organisations, a CA Score should be obtained. Additionally, all or some of the suggested software security culture questions

should be asked. This is a more detailed echo of some of the recommendations resulting from the study in Chapter 4 that uncovered unhelpful assumptions. Researchers should consciously look up the hierarchy at the direction and support that is received (or not) from management and senior management, rather than training the secure-coding microscope solely on developers. For organisations, I recommended awareness of the need for secure-coding support in terms of time and budget. For industry I recommended a holistic approach to secure-coding direction. This advice applies to legislators also, who are best placed to influence secure-coding practice at scale.

Provision of secure-coding training can allow the organisation to enhance developers' software security expertise. One of the assumptions that I discussed in Chapter 4 is that '*All developers know how to code securely*.' I found that in some research, there is little discussion of whether individual developers have the knowledge to incorporate secure coding into their routine [321]. Some aspects of secure coding require relatively little specific technical skill. A simple, arguably naive, threat-modelling exercise could in theory be done by anyone who understands how a piece of software should behave. However, the avoidance of insecure software, as a purely technical coding matter, is non-trivial [199]. For example, the 'OWASP Top Ten' lists the most common and serious vulnerabilities in web applications[2]. Techniques to avoid these problems typically differ between platforms, coding languages and specific implementations. Developers need support, including formal training, to properly understand and avoid these and similar coding issues.

The worrying current state of software security training was clear from two questions on training in the developer survey, and is outlined in Chapter 7. The first question asked '*Have you ever been offered training relevant to software security or privacy in your environment?*' Only 39.6% of respondents chose '*Yes*.' The second question asked these respondents to '*please describe the training here*,' supplying a free text option. I found that 22 respondents described their training using a negative or very negative tone, while 34 conveyed a positive or very positive experience. There were descriptions of infrequent, irrelevant or compliance-focused training. Five respondents mentioned that they did not attend the training, usually mentioning deadline pressures as a reason. This is an indicator of very poor software security culture. Experts recommend that training should be mandatory [58]. Chapter 7 breaks down the results in greater detail. This study

[2] `https://owasp.org/www-project-top-ten/`

was the first of which I am aware to provide a list of criteria by which software security training can be judged. It is also the first large-scale survey that I have seen that assesses software security training in industry. Due to the free-text nature of the responses, in many cases I could only assign a neutral or unknown status for the items. Future work could adopt the criteria list and ask about each item individually.

The result of most concern was the large number of respondents who did not state that they had been offered secure-coding training and who therefore may not be aware of even the most elementary security precautions when coding. While some may work in environments where security is genuinely not a concern, the number of such environments is rapidly diminishing in this era of digital transformation [297]. It is more likely that they are '*Optimists*,' security-unaware developers working in security-unaware environments, as described in Chapter 3, which presented the Developer Security Archetypes. An absence of security training suggests a more general absence of secure-coding support. If they are attempting to code securely despite a lack of training, they may well be '*Heroes*.' Heroes work in security-unaware environments, but are security enthusiasts themselves. They can become burned out due to the lack of support.

9.5 Results of Study C: Motivating Organisations

One approach to enhancing secure coding in industry is devising interventions designed to improve developers' secure-coding practice [320, 227]. A question that arises in relation to such interventions is whether organisations would allow or implement them. Organisational secure-coding motivation may be even more important than that of individual developers. Organisations can mandate secure coding, rendering developers' personal interest and enthusiasm relatively irrelevant, since developers are accustomed to following specifications and procedures such as internal coding standards. Yet I did not find any formal study on the effect of organisational secure-coding priorities on secure development. Even more interesting, I could not find a study on organisational secure-coding motivation, and how it can be improved. Since organisations are composed of individuals, I reframed this second question in terms of whether and how senior managers in organisations — often called the C-suite — can be motivated to prioritise code security.

Considering the best approach to exploring these questions, I could not see value in a questionnaire. There were two reasons for this. First, there was little literature from

which to derive a set of questions. Second, it would be difficult to get interesting information from senior managers via a questionnaire, particularly an anonymous one. Therefore, I undertook an interview study, interviewing 21 people with insight into software development in organisations, including developers, consultants and senior managers. The protocol for each included questions about the effect of management prioritisation on secure coding and about the secure-coding motivations of senior management, or their absence.

Another area that needs urgent attention is the question of organisational turbulence and its effect on software security. Industry models such as the BSIMM tend to be aspirational. They describe a set of practices which, once implemented correctly, will ensure that there is a good secure-coding process in place. But real organisations are subject to turbulence. I conducted 18 of the interviews in 2023. Repeated rounds of layoffs in tech organisations had begun in late 2022 and were ongoing. There was general uncertainty in the industry about the sustainability and security of certain organisations' products due to the high level of job losses. I therefore asked a set of questions designed to discover whether interview subjects had observed any effect on software security from organisational turbulence.

I used reflexive thematic analysis, as described by Braun and Clarke in a series of papers [30, 31], to analyse the interview data. The details of this study can be found in Chapter 8. I found that senior management's priorities are important and influence the priorities set at other levels of the organisation. Where prioritisation of secure coding is missing, developers tend to feel lost and unsupported when encountering software security issues. The participants had encountered a range of attitudes to secure coding amongst senior managers. While some managers make it a high priority, others simply do not understand it, lacking the relevant experience. Still others claim to prioritise software security in public while in reality ignoring or even subverting it.

Interview participants observed that senior managers are more likely to prioritise code security when they have the education to understand it, when they observe breaches in other organisations, and when there are legislative obligations that mandate it. With regard to organisational turbulence, poor software security is often related to downsizing or to bankruptcy or other funding issues. These findings appear to support the existing drive to increase software security legislation and liability in both the U.S. and Europe [38].

9.6 Implications for Stakeholders

The work presented in this thesis has a number of implications for different stakeholders. First, software developers with an interest in software security should note that their ability to implement secure practices is somewhat dependent on their work environment. If they meet with resistance when raising secure-coding issues, they should be aware that this is not a universal experience. Trying to be a software security 'Hero' is exhausting and thankless. A different organisation might be a better fit for them.

This research also has implications for researchers. The study of secure software development in recent years has emphasised individual developers more than the work environment. This may be partly due to a centring of usable security for developer tools. While much needed, the focus on usable tools and developer activity may have sometimes blurred the importance of organisational concerns. It is hoped that this thesis will facilitate a renewed emphasis on the importance of the organisation, and an awareness of the Software Developer Security Archetypes may provide a context and vocabulary for this.

Researchers should particularly enjoy Chapter 4. Its topic, unhelpful assumptions in secure software development research, is particularly pertinent to them. It identifies 25 assumptions that are sometimes made by software security researchers and that can affect their research. This chapter should help researchers to avoid such assumptions in their own work, and perhaps to identify other unhelpful research assumptions.

There are implications for senior managers in organisations that develop software also. They should ensure that they are familiar with secure software development principles and best practice, and prioritise them. They should be aware of the danger of paying lip service and focusing too much on checking compliance boxes rather than embracing software security as a high priority. Insecure software could pose a systemic threat to their organisation. This risk is increasing as the catastrophic effects of poorly-secured software are felt worldwide, attracting the attention of governments and regulators.

More generally, the various institutions that concern themselves with software development must embrace the seriousness of software security. Universities and other educators should include mandatory software security training on their computer science and business syllabuses. Software development advocates should become familiar with software security concerns and factor them into all considerations.

My research shows that in many organisations, development of insecure software will continue unless it becomes financially unsustainable, and/or too personally costly to upper management. This should be considered by governments who are debating whether releasing software without taking software security precautions should incur a cost. Stringent regulations could stifle innovation. Nevertheless, the current situation is not sustainable. There are perverse incentives to release insecure software, due to the absence of clear deterrents.

Finally, the geopolitical adverse consequences of insecure software are chilling. More and more of our data and critical infrastructure are moving online. Failure to tackle the issue of turbulence, and the other organisational concerns I investigated in this thesis, will have increasingly disastrous implications and effects. These issues must be urgently addressed.

Appendices

A Protocol for Literature Review on Assumptions in Software Security Research

Note: This protocol is publicly available online at `https://osf.io/7a2dr/?view_only=e39e7822618f47f0a9bca7e5ec46d86c`. What follows is the exact text of the online protocol for the paper in Chapter 4.

This is the protocol for the systematic literature review focusing on software security research. This is a 'living document'; as decisions were made, this document was updated. The protocol follows the recommended structure by Kitchenham and Charters [138]. This version is dated 30 June 2023.

A.1 Background

In an increasingly-connected world, online security is ever more important. Many hacking techniques depend on the existence of vulnerabilities in software. Their presence facilitates cybercrime, from ransomware to cyber espionage [238]. The field of DCS explores reasons why developers write vulnerable code [267], seeking to understand developers, how they can be helped, and what problems they face in practice. Seeking to gain greater understanding of DCS challenges, we conducted a systematic literature review focusing on papers published in known DCS venues since 2016. The objective of the review was to identify reasons why, despite over two decades of software security research, the discovery rate of exploitable software vulnerabilities continues to increase [215].

A.1.1 Previous Reviews

Several previous reviews that are relevant to our study are noteworthy. Kaur et al. [136] present a number of lessons learned about human factors in security research. Mokhberi and Beznosov identified '*Human, Organizational, and Technological*' dimensions, containing 17 categories of challenges [173]. They viewed their work as the beginning of a systematic way of thinking about the challenges in software security. Some of the challenges have wide-ranging effects; for example, managers set and enforce priorities, contributing to culture, and therefore have a substantial effect on an organisation's security stance. To develop a security culture, both internal motivations such as valuing security and external motivations such as a belief that security will increase market share are needed. Security is a non-functional requirement, and Mokhberi and Beznosov reflected on the additional challenges entailed by its lack of visibility and the constantly changing nature of the threat landscape. Wide-ranging recommendations from this study included using organisational strategies to improve security; for example, making software security visible, ensuring developers know that they have responsibility for it, and designing tool-friendly policies. Providing training and support to developers was also advocated.

Das Chowdhury et al. analysed developer-centred studies, identifying 72 papers for their analysis that were written over 13 years [62]. They sought to find abstract themes in the literature, identifying challenges, behaviours and interventions, and how challenges and behaviours interact. They also analysed five '*tropes*,' or '*misplaced beliefs about how developers behave*.' They argued that these five misplaced beliefs strongly influence the design of libraries and tools, and the secure coding interventions that developers endure. These tropes are, that '*developers are experts*,' that '*everyone has the same security and privacy needs*,' that '*security is easy*,' '*security is a universal priority*,' and '*maintenance is fun*.'

A.1.2 Assumptions

Informed by guidelines on conducting systematic literature reviews [138, 312], we conducted a literature survey to determine the secure coding aids currently available to coders when writing software, and any secure coding barriers. We also sought to evaluate the quality of studies included in the review. We did this by evaluating logical

consistency, study populations and assumptions made within the studies. In the course of the review we observed a number of unconscious assumptions within the field of software security research. Some are common, some less so, but all are unhelpful when attempting to understand developer behaviour. Unconscious assumptions may lead researchers to miss relevant information, or to jump too hastily to conclusions.

Thus, whereas initially we sought to conduct a descriptive review [203], once we observed and identified the notion of assumptions in the literature, we adopted a more critical stance. Still relying on content analysis of selected papers, we also adopted a more critical interpretive method [203]. Paré et al. [203] describe that a critical review:

> "aim[s] to critically analyze the extant literature on a broad topic to reveal weaknesses, contradictions, controversies, or inconsistencies. [...] The strength of a critical review lies in its ability to highlight problems, [and] discrepancies [...]"

Assumptions are important to identify explicitly in any scientific field, at different levels of granularity. The process of identifying and challenging assumptions will have different implications for different fields, but the basic principle stands. A well known (and simple) example is assumptions in statistics: most traditional parametric statistical tests rely on several assumptions, a common one being that data are normally distributed. Using statistical tests while violating these assumptions will lead to biased (i.e., incorrect) results. Another example is the field of Organizational Management [9]. Quite different from the field of software security research, similar observations can be made in that studies frequently reinforce long-held assumptions which are never tested or challenged, and may lead to studies that make only small increments on the state of the literature. Within the field of software engineering, closer to software security, identifying assumptions has also proven useful. For example, Rolland et al. [233] identified a number of assumptions regarding agile methods in large-scale settings.

The current document is the study protocol for the literature review, and is structured following the Kitchenham and Charters guidelines [138].

A.2 Research Questions

Our literature review defined the following research questions at the outset:

RQ1 What technologies, strategies or techniques exist to help software developers to avoid errors that cause security bugs?

RQ2 Are there factors that have been found to make security bugs more likely?

RQ3 What are the characteristics of the studies that exist for each factor?

The analysis of data to address these three research questions is, as of now, currently ongoing. During the capturing of the data (see Sec. A.7) and analysis of the papers, a number of apparent contradictions emerged, and in other cases, obvious interpretations of study findings appeared to be missed by the original authors. This led us to focus on these issues, which we traced to the notion of "assumptions." That is, it appeared that many prior studies relied on certain assumptions related to the three research questions above; that is, assumptions about software developers, organizations, or the tools that might be available for software security-related job, as well as assumptions about how to measure key concepts such as software security.

The current protocol document seeks to capture the overall process, including literature coverage, selection process, data extraction and synthesis procedures as it pertained to our identification of assumptions in the selected papers.

The remainder of the current document focuses exclusively on this, and serves as an auxiliary document to the manuscript "Unhelpful Assumptions in Software Security Research."

A.3 Search Strategy

Our search strategy was an automated search within a pre-defined list of venues that are relevant to software security. This strategy is a hybrid between a manual search of pre-defined venues, which requires a researcher to read all titles (and possibly abstracts) of papers published in a particular venue, and an automated search which uses a search string in one or more of the major digital libraries. Using our hybrid strategy, we conducted automated searches while pre-selecting the venues. This strategy helps to prevent the need to evaluate very large numbers of papers (e.g. potentially over 10,000) if the search is fully 'open' (without pre-selecting the venues).

