Full citation: Kirk, D.C., MacDonell, S.G., & Tempero, E. (2009) Modelling software processes - a focus on objectives, in Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications. Orlando FL, USA, ACM Press, pp.941-948. http://dx.doi.org/10.1145/1639950.1640061

Modelling Software Processes - a Focus on Objectives

Diana Kirk, Stephen MacDonell

SERL, Auckland University of Technology
Private Bag 92006, Auckland 1142,
New Zealand
dkirk@aut.ac.nz, stephen.macdonell@aut.ac.nz

Ewan Tempero

Department of Computer Science
University of Auckland
New Zealand
e.tempero@cs.auckland.ac.nz

Abstract

Existing software process models such as Waterfall and XP are characterised by unstated assumptions, a consequence of which is that we cannot easily compare models or transfer data from one model to another. This means that software planners have no mechanism for selecting process activities that are best suited to individual projects. In this paper, we propose a framework for modelling software processes that supports representation and comparison of different kinds of software process. Our framework is based on a lift in focus from 'choosing activities' to 'identifying project objectives and selecting activities to meet those objectives'. We overview some evidence to support the claims of representation and comparison and discuss benefits and limitations of the approach.

Keywords: *software process modelling, software project planning, software process improvement, project objectives*

1. INTRODUCTION

Many models of the software development process have been proposed for a number of purposes. For example, some models are intended to proscribe how software is to be developed, such as *Waterfall* or *XP*. Some models are intended to help stakeholders understand consequences of decisions, such as simulation models. A third set of models supports the management of outcomes of software development projects, such as for cost estimation or defect prediction. Each kind of model reflects a particular perspective on software processes and their instantiation. Proscriptive models range from viewing the software development process as a traditional engineering process to viewing software development as an activity in which the most important characteristics relate to the humans carrying out the process. Simulation models are based on several modelling paradigms, for example, *system dynamics* and *event-driven* modelling, each characterised by a number of strengths and limitations. Predictive models generally apply statistical manipulations on existing data sets to predict future outcomes, and apply a variety of techniques to address the sparseness or uncertainty of the available data sets.

Although models differ in kind and approach, they all share a common characteristic in that each embeds (often unstated) assumptions about how the process operates or what information is needed. For the proscriptive models, the more traditional models such as Waterfall assume objectives of high quality and low cost and realise these by implementing a defined set of *engineering activities*, for example, 'specification', 'design', 'code' and 'test'. However, if a project is to deliver a new product based on novel technology to an early adopter marketplace in advance of the competition, the goal of early delivery may take priority over that of quality and a traditional process may hinder rather than support this prioritisation. Proponents of the *agile* approaches reject the 'engineering' viewpoint and talk about 'principles' and 'practices' rather than 'activities'. However, these approaches also embed assumptions about objectives and project contexts. For example, it is claimed that the *XP* approach 'reduces risk' and specific practices are mandated to mitigate against specific risks. One practice, *planning game*, involves a regular meeting between developers and client representative to ensure changing requirements are understood. Another practice, *pair programming*, mandates that all design and code is carried out by two developers working together in a specific way in order to ensure code quality. Some assumptions embedded in the XP approach are that the project must deal with fast-changing requirements, an effective client representative is available, and developers are sufficiently experienced and personality-matched to render pair programming effective. Again, if the project profile is inconsistent with the profile assumed by the approach, the process will hinder, rather than support, project objectives.

Simulation models [Drappa and Ludewig 1999, Lakey 2003, Munch 2005, Raffo 2005, Storrle 2003] generally

define a specific process and support only small, local perturbations. Assumptions are thus generally embedded in the model structure and relate both to which activities are carried out and which factors are most likely to affect outcomes. For example it may be possible to change the value that defines the effectiveness of an inspection, but not to omit the inspection from the model.

Cost estimation [Boehm 1981, Briand et al. 1999, Londeix 1987] and defect prediction [Khoshgoftaar et al. 1996, Lanubile and Visaggio 1997, Zhang and Pham 2000] models generally apply a statistical manipulation on an existing data set. However, the activities that were carried out during the process represented by the data set are generally unknown, or are known only by very high-level labels such as 'design effort'. Predictions are thus based on assumptions that the processes of the predicted projects involve the same activities as the original ones from which the data was extracted. This is seldom likely to be the case because, even if an organisation has a fixed process for all projects, it is unlikely that all developers on the new project will have the same experience and subject-area knowledge as those from earlier projects.

