Planning a Portfolio of Controls for Software Development

Onkar Malgonde, Alan Hevner, and Rosann Webb Collins

Muma College of Business, University of South Florida, USA
{omalgonde, ahevner, rwcollins}@usf.edu

Abstract. A growing number of software development projects successfully exhibit a mix of agile and traditional software development methodologies. Many of these mixed methodologies are organization specific and tailored to a specific project. Our objective in this research-in-progress paper is to develop an artifact that can guide the development of such a mixed methodology. Using control theory, we design a process model that provides theoretical guidance to build a portfolio of controls that can support the development of a mixed methodology for software development. Controls, embedded in methods, provide a generalizable and adaptable framework for project managers to develop their mixed methodology specific to the demands of the project. A research methodology is proposed to test the model. Finally, future directions and contributions are discussed.

Keywords: Control theory, Portfolio of controls, Method engineering, Design science

1 Introduction

Increasingly, software development teams want control and flexibility to co-exist in their development process. Such a controlled-flexible approach allows them to handle uncertainty in market and produce a better market-product match [1]. A recent industry trend report [2] on enterprise software quality reports that a mix of Agile and Waterfall (plan-driven) methods produces higher structural quality for business critical applications than either Agile or Waterfall methods alone. Similarly, Baskerville, Heje-Pries and Madsen [3] note that companies are successfully combining agile and plan-driven approaches, consolidating the lessons learnt, and developing an organizational software development process. Such an organizational development process can then be tailored to specific projects to meet project goals.

A mixed methodology is desirable for software development teams because they find that adhering to a specific software development approach may not provide an adequate fit to the project needs. For example, within agile methods, Fitzgerald, Hartnett and Conboy [4] combine extreme programming (XP) and Scrum to develop an effective software development methodology. They select 6 existing XP practices out of 12, based on their applicability to the project environment. These 6 XP practic-

es are then supplemented with 6 practices from the Scrum methodology. The rationale behind such a combination is that XP provides support for technical aspects whereas Scrum provides better support for planning and tracking for the projects progress.

In this research-in-progress paper, we aim to provide theoretical guidance on developing a mixed methodology that is tailored for a specific project. The key focus is a portfolio of controls that is initially developed based on the critical factors found in the project [5]. Controls, which are embedded in method fragments, are used to identify method fragments to develop the methodology. The focus of this manuscript, then, is to describe our research-in-progress on designing an artifact that provides guidance and understanding of controls needed to build a mixed methodology. Our goal is to improve current practices in developing mixed methods for software development, as positioned in design science research [6]. In the following sections we explore related literature, provide an example of mixed methodology development, develop our design artifact, propose a research methodology to evaluate that artifact, and discuss future work and contributions of our research.

2 Tailoring a Software Development Method

Traditionally, two method-tailoring (situational methodology) approaches have been employed to develop organization-wide and project-specific methodologies from existing methods: *contingency factors* and *method engineering* [4, 7]. Both emphasize that method-tailoring is driven by critical factors in the project and organizational context. Proposed by Davis [8], *contingency factors* require the development team to analyze the project environment (source of contingency). Upon analyzing the project environment, the project team would then identify critical contingency factors. Based on the identified contingency factors, methods of software development are compared that are available in an organizational repository of methods. Typically, organizational repository of methods is a function of successful prior utilization of methods. Based on the analysis and identification of a suitable fit, the methodology is chosen for software development, and is tailored to the project specific environment [4].

Method engineering [9] involves developing a software development methodology using method fragments from existing methodologies that are present in the organizational repository of methods [10]. Based on the project performance, an organizational method repository is continually updated with new method fragments. An important decision point in method engineering is the concept of situation specific selection of method fragments [10], where method fragments are replaced or added to the existing method based on particular situations that arise during project execution. The selection of method fragments is based on contingencies, similar to the previous approach.

Project teams face multiple challenges while employing contingency factors and method engineering approaches. First, these approaches advance an organization-specific development approach that can be challenging in situations where multiple organizations are involved. Distributed or culturally diverse teams can find it challenging to adapt to tailored methodologies. Second, these approaches to methodology development do not provide practical risk-benefit analysis of adding, substituting,

deleting, and combining methods from different methodologies. Third, much rests on the project manager's experience about how the methodology can be tailored to situation specific needs. Finally, these approaches lack formal theory to describe how the selected method achieves a balance between control and flexibility [1].

3 Control Theory

A central responsibility of any manager is to exercise control over employees and organizational activities. Control theory [11-13] explains different control modes available to managers, including project managers. It provides the lens that guides the development of a project-specific methodology. Control modes are categorized into two types: *formal* and *informal*. Formal modes of controls are viewed as performance rewarding strategies by the management [14, 15]. In formal control mode, the management specifies a goal and reward for the team upon completion of the project goal.