However, the trade-off is that we could have missed important and relevant papers that were not published in one of the pre-selected venues. To mitigate against this

risk, we also conducted a snowballing step [331] during our paper selection process (see Sec. A.5); snowballing refers to inspecting the bibliographies of already selected papers. Snowballing can thus mitigate against the risk of excluding papers a priori by not searching certain venues. Important related papers would be cited by those papers that we did select.

Software security research is published in a wide range of venues. In order to identify as many relevant papers as possible, we compiled a list of these venues with the help of a software security expert, who had published widely in the field as a researcher, but has also extensive security consulting experience. Workshops for these venues were also included. Table A.1 presents the list of venues.

We included papers published from 2016 on. Our initial search term was:

> Software AND (developer OR development) AND security

We ran trial searches on our target venues with this term. We found that several well-known and relevant studies that we were expecting to find were missing from search results. Through an iterative process we revised our search string, concluding with:

> (software OR code) AND (developer OR development) AND security

Because all search terms were singular words, there was no need to enforce an exact match of multiple words jointly using quotation marks.

The date range was enforced using the digital libraries' options to set the 'start' and 'end' dates.

We allowed autostemming, meaning that word stems for the search terms could be used during the search, which means that the terms had a wildcard implied at the end, as in "developer*"; this meant that the search included 'developers' as well as 'developer'.

Since no one search engine provided papers from all of our venues, we used the following three databases:

- ACM Digital Library
- IEEE Xplore
- Scopus

Table A.1: List of venues

Acronym	Type	Full Name
ARES	Conference	Availability, Reliability and Security
C&S	Journal	Computers & Security
CCS	Conference	Computer and Communications Security
CHI	Conference	ACM CHI Conference on Human Factors in Computing Systems
COMPSAC	Conference	Computer Software and Applications Conference
DSN	Conference	IEEE/IFIP International Conference on Dependable Systems and Networks
ESE	Journal	Empirical Software Engineering
ESORICS	Conference	European Symposium on Research in Computer Security
EURO S&P	Conference	European Symposium on Security and Privacy
FSE	Conference	Foundations of Software Engineering
ICSE	Conference	International Conference on Software Engineering
IEEE S&P[1]	Journal	IEEE Security and Privacy
JOC	Journal	Journal of Cybersecurity
NSPW	Workshop	New Security Paradigms Workshop
SecDev	Conference	IEEE Secure Development Conference
SOUPS	Conference	Symposium on Usable Privacy and Security
S&P "Oakland"	Conference	IEEE Symposium on Security & Privacy (commonly known as the "Oakland" conference)
SPE	Journal	Software – Practices and Experience
SPLASH	Conference	Conference on Systems, Programming, Languages, and Applications, Software for Humanity
TOPS	Journal	ACM Transactions on Privacy and Security
TOSEM	Journal	ACM Transactions on Software Engineering and Methodology
TSE	Journal	IEEE Transactions on Software Engineering
USENIX Security	Conference	USENIX Security Symposium

[1]We included the sponsor/publisher here to differentiate from the similarly named EURO S&P; we excluded the publishers from all other acronyms.

The exact syntax used depended on the search engine. The search resulted in an initial total of 852 papers (see Sec. A.5).

A.4 Study Selection Criteria

We focused on papers published since 2016; we selected this date for a few reasons. First, we decided that some start date (rather than leaving it open-ended) was necessary to be able to manage the number of papers. Second, we felt this was warranted because our research questions (see Sec. A.2) focused on technologies and tools to assist developers, and thus focusing on very old studies that may not be 'current' anymore would not contribute much to our study.

We defined the following inclusion and exclusion criteria. We started the literature review in 2020, and it was conducted over a very extensive time period of circa 2 years. During the process, we conducted searches at several points in time (see details in Sec. A.5), continuously monitoring whether new relevant papers were published. The cutoff date for papers to be included in the review was September 30, 2022.

A.4.1 Inclusion criteria

I1 papers that identify one or more factors that reduce the incidence of security vulnerabilities in code while it is being written

I2 papers that identify one or more factors that increase the incidence of security vulnerabilities in code while it is being written

A.4.2 Exclusion criteria

E1 Abstracts, posters, talks, proceeding entries, tutorials

E2 Not peer reviewed papers

E3 Two-page outline papers and other very short papers.

E4 Duplicates. Where two papers described the same study, the most recent paper was used.

E5 Papers that did not identify factors that reduce or increase the incidence of security issues in code.

E6 Papers describing tools that help to write secure code in specific problem domains with no application for the broader developer community (for example a tool to verify contract rules on the Ethereum blockchain). Further, papers describing specific implementations of tool types that are already well-known. For example, a paper describing a new static analysis tool for a specific language.

E7 Proposals for new processes or tools that have not been empirically assessed either in a laboratory setting or in industry. An example would be a proposed method for integrating security into Agile that has not been put into practice anywhere.

A.5 Study Selection Procedures

This section describes the steps to identify and select relevant papers. Figure A.1 presents a summary. The first author undertook the selection procedure, with consultation with the other authors where necessary. In each of the steps, when in doubt about a paper, it was reviewed by the research team.

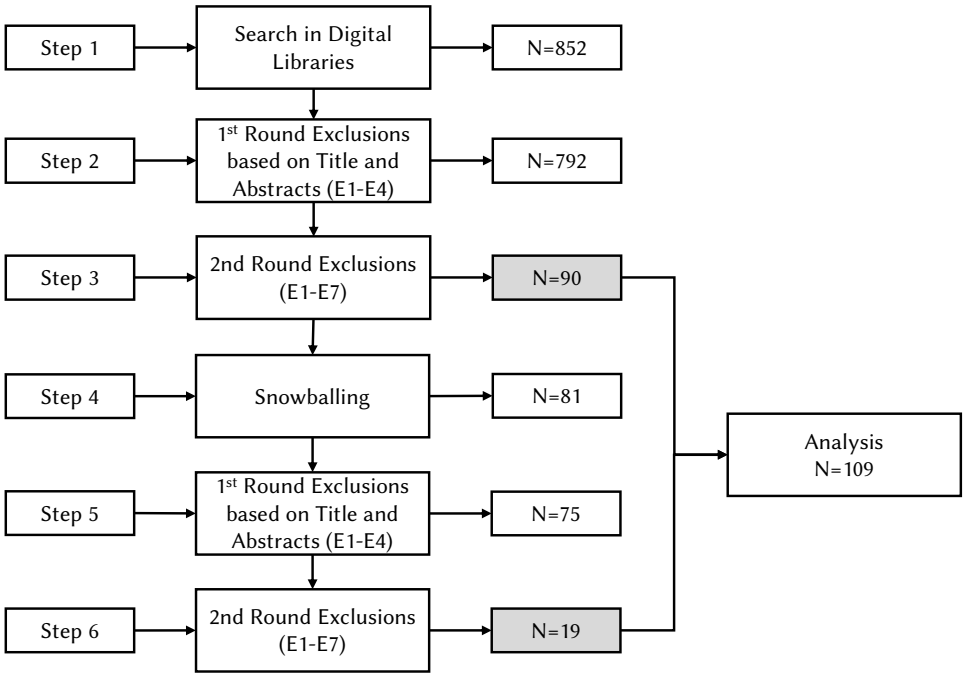


Figure A.1: Study selection process

A.5.1 Step 1: Search in Digital Libraries

Step 1 involved conducting the searches in the digital libraries (listed in Section A.3). These searches led to a combined total of 852 studies.

As alluded to in Sec. A.4, we conducted the searches several times; Vom Brocke et al. [312] refer to this as an iterative process, which they describe as requiring:

> "the researcher to search for literature in a continuous, repeated process that is intertwined with reviewing and synthesizing the literature."

We first conducted the searches in 2020, when we started the process. We re-ran the search in 2021, on the years 2020-2021, so as to ensure that no publications were missed, and again in 2022 on the years 2021-2022. We meticulously updated our database of papers, keeping a detailed administration of the steps we followed.

A.5.2 Step 2: First Round Exclusions

Step 2 was a first round of "quick" exclusions, which focused on exclusion criteria E1 to E4 (see Sec. A.4). All 852 papers resulting from Step 1 were inspected based on title and abstract. Abstracts, posters, and short, two-page papers were removed, and duplicates were removed as well at this stage. Three papers should not have appeared in search engine results. For example, one paper appeared twice in the same search result. We excluded these and labelled the reason for exclusion "search engine error." A total of 792 papers passed Step 2 and were evaluated for further selection in Step 3.

A.5.3 Step 3: Second Round Exclusions

Step 3 was a second round of exclusions that required more time, as inclusion/exclusion decisions could not be taken quickly. This step now considered all exclusion criteria E1 to E7. Table A.3 presents details. During this step, we excluded 702 papers, leaving 90 for further analysis.

A.5.4 Step 4: Snowballing

In step 4 we used snowballing as a technique to identify additional papers that were either by the same authors or cited by the papers included.

Table A.2: Details of exclusions in Step 2

Exclusion criterion	No. Exclusions
E1 Proceedings entries	19
E1 Talks	7
E1 Tutorial	8
E1 Poster	5
E2 Not peer reviewed	2
E3 Very short papers	15
E4 Duplicate	1
Search Engine Error	3
Total excluded	60

Table A.3: Details of exclusions in Step 3

Exclusion criterion	No. Exclusions
E5 Off-topic	496
E6 Specific domains and tools	155
E7 Proposals	48
Duplicate	3
Total excluded	702

We identified authors who had worked on three or more of the papers reviewed. (We set this threshold to three, in order to remain pragmatic; a threshold of one would obviously lead to a list of all authors, which would not be workable.) This resulted in a list of 36 authors. Other papers by these authors were assessed for inclusion in the study. After a pilot with three authors using ACM, IEEE, Google Scholar, and Scopus for this search, Google Scholar was identified as capturing the most papers for each author. Google Scholar was therefore used for the author snowballing. If in doubt about a paper that was found in Google Scholar, we included it. This led to a list of 78 papers. These were evaluated for inclusion as shown in Figure A.1, yielding 16 additional papers.

The references sections of all our selected papers were processed using an extraction tool named CERMINE [278]. CERMINE is described as [278]: *"a comprehensive tool for automatic metadata extraction from born-digital scientific literature."* Its main

purpose is to extract metadata from a document, as well as a list of bibliographic references with their metadata.

The bibliographies were scanned and cited papers that seemed relevant were screened for relevance to our review. Seventeen of these papers were already included in the review. Therefore, this process resulted in only three additional papers.

To ensure that we had not missed important relevant venues, we analysed the venues where our 19 (=16+3, see above) snowballed papers were published. We looked for venues that had published two or more snowballed papers and that we had not already searched. We did not find any such venues amongst the snowballed papers.

A.5.5 Step 5: First Round Exclusions after Snowballing

Step 5 is a repeat of Step 2, but only on the papers identified during snowballing (see Step 4). Again, only focusing on exclusion criteria E1-E4, we excluded 6 of the 81 papers. Further analysis was done on 75 of the papers in Step 6. Table A.4 presents details.

Table A.4: Details of exclusions in Step 5

Exclusion criterion	No. Exclusions
E1 Proceedings entries	0
E1 Talks	1
E1 Tutorial	0
E1 Poster	0
E2 Not peer reviewed	0
E3 Very short papers	2
E4 Duplicate	3
Search Engine Error	n/a
Total excluded	6

A.5.6 Step 6: Second Round Exclusions after Snowballing

Step 6 is a repeat of Step 3, but only on the papers identified during snowballing (see Step 4). This reduced the number of relevant papers to 19. These 19 were combined with the original set of 90 resulting from Step 3. The total number of papers that were analysed for the review was thus 90 + 19 = 109. Table A.5 presents details.

Table A.5: Details of Step 6

Exclusion criterion	No. Exclusions
E5 Off-topic	27
E6 Specific domains and tools	11
E7 Proposals	7
Duplicate	11
Total excluded	56

Table A.6 presents the initial number of papers per venue (see Table A.1) that were identified via search, and the final number remaining after exclusions. Initial and final number of papers that were identified through author snowballing and references snowballing are at the bottom (see Sec. A.5.4). As noted in Step 1, we conducted an iterative search process to stay up-to-date during the review process. The numbers presented here are cumulative after the three searches.

A.6 Study Quality Assessment

The SLR procedure originated from evidence-based medicine, where-by all papers presenting evidence for a particular intervention are identified. All evidence must be critically evaluated, because evidence from poorly conducted studies should not be included so as to prevent drawing the wrong conclusion.

A quality assessment in this SLR was not deemed appropriate for a few reasons. First, this SLR does not seek to draw any firm conclusion on specific interventions (specific aids in the development of secure software). Second, the methods used in the selected papers varied widely; unlike in evidence based medicine, which relies primarily on randomized controlled trials (RTC), we observed a wider range of methods, including interview studies and surveys.

While we wrote down extensive notes regarding our thoughts, comments, and ideas when reading the papers, including our opinions regarding the quality of papers, no papers were excluded based on the criterion of what we perceived to be low quality.

Table A.6: Selected papers per venue

Source	Candidates Identified	Final Selection
ARES	64	9
C&S	65	4
CCS	86	5
CHI	14	6
COMPSAC	26	0
DSN	15	0
ESE	40	4
ESORICS	43	1
EURO S&P	10	1
FSE	43	4
ICSE	150	17
IEEE S&P	21	6
JOC	3	1
NSPW	6	1
SECDEV	31	4
SOUPS	16	8
S&P (Oakland)	48	8
SPE	30	1
SPLASH	11	0
TOPS	8	0
TOSEM	9	3
TSE	32	4
USENIX Security	81	3
Author snowballing	78	16
Reference snowballing	3	3
Total	933	109

A.7 Data Extraction Strategy

Data extracted from each paper was as follows:

- Title
- Authors

- Publication title
- Date inclusion decision made
- Date analysis complete
- Summary of study research questions
- Summary of findings
- Study design critique
- Secure coding aids
- Barriers to secure coding

All data were captured in spreadsheets. We also kept running notes during the project, recording issues and solutions as they arose. We kept a record of outstanding issues using Trello (`https://trello.com/`).

