Department of Electrical & Computer Engineering

Software Engineering
The University of Auckland
New Zealand

A Flexible Software Process Model

Diana Kirk

April 2007

Supervisor: Associate Professor Ewan Tempero

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF DOCTOR OF PHILOSOPHY IN ENGINEERING

The University of Auckland

Thesis Consent Form

Signed: . . . .

Date: . . . .

Created:	5 July 2001
Last updated:	9 August 2001

Abstract

Many different kinds of process are used to develop software intensive products, but there is little agreement as to which processes give the best results under which circumstances. Practitioners and researchers believe that project outcomes would be improved if the development process was constructed according to project-specific factors. In order to achieve this goal, greater understanding of the factors that most affect outcomes is needed. To improve understanding, researchers build models of the process and carry out studies based on these models. However, current models contain many ambiguities and assumptions, and so it is not clear what the results of the studies mean. The statement of this thesis is that it is possible to create an abstraction of the software development process that will provide a mechanism for comparing software processes and software process models. The long term goal of the research is to provide planners with a means of tailoring the development process on a project by project basis, with the aim of reducing risk and improving outcomes.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Ewan Tempero, my supervior for this dissertation, for his superb support and guidance during the time of my research. Ewan always succeeded in providing advice and direction that were appropriate and timely and was always available for discussion. I am very grateful for his encouragement.

During the time of this research, I attended a number of conferences and workshops to present papers based on the research. I would like to thank Ita Richardson, Juan Ramil, Mary Shaw, James Noble and Stephen MacDonell for taking the time to provide me with feedback on some of these papers and Judith Segal, David Weiss and Colin Coghill for some interesting and, from my perspective, fruitful discussions about various aspects of the research. I would also like to thank Barbara Kitchenham for her willingness to share with me some of her vast knowledge relating to research techniques and evidence and thus helping me on my beginner's journey towards an understanding of the current state of software research.

Finally, I would like to thank the Department of Computer Science at the University of Auckland for providing me with funding to support my research and the department staff for providing an excellent atmosphere to work in.

1 Introduction

1.1 Research Area Overview

Many researchers and practitioners are interested in exploring different ways of producing software-intensive products. The reason is that it is generally agreed in the software industry that the kind of process used in a software project is a key factor in determining what are the outcomes for the project. Example outcomes are the ability of the project to deliver the software product on-time and within budget.

At the present time, there are a number of different kinds of process in use in industry. These are often categorised as either ‘traditional’ (commonly referred to as ‘heavyweight’) or ‘lightweight’. Traditional processes were created to help control very large software projects spanning several years, many of which exhibited safety-critical or other ‘large loss’ aspects. These processes are based on a manufacturing paradigm and are characterised by a phased approach, in which, for example, design tasks are strictly separated from coding tasks. Different phases tend to be carried out by different people, for example, ‘systems analysts’, ‘architects’, ‘coders’ and ‘testers’. As a result of the strict separation, large quantities of documentation are required to communicate decisions among the various parties. The well-known ‘Waterfall’ model represents an example of this kind of process.

The ‘light’ processes have emerged more recently as a response to the perceived ineffectiveness of the traditional methods when applied to, for example, Web development. These processes tend to be more responsive to change in product requirements and are characterised by a strong people focus. Because of the close relationships between developers and customers, the underlying development paradigm for these methodologies is presented as ‘software-as-a-service’ and communications are generally face-to-face. ‘XP’ (eXtreme Programming) is an example of this kind of process.

Traditional and light processes are most commonly applied to different kinds of projects. Traditional processes are most often used in very large projects or for safety-critical products. Light processes are most often used for small projects or products that can be produced and changed quickly, for example, Web applications. However, there is much discussion about how to apply individual processes to projects other than those for which they appear to be most suited. For example, is it appropriate to use an XP process for developing a product that is expected to undergo further change in the future i.e. is part of a product line? There is also much discussion about the possibility of embedding elements of one process into another. For example, would deadlines be more likely to be met if a ‘Pair Programming’ technique from XP was used within a Waterfall process?

In this dissertation, I address the possibility of synthesising a new process from elements of existing processes. In order to achieve this, I study the work of those researchers who model the software process for the purpose of predicting outcomes and uncover limitations of the models that render the models inappropriate as a basis for synthesis. I present an abstraction of the process that supports such synthesis.