Two forms of formal control are outcome control and behavior control. *Outcome control* specifies establishing prior set of goals and determining reward levels based on the extent to which established goals have been accomplished. For example, specified software load time is a system goal. If such a load time is consistently achieved, the software team has met the outcome goal and can be rewarded based on a prespecified contract. *Behavior control* specifies adherence to established processes that software development teams should follow in order to achieve the outcome goals. In such a control mode, management's emphasis is on observing team's behavior. For example, presence in daily Scrum meetings is expected from team members so that information can be shared.

In contrast to formal modes of control an informal mode of control relies on a social strategy to achieve the goal of aligning organizational and employee goals. Two forms of informal control are clan control and self-control. *Clan control* relies on the team to foster a unique set of rules, applying to all, that help in achieving the common goal for the team. Management has limited leverage on such a control since it is loosely coupled from the organization goals and is highly influenced by interactions within the team. S*elf-control* emphasizes individual autonomy to achieve goals set by the individual. In a software development team, individuals are required to be creative and govern their own individual processes to meet deadlines [16]. In professional settings like software development informal modes of control are also influenced by developers' education and socialization to the profession.

In order to extend Control Theory to handle situations with high risk and uncertainty, Harris, Collins and Hevner [1] propose a new mode of control: emergent outcome control (EOC). They identify two EOC mechanisms. *Scope boundaries* limit the feasible solution such that the development team has the flexibility to explore but is constrained within a boundary. However, the project team is unconstrained within the boundaries thereby maintaining creativity. *Ongoing feedback* is provided to the team, from users, or the market, to steer development so that specifications are closely met. For example, feedback can be provided to the team via meetings, documentation, user reviews, or market orientation. Such feedback allows them to adjust their development to specific needs of the market and achieve their goal.

Project managers employ *control mechanisms* to implement control modes [15, 17]. For example, delivering a working prototype every 2 weeks implements outcome control by specifying a target for every development cycle. Also, it implements behavioral control by providing a sense of urgency within the team. In a software development project, control mechanisms are embedded in method fragments [18].

Kirsch [15] posits that construction of a portfolio of controls is driven by four influencing factors: availability of pre-existing mechanisms, task characteristics, role expectations, and project-related knowledge and skills. This critical factor focus is congruent with the contingency theory and methods engineering approaches to selecting project methods. However, there are three limitations with this approach. First, the approach is highly biased towards selecting preexisting mechanisms without any analysis of their aptness to the project. Second, the approach does not focus on what controls are needed for the successful completion of the project. Rather, the approach is focused on factors that aim to fit existing controls to project needs. Finally, as the project unfolds, Kirsch [19] attributes changes in the configuration of portfolio of controls, across project phases, to the influencing factors, but does not explain how project teams can proactively change the configuration to steer project development towards its goal.

Thus, there is still a gap in our understanding about which controls should be included in the initial portfolio of controls, and how the portfolio should be manipulated over the execution of the project to best adapt to change. Addressing the first gap here, we now discuss our process model to develop an initial portfolio of controls.

4 Designing a Portfolio of Controls

Figure 1 describes our work-to-date on a process model for developing a portfolio of project controls. Initially, the project manager should analyze the project needs for control, based on the critical factors in the context. Boehm and Turner [5] provide five critical factors to analyze a project's needs for its suitability to plan-driven or agile approach: size (number of personnel), criticality (loss due to impact of defects), level of skilled personnel, dynamism (change in requirements), and the culture (people feel comfortable under chaos or order). Three of these factors overlap with Kirsch's influencing factors of *project-related knowledge and skills* and *role expectations* (size, level of skilled personnel and culture). An influencing factor to add to Boehm and Turner's factor set is *task characteristics*. Analyzing the project on the resulting six critical factors using a polar graph [5], the project manager can identify needed controls to accomplish the project goal.

Based on the analysis using critical factors, the project team selects desired control modes and mechanisms (controls) from the controls base. The control base is the repository of control modes and mechanisms available to the project manager. At this point, the selection of controls is completely based on the desired outcomes envisioned by the project manager. Selection of desired controls is due to the high temporal distance between the present state of the project and the project goal. For example, in Intel Shannon [4], the development team had formed a cohesive group over

many years of working together. For such a project, relying primarily on informal controls while supplementing it with formal controls would maintain the comradery and cohesion, and help attain the project goal.

Following the initial selection of controls, the project manager then selects method fragments from the methods base [9]. The selection of method fragments is governed by desirability of the method fragment and the extent to which the method fragment embeds the control mechanisms identified.