Once the papers for the review were identified, each paper was analysed and factors affecting the security of code at coding time were extracted and categorised. Such factors could affect the code either positively or adversely. During this process, a number of apparent contradictions between papers came to light. In other cases, an obvious interpretation of the data collected by the researchers appeared to be missed. On analysing these contradictions and missed opportunities, it became apparent that in some cases the root cause was an unconscious assumption made by researchers.

We identified these assumptions by documenting key phrases that were of particular interest from the papers, or that suggested that the authors made a certain assumption. Further investigation was done to confirm the presence of the assumptions. Table A.7 provides examples where the assumptions were confirmed.

Once all data was captured, we further analysed and synthesized the data; Sec. A.8 describes our procedures.

A.8 Synthesis of Extracted Data

A.8.1 Synthesizing Assumptions

Table A.7 presents an example of how assumptions were identified in the extracted data. The first example, the assumption that "Keeping software updated is always good," shows two sources. The first source is a paper by Derr et al. [67]. From this paper, we extracted:

> "To provide end-users reliable software, it is therefore of utmost importance to keep third-party libraries up-to-date."

However, the paper makes no mention of any potential danger that updating could introduce new vulnerabilities. We observed a similar issue in the paper by Huang et al. [121]; the text of interest from this paper is:

> "As old third-party library versions inevitably contain bugs (e.g., functional bugs and security bugs), they might cause crashes or increase attack surfaces in client projects. As new third-party library versions refactor code, fix bugs and add features, they might break library APIs."

While the last sentence in this excerpt describes possible issues with new third-party library versions, it makes no mention of the danger that a new version might also contain security bugs.

Once the assumption was identified and labelled, we then proceeded to report it, and identified counter-examples, or "black swans" that demonstrate the opposite. In other words, certain papers make assumptions, but others demonstrate that these are, indeed, assumptions, and not reality. The juxtaposition of the counter-examples with the assumptions that we identified demonstrate the contrast and divergence that exist in the literature.

For each of the included papers in the review, we extracted relevant excerpts. Where unwarranted assumptions appeared to be made, we investigated further (as in Table A.7).

Table A.7: Examples of analyses underpinning the identification of unhelpful assumptions

Assumption: Keeping software updated is always good	
Paper	Keep me Updated: An Empirical Study of Third-Party Library Updatability on Android, Derr et al. 2017 [67]
Extracts	"To provide end-users reliable software, it is therefore of utmost importance to keep third-party libraries up-to-date."
Analysis	No mention of potential danger that updating swiftly could introduce new vulnerabilities. No mention of this was found elsewhere in the paper.
Paper	Characterizing usages, updates and risks of third-party libraries in Java projects. Huang et al. 2022 [121]
Extracts	"As old third-party library versions inevitably contain bugs (e.g., functional bugs and security bugs), they might cause crashes or increase attack surfaces in client projects. As new third-party library versions refactor code, fix bugs and add features, they might break library APIs."
Analysis	The second sentence describes possible issues with new third-party library versions, but does not include the danger that the new version might also contain security bugs or increase attack surfaces. No mention of this fact was found elsewhere in the paper.
Assumption: Developers can introduce secure coding practices and tools to organisations easily	
Paper	Surveying Secure Software Development Practices in Finland. Kalle Rindell et al. 2018 [232]
Extracts	"penetration testing is perceived to have a very high impact [in our survey of Finnish software professionals], although this activity is only seldom systematically used in Finland."

Continued on next page

Table A.7 *Continued from previous page*

	"it is worth pointing out that penetration testing was perceived as particularly important for security. This result is in contrast to the [referenced] mentioned survey, which rather pointed out the limitations of penetration testing when compared to code reviews. Given that penetration testing is not widely used in Finland according to the results, the reason may relate to the concept of perceived impact on software security. In other words, an activity that is widely promoted by consultants and industry associations may not correlate with the actual impact of the activity. This remark leads to a couple of important points for both research and practice..."
Analysis	The authors here do not discuss the possibility that the software professionals would have liked to introduce penetration testing, but could not get approval for it. The authors hypothesise an alternative solution, that penetration testing may not be widely used because it is ineffective, even though 'widely promoted by consultants and industry associations'. They go on to draw conclusions based on this hypothesised ineffectiveness. While the logic used is somewhat circular, we believe that the root cause of the confusion is an assumption that the surveyed software professionals would have been able to introduce the practice if they wished to.

A.8.2 Categorising Assumptions

We identified a total of 25 assumptions; we do not claim that these are necessarily all assumptions that exist in this or other literature. The goal here is not necessarily to be complete, but rather to start a further conversation about the nature of these assumptions.

To organize the discussion of these 25 assumptions, we organized them into four categories. Three of these were based on categories identified by Mokhberi and Beznosov [173], which we found useful in our own work. Mokhberi and Beznosov refer to the elements in each category as *'dimensions.'* Their *'human dimensions'* include, for example, *Role of personality*, *Experience*, and *Attitudes & Perceptions*. We relabelled the category *'Human'* as *'Developer'* because we wanted to be explicit that this category refers to developers, not people in the organization who do not write code such as managers. This is because managers are frequently the people responsible for implementing organisational policy and strategy, and there is a separate category for assumptions related to organisations. We identified a fourth category that seeks to capture assumptions about the research process, and so we labelled this category 'Research Assumptions'. We describe the four categories briefly:

1. Technological Assumptions: assumptions that refer to the technology and tools that are available, their characteristics, or any practices that refer to tools and techology.

2. Organisational Assumptions: assumptions that relate to organisations as a whole, in relation to their priorities, policies, and strategies. Assumptions held by managers would also fall into this category, rather than the developer category, because the manager here is seen as a proxy for the organization: someone who represents the organization and implements its policy.

3. Developer Assumptions: assumptions that refer to a developer's characteristics, preferences, knowledge, and expertise. Taken together, these assumptions would have readers believe that developers behave in a certain way, without providing any evidence for such assumptions or generalizations.

4. Research Assumptions: assumptions that relate to the study of secure software development. These assumptions encapsulate beliefs about how to measure things, how to study certain topics, what questions to ask, etc. All relate to the study, or research enterprise, of secure software.

It is important to note that there is no inherent value in this particular set of categories: others may identify a set of alternative categories—the process of categorising is ultimately subjective. The purpose is not to present a definitive list of categories, but rather one that is helpful for the purpose of presenting these assumptions in a way that can be more easily digested than a list of 25 entries without further structure.

A.9 Dissemination Strategy

This protocol document is made freely available on an Open Science Foundation (OSF) repository. We submitted the paper that focuses on *'unhelpful assumptions'* to CCS 2023 and it was accepted.

B Protocol for Interview Questions

These are the questions asked of interviewees in Chapter 8.

B.1 All Participants: Opening Question

As you know, this interview is being recorded. Is that OK with you?

B.2 Questions for Developers

B.2.1 Demographic information

- What country do you live in?
- What job title and role do you have in your current organisation?
- How many years of development experience do you have?
- Do you have other relevant experience?
- What kind of software do you mostly work on?
- What technologies do you use?

This section is for solo developers only

- You are self-employed, is that right?
- Do you ever work embedded within organisations?
- Do you ever work with other software developers?

This section is for developers in organisations only

- Are you contract or permanent?
- What kind of environment do you currently work in, big company or small? Multinational?
- Is it a software organisation?
- What is the organisation's main focus?
- How many other software developers do you work with regularly?

B.2.2 Working environment information

Read this definition to the interviewee: Software security is the ability of software to withstand malicious attack. Good ability to withstand attack is achieved by considering security when coding software.

- What do you think are the main security risks to the software you work on?
- Do you try to code securely?

If yes:

- What motivates you to code securely? How did you start?
- Do you code security tools, or do you focus on writing code securely, or both?
- Do you feel that you succeed in making your code secure? How do you check?

This section is for solo developers only

- Where do you look for support to write secure code?
- Do your clients ask about code security?

This section is for developers in organisations only

- Do you generally feel supported to write secure code?
- Have you ever been asked to focus on functionality and ignore secure coding for a while?

- Would you consider yourself to be a software security driver in your organisation?
- How is that manifested?
- How is it viewed – by colleagues – by management?

If no:

- If you wanted to write secure code, would you have support to do that?

B.2.3 Information on specific practices

I have a list of secure coding practices here. Can you tell me whether you have encountered them in your coding environment? (For solo developers: can you tell me whether you use them?)

- (**Solo not asked**): Secure coding procedures that must be followed
- A written security checklist
- Use of security tools
  - Do you think there is overhead attached to using them?
  - (**Solo**): How did you find them? Did you find setup difficult? Why?

    (**Others**): Did you introduce any yourself? Was that difficult? Why?
- Automation such as DevSecOps. What are the benefits and drawbacks to this?
- A policy on keeping dependencies such as libraries up-to-date? (**Solo**): Do you follow it? (**Others**): Is it followed?
- (**Solo**): Secure coding training or research

  (**Others**): Secure coding training, or support for secure coding research if you are already expert
- (**Solo**): Discussion of code security aspects with others

  (**Others**): Frequent discussion of code security aspects (more than once a week)
- Methodical consideration of code security when a project begins

- (**Solo**): Taking time to do a good job on code security, in your opinion

  (**Others**): Enough time allocated to do a good job on code security, in your opinion

This section is for developers in organisations only:

- Rewards or thanks for doing a good job on code security
- Adverse personal consequences if you ignore code security
- Does your immediate manager seem to care about software security?

All developers:

- Do you feel you've learned anything about software security from your development role?
- Do software security requirements get in the way of getting your job done?
- Are there other security aspects in your workplace that you'd like to mention? (Example if asked, requirements evaluations or code reviews that consider security).
- As far as you know, has code you wrote ever caused a security breach? Do you want to tell us about it?
- (**Solo**): As far as you know, has a company or team you dealt with ever experienced a security breach? Explain.

  (**Others**): As far as you know, has your company or team ever experienced a security breach? Explain.
- Do you work with open source? What are the differences in secure coding practice in that environment?
- What do you think would be the best possible improvement to support you [**not solo**: and your team] to write secure code? At any level; industry, organisation or tool.
- Do you work with cloud services? If so, do you find that there are any security implications?

- What do you feel is the biggest barrier to writing secure code at the moment?

This section is for developers in organisations only

- Do you think that the importance you attach to software security is the same as the importance your organisation attaches to it?
- What do you think is the attitude of top management at your company – like the CEO, or the C-Suite – to software security?
- Do you have any theories as to why they have that attitude?
- Are there inconsistencies between what they say and what they do?
- Do you have any thoughts on a difference in attitude between contract and permanent employees when it comes to software security?

B.3 Questions for consultants with industry knowledge

- What is your current job title?
- What is your role in your current organisation?
- Approximately what size is your organisation?
- What role do you have in relation to software security?
- Software security is different from general cybersecurity, because it involves taking steps during the software development process to ensure that the code produced is secure. Do you think that the organisations you work with prioritise software security?

  If no:

  - Why not?

  If yes:

  - what do you think are the drivers for this? (If not mentioned, ask about:
    - Client expectations

    - Regulation
    - Legal
    - Reputation
    - Committed specialist)

- How old is the drive for software security in the industry?
- What do you think is the attitude of top management at most organisations – like the CEO, or the C-Suite – to software security?
- Do you have any theories as to why they have that attitude?
- Are there inconsistencies between what they say and what they do with regards to software security?
- Speaking about organisations in general, what one or two developments do you think could push software security higher up the agenda for all management C-suites? (If they are not sure what I mean: For example, the ability for ransomwared customers to sue / removal of insurance support for ransomware recovery costs).
- Is your assessment of the importance of software security aligned to that of your organisation?
- What do you think is the biggest barrier to developers and organisations in writing secure code?
- Have you encountered any security implications to working with cloud services?
- Have you encountered issues with the open source supply chain? How are they addressed?
- What do you think is the most important thing a management team can contribute to software security within the organisation?
- The BSIMM authors suggest that historically, a drive towards secure software has only succeeded when top-down – i.e., when the main instigator is in management.

In recent years they've said that this is changing and that an engineering-led software security drive now has a chance of success within an organisation, probably due to DevSecOps. What is your opinion on this?

- As far as you know, has a company or team you worked with ever experienced a security breach? Explain.

B.4 Questions for senior managers

- What is your current job title?
- What role do you have in relation to software security in your current organisation?
- Approximately what size is your organisation?
- How many developers are there in the organisation?
- Software security is different from general cybersecurity, because it involves taking steps during the software development process to ensure that the code produced is secure. Does your organisation prioritise software security?

  If no:

  - Why not?

  If yes:

  - what do you think are the drivers for this? (If not mentioned, ask about:
    - Client expectations
    - Regulation
    - Legal
    - Reputation
    - Committed specialist)
- How old is the drive for software security in your organisation?

- Speaking about organisations in general, what one or two developments do you think could push software security higher up the agenda for all management C-suites? (If they are not sure what I mean: For example, the ability for ransomwared customers to sue / removal of insurance support for ransomware recovery costs).
- Is your assessment of the importance of software security aligned to that of your organisation?
- Would you consider yourself to be a software security driver in your organisation?
- What do you think would be the best possible improvement to help you to support your team in writing secure code?
- What do you think is the biggest barrier to you supporting your team in writing secure code?
- What has been your experience of software security in previous roles?
- What do you think is the most important thing a management team can contribute to software security within the organisation?
- The BSIMM authors suggest that historically, a drive towards secure software has only succeeded when top-down – i.e., when the main instigator is in management. In recent years they've said that this is changing and that an engineering-led software security drive now has a chance of success within an organisation, probably due to DevSecOps. What is your opinion on this?

B.5 All Participants: Broader industry

- Has organisational turbulence – downsizing, mergers, etc – ever had an effect on secure coding practice in your experience?
- Has there been any occasion in your career where you've encountered a big increase or decrease in secure coding behaviour in an organisation or team? What do you think were the causes?