1.2 Problem to be Addressed

It is widely acknowledged in the software industry that no one process is appropriate for all software development projects [10, 35, 38, 57]. Some believe that each kind of process is appropriate for specific kinds of project and should be used only within such projects. This requires an assumption that any project can be classified as one of a number of discrete types, each with fixed boundaries. Others are adamant that their ‘favourite’ process may be applied to any project, with only minor adaptations required. Of greater interest is the possibility of synthesising new processes from existing ones. Many researchers and practitioners believe that the chance of project success can be improved by selecting process elements from different processes in order to tailor the process according to project-specific factors [13, 35, 46, 93, 115, 144, 153].

The interest in customisation comes from two directions. The first is the ‘traditional’ versus ‘light’ discussions [16, 21]. Practitioners understand that different kinds of process have different strengths, but would like to know under which circumstances elements from one process may be embedded in another. The kinds of questions asked include: “How well does XP perform if the customer is unable to be on-site?”; “How would delivery schedules be affected if developers practice Test Driven Design within a Waterfall process?”. The second source of interest is from the study of software economics. The suggestion here is that a project should maximise value creation [102] and should use a process that is no more costly than necessary [24]. The kinds of questions that represent this kind of interest include: “What will be the effect on the quality of the delivered product if we replace code inspections by automated checking to reduce cost?”; “Can we customise a process to give best outcomes for a specific project by combining elements from existing processes?”.

Although the industry would like to answer the kinds of questions illustrated above, this is not possible at the present time. Before we can answer such questions, we must first be able to represent any process element from any process in a way that facilitates composition and prediction. It is contended here that no suitable abstraction exists.

One reason is that the problem space is not yet well-understood. Current processes have emerged in response to perceived need and, although attempts have been made to understand what are the key factors that affect outcomes, there is little data to support claims. Efforts have been made to collect supporting data, but the complexity of the problem space has rendered this difficult. In addition to the many technical challenges presented by a fast-changing industry, software process tasks are carried out by people rather than machines, and so issues of psychology and social behaviour are relevant.

It is now generally accepted that human factors, for example, management style and developer experience and motivation, have a major impact on the success of a software development effort [3, 24, 37, 34, 155, 157]. Curtis et. al. suggested in 1988 that process problems were overwhelmingly caused by people-related factors and recognised at that time the need for a behavioural model of the software development process [36]. However, there are no such models on which to base formal research into the effects of human factors on outcomes and current processes either assume a tendency to the mean or make assumptions about which factors are key. For example, the Waterfall process does not include any consideration of human factors. This is perhaps because of its traditional use for very large projects where human effects ‘average out’ over the course of the project. XP, on the other hand, embeds assumptions about developer performance, for example, that all developers work more efficiently and effectively together than alone. Although this represents some consideration of human factors, there is no mechanism for accommodating differences between developers.

In an attempt to accumulate data to increase the industry’s ability to predict process outcomes, researchers have carried out different kinds of studies. There are a number of concerns that relate to these studies. One such concern is the issue of how to measure the various factors and attributes that apply to the software process. In 1995, Kitchenham, Pfleeger and Fenton identified a lack of integrity in the way in which software practitioners and researchers measure software-related attributes [89]. It is important to work with validated software metrics and at that time there was no agreed way to perform such validation. The authors made a plea for the industry to agree on a way to bridge the gap between measurement theory and software metrics. Although the plea appeared to spawn some heated discussions, it seems that no consensus has been reached, and the industry continues to work with metrics based on disputed foundations [91]. A second concern is that, as software engineering is a relatively immature discipline, researchers have not yet learned to routinely apply sound practices when conducting studies, and so resulting data is scarce, fragmented and of varying quality [7, 14, 46, 90, 125]. Gilmore describes four modes of research data collection and states that only one of these, hypothesis testing, results in establishing causal connections. He and others agree that, in order to carry out this kind of research, a theoretical framework is essential from which to spawn hypotheses [38, 55, 90]. As discussed above, there is no suitable abstraction for elements of a software process and so it is difficult to carry out hypothesis testing experiments and establish causal connections.