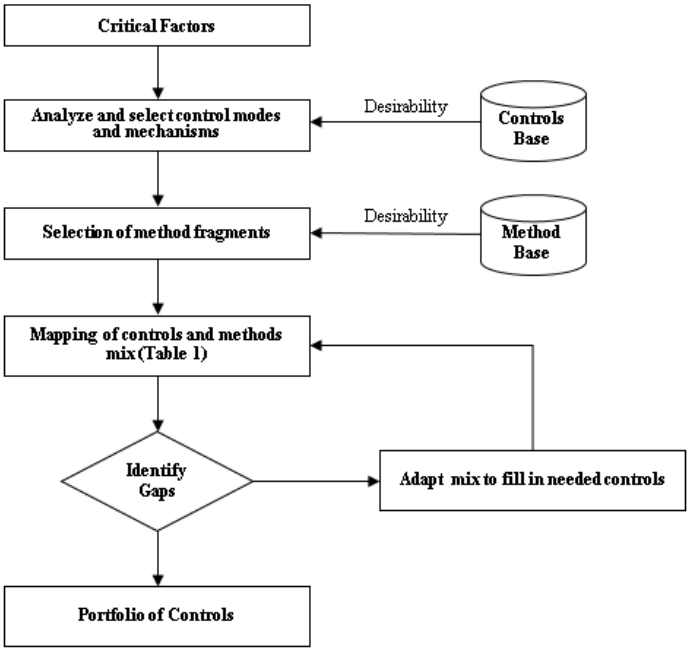

Figure 1. Process Model for a Portfolio of Controls

After selecting the required method fragments that embed the desired control mechanisms, the project manager should conduct a mapping analysis of needed controls versus support available for them via method fragments. Table 1 demonstrates such an analysis using Intel Shannon as the example [4]. Note that we have not included all method fragments and control mechanisms due to space constraints. Columns represent the required control modes. Rows represent method fragments which are selected to support the control modes. Mapping of controls and supporting method fragments reveals high reliance on informal control modes in the selected methods. Though such a portfolio is beneficial based on the cohesive group, inclusion of formal controls will allow the project manager to provide product demos and delivery dates to the customer. After identifying such a gap in control-method mapping, the project team can add appropriate method fragments to fill those gaps. During instances where appropriate method fragments are not available, the project manager can adapt existing control mechanisms to fill those gaps. For example, *on-site customer* method

fragment was not feasible for the Intel Shannon team. They can adapt the *post-game closure* fragment to incorporate customer feedback after every sprint.

Table 1. Mapping Controls and Method Fragments (constructed from [4])

Method Fragments	**Control Modes**					
	Outcome Controls	**Behavioral Controls**	**Clan Controls**	**Self-Controls**	**Scope Boundaries**	**Ongoing Feedback**
Pair Programming		Work together	Activities transparent to team members			New ideas tested with partner
Testing				Provides feedback On re-quired work	Development bounded by test constraints	
Post-Game Closure	Specify com-pletes and incompletes at the end of each sprint					Progress visible
Scrum Sprints	Specify sprint outcomes at the start					

5 Future Research Directions and Contributions

Our on-going research plan is to first validate the process model design in Figure 1. In the selection of the portfolio of controls, we need to: (1) determine if the six critical factors set is both accurate and reasonably complete, (2) understand the processes of analyzing the need for controls and the selection of method fragments, and (3) develop the mapping of controls and methods. In addition we need to understand how the role of a need for flexibility is balanced against control in this process. Our goal is to design a model that is prescriptive in nature. Interviews with project managers that have experience in developing mixed approaches for software development will provide rich data for model evaluation. Organizations that have a specified organizational-methodology and allow managers to customize it based on the project would be ideal places for conducting interviews. Also, projects with multi-organizational or multi-cultural involvement provide additional testing areas for our process model.

This project provides multiple avenues for future research. The research proposed in this paper supports the planning stage of a software development project. Our future directions will develop a similar process model that adapts the initial control portfolio to the changes found while executing the project. Specifically, we will draw upon Construal-Level Theory [20] which argues that objects that are at a higher tem-