- What do you think will be the impact of recent AI developments on software security?
- **(Except consultants)**: What has been your experience of software security in previous roles?
- How do you feel about secure coding in the general developer population?
- Do you think that more legislation or regulation would increase overall code security?
- Do you have anything more you would like to add about this topic?

B.5.1 Concluding Questions

Thanks for doing this interview. How did you find the experience? Did it raise any issues for you?

You may have further thoughts about this later. If you'd like to fill me in on anything else, you can contact me at itaryan@gmail.com, or at the university email address in the interview doc.

References

[1] Yasemin Acar, Michael Backes, Sven Bugiel, Sascha Fahl, Patrick McDaniel, and Matthew Smith. SoK: Lessons learned from Android security research for appified software platforms. In *2016 IEEE Symposium on Security and Privacy (SP)*, page 433–451, 2016.

[2] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. Comparing the usability of cryptographic APIs. In *2017 IEEE Symposium on Security and Privacy (SP)*, page 154–171. IEEE, 2017.

[3] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, page 289–305. IEEE, 2016.

[4] Yasemin Acar, Sascha Fahl, and Michelle L. Mazurek. You are not your developer, either: A research agenda for usable security and privacy research beyond end users. In *2016 IEEE Cybersecurity Development (SecDev)*, page 3–8. IEEE, 2016.

[5] Yasemin Acar, Christian Stransky, Dominik Wermke, Michelle L Mazurek, and Sascha Fahl. Security developer studies with GitHub users: Exploring a convenience sample. In *Thirteenth Symposium on Usable Privacy and Security, (SOUPS 2017)*, page 81–95, 2017.

[6] Yasemin Acar, Christian Stransky, Dominik Wermke, Charles Weir, Michelle L. Mazurek, and Sascha Fahl. Developers need support, too: A survey of security advice for software developers. In *2017 IEEE Cybersecurity Development (SecDev)*, page 22–26. IEEE, 2017.

[7] Abeer AlJarrah and Mohamed Shehab. The demon is in the configuration: Revisiting hybrid mobile apps configuration model. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, page 1–10. ACM, 2017.

[8] Majed Almansoori, Jessica Lam, Elias Fang, Kieran Mulligan, Adalbert Gerald Soosai Raj, and Rahul Chatterjee. How secure are our computer systems courses?

In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, 2020.

[9] Mats Alvesson and Jörgen Sandberg. Generating research questions through problematization. *Academy of Management Review*, 36(2):247–271, 2011.

[10] Renana Arizon-Peretz, Irit Hadar, and Gil Luria. The importance of security is in the eye of the beholder: Cultural, organizational, and personal factors affecting the implementation of security by design. *IEEE Transactions on Software Engineering*, 48(11):4433–4446, 2022.

[11] Renana Arizon-Peretz, Irit Hadar, Gil Luria, and Sofia Sherman. Understanding developers privacy and security mindsets via climate theory. *Empirical Software Engineering*, 26(6):34, 2021.

[12] Debi Ashenden and Darren Lawrence. Security dialogues: Building better relationships between security and business. *IEEE Security Privacy*, 14(3):82–87, 2016.

[13] Hala Assal and Sonia Chiasson. Security in the software development lifecycle. In *Proceedings of the Fourteenth USENIX Conference on Usable Privacy and Security*, page 281–296, USA, 2018. USENIX Association.

[14] Hala Assal and Sonia Chiasson. "Think secure from the beginning": A survey with software developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, page 1–13. ACM, 2019.

[15] Betsy Atkins. SEC charges CISO with fraud in landmark cybersecurity case. `https://www.forbes.com/sites/betsyatkins/2023/11/16/sec-charges-ciso-with-fraud-in-landmark-cybersecurity-case/`, 2023.

[16] Maria Bada and Ildiko Pete. An exploration of the cybercrime ecosystem around Shodan. In *2020 7th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, page 1–8, 2020.

[17] Wei Bai, Omer Akgul, and Michelle L. Mazurek. A qualitative investigation of insecure code propagation from online forums. In *2019 IEEE Cybersecurity Development (SecDev)*, page 34–48. IEEE, 2019.

[18] Frank Bajak. Chinese spies breached hundreds of public, private networks, security firm says. `https://apnews.com/article/barracuda-mandiant-cybersecurity-china-hackers-a52d1595c9108d2c58df11e38756600d`, 2023.

[19] Bojana Bakić, Miloš Milić, Ilija Antović, Dušan Savić, and Tatjana Stojanović. 10 years since Stuxnet: What have we learned from this mysterious computer software worm? In *2021 25th International Conference on Information Technology (IT)*, page 1–4, 2021.

[20] Aniqua Z. Baset and Tamara Denning. IDE plugins for detecting input-validation vulnerabilities. In *2017 IEEE Security and Privacy Workshops (SPW)*, page 143–146, 2017.

[21] BBC. Russia bans sale of gadgets without Russian-made software. https://www.bbc.com/news/world-europe-50507849, 2019.

[22] BBC. NSO Group: Israeli spyware company added to US trade blacklist. https://www.bbc.com/news/technology-59149651, 2021.

[23] BBC. Kazakhstan unrest: Internet cut amid fuel protests. https://www.bbc.com/news/world-asia-59876093, 2022.

[24] Kellen Beck. Hackers exploit casino's smart thermometer to steal database info. https://mashable.com/article/casino-smart-thermometer-hacked, 2018.

[25] Tim Bender, Rolf Huesmann, and Andreas Heinemann. Software development processes for ADs, SMCs and OSCs supporting usability, security, and privacy goals - an overview. In *The 16th International Conference on Availability, Reliability and Security*, page 1–6. ACM, 2021.

[26] Shay Berkovich, Jeffrey Kam, and Glenn Wurster. UBCIS: Ultimate benchmark for container image scanning. In *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20)*, page 6, 2020.

[27] Chris Bing. China's government is keeping its security researchers from attending conferences. https://www.cyberscoop.com/pwn2own-chinese-researchers-360-technologies-trend-micro/, 2018.

[28] David Bisson. NotPetya: Timeline of a ransomworm. https://www.tripwire.com/state-of-security/security-data-protection/cyber-security/notpetya-timeline-of-a-ransomworm/, 2017.

[29] Nandita Bose. Biden: If U.S. has 'real shooting war' it could be result of cyber attacks. https://www.reuters.com/world/biden-warns-cyber-attacks-could-lead-a-real-shooting-war-2021-07-27/, 2021.

[30] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101, 2006.

[31] Virginia Braun and Victoria Clarke. One size fits all? What counts as quality practice in (reflexive) thematic analysis? *Qualitative Research in Psychology*, 18(3):328–352, 2020.

[32] Larissa Braz, Enrico Fregnan, Gül Çalikli, and Alberto Bacchelli. Why don't developers detect improper input validation?': DROP TABLE Papers. In *Proceedings of the 43rd International Conference on Software Engineering*, page 499–511. IEEE Press, 2021.

[33] Elizabeth Buchanan. Subsea cables in a thawing Arctic. `https://www.maritime-executive.com/editorials/subsea-cables-in-a-thawing-arctic`, 2018.

[34] Marte Pettersen Buvik and Anastasiia Tkalich. Psychological safety in Agile software development teams: Work design antecedents and performance consequences. *CoRR*, abs/2109.15034, 2021.

[35] David Byrne. A worked example of Braun and Clarke's approach to reflexive thematic analysis. *Quality & Quantity*, 56(3):1391–1412, 2022.

[36] Oliver Carroll. Hacktivists vs The Dictator: How Belarus cyber army is taking on Alexander Lukashenko and his goons. `https://www.independent.co.uk/news/world/europe/belarus-lukashenko-protests-cyber-attacks-minsk-b807184.html`, 2021.

[37] Madison Cartwright. Internationalising state power through the internet: Google, Huawei and geopolitical struggle. *Internet Policy Review*, 9(3), 2020.

[38] Centre for Cybersecurity. Cybersecurity agencies publish new guidance on safe software design: Here's why it matters. `https://www.weforum.org/agenda/2023/04/cybersecurity-secure-by-design-software-guidance/`, 2023.

[39] Chainalysis Team. As ransomware payments continue to grow, so too does ransomware's role in geopolitical conflict. `https://blog.chainalysis.com/reports/2022-crypto-crime-report-preview-ransomware/`, 2022.

[40] Zhe Chen, Chuang Wang, Guanwen Li, Zhe Lou, Sheng Jiang, and Alex Galis. NEW IP framework and protocol for future applications. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, page 1–5, 2020.

[41] Bodin Chinthanet, Raula Gaikovina Kula, Shane McIntosh, Takashi Ishio, Akinori Ihara, and Kenichi Matsumoto. Lags in the release, adoption, and propagation of npm vulnerability fixes. *Empirical Software Engineering*, 26(3), 2021.

[42] Maria Christakis and Christian Bird. What developers want and need from program analysis: An empirical study. In *ASE 2016 - Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, page 332–343, 2016.

[43] Sam Chung, Leo Hansel, Yan Bai, Elizabeth Moore, Carol Taylor, Martha Crosby, Rachelle Heller, Viatcheslav Popovsky, and Barbara Endicott-Popovsky. What approaches work best for teaching secure coding practices? In *Proceedings of the 2014 HUIC Education & STEM Conference*, 2014.

[44] Catalin Cimpanu. Oracle: China's internet is designed more like an intranet. https://www.zdnet.com/article/oracle-chinas-internet-is-designed-more-like-an-intranet/, 2019.

[45] Catalin Cimpanu. Two of China's largest tech firms are uniting to create a new 'domestic OS'. https://www.zdnet.com/article/two-of-chinas-largest-tech-firms-are-uniting-to-create-a-new-domestic-os/, 2019.

[46] Catalin Cimpanu. Finland says hackers accessed MPs' emails accounts. https://www.zdnet.com/article/finland-says-hackers-accessed-mps-emails-accounts/, 2020.

[47] Catalin Cimpanu. Chinese government lays out new vulnerability disclosure rules. https://therecord.media/chinese-government-lays-out-new-vulnerability-disclosure-rules/, 2021.

[48] Catalin Cimpanu. Israel restricts cyberweapons export list by two-thirds, from 102 to 37 countries. https://therecord.media/israel-restricts-cyberweapons-export-list-by-two-thirds-from-102-to-37-countries/, 2021.

[49] Catalin Cimpanu. Some ransomware gangs are going after top execs to pressure companies into paying. https://www.zdnet.com/article/some-ransomware-gangs-are-going-after-top-execs-to-pressure-companies-into-paying/, 2021.

[50] Catalin Cimpanu. Microsoft: Data-wiping malware disguised as ransomware targets Ukraine again. https://therecord.media/microsoft-data

-wiping-malware-disguised-as-ransomware-targets-ukraine-again/, 2022.

[51] Catalin Cimpanu. Ukraine dismantles social media bot farm spreading "panic". https://therecord.media/ukraine-dismantles-social-media-bot-farm-spreading-panic/, 2022.

[52] Eva Claessen. Reshaping the internet – the impact of the securitisation of internet infrastructure on approaches to internet governance: the case of Russia and the EU. *Journal of Cyber Policy*, 5(1):140–157, 2020.

[53] Victoria Clarke and Virginia Braun. Thematic analysis. *The Journal of Positive Psychology*, 12(3):297–298, 2017.

[54] Cloudflare. What is an Internet exchange point? — how do IXPs work? https://www.cloudflare.com/learning/cdn/glossary/internet-exchange-point-ixp/, 2019.

[55] Carolyn Cohn. Focus: Insurers run from ransomware cover as losses mount. https://www.reuters.com/markets/europe/insurers-run-ransomware-cover-losses-mount-2021-11-19/, 2021.

[56] Lucian Constantin. Clop ransomware dominates ransomware space after MOVEit exploit campaign. https://www.csoonline.com/article/650272/clop-ransomware-dominates-ransomware-space-after-moveit-exploit-campaign.html, 2023.

[57] Crowdstrike. What Are Living off the Land (LOTL) Attacks? https://www.crowdstrike.com/cybersecurity-101/living-off-the-land-attacks-lotl/, 2023.

[58] Kenneth G. Crowther and Brian Rust. Built-in cybersecurity: Insights into product security for cyberphysical systems at a large company. *IEEE Security and Privacy*, 18(5):74–79, 2020.

[59] Anastasia Danilova, Alena Naiakshina, Johanna Deuter, and Matthew Smith. Replication: On the ecological validity of online security developer studies: Exploring deception in a password-storage study with freelancers. In *Proceedings of the 16th Symposium on Usable Privacy and Security, SOUPS 2020*, pages 165–184, 2020.

[60] Anastasia Danilova, Alena Naiakshina, Stefan Horstmann, and Matthew Smith. Do you really code? Designing and evaluating screening questions for online surveys with programmers. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, page 537–548, 2021.

[61] Anastasia Danilova, Alena Naiakshina, and Matthew Smith. One size does not fit all: A grounded theory and online survey study of developer preferences for security warning types. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, page 136–148. ACM, 2020.

[62] Partha Das Chowdhury, Joseph Hallett, Nikhil Patnaik, Mohammad Tahaei, and Awais Rashid. Developers are neither enemies nor users: They are collaborators. In *2021 IEEE Secure Development Conference (SecDev)*, page 47–55, 2021.

[63] Nicolai Davidsson, Andre Pawlowski, and Thorsten Holz. Towards automated application-specific software stacks. In *Computer Security – ESORICS 2019: 24th European Symposium on Research in Computer Security, Proceedings, Part II*, page 88–109. Springer-Verlag, 2019.

[64] DAXX. How many software developers are in the US and the world? `https://www.daxx.com/blog/development-trends/number-software-developers-world`, 2020.

[65] Giuseppe Pino De Francesco. The General Data Protection Regulation's practical impact on software architecture. *Computer*, 52(4):32–39, Apr 2019.

[66] Autumn Demberger. Merck Awarded $1.4 Billion for NotPetya After 5 Years of Legal Battle. `https://riskandinsurance.com/merck-awarded-1-4-billion-for-notpetya-after-5-years-of-legal-battle/`, 2022.