In summary, the industry at the present time has no abstraction or theoretical framework for software development processes. This means that practitioners are unable to combine process elements and predict outcomes and researchers find it difficult to investigate causal relationships.

1.3 Modelling for Understanding

In the previous Section, I described a problem of lack of a theoretical model of the software process. In this Section, I consider the act of model building from an historical perspective and conclude that an appropriate model should be explanatory rather than predictive and that such a model will be holistic rather than fragmented.

Rivett [139], when describing the status of model building in the field of operations research in 1972, reminds us that, throughout history, man has constantly searched for pattern and generalisation. From around 700 BC, the Babylonians measured and recorded the motions of the stars and planets, analysed these, and were successful in forecasting planetary events with great precision. Their recordings of hundreds of years of planetary data enabled them to estimate the value of the motion of the sun from the node with an error of only five seconds. The large quantity of data collected by the Babylonians supported accurate prediction. In fact, two thousand years later, the same estimation, based on models of planetary motion, yielded an accuracy of only seven seconds. The observation is that large quantities of accurate data often yield more accurate predictions than those based on models.

Although the Babylonians recorded events with care, they made no attempt to theorise. The Greeks, on the other hand, followed a different approach, and built first mechanical and then geometrical models of planetary motion in an attempt to understand and explain. However, their models were made up of a number of parts and the Greeks had no success in unifying these. When applied to the Babylonian data, the models were found to be incorrect [139]. This illustrates that it is often difficult to achieve consistent results when a fragmented approach is taken i.e. a model of a part of a system may yield results that are invalid in the context of the bigger system.

Rivett presents another example from more recent times that illustrates that consistent and complete results will be achieved only if a model is based on an underlying theory. Kepler proposed three laws of planetary motion based on data that had been collected by Tycho Brahe. He applied an elliptical model to the motion of the planets and from this model produced laws that appeared to work. No-one knew the fundamental reason why the laws worked. I notice that, as the laws were based on planetary data, these laws could not predict the movements of other celestial objects, for example, comets. Newton later brought some understanding to bear on celestial motion when he postulated a force that acted between all objects with mass in the universe. From this understanding and unification of ideas from physics and astronomy, he was able to show that orbits for celestial objects, for example comets, were not only elliptical, but could be hyperbolic and parabolic. He was able to predict accurately for all celestial bodies, show that Kepler's Laws were a special case of Newton's Laws and improve the accuracy of Kepler's calculations.

Rivett summarises by stating that a model may be predictive without being explanatory, but an holistic, explanatory model is always predictive. When I apply this idea to the software process, it follows that previous process data may be successfully used to predict the outcomes of future projects that are based on similar processes. I understand that, if we wish to predict in a more general way, our predictive models must be holistic and explanatory. This means our models must be able to represent any element of any process, including both existing elements and those defined at some future time.

I have identified the need to represent different process elements for synthesis and prediction. In order to meet these goals, I want to capture process elements in a descriptive way i.e. capture elements as they actually happen.

1.4 Thesis Statement

Before we can synthesise processes from existing and future process elements, we must first be able to capture elements of processes and process models. Before we can predict outcomes of applying process elements, we must first be able to compare the effects on these outcomes of different elements.

I believe that it is possible to capture software development processes and process models in a way that allows us to compare processes and process models for the purpose of constructing new processes.

My thesis is realised as a conceptual modelling framework, *KiTe*, the elements of which are themselves models. The framework supports capture of, and facilitates comparison and composition of, processes and process models.

1.5 Approach

I have postulated the need for a model of the software development process that allows capture of any element from any process or process model and facilitates comparison between, and composition of, elements. The long term goal for such a model is the ability to predict outcomes when process elements are combined in various ways. Rivett and others argue that a model for prediction must be holistic and explanatory. 'Holistic' suggests that all relevant aspects of the process, for example, behavioural aspects, must be included. 'Explanatory' suggests that the model should be based on a theoretical abstraction rather than on specific data.

The creation of such a model is difficult. If the industry is to create such a model, it must first identify what are the characteristics of existing processes that must be included in a representation and understand what are the limitations of existing predictive models that render them inappropriate for general process representation.