A consequence of the variation in implicit assumptions embedded in the different models is that it is not easy to compare models or transfer knowledge derived from one model to another. This is problematic for the following reason. There is general agreement in the software industry that software project outcomes can be maximised by tailoring the process to the specifics of individual projects. For example, a project manager has heard that applying the agile technique of pair programming results in an increase in quality in the software product. (S)he would like to know whether replacing design inspections by pair programming in a traditional waterfall process would improve quality in the delivered product. In order to model this scenario using existing modelling techniques, a model specific to the scenario would be built. If the project manager then wishes to check if a better outcome would be achieved by improving the testing process, a new model would be built to describe the new scenario. The problem is that we have no mechanism for choosing activities and parameters in a flexible way and building process-specific models is expensive and time-consuming [Raffo et al. 2005].

We believe that the existence of so many disparate approaches is a consequence of our lack of a real understanding of the software process. Kitchenham and Cairn state that ". . . before software engineering can mature as an engineering discipline, practitioners need a better understanding of the process by which software is created" [Kitchenham and Cairn 1990]. We suggest that a first step towards such understanding is to attempt to create a more general model of the process that can be applied and validated in a local way, but that forces a consideration and declaration of the specific factors that characterise the local application.

In this paper, we consider the form and some desired properties of such a model and propose a framework that we believe satisfies the properties. The motivation for the form of our framework is an observation that all current approaches are based on a communal and persistent mindset of 'we must define what we must do without giving too much thought about what we want to achieve or to project contexts'. We suggest that both traditional and agile approaches represent local 'solutions' to unstated process 'problems', where process authors have created a set of activities or practices based on specific real-world experiences without characterising the real-world aspects that informed the process. For example, traditional models have their origin in a military context, and relate to software projects that were believed to be relatively stable and were many person-years in size. Agile models have their roots in the context of smaller software organisations whose projects are inherently less stable and of smaller size and where there is increased reliance on individuals. The objectives and contexts relating to each have become secondary to the defined activities and practices and so have become unstated assumptions.

We believe that the future for software process modelling lies in a change in paradigm from 'choosing activities' to 'understanding objectives and constraints', a paradigm that inherently forces a more holistic approach to software system modelling. Our proposed framework is based on 'system variables-of-interest' (SVoI) that makes no assumptions about what constitutes a process or what the model is intended for. In contrast with the *activity-centric* nature of the above models, our approach involves first identifying the key objectives for the software project, for example 'deliver high quality artifacts' and 'increase developer product knowledge', and then establishing each as a *system factor-of-interest*. Each key factor is then operationalised as at least one *system variable-of-interest (SVoI)* and assigned an appropriate target value, for example, 'no more than ten known defects'. Process and management activities are then viewed as transformations on these SVoI. The aim of a software development process is to 'move' the values of the SVoI towards the desired outcome values. The progress that is actually made depends on the actual transformations i.e. on the contexts that affect how well the engineers are able to carry out tasks.

The SVoI represent the aspects of the project system in which we have an interest i.e. the *state* of our project system. This means that transformations are definitions of state change and can be applied one after the other, as required. Activity transformations can be based on available evidence, if some exists, or on expert opinion. This means the framework may be applied in projects for which existing organisational data is available to indicate the expected effects of an activity on, for example, effort or defect numbers, and may also be applied when no data is available but the project manager chooses to estimate expected effects based on experience.

The paper is organised as follows. In Section 2, we overview some process frameworks and models that aim to support process flexibility. In Section 3, we examine the kinds of properties we would like our model to exhibit, for example, extensibility of objectives and in Section 4, we present a framework that we believe meets these objectives. In Section 5, we discuss the claimed benefits and limitations of our approach and present some evidence. Finally, in Section 6, we summarise the paper and indicate directions for future research.

2. RELATED WORK

In this Section, we overview some process frameworks and models that aim to support the kind of process tailoring described in Section 1.