[67] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. Keep me updated: An empirical study of third-party library updatability on Android. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17*, page 2187–2200. ACM Press, 2017.

[68] Ben Dickson. AV Oracle: New hacking technique leverages antivirus to steal secrets. `https://portswigger.net/daily-swig/av-oracle-new-hacking-technique-leverages-antivirus-to-steal-secrets`, 2019.

[69] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. Scaling static analyses at Facebook. *Communications of the ACM*, 62(8):62–70, 2019.

[70] Dale Dominey-Howes. The Tonga volcanic eruption reveals the vulnerabilities in our global telecommunication system. `https://techxplore.com/news/2022-01-tonga-volcanic-eruption-reveals-vulnerabilities.html`, 2022.

[71] William J Drake, Vinton G. Cerf, and Wolfgang Kleinwächter. Internet fragmentation: An overview. *World Economic Forum*, page 80, 2016.

[72] Po-Yi Du, Ning Zhang, Mohammedreza Ebrahimi, Sagar Samtani, Ben Lazarine, Nolan Arnold, Rachael Dunn, Sandeep Suntwal, Guadalupe Angeles, Robert Schweitzer, and et al. Identifying, collecting, and presenting hacker community data: Forums, IRC, carding shops, and DNMs. In *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, page 70–75, 2018.

[73] Briana Duggan. Uganda shuts down social media; candidates arrested on election day. `https://edition.cnn.com/2016/02/18/world/uganda-election-social-media-shutdown/index.html`, 2016.

[74] Jerry Dunleavy. FCC designates Huawei and four other Chinese tech companies as national security threats. `https://www.washingtonexaminer.com/news/fcc-huawei-four-other-chinese-tech-companies-national-security-threats`, 2021.

[75] Editorial. It's time to talk about ditching statistical significance. *Nature*, 567:283, 2019.

[76] Wassim El-Hajj, Ghassen Ben Brahim, Hazem Hajj, Haidar Safa, and Ralph Adaimy. Security-by-construction in web applications development via database annotations. *Computers & Security*, 59(C):151–165, 2016.

[77] William Enck and Laurie Williams. Top five challenges in software supply chain security: Observations from 30 industry and government organizations. *IEEE Security & Privacy*, 20(2):96–100, 2022.

[78] Enterprise Ireland. SME Definition. `https://www.enterprise-ireland.com/en/About-Us/Our-Clients/SME-Definition.html`, 2023.

[79] Ksenia Ermoshina, Benjamin Loveluck, and Francesca Musiani. A market of black boxes: The political economy of Internet surveillance and censorship in Russia. *Journal of Information Technology & Politics*, 19(1):18–33, 2021.

[80] Tiago Espinha Gasiba, Iosif Andrei-Cristian, Ulrike Lechner, and Maria Pinto-Albuquerque. Raising security awareness of cloud deployments using infrastructure as code through cybersecurity challenges. In *The 16th International Conference on Availability, Reliability and Security*, page 1–8. ACM, 2021.

[81] Tiago Espinha Gasiba, Ulrike Lechner, Maria Pinto-Albuquerque, and Daniel Méndez. Is secure coding education in the industry needed? An investigation

through a large scale survey. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, page 241–252, 2021.

[82] EU. Directive (EU) 2016/1148 of the European Parliament and of the Council. https://eur-lex.europa.eu/eli/dir/2016/1148/oj, 2016.

[83] European Commission. New EU cybersecurity rules ensure more secure hardware and software. https://digital-strategy.ec.europa.eu/en/news/new-eu-cybersecurity-rules-ensure-more-secure-hardware-and-software-products, 2023.

[84] European Union Agency for Cybersecurity. *ENISA threat landscape 2021: April 2020 to mid July 2021*. Publications Office, 2021.

[85] Sead Fadilpašić. ICANN rejects call to remove Russian domains from the Internet. https://www.techradar.com/news/icann-rejects-call-to-remove-russian-domains-from-the-internet, 2022.

[86] Federal Communications Commission (FCC). Circuit capacity data for U.S.-international submarine cables. https://www.fcc.gov/international/circuit-capacity-data-us-international-submarine-cables, 2022.

[87] Federal Office for Information Security. What are Critical Infrastructures? https://www.bsi.bund.de/EN/Themen/KRITIS-und-regulierte-Unternehmen/Kritische-Infrastrukturen/Allgemeine-Infos-zu-KRITIS/allgemeine-infos-zu-kritis_node.html, 2024.

[88] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack Overflow considered harmful? The impact of copy&paste on Android application security. In *2017 IEEE Symposium on Security and Privacy (SP)*, page 121–136, 2017.

[89] Felix Fischer, Yannick Stachelscheid, and Jens Grossklags. The effect of Google search on software security: Unobtrusive security interventions via content re-ranking. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, page 3070–3084. ACM, 2021.

[90] FLD. Report: Jordanian human rights defenders and journalists hacked with Pegasus spyware. https://www.frontlinedefenders.org/en/statement-report/report-jordanian-human-rights-defenders-and-journalists-hacked-pegasus-spyware, 2022.

[91] William Foddy. *Constructing Questions for Interviews and Questionnaires: Theory and Practice in Social Research*. Cambridge University Press, 1994.

[92] Neil Ford. List of Data Breaches and Cyber Attacks in 2023. `https://www.itgovernance.co.uk/blog/list-of-data-breaches-and-cyber-attacks-in-2023`, 2023.

[93] Helene Fouquet. China's 7,500-mile undersea cable to Europe fuels Internet feud. `https://www.bloomberg.com/news/articles/2021-03-05/china-s-peace-cable-in-europe-raises-tensions-with-the-u-s`, 2021.

[94] Freedom House. Countries. `https://freedomhouse.org/countries/freedom-net/scores`, 2021.

[95] Kelsey R. Fulton, Daniel Votipka, Desiree Abrokwa, Michelle L. Mazurek, Michael Hicks, and James Parker. Understanding the how and the why: Exploring secure development practices through a course competition. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, page 1141–1155. ACM, 2022.

[96] Conor Gallagher. Russian military drills pose strategic and environmental risks to Ireland. `https://www.irishtimes.com/news/ireland/irish-news/russian-military-drills-pose-strategic-and-environmental-risks-to-ireland-1.4784787`, 2022.

[97] Lisa Geierhaas, Anna-Marie Ortloff, Matthew Smith, and Alena Naiakshina. Let's hash: Helping developers with password security. In *Proceedings of the Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022)*, page 503–522, 2022.

[98] Dan Goodin. FCC puts Kaspersky on security threat list, says it poses "unacceptable risk". `https://arstechnica.com/information-technology/2022/03/fcc-puts-kaspersky-on-security-threat-list-says-it-poses-unacceptable-risk/`, 2022.

[99] Peter Leo Gorski, Yasemin Acar, Luigi Lo Iacono, and Sascha Fahl. Listen to developers! A participatory design study on security warnings for cryptographic APIs. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, page 1–13. ACM, 2020.

[100] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Moeller, Yasemin Acar, and Sascha Fahl. Developers deserve security warnings, too: On the effect of integrated security advice on cryptographic API

misuse. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, page 265–281, 2018.

[101] Peter Leo Gorski, Sebastian Moller, Stephan Wiefling, and Luigi Lo Iacono. "I just looked for the solution!" On integrating security-relevant information in non-security API documentation to support secure coding practices. *IEEE Transactions on Software Engineering*, 48(9), 2021.

[102] Matthew Green and Matthew Smith. Developers are not the enemy! The need for usable security APIs. *IEEE Security and Privacy*, 14(5):40–46, 2016.

[103] Andy Greenberg. The full story of the stunning RSA hack can finally be told. `https://www.wired.com/story/the-full-story-of-the-stunning-rsa-hack-can-finally-be-told/`, 2021.

[104] Samuel Greengard. The worsening state of ransomware. *Communications of the ACM*, 64(4):15–17, 2021.

[105] Jonathan Greig. Average time to fix critical cybersecurity vulnerabilities is 205 days: report. `https://www.zdnet.com/article/average-time-to-fix-critical-cybersecurity-vulnerabilities-is-205-days-report/`, 2019.

[106] Egon G Guba. Criteria for assessing the trustworthiness of naturalistic inquiries. *ECTJ*, 29(2):75–91, 1981.

[107] Juan Andrés Guerrero-Saade. AcidRain — a modem wiper rains down on Europe. `https://www.sentinelone.com/labs/acidrain-a-modem-wiper-rains-down-on-europe/`, 2022.

[108] Marco Gutfleisch, Markus Schöps, Sibel Sayin, Frederic Wende, and Martina Angela Sasse. Putting security on the table: The digitalisation of security tabletop games and its challenging aftertaste. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, page 217–222, 2022.

[109] Mark Guzdial. Does contextualized computing education help? *ACM Inroads*, 1(4):4–6, Dec 2010.

[110] Joseph Hallett, Nikhil Patnaik, Benjamin Shreeve, and Awais Rashid. "Do this! Do that!, And nothing will happen" Do specifications lead to securely stored passwords? In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, page 486–498. IEEE, 2021.

[111] Julie Haney, Mary Theofanos, Yasemin Acar, and Sandra Spickard Prettyman. "We make it a big deal in the company": Security mindsets in organizations that develop cryptographic products. In *Proceedings of the Fourteenth Symposium on Usable Privacy and Security*, page 357–373. USENIX Association, 2018.

[112] Julie M. Haney and Wayne G. Lutters. 'It's scary... it's confusing... it's dull': How cybersecurity advocates overcome negative perceptions of security. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, page 411–425, 2018.

[113] Robbie Harb. India bans a further 118 Chinese apps as physical and online tensions escalate. `https://www.theregister.com/2020/09/03/india_bans_chinese_apps/`, 2020.

[114] Shane Harrison. Cyber attack: When will the Irish health service get a resolution? `https://www.bbc.com/news/world-europe-57193160`, 2020.

[115] Chad Heitzenrater and Andrew Simpson. A case for the economics of secure software development. In *Proceedings of the 2016 New Security Paradigms Workshop*, page 92–105. ACM, 2016.

[116] Simon Hendery. Email hack prompts call for Microsoft to make security logs free. `https://www.scmagazine.com/analysis/email-hack-prompts-call-for-microsoft-to-make-security-logs-free`, 2023.

[117] HM Government. Government Cyber Security Strategy, 2022.

[118] Stacie Hoffmann, Dominique Lazanski, and Emily Taylor. Standardising the splinternet: how China's technical standards could fragment the internet. *Journal of Cyber Policy*, 5:239–264, 2020.

[119] Alicia Hope. Ransomware trends and predictions for 2024. `https://www.cpomagazine.com/cyber-security/ransomware-trends-and-predictions-for-2024/`, 2024.

[120] HR For Ukraine. Companies suspending operations in Russia and Belarus. `https://hrforukraine.notion.site/Companies-Suspending-Operations-in-Russia-and-Belarus-93d42cbb7a234438b663bded9f1f7f4c`, 2022.

[121] Kaifeng Huang, Bihuan Chen, Congying Xu, Ying Wang, Bowen Shi, Xin Peng, Yijian Wu, and Yang Liu. Characterizing usages, updates and risks of third-party libraries in Java projects. *Empirical Software Engineering*, 27(4), 2022.

[122] Sylvia Hui and Eric Tucker. US and UK go after Chinese hackers accused of state-backed operation against politicians, dissidents. https://apnews.com/article/uk-china-cyberattacks-parliament-election-770e7b00454b63ad424000feecddd0c1, 2024.

[123] Human Rights Watch. Belarus: Internet disruptions, online censorship. https://www.hrw.org/news/2020/08/28/belarus-internet-disruptions-online-censorship, 2020.

[124] Shane Huntley. Buying Spying: How the commercial surveillance industry works and what can be done about it. https://blog.google/threat-analysis-group/commercial-surveillance-vendors-google-tag-report/, 2024.

[125] Thorvald Hærem, Brian T. Pentland, and Kent D. Miller. Task complexity: Extending a core concept. *Academy of Management Review*, 40(3):446–460, 2015.

[126] Nasif Imtiaz, Brendan Murphy, and Laurie Williams. How do developers act on static analysis alerts? An empirical study of Coverity usage. In *IEEE 30th International Symposium on Software Reliability Engineering*, pages 323–333, 2019.

[127] ISO. ISO/IEC 29110 Series. https://committee.iso.org/sites/jtc1sc7/home/projects/flagship-standards/isoiec-29110-series.html, 2023.

[128] Martin Gilje Jaatun. Hunting for aardvarks: Can software security be measured? In Gerald Quirchmayr, Josef Basl, Ilsun You, Lida Xu, and Edgar Weippl, editors, *Multidisciplinary Research and Practice for Information Systems*, page 85–92. Springer, 2012.

[129] Martin Gilje Jaatun, Daniela S. Cruzes, Karin Bernsmed, Inger Anne Tøndel, and Lillian Røstad. Software security maturity in public organisations. In Javier Lopez and Chris J. Mitchell, editors, *Information Security*, Lecture Notes in Computer Science, page 120–138. Springer International Publishing, 2015.

[130] Martin Gilje Jaatun and Daniela Soares Cruzes. Care and feeding of your security champion. In *2021 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, page 1–7. IEEE, 2021.

[131] Jan Jancar, Marcel Fourné, Daniel De Almeida Braga, Mohamed Sabt, Peter Schwabe, Gilles Barthe, Pierre-Alain Fouque, and Yasemin Acar. "They're not that hard to mitigate": What cryptographic library developers think about timing

attacks. In *2022 IEEE Symposium on Security and Privacy (SP)*, page 632–649, 2022.

[132] Sam Jarman. How pulling out of Russia's internet could further isolate its citizens. https://bigthink.com/the-present/russian-internet-runet/?utm_medium=Social&utm_source=Twitter#Echobox=1648510641-2, 2022.

[133] Connor Jones. Two years on, 1 in 4 apps still vulnerable to Log4Shell. https://www.theregister.com/2023/12/11/log4j_vulnerabilities/, 2023.