I thus examine the characteristics of existing processes and process models and create from this examination a set of 'desirable properties'. These properties will act as preliminary criteria against which to judge any candidate model. This provides an informal mechanism for evaluation, in that the criteria are subjective in nature. The aim is to gain some confidence prior to any evidence-gathering attempt that the candidate model is likely to support the objectives of capture and comparison.

Once a candidate model is proposed and evaluated against the preliminary criteria, some evidence must be presented to support the thesis that the model supports capture and comparison for the purpose of synthesis and prediction. As there are many different kinds of process element, there is a rich space for investigation. Possible kinds of element include those from traditional and agile processes, large and small projects, elements from process models, and many more. As a means of structuring the evidence that represents the model's ability to capture and compare process elements, I have chosen an approach called *argumentation* along with an established notation for structuring arguments, *Goal Structuring Notation (GSN)*. *Argumentation* is "an approach which can be used for describing how evidence satisfies requirements and objectives" [160]. The use of a suitable notation such as *GSN* helps researchers to easily identify what evidence is required and helps stakeholders see at a glance what is the 'evidence coverage'. This approach has been used for many years in the safety critical domain and has recently been applied in the software domain [160].

The need to capture existing models that are the basis of various studies means that I must provide a means of representing studies that vary in integrity. One consequence of this is that it must be possible to capture models based on different beliefs, for example, beliefs about which contextual factors most affect outcomes. This will be necessary until the industry has progressed to a better understanding of these factors. This suggests that I must find an abstraction that accommodates a potentially huge variation in the statement of possible influencing factors. I am also required to capture processes that may have different kinds of objectives, for example, relating to cost or quality. For these kinds of reasons, the solution model will be realised as a framework, the elements of which are models in their own right. For example, there is a model (abstraction) for the contextual factors and one for the process objectives.

Evaluation of the framework will involve:

1. Identification of the range of processes and process models that must be successfully represented.
2. Discussion about how the framework meets the criteria established as a result of examination of process characteristics and process model limitations.
3. Presentation of evidence to support the claim that identified processes and models can be represented and that representation supports comparison.
4. Discussion about some limitations inherent in the approach.

1.6 Overview of Contributions

The main contribution of this thesis is the identification of the need for a holistic approach to modelling the sofware development process in a descriptive way and the presentation of a candidate modelling framework that provides a way of representing and comparing different kinds of process elements.

A second contribution is the identification of the various research groups that model the software development process to predict outcomes and the understanding of how these groups differ in approach and what are the limitations inherent in the work of each. The contribution also includes a realisation that the narrow approach taken by each of the groups is a symptom of lack of real understanding and is the basis for the case for a more holistic approach.

A third contribution is the establishment of an approach for developing and evaluating models that claim to describe systems in a holistic and explanatory way. The strategy is to first establish model objectives and identify a comprehensive range of example systems to be described by the model. The next step is to examine the characteristics for, and problems with, the example systems to help identify key model properties. These provide some basis for establishment of a suitable model structure and may be used as criteria against which to evaluate candidate models in a preliminary evaluation step. Finally, the ability of the candidate model to satisfy objectives is established by accumulating a portfolio of different kinds of evidence relating to the example systems.

A final contribution is the understanding that the existence of a suitable framework gives rise to a number of unplanned research directions and the addressing of one such possibility, that of process-specific risk profiles. Remaining directions include the use of the framework to support research. Such directions are suggested as areas for future research.

1.7 Terminology

The issue of terminology in the field of software engineering is problematic. Words such as ‘task’ and ‘activity’ are used by different authors to mean the same thing. ‘Process’ and ‘product’ also tend to be undefined and many other terms are applied without any definition of what they mean.

My approach in this dissertation is to define terms used in a general way in a Glossary (see Appendix A). Such terms are *italicised* and defined on first use and subsequently italicised only when this helps clarify content. I also use *italicised* text when emphasising a word or phrase, even if not included in the Glossary.

For terms used by authors of a study being described, I include the term in ‘single quotes’. For example, a process might be described by an author as comprising a number of ‘Activities’ and ‘Tasks’. In such cases, I do not try to define exactly what the term means, unless this is necessary for the discussion.

I also use ‘single quotes’ when paraphrasing and “double quotes” when quoting phrases from other sources.