poral distance are perceived as abstract concepts, whereas objects with lower temporal distance are perceived as concrete concepts. Evaluation of a decision alternative for an abstract concept tends to focus on a holistic and desired view for the object. On the other hand, evaluation of a decision alternative for a concrete concept tends to focus on feasibility and precise view of the object. Desirable alternatives are the long-term ideal actions that are coveted at the outset when temporal distance between the decision and the goal is greater. Feasible alternatives, on the other hand, are the short-term actions that are required to attain the desired goal which is temporally close. With increasing temporal distance, desirable alternatives are preferred over feasible alternatives. Conversely, decreasing temporal distance to the goal leads to greater acceptance of feasible alternatives [21]. For example, Liberman and Trope [22] find empirical evidence for student's preference for a desirable (interesting) assignment over feasible (simple) assignment as a choice over distant future. In software development projects, the initial portfolio of controls consists of desirable control modes since the project goal is at a higher temporal distance. However, as the project is being performed and the project goal is at a lower temporal distance, project characteristics change over time. This requires changes in the control portfolio that can adjust to the changing project characteristics. With decreasing temporal distance, desirable controls are replaced by feasible controls to attain the project goal. Thus, it is important to identify the conditions under which method fragments need to be added, deleted, or replaced with other fragments over time. In addition, the impacts of adverse situations like time or budget pressure on portfolio of controls and possible mitigating strategies are other important areas that need further research.

Adhering to a single software development approach is increasingly challenging when project characteristics and market needs change [9]. We have presented our artifact as a model that guides the process of constructing an initial portfolio of controls. The application area for our model is the development of mixed methodology but with guidance of using controls as the driving force rather than methods fragments themselves. Further, our process model provides a risk-benefit analysis for project manager that can be used to develop mixed methodologies. We have also proposed a research methodology to evaluate our process model that will serve as an evaluation mechanism.

6 References

1. Harris, M.L., Collins, R.W., Hevner, A.R.: Control of Flexible Software Development Under Uncertainty. Information Systems Research 20, 400-419 (2009)
2. Curtis, B., Szynkarshi, A., Lesokhin, L., Duthoit, S.: The CRASH Report - 2014. CAST Research on Application Software Health (2014)
3. Baskerville, R., Heje-Pries, J., Madsen, S.: Post-agility: What follows a decade of agility? Information and Software Technology 53, 543-555 (2011)
4. Fitzgerald, B., Hartnett, G., Conboy, K.: Customising agile methods to software practices at Intel Shannon. European Journal of Information Systems 15, 200-213 (2006)
5. Boehm, B., Turner, R.: Balancing Agility and Discipline (2004)
6. Gregor, S., Hevner, A.R.: Positioning And Presenting Design Science Research For Maximum Impact. MIS Quarterly 37, (2013)

7. Fitzgerald, B., Russo, N., O'Kane, T.: Software Development Method Tailoring at Motorola. Communications of the Association for Information Systems 46, (2003)
8. Davis, G.: Strategies for information requirements determination. IBM Systems Journal 21, 4-30 (1982)
9. Brinkkemper, S.: Method Engineering: engineering for information systems development methods and tools. Information and Software Technology 38, 275-280 (1996)
10. Kumar, K., Welke, R.: Methodology Engineering: A proposal for situation-specific methodology construction. In: Cotterman, W., Senn, J. (eds.) Challenges and Strategies for Research in Systems Development, (1992)
11. Ouchi, W.: The relationship between organizational structure and organizational control. Administrative Science Quarterly 22, 95-113 (1977)
12. Ouchi, W.: A conceptual framework for the design of organizational control mechanisms. Management Science 25, 833-848 (1979)
13. Ouchi, W.: Markets, bureaucracies, and clans. Administrative Science Quarterly 25, 129-141 (1980)
14. Eisenhardt, K.M.: Control: Organizational and Economic Approaches. Management Science 31, 134-149 (1985)
15. Kirsch, L.J.: Portfolios of Control Modes and IS Project Management. Information Systems Research 8, 215-239 (1997)
16. Henderson, J., Lee, S.: Managing I/S Design Teams: A Control Theories Perspective. Management Science 38, 757-777 (1992)
17. Choudhury, V., Sabherwal, R.: Portfolios of Control in OUtsourced Software Development Projects. Information Systems Research 14, 291-314 (2003)
18. Harris, M.L., Collins, R.W., Hevner, A.R.: Controls in Flexible Software Development. Communications of the Association for Information Systems 24, 757-776 (2009)
19. Kirsch, L.J.: Deploying Common Systems Globally: The Dynamics of Control. Information Systems Research 15, 374-395 (2004)
20. Trope, Y., Liberman, N.: Construal-Level Theory of Psychological Distance. Psychological Review 117, 440-463 (2010)
21. Ariely, D., Zakay, D.: A timely account of the role of duration in decision making. Acta Psychologica 108, 187-207 (2001)
22. Liberman, N., Trope, Y.: The Role of Feasibility and Desirability Considerations in Near and Distant Future Decisions: A Test of Temporal Construal Theory. Journal of Personality and Social Psychology 75, 5-18 (1998)