[134] Sri Lakshmi Kanniah and Mohd Naz'ri Mahrin. Secure software development practice adoption model: A Delphi study. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 10(2–82–8):71–75, July 2018.

[135] Harjot Kaur, Sabrina Amft, Daniel Votipka, Yasemin Acar, and Sascha Fahl. Where to recruit for security development studies from: Comparing six software developer samples. In *31st USENIX Security Symposium (USENIX Security 22)*, page 24, 2022.

[136] Mannat Kaur, Michel van Eeten, Marijn Janssen, Kevin Borgolte, and Tobias Fiebig. Human factors in security research: Lessons learned from 2008-2018. *arXiv preprint arXiv:2103.13287*, 2021.

[137] Stephanie Kirchgaessner. Dozens in Jordan targeted by authorities using NSO spyware, report finds. https://www.theguardian.com/world/2024/feb/01/jordan-targeting-lawyers-jouranlists-israeli-nso-spyware-report, 2024.

[138] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering, 2007.

[139] Michael T. Klare. Cyber battles, nuclear outcomes? Dangerous new pathways to escalation. https://www.armscontrol.org/act/2019-11/features/cyber-battles-nuclear-outcomes-dangerous-new-pathways-escalation, 2019.

[140] Maria Korolov. Cisco router vulnerability puts network segmentation at risk. https://www.datacenterknowledge.com/security/cisco-router-vulnerability-puts-network-segmentation-risk, 2020.

[141] Brian Krebs. Try this one weird trick Russian hackers hate. https://krebsonsecurity.com/2021/05/try-this-one-weird-trick-russian-hackers-hate, 2021.

[142] Brian Krebs. Pro-Ukraine 'protestware' pushes antiwar ads, geo-targeted malware. https://krebsonsecurity.com/2022/03/pro-ukraine-protestware-pushes-antiwar-ads-geo-targeted-malware/, 2022.

[143] Richard A. Krueger and Mary Anne Casey. *Focus Groups, A Practical Guide for Applied Research*. SAGE Publications, Inc., 5 edition, 2000.

[144] Ravie Lakshmanan. Exclusive: SonicWall hacked using 0-day bugs in its own VPN product. https://thehackernews.com/2021/01/exclusive-sonicwall-hacked-using-0-day.html, 2021.

[145] Ravie Lakshmanan. Chinese hackers exploit VMWare zero-day to backdoor Windows and Linux systems. https://thehackernews.com/2023/06/chinese-hackers-exploit-vmware-zero-day.html, 2023.

[146] Enrique Larios-Vargas, Omar Elazhary, Soroush Yousefi, Derek Lowlind, Michael L. W. Vliek, and Margaret-Anne Storey. DASP: A framework for driving the adoption of software security practices. *IEEE Transactions on Software Engineering*, page 1–29, 2023.

[147] Frank Li and Vern Paxson. A large-scale empirical study of security patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, page 2201–2215. ACM, 2017.

[148] Kevin Limonier, Frédérick Douzet, Louis Pétiniaud, Loqman Salamatian, and Kave Salamatian. Mapping the routes of the Internet for geopolitics: The case of Eastern Ukraine. *First Monday*, 2021.

[149] Luigi Lo Iacono and Peter Leo Gorski. I do and I understand. Not yet true for security APIs. So sad. In *Proceedings 2nd European Workshop on Usable Security*, page 11. Internet Society, 2017.

[150] Tamara Lopez, Helen Sharp, Thein Tun, Arosha Bandara, Mark Levine, and Bashar Nuseibeh. Talking about security with professional developers. In *Proceedings of the Joint 7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice*, page 34–40. IEEE Press, 2019.

[151] Tamara Lopez, Helen Sharp, Thein Tun, Arosha Bandara, Mark Levine, and Bashar Nuseibeh. Security responses in software development. *ACM Transactions on Software Engineering and Methodology*, 2022.

[152] Tamara Lopez, Helen Sharp, Thein Tun, Arosha K. Bandara, Mark Levine, and Bashar Nuseibeh. "Hopefully we are mostly secure": Views on secure code in professional practice. In *Proceedings of the 12th International Workshop on Cooperative and Human Aspects of Software Engineering*, page 61–68. IEEE Press, 2019.

[153] Tamara Lopez, Thein Tun, Arosha Bandara, Mark Levine, Bashar Nuseibeh, and Helen Sharp. An anatomy of security conversations in Stack Overflow. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society*, page 31–40. IEEE Press, 2019.

[154] Tamara Lopez, Thein T. Tun, Arosha K. Bandara, Mark Levine, Bashar Nuseibeh, and Helen Sharp. Taking the middle path: Learning about security through online social interaction. *IEEE Software*, 37(1):25–30, 2020.

[155] Thomas Loruenser, Henrich C. Pöhls, Leon Sell, and Thomas Laenger. CryptSDLC: Embedding cryptographic engineering into secure software development lifecycle. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*. ACM, 2018.

[156] Linghui Luo, Martin Schäf, Daniel Sanchez, and Eric Bodden. IDE support for cloud-based static analyses. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, page 1178–1189. ACM, 2021.

[157] Sean Lyngaas. 'I can fight with a keyboard': How one Ukrainian IT specialist exposed a notorious Russian ransomware gang. `https://edition.cnn.com/2022/03/30/politics/ukraine-hack-russian-ransomware-gang/index.html`, 2022.

[158] Sean Lyngaas. Cybercriminals raked in record $1.1 billion in ransom payments in 2023. `https://edition.cnn.com/2024/02/07/politics/cybercriminals-record-ransom-payments-2023/index.html`, 2024.

[159] Jamie MacColl, Pia Hüsch, Gareth Mott, James Sullivan, Jason R. C. Nurse, Sarah Turner, and Nandita Pattnaik. *Ransomware: Victim Insights on Harms to Individuals, Organisations and Society*. RUSI, 2024.

[160] Ali B. Mahmoud, William D. Reisel, Leonora Fuxman, and Iris Mohr. A motivational standpoint of job insecurity effects on organizational citizenship behaviors: A generational study. *Scandinavian Journal of Psychology*, 62(2):267–275, 2021.

[161] Hari Prasad Mariswam. 2024 ransomware attacks on healthcare: A wake-up call for healthcare data security. `https://www.skyhighsecurity.com/in`

dustry-perspectives/2024-ransomware-attacks-on-healthcare.html, 2024.

[162] Fabio Massacci. Is "Deny Access" a valid "Fail-Safe Default" principle for building security in cyberphysical systems? *IEEE Security and Privacy*, 17(5):90–93, 2019.

[163] Declan McCullagh. How Pakistan knocked YouTube offline (and how to make sure it never happens again). https://www.cnet.com/news/how-pakistan-knocked-youtube-offline-and-how-to-make-sure-it-never-happens-again/, 2008.

[164] Will McCurdy. Lazarus hackers are using Log4j to hack US energy companies. https://www.techradar.com/news/lazarus-hackers-are-using-log4j-to-hack-us-energy-companies, 2023.

[165] Gary McGraw. Software security. *IEEE Security and Privacy*, 2(5):80–83, 2004.

[166] Adam McNeil. How did the WannaCry ransomworm spread? https://blog.malwarebytes.com/cybercrime/2017/05/how-did-wannacry-ransomworm-spread/, 2019.

[167] Jezreel Mejía, Mirna Muñoz, Perla Maciel-Gallegos, and Yadira Quiñonez. Proposal to integrate security practices into the ISO/IEC 29110 standard to develop mobile apps. In *New Perspectives in Software Engineering*, page 29–40. Springer International Publishing, 2022.

[168] Per Håkon Meland, Yara Fareed Fahmy Bayoumy, and Guttorm Sindre. The Ransomware-as-a-Service economy within the darknet. *Computers & Security*, 92:101762, 2020.

[169] Na Meng, Stefan Nagy, Danfeng (Daphne) Yao, Wenjie Zhuang, and Gustavo Arango Argoty. Secure coding practices in Java: challenges and vulnerabilities. In *Proceedings of the 40th International Conference on Software Engineering*, page 372–383. ACM, 2018.

[170] Microsoft. Microsoft Security Development Lifecycle. https://www.microsoft.com/en-us/securityengineering/sdl/, 2022.

[171] Sammy Migues, John Steven, and Mike Ware. BSIMM. https://www.bsimm.com/, 2022.

[172] Vaishnavi Mohan and Lotfi Ben Othmane. SecDevOps: Is it a marketing buzzword? Mapping research on security in DevOps. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, page 542–547, 2016.

[173] Azadeh Mokhberi and Konstantin Beznosov. SoK: Human, organizational, and technological dimensions of developers' challenges in engineering secure software. In *European Symposium on Usable Security 2021*, page 59–75, Karlsruhe Germany, 2021. ACM.

[174] Jose Andre Morales, Thomas P. Scanlon, Aaron Volkmann, Joseph Yankel, and Hasan Yasar. Security impacts of sub-optimal DevSecOps implementations in a highly regulated environment. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, page 8. ACM, 2020.

[175] Patrick Morrison. A security practices evaluation framework. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, page 935–938. IEEE Press, 2015.

[176] Patrick Morrison, David Moye, Rahul Pandita, and Laurie Williams. Mapping the field of software life cycle security metrics. *Information and Software Technology*, 102(May):146–159, 2018.

[177] Patrick Morrison, Benjamin H. Smith, and Laurie Williams. Measuring security practice use: A case study at IBM. In *2017 IEEE/ACM 5th International Workshop on Conducting Empirical Studies in Industry (CESI)*, page 16–22. IEEE Press, 2017.

[178] Marcus R. Munafò and George Davey Smith. Robust research needs many lines of evidence. *Nature*, 553(7689):399–401, 2018.

[179] Phil Muncaster. Half of government security incidents caused by missing patches. `https://www.infosecurity-magazine.com/news/half-government-incidents-missing/`, 2021.

[180] Emerson Murphy-Hill, Ciera Jaspan, Caitlin Sadowski, David Shepherd, Michael Phillips, Collin Winter, Andrea Knight, Edward Smith, and Matt Jorde. What predicts software developers' productivity? *IEEE Transactions on Software Engineering*, page 1–23, 2019.

[181] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. Jumping through hoops: Why do Java developers struggle with cryptography APIs? In *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*, page 935–946. ACM Press, 2016.

[182] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, and Matthew Smith. On conducting security developer studies with CS students: Examining a password-storage study with CS students, freelancers, and company developers. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, page 1–13. ACM, 2020.

[183] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, and Matthew Smith. "If you want, I can store the encrypted password": A password-storage field study with freelance developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, page 1–12. ACM, 2019.

[184] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. Why do developers get password storage wrong? A qualitative usability study. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, page 311–328. ACM, 2017.

[185] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, and Matthew Smith. Deception task design in developer password studies: Exploring a student sample. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 297–313. USENIX Association, 2018.

[186] National Security and Defense Council of Ukraine. The NCCC at the NSDC of Ukraine warns of a cyberattack on the document management system of state bodies. `https://www.rnbo.gov.ua/en/Diialnist/4823.html`, 2021.

[187] Lily Hay Newman. The leaked NSA spy tool that hacked the world. `https://www.wired.com/story/eternalblue-leaked-nsa-spy-tool-hacked-world/`, 2018.

[188] Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. A stitch in time: Supporting Android developers in writing secure code. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17*, page 1065–1077. ACM Press, 2017.

[189] Lisa Nguyen Quang Do and Eric Bodden. Gamifying static analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, page 714–718. ACM, 2018.

[190] Arian Akhavan Niaki, Shinyoung Cho, Zachary Weinberg, Nguyen Phong Hoang, Abbas Razaghpanah, Nicolas Christin, and Phillipa Gill. ICLab: A global, longitudinal Internet censorship measurement platform. In *2020 IEEE Symposium on Security and Privacy (SP)*, page 135–151, 2020.

[191] Cybereason Nocturnus. Cybereason vs. DarkSide ransomware. `https://www.cybereason.com/blog/cybereason-vs-darkside-ransomware`, 2021.

[192] Geoff Norman. Likert scales, levels of measurement and the "laws" of statistics. *Advances in Health Sciences Education*, 15(5):625–632, Dec 2010.

[193] Aimee O'Driscoll. 25+ cyber security vulnerability statistics and facts of 2021. `https://www.comparitech.com/blog/information-security/cybersecurity-vulnerability-statistics/`, 2021.

[194] Marc Ohm, Arnold Sykosch, and Michael Meier. Towards detection of software supply chain attacks by forensic artifacts. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*. ACM, 2020.

[195] Daniela Oliveira, Marissa Rosenthal, Nicole Morin, Kuo-Chuan Yeh, Justin Cappos, and Yanyan Zhuang. It's the psychology stupid: How heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots. In *Proceedings of the 30th Annual Computer Security Applications Conference*, page 296–305. ACM Press, 2014.

[196] Daniela Seabra Oliveira, Tian Lin, Muhammad Sajidur Rahman, Rad Akefirad, Donovan Ellis, Eliany Perez, Rahul Bobhate, Lois A. DeLong, Justin Cappos, and Yuriy Brun. API blindspots: Why experienced developers write vulnerable code. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, page 315–328, 2018.

[197] OpenAI. ChatGPT. `https://openai.com/chatgpt`, 2024.

[198] OWASP. SAMM. `https://www.opensamm.org/`, 2021.

[199] OWASP. Owasp developer guide. `https://owasp.org/www-project-developer-guide/release/`, 2024.

[200] Tosin Daniel Oyetoyan, Daniela Soares Cruzes, and Martin Gilje Jaatun. An empirical study on the relationship between software security skills, usage and training needs in Agile settings. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, page 548–555, 2016.

[201] Tosin Daniel Oyetoyan, Bisera Milosheska, Mari Grini, and Daniela Soares Cruzes. Myths and facts about static application security testing tools: An action research at Telenor Digital. In *Agile Processes in Software Engineering and Extreme Programming: 19th International Conference, XP 2018*, page 86–103. Springer International Publishing, 2018.