For elements of the model that is the subject of this thesis, I use *Slanting Text*.

1.8 Document Roadmap

This dissertation is placed in the area of *software processes* used by *projects* to produce software-intensive *products*. A *project* is ".. a temporary endeavour to create a unique service or product and with a definite beginning and end" [135]. I note that this definition says nothing about the form of the service or product delivery and, in this thesis, I view a project as any effort that makes a delivery of any kind to any stakeholder. For example, a project might deliver a finished product to a customer, a prototype to the development group or a test plan to the test group. In other words, project objectives and scope are decided by the interested parties and project definition is constrained only by the need to have a defined start and end and agreed delivery. A *software process* is "...the set of all activities which are carried out in the context of a concrete software development project" [59]. A *product* comprises the artifacts that implement a software-intensive system and are the deliverables from a project.

This thesis is organised as follows.

In Chapter 2, I consider some different viewpoints on what is software development and note that the range of proposed paradigms indicates a lack of real understanding of the essential nature of the activity. I then provide an overview of some common software development processes. I finally discuss some ways in which processes are categorised. I suggest that the interest in categorising is a symptom of lack of understanding and that focus should return to the more important task of understanding what are the key characteristics common to all software processes.

In Chapter 3, I examine the various kinds of study carried out by researchers with the goal of predicting software process outcomes. I learn that there are three separate communities and each applies a different approach and builds different kinds of models of the process. I expose some limitations inherent in the work of the different groups by showing that the kinds of models applied by each contain ambiguities and assumptions that render impossible comparison of results.

In Chapter 4, I examine research related to the goal of process flexibility. This related work spans several research areas, some directly related and others more indirectly. I first discuss processes and process frameworks for which claims of flexibility are made and suggest limitations based on an inability to capture the different kinds of process presented in Chapter 2. I then examine process models for which claims of flexibility are made and show how each is limited according to its underlying approach as discussed in the previous Chapter. I finally discuss attempts to model human-related factors and note that research is progressing in this direction but, as yet, no suitable model of the human aspects of the process exists.

In Chapter 5, I present a justification of the need for a theoretical model of the software development process. I first provide a general overview of the different approaches to gathering data and discover that, if the aim is to establish causal connections between factors, a theoretical framework is required. I next present some quotes from a number of researchers stating the need for a holistic theory and reminding us that a characteristic of studies that are not based on such a theory is an inability to achieve consistent results. Finally, I suggest that current software process research is achieving inconsistent results because the research is fragmented and is not based on an underlying theory. I conclude that a formal model of the software process is required and formalise the objectives for such a model in the context of this thesis.

In Chapter 6, I analyse the various processes and process models described in Chapters 2 – 5 with the aim of understanding what might be the properties of a model that is a solution to the problem of process customisation. I also consider some 'real-life' scenarios from industry to help with identification of such properties. I then list the desired properties to be used as criteria against which to evaluate a candidate model.

In Chapter 7, I present a candidate model, *KiTe*. My approach to presentation is to first provide a schematic overview as a 'gentle' introduction, and to then present and formalise the abstract model.

In Chapter 8, I present some evidence to support the proposed model. This includes evidence to support the claim that *KiTe* may be used to capture any process or process model and that it supports comparison of studies.

In Chapter 9, I discuss how the existence of a suitable model provides some benefits not originally planned or realised. As illustration, I show how *KiTe* can be applied to the identification of risks specific to XP (eXtreme Programming) processes.

In Chapter 10, I evaluate the candidate model *KiTe*. I first discuss how well *KiTe* fulfils the criteria stated in Chapter 6. The aim of this discussion is to provide some preliminary confidence that *KiTe* will address the various process characteristics described in Chapter 2 and overcome the limitations of current process models described in Chapters 3 and 4. I then examine the evidence presented in Chapter 8 and discuss the strengths and weaknesses of the evidence. The studies that comprise this evidence represent attempts to find inadequacies with *KiTe* as a solution to the problem of capture and comparison. I find that the evidence is reasonably strong, but there are some serious gaps. I finally discuss some limitations inherent in the approach taken.

Chapter 11, I summarise the thesis and suggest some future research directions resulting from the research.