[202] Hernan Palombo, Armin Ziaie Tabari, Daniel Lende, Jay Ligatti, and Xinming Ou. An ethnographic understanding of software (in)security and a co-creation model to improve secure software development. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, pages 205–220. USENIX Association, 2020.

[203] Guy Paré, Marie-Claude Trudel, Mirou Jaana, and Spyros Kitsiou. Synthesizing information systems knowledge: A typology of literature reviews. *Information & Management*, 52(2):183–199, 2015.

[204] James Parker, Michael Hicks, Andrew Ruef, Michelle L. Mazurek, Dave Levin, Daniel Votipka, Piotr Mardziel, and Kelsey R. Fulton. Build It, Break It, Fix It: Contesting secure development. *ACM Transactions on Privacy and Security*, 23(2):10:1–10:36, 2020.

[205] David Parsons, Teo Susnjak, and Anuradha Mathrani. The Software Developer Cycle: Career demographics and the market clock: or, is SQL the new COBOL? In *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference*, page 86–90. ACM, 2015.

[206] Jeff Parsons. Russians 'to be disconnected from global Internet from Friday'. `https://metro.co.uk/2022/03/07/russia-preparing-to-disconnect-from-global-internet-on-march-11-16230918/`, 2022.

[207] Ivan Pashchenko, Duc-Ly Vu, and Fabio Massacci. A qualitative study of dependency management and its security implications. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, page 1513–1531. ACM, 2020.

[208] Sergio Pastrana and Guillermo Suarez-Tangil. A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth. In *Proceedings of the Internet Measurement Conference*, page 73–86. ACM, Oct 2019.

[209] Nikhil Patnaik, Andrew C. Dwyer, Joseph Hallett, and Awais Rashid. SLR: From Saltzer & Schroeder to 2021... *ACM Transactions on Software Engineering and Methodology*, 32(3), 2022.

[210] Nikhil Patnaik, Joseph Hallett, and Awais Rashid. Usability smells: An analysis of developers' struggle with crypto libraries. In *Proceedings of the Fifteenth Symposium on Usable Privacy and Security*, page 245–257, 2019.

[211] Rajshakhar Paul, Asif Kamal Turzo, and Amiangshu Bosu. Why security defects go unnoticed during code reviews? A case-control study of the Chromium OS project. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, page 1373–1385, 2021.

[212] Aleksandra Pawlicka, Michał Choraś, and Marek Pawlicki. Cyberspace threats: Not only hackers and criminals. Raising the awareness of selected unusual cyberspace actors - cybersecurity researchers' perspective. In *Proceedings of the*

15th International Conference on Availability, Reliability and Security, page 1–11. ACM, 2020.

[213] PCI Security Standards Council. PCI DSS. `https://www.pcisecuritystandards.org/document_library/?category=pcidss&document=pci_dss`.

[214] Anelis Pereira-Vale, Eduardo B. Fernandez, Raul Monge, Hernan Astudillo, and Gaston Marquez. Security in microservice-based systems: A multivocal literature review. *Computers and Security*, 103:22, 2021.

[215] Ani Petrosyan. Number of common IT security vulnerabilities and exposures (CVEs) worldwide from 2009 to 2023 YTD. `https://www.statista.com/statistics/500755/worldwide-common-vulnerabilities-and-exposures/`, 2023.

[216] Olgierd Pieczul, Simon Foley, and Mary Ellen Zurko. Developer-centered security and the symmetry of ignorance. In *Proceedings of the 2017 New Security Paradigms Workshop*, page 46–56. ACM Press, 2017.

[217] Vladislav Pikulin, Daiki Kubo, Kaveesha Nissanka, Sadeeptha Bandara, Muhammad A Shamsiemon, Arissha Yasmin, Asangi Jayatilaka, Anuradha Madugalla, and Tanjila Kanij. Towards developer-centered secure coding training. In *38th IEEE/ACM ASE Workshops*, page 24–31, 2023.

[218] Scott Piper. Setting secure defaults on AWS and avoiding misconfigurations. `https://www.wiz.io/blog/how-to-set-secure-defaults-on-aws`, 2024.

[219] Andreas Poller, Laura Kocksch, Katharina Kinder-Kurlanda, and Felix Anand Epp. First-time security audits as a turning point? Challenges for security practices in an industry software development team. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, page 1288–1294. ACM, 2016.

[220] Andreas Poller, Laura Kocksch, Sven Türpe, Felix Anand Epp, and Katharina Kinder-Kurlanda. Can security become a routine? A study of organizational change in an Agile software development group. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, page 2489–2503. ACM, 2017.

[221] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2022.

[222] Sazzadur Rahaman, Gang Wang, and Danfeng Daphne Yao. Security certification in payment card industry: Testbeds, measurements, and recommendations. *ACM CCS*, page 481–498, 2019.

[223] Akond Rahman, Chris Parnin, and Laurie Williams. The seven sins: Security smells in infrastructure as code scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, page 164–175, 2019.

[224] Md Rayhanur Rahman, Nasif Imtiaz, Margaret-Anne Storey, and Laurie Williams. Why secret detection tools are not enough: It's not just about false positives - an industrial case study. *Empirical Software Engineering*, 27(3), 2022.

[225] Pawel Rajba. Challenges and mitigation approaches for getting secured applications in an enterprise company. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, page 6. ACM, 2018.

[226] Irum Rauf, Tamara Lopez, Helen Sharp, Marian Petre, Thein Tun, Mark Levine, John Towse, Dirk van der Linden, Awais Rashid, and Bashar Nuseibeh. Influences of developers' perspectives on their engagement with security in code. In *Proceedings of the 15th International Conference on Cooperative and Human Aspects of Software Engineering*, page 86–95. ACM, 2022.

[227] Irum Rauf, Marian Petre, Thein Tun, Tamara Lopez, Paul Lunn, Dirk van der Linden, John Towse, Helen Sharp, Mark Levine, Awais Rashid, and et al. The case for adaptive security interventions. *ACM Transactions on Software Engineering and Methodology*, 31(1):9:1–9:52, 2021.

[228] Irum Rauf, Dirk van der Linden, Mark Levine, John Towse, Bashar Nuseibeh, and Awais Rashid. Security but not for security's sake: The impact of social considerations on app developers' choices. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, page 141–144. ACM, 2020.

[229] Claire Reilly. Ransomware shuts down 1 in 5 small businesses after it hits. `https://www.cnet.com/news/privacy/malwarebytes-state-of-ransomware-shutting-down-1-in-5-affected-small-businesses/`, 2024.

[230] Reuters. Israel ramps up scrutiny of police as NSO scandal spreads. `https://www.euronews.com/2022/02/07/us-israel-nso`, 2022.

[231] National Law Review. Federal government outlines new security and attestation requirements for software. `https://www.natlawreview.com/artic`

le/federal-government-outlines-new-security-and-attestation-requirements-software, 2023.

[232] Kalle Rindell, Jukka Ruohonen, and Sami Hyrynsalmi. Surveying secure software development practices in Finland. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*. ACM, 2018.

[233] Knut Rolland, Torgeir Dingsoyr, Brian Fitzgerald, and Klaas-Jan Stol. Problematizing Agile in the large: Alternative assumptions for large-scale Agile development. In *39th International Conference on Information Systems*, pages 1–21. Association for Information Systems (AIS), 2016.

[234] Seth Rosenblatt. How the shady zero-day sales game is evolving. https://www.darkreading.com/edge-articles/how-the-shady-zero-day-sales-game-is-evolving, 2021.

[235] RT. Russia to launch 'independent internet' for BRICS nations - report. https://www.rt.com/russia/411156-russia-to-launch-independent-internet/, 2017.

[236] Andrew Ruef, Michael Hicks, James Parker, Dave Levin, Michelle L. Mazurek, and Piotr Mardziel. Build It, Break It, Fix It: Contesting secure development. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, page 690–703. ACM, 2016.

[237] Ita Ryan, Utz Roedig, and Klaas-Jan Stol. Understanding developer security archetypes. In *International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*. Association of Computer Machinery, 2021.

[238] Ita Ryan, Utz Roedig, and Klaas-Jan Stol. Insecure software on a fragmenting Internet. In *2022 Cyber Research Conference - Ireland (Cyber-RCI)*, 2022.

[239] Ita Ryan, Utz Roedig, and Klaas-Jan Stol. Measuring secure coding practice and culture: A finger pointing at the moon is not the moon. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023.

[240] Ita Ryan, Utz Roedig, and Klaas-Jan Stol. Measuring secure coding practice and culture: Supplementary material. https://osf.io/mz29b, 2023.

[241] Ita Ryan, Utz Roedig, and Klaas-Jan Stol. Unhelpful assumptions in software security research. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2023.

[242] Ita Ryan, Utz Roedig, and Klaas-Jan Stol. Unhelpful assumptions in software security research: Study protocol. https://osf.io/7a2dr/, 2023.

[243] Ita Ryan, Klaas-Jan Stol, and Utz Roedig. The state of secure coding practice: Small organisations and 'lone, rogue coders'. In *2023 IEEE/ACM 4th International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*, page 37–44, 2023.

[244] Ita Ryan, Klaas-Jan Stol, and Utz Roedig. Studying secure coding in the laboratory: Why, what, where, how, and who? In *2023 IEEE/ACM 4th International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*, page 23–30, 2023.

[245] Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, and Ciera Jaspan. Lessons from building static analysis tools at Google. *Communications of the ACM*, 61(4):58–66, 2018.

[246] SAFECode. SAFECode Homepage. `https://safecode.org/`, 2022.

[247] Merve Sahin, Tolga Ünlü, Cédric Hébert, Lynsay A. Shepherd, Natalie Coull, and Colin Mc Lean. Measuring developers' web security awareness from attack and defense perspectives. In *2022 IEEE Security and Privacy Workshops (SPW)*, page 31–43, 2022.

[248] Lavanya Sajwan, James Noble, Craig Anslow, and Robert Biddle. Why do programmers do what they do? A theory of influences on security practices. In *Workshop on Usable Security and Privacy*, 2021.

[249] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[250] Raphael Satter and James Pearson. What is Volt Typhoon, the Chinese hacking group targeting US infrastructure? `https://www.reuters.com/technology/what-is-volt-typhoon-alleged-china-backed-hacking-group-2023-05-25/`, 2024.

[251] Benjamin Schneider, Mark G. Ehrhart, and William H. Macey. Organizational climate and culture. *Annual Review of Psychology*, 64(1):361–388, 2013.

[252] Security Encyclopedia. NotPetya. `https://www.hypr.com/notpetya/`, 2022.

[253] SentinelOne. EternalBlue exploit: What it is and how it works. `https://www.sentinelone.com/blog/eternalblue-nsa-developed-exploit-just-wont-die/`, 2019.

[254] Nicole Sganga and Mary Ilyushina. Russia arrests 14 alleged members of REvil ransomware gang. https://www.cbsnews.com/news/ransomware-russia-arrests-revil/, 2022.

[255] Mayank Sharma. US will give ransomware hacks similar priority to terrorist attacks. https://www.techradar.com/uk/news/us-will-give-ransomware-hacks-similar-priority-to-terrorist-attacks, 2021.

[256] Xinmei Shen. Apache Log4j bug: China's industry ministry pulls support from Alibaba Cloud for not reporting flaw to government first. https://www.scmp.com/tech/big-tech/article/3160670/apache-log4j-bug-chinas-industry-ministry-pulls-support-alibaba-cloud, 2021.

[257] Justin Sherman. The US-China battle over the Internet goes under the sea. https://www.wired.com/story/opinion-the-us-china-battle-over-the-internet-goes-under-the-sea/, 2020.

[258] Joao Silva. LinkedIn is the latest big tech service being banned from China. https://www.techspot.com/news/91754-linkedin-shutting-down-china-replaced-new-app-called.html, 2021.

[259] Jennifer Smith. Internet Atlas maps the physical internet to enhance security. https://news.wisc.edu/internet-atlas-maps-the-physical-internet-to-enhance-security/, 2021.

[260] Justin Smith, Brittany Johnson, Emerson Murphy-Hill, Bill Chu, and Heather R. Lipford. How developers diagnose potential security vulnerabilities with a static analysis tool. *IEEE Transactions on Software Engineering*, 45(9):877–897, 2019.

[261] Stack Overflow. Stack Overflow Developer Survey - Education. https://insights.stackoverflow.com/survey/2020#education, 2020.

[262] Stack Overflow. Stack Overflow Developer Survey - Gender. https://insights.stackoverflow.com/survey/2020/#demographics, 2020.

[263] Phil Stewart and Idrees Ali. How the US is preparing for a Chinese invasion of Taiwan. https://www.reuters.com/world/china/logistics-war-how-washington-is-preparing-chinese-invasion-taiwan-2024-01-31/, 2024.

[264] Klaas-Jan Stol and Brian Fitzgerald. The ABC of software engineering research. *ACM Transactions on Software Engineering and Methodology*, 27(3):1–51, 2018.

[265] Mohammad Tahaei, Alisa Frik, and Kami Vaniea. Privacy champions in software teams: Understanding their motivations, strategies, and challenges. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, page 16. ACM, 2021.

[266] Mohammad Tahaei, Adam Jenkins, Kami Vaniea, and Maria Wolters. "I don't know too much about it": On the security mindsets of computer science students. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 350–350, 2019.

[267] Mohammad Tahaei and Kami Vaniea. A survey on developer-centred security. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 129–138. IEEE, 2019.

[268] Mohammad Tahaei, Kami Vaniea, Konstantin (Kosta) Beznosov, and Maria K Wolters. Security notifications in static analysis tools: Developers' attitudes, comprehension, and ability to act on them. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, page 1–17. ACM, May 2021.

[269] Amanda Tanner, Alex Hinchliffe, and Doel Santos. Threat assessment: Blackcat ransomware. `https://unit42.paloaltonetworks.com/blackcat-ransomware/`, 2022.

[270] Richard D. Taylor. "Data localization": The internet in the balance. *Telecommunications Policy*, 44(8):102003, 2020.

[271] Dina Temple-Raston. Report: Beijing, Moscow step up efforts to control the Internet's backbone. `https://therecord.media/report-beijing-moscow-step-up-efforts-to-control-the-internets-backbone/`, 2021.

[272] Tom Tervoort. Zerologon: Unauthenticated domain controller compromise by subverting Netlogon cryptography (CVE-2020-1472). `https://www.secura.com/uploads/whitepapers/Zerologon.pdf`, 2020.

[273] The Japan Times. Kawasaki heavy hack may have targeted defense-linked information. `https://www.japantimes.co.jp/news/2020/12/28/national/kawasaki-heavy-data-breach/`, 2020.

[274] The Straits Times. Malaysia's armed forces confirms cyber-attack on network. `https://www.straitstimes.com/asia/se-asia/malaysias-armed-forces-confirms-cyber-attack-on-network`, 2020.

[275] The White House. Joint statement of the ministers and representatives from the counter ransomware initiative meeting October 2021. https://www.whitehouse.gov/briefing-room/statements-releases/2021/10/14/joint-statement-of-the-ministers-and-representatives-from-the-counter-ransomware-initiative-meeting-october-2021/, 2021.

[276] Tyler W. Thomas, Madiha Tabassum, Bill Chu, and Heather Lipford. Security during application development: An application security expert perspective. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 1–12. ACM, 2018.

[277] Christopher Thompson and David Wagner. A large-scale study of modern code review and security in open source projects. In *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, page 83–92. ACM, 2017.

[278] Dominika Tkaczyk, Paweł Szostek, Mateusz Fedoryszak, Piotr Jan Dendek, and Łukasz Bolikowski. CERMINE: automatic extraction of structured metadata from scientific literature. *International Journal on Document Analysis and Recognition (IJDAR)*, 18:317–335, 2015.

[279] Nora Tomas, Jingyue Li, and Huang Huang. An empirical study on culture, automation, measurement, and sharing of DevSecOps. In *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, page 1–8, 2019.

[280] Inger A. Tondel, Martin Gilje Jaatun, and Daniela Soares Cruzes. IT security is from Mars, software security is from Venus. *IEEE Security Privacy*, 18(4):48–54, 2020.

[281] Stephen Treacy. How Ukraine war is seeing a new and alarming kind of hybrid warfare. https://www.rte.ie/brainstorm/2024/0122/1427899-ukraine-russia-hybrid-warfare-cyberattacks-blackouts-psychological-damage/, 2024.

[282] Charlotte Trueman. Tech layoffs in 2023: A timeline. https://www.computerworld.com/article/3685936/tech-layoffs-in-2023-a-timeline.html, 2023.

[283] Trustwave SpiderLabs. Law enforcement collaboration has Eastern-European cybercriminals questioning whether there is a safe haven anymore. https://www.trustwave.com/en-us/resources/blogs/spiderlabs

-blog/law-enforcement-collaboration-has-eastern-european-cybercriminals-questioning-whether-there-is-a-safe-haven-anymore/, 2021.

[284] Anwesh Tuladhar, Daniel Lende, Jay Ligatti, and Xinming Ou. An analysis of the role of situated learning in starting a security culture in a software company. In *Proceedings of the Seventeenth Symposium on Usable Privacy and Security*, page 617–631, 2021.

[285] Inger A. Tøndel, Daniela S. Cruzes, Martin G. Jaatun, and Guttorm Sindre. Influencing the security prioritisation of an Agile software development project. *Computers and Security*, 118(C), 2022.

[286] Inger Anne Tøndel, Daniela Soares Cruzes, Martin Gilje Jaatun, and Kalle Rindell. The security intention meeting series as a way to increase visibility of software security decisions in Agile development projects. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES)*. ACM, 2019.

[287] Joe Uchill. After Conti backs war, ransomware gangs realize peril of patriotism amid infighting. https://www.scmagazine.com/analysis/ransomware/after-conti-backs-war-ransomware-gangs-realize-peril-of-patriotism-amid-infighting, 2022.

[288] UK Foreign and Commonwealth Office. Foreign office minister condemns Russia for NotPetya attacks. https://www.gov.uk/government/news/foreign-office-minister-condemns-russia-for-notpetya-attacks?mid=1, 2018.

[289] UK National Cyber Security Centre. Cyber Essentials. https://www.ncsc.gov.uk/cyberessentials/overview, 2023.

[290] Akond Ashfaque Ur Rahman and Laurie Williams. Security practices in DevOps. In *Proceedings of the Symposium and Bootcamp on the Science of Security*, page 109–111. ACM, 2016.

[291] Akond Ashfaque Ur Rahman and Laurie Williams. Software security in DevOps: Synthesizing practitioners' perceptions and practices. In *Proceedings of the International Workshop on Continuous Software Evolution and Delivery*, page 70–76. ACM, 2016.

[292] Tom Uren. Srsly Risky Biz: Thursday January 13. https://srslyriskybiz.substack.com/p/srsly-risky-biz-thursday-january-39c, 2022.

[293] U.S. Cybersecurity & Infrastructure Security Agency (CISA). Apache Log4j vulnerability guidance. https://www.cisa.gov/uscert/apache-log4j-vulnerability-guidance, 2021.

[294] U.S. Cybersecurity & Infrastructure Security Agency (CISA). Top routinely exploited vulnerabilities. https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-209a, 2021.

[295] U.S. Cybersecurity & Infrastructure Security Agency (CISA). Critical Infrastructure Sectors. https://www.cisa.gov/topics/critical-infrastructure-security-and-resilience/critical-infrastructure-sectors, 2024.

[296] U.S. Cybersecurity & Infrastructure Security Agency (CISA). PRC State-Sponsored Actors Compromise and Maintain Persistent Access to U.S. Critical Infrastructure. https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-038a, 2024.

[297] U.S. Cybersecurity & Infrastructure Security Agency (CISA). Secure your products. https://www.cisa.gov/secure-our-world/secure-your-products, 2024.

[298] U.S. Department of Justice. Six Russian GRU officers charged in connection with worldwide deployment of destructive malware and other disruptive actions in cyberspace. https://www.justice.gov/opa/pr/six-russian-gru-officers-charged-connection-worldwide-deployment-destructive-malware-and, 2020.

[299] U.S. Department of the Treasury. Treasury continues to counter ransomware as part of whole-of-government effort; sanctions ransomware operators and virtual currency exchange. https://home.treasury.gov/news/press-releases/jy0471, 2021.

[300] U.S. Government Accountability Office. Critical infrastructure protection: Agencies need to enhance oversight of ransomware practices and assess federal support. https://www.gao.gov/products/gao-24-106221, 2024.

[301] U.S. National Institute of Standards and Technology. Security and privacy controls for information systems and organizations. Technical report, U.S. Department of Commerce, Washington, D.C., 2020.

[302] U.S. Nuclear Regulatory Commission. Cyber security programs for nuclear facilities. https://scp.nrc.gov/slo/regguide571.pdf, 2010.

[303] Lisa Vaas. BillQuick billing app rigged to inflict ransomware. https://vulners.com/threatpost/THREATPOST:94E54481AD472743701D499DC7677008, 2021.

[304] Emma Vail. Russia or Ukraine: Hacking groups take sides. https://therecord.media/russia-or-ukraine-hacking-groups-take-sides/, 2022.

[305] Dirk van der Linden, Pauline Anthonysamy, Bashar Nuseibeh, Thein T Tun, Marian Petre, Mark Levine, John Towse, and Awais Rashid. Schrödinger's security: Opening the box on app developers' security rationale. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*, page 12, 2020.

[306] Dirk van der Linden, Emma Williams, Joseph Hallett, and Awais Rashid. The impact of surface features on choice of (in)secure answers by stackoverflow readers. *IEEE Transactions on Software Engineering*, 2022.

[307] Rob van der Veer, Menno Valkema, Fabio Guasconi, and Prokopios Drogkaris. *Advancing software security in the EU: The role of the EU cybersecurity certification framework*. ENISA, 2020.

[308] K.R. van Wyk and J. Steven. Essential factors for successful software security awareness training. *IEEE Security & Privacy*, 4(5):80–83, 2006.

[309] Christian Vasquez. CISA's Goldstein wants to ditch 'patch faster, fix faster' model. https://cyberscoop.com/cisa-goldstein-secure-by-design/, 2023.

[310] Vasilis Ververis, Sophia Marguel, and Benjamin Fabian. Cross-country comparison of Internet censorship: A literature review. *Policy & Internet*, 12(4):450–473, 2020.

[311] AJ Vicens. Russian government continues crackdown on cybercriminals. https://www.cyberscoop.com/sky-fraud-takedown-russia-cybercrime/, 2022.

[312] Jan Vom Brocke, Alexander Simons, Kai Riemer, Bjoern Niehaves, Ralf Plattfaut, and Anne Cleven. Standing on the shoulders of giants: Challenges and recommendations of literature search in information systems research. *Communications of the association for information systems*, 37(1):9, 2015.

[313] Daniel Votipka, Desiree Abrokwa, and Michelle L. Mazurek. Building and validating a scale for secure software development self-efficacy. In *Proceedings*

of the 2020 CHI Conference on Human Factors in Computing Systems, page 1–20. ACM, 2020.

[314] Daniel Votipka, Kelsey R Fulton, James Parker, Matthew Hou, Michelle L Mazurek, and Michael Hicks. Understanding security mistakes developers make: Qualitative analysis from Build It, Break It, Fix It. In *Proceedings of the 29th USENIX Security Symposium*, page 109–126. USENIX Association, 2020.

[315] Daniel Votipka, Michelle L Mazurek, Hongyi Hu, and Bryan Eastes. Toward a field study on the impact of hacking competitions on secure development. In *Proceedings of the Workshop on Security Information Workers (WSIW) 2018*, page 6, 2018.

[316] Tillmann Wagner, Daniel Korschun, and Cord-Christian Troebs. Deconstructing corporate hypocrisy: A delineation of its behavioral, moral, and attributional facets. *Journal of Business Research*, 114:385–394, 2020.

[317] Pei Wang, Julian Bangert, and Christoph Kern. If it's not secure, it should not compile: Preventing DOM-based XSS in large-scale web development with API hardening. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, page 1360–1372, 2021.

[318] Charles Weir, Ingolf Becker, and Lynne Blair. A passion for security: Intervening to help software developers. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, page 21–30, 2021.

[319] Charles Weir, Ingolf Becker, James Noble, Lynne Blair, Angela Sasse, and Awais Rashid. Interventions for software security: Creating a lightweight program of assurance techniques for developers. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 41–50. IEEE, ACM, 2019.

[320] Charles Weir, Ingolf Becker, James Noble, Lynne Blair, M. Angela Sasse, and Awais Rashid. Interventions for long-term software security: Creating a lightweight program of assurance techniques for developers. *Software - Practice and Experience*, 50(3):275–298, 2020.

[321] Charles Weir, Lynne Blair, Ingolf Becker, Angela Sasse, and James Noble. Light-touch interventions to improve software development security. In *2018 IEEE Cybersecurity Development (SecDev)*, page 85–93, 2018.

[322] Charles Weir and Ben Hermann. From needs to actions to secure apps? The effect of requirements and developer practices on app security. In *Proceedings of the 29th USENIX Security Symposium*, page 17. USENIX Association, 2020.

[323] Charles Weir, Sammy Migues, Mike Ware, and Laurie Williams. Infiltrating security into development: Exploring the world's largest software security study. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, page 1326–1336. ACM, 2021.

[324] Dominik Wermke, Noah Wöhler, Jan H. Klemmer, Marcel Fourné, Yasemin Acar, and Sascha Fahl. Committed to trust: A qualitative study on security & trust in open source software projects. In *2022 IEEE Symposium on Security and Privacy (SP)*, page 1880–1896, 2022.

[325] Tarah Wheeler. The danger in calling the SolarWinds breach an 'act of war'. `https://www.brookings.edu/techstream/the-danger-in-calling-the-solarwinds-breach-an-act-of-war/`, 2021.

[326] Jim Whitmore and Will Tobin. Improving attention to security in software design with analytics and cognitive techniques. In *2017 IEEE Cybersecurity Development (SecDev)*, page 16–21, 2017.

[327] Stephen B. Wicker. The ethics of zero-day exploits—: the NSA meets the trolley car. *Communications of the ACM*, 64(1):97–103, 2020.

[328] Chamila Wijayarathna and Nalin Asanka Gamagedara Arachchilage. Why Johnny can't develop a secure application? A usability analysis of Java Secure Socket Extension API. *Computers and Security*, 80:54–73, 2019.

[329] Jim Witschey, Shundan Xiao, and Emerson Murphy-Hill. Technical and personal factors influencing developers' adoption of security tools. In *Proceedings of the 2014 ACM Workshop on Security Information Workers - SIW '14*, page 23–26. ACM Press, 2014.

[330] Jim Witschey, Olga Zielinska, Allaire Welk, Emerson Murphy-Hill, Chris Mayhorn, and Thomas Zimmermann. Quantifying developers' adoption of security tools. In *2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2015 - Proceedings*, page 260–271, 2015.

[331] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international*

conference on evaluation and assessment in software engineering, pages 1–10, 2014.

[332] Josephine Wolff. Should insurance companies pay out for damage caused by state-sponsored cyberattacks? `https://slate.com/technology/2022/01/merck-notpetya-cyberattack-insurance-russia.html`, 2022.

[333] Glenn Wurster and P. C. van Oorschot. The developer is the enemy. In *Proceedings of the 2008 New Security Paradigms Workshop*, page 89–97. ACM, 2008.

[334] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. Social influences on secure development tool adoption: Why security tools spread. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work and Social Computing*, page 1095–1106. ACM, 2014.

[335] Jing Xie, Heather Richter Lipford, and Bill Chu. Why do programmers make security errors? In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, page 161–164. IEEE, 2011.

[336] Zeljka Zorz. Cyber attacks on Ukraine: DDoS, new data wiper, cloned websites, and Cyclops Blink. `https://www.helpnetsecurity.com/2022/02/24/cyber-attacks-ukraine/`, 2022.