2.1 Modelling and simulation

The main source of related work is the modelling and simulation community. Software process simulation and modelling has become an "increasingly active research area" with growing numbers of publications and related activities [Zhang et al. 2008]. Techniques applied include *discrete event simulation* and *system dynamics*. The specific models overviewed below address the issue of flexibility by building processes from a number of pre-defined activity 'building blocks'.

Lakey [Lakey 2003] introduces a model to support software project prediction and management. The model is intended as a theoretical framework. It comprises four building blocks, 'preliminary design', 'detailed design', 'code and unit test' and 'subsystem integration and test'. In this framework, project-specific process models are built by creating an appropriate number of building blocks and calibrating the equations for each with project, process and product data from the project to be modelled. Examples of project factors included are communication overhead', 'tool support' and 'skill levels'. Examples of process factors are 'defects injected' and 'estimated calendar weeks'. Product factors include 'size' and 'quality'. A strength of this framework is the inclusion of cost, schedule and quality performance parameters in a holistic system as "the primary software project performance parameters of cost, schedule and quality are not independent, and they cannot be tracked and managed independently". However, customisation is achieved by copying and renaming building blocks to achieve the correct process structure and then providing the relevant input values. This means that there is no possibility of representing any activities that do not comply with one of these blocks. We suggest that customisation thus refers to changing input values rather than changing the process. Another limitation is in the predefinition of the factors that are believed to affect outcomes. The beliefs are effectively model assumptions.

Munch applies a patterns approach to the development of custom-tailored process models [Munch 2005]. He believes that "The development of high-quality software or software-intensive systems requires custom-tailored process models that fit the organizational and project goals as well as the independent contexts" (page 1). In Munch's solution, a process pattern is a reusable fragment of a process model that represents an activity. Patterns can be combined to represent combinations of process models. Information for each pattern includes attributes and a description of how attributes change when the pattern is applied, for example, causing a change to 'reliability' [Munch 2005]. In this model attributes may relate to process state (for example, 'not in maintenance activity') or process goals (for example, 'Maximal effort is less than 2000'). Required goals are thus modelled as restrictions on project attributes and include only those over which the project has control. This means that the model does not support objectives such as 'developer subject area knowledge' and other human-related goals. In addition, the rules for transformation form an integral part of the model i.e. assumptions are embedded in the model.

Raffo et. al. describe an approach for creating *Generalised Process Simulation Models (GPSM)* [Raffo et al. 2005]. The approach consists of constructing a process from a library of generic process building blocks, for example, relating to 'Design', configuring the inputs to blocks for specific environments and viewing outputs relating to time, cost, quality and functionality. Although the approach supports a degree of flexibility in process construction, there is an assumption of 'traditional process' and a restriction of outputs to those relating to time, cost, quality or functionality [Raffo 2005]. This means that the GPSM model as constructed cannot be used for simulating less traditional processes or for modelling, for example, the effects of 'team meeting' on 'developer product knowledge'.

2.2 Other frameworks

Several authors have proposed approaches that reduce risk by enabling a planner to select activities that will support organisational objectives. Models such as *Spiral* [Boehm 1988] and *Rational Unified Process (RUP)* [Kruchten 2000] aim to address risk by facilitating flexibility as regards which development activities are performed. However, there is an assumption of 'traditional' objectives and so the approaches cannot represent, for example, architectural discussions to increase developer understanding.

Recent contributions from collaborations involving the University of Southern California include combining process elements [Bhuta et al. 2005], tailoring the process according to business cases [Huang et al. 2006, Yang et al. 2007] and dealing with uncertainty by fixing the variable 'Schedule' [Yang et al. 2007]. The underlying paradigm for these contributions is that of *value-based engineering*, where key mechanisms include understanding what is the key objective for a project from a value perspective (for example, cost, quality), selecting activities that will ensure the objective is reached in the most cost-effective way and monitoring the project to ensure both objective and activity selection remain appropriate. The modelling of objectives is not formalised and so the contributions support flexibility in a limited way only.

Other tailoring approaches include Basili and Rombach's approach for tailoring processes towards project goals and environments [Basili and Rombach 1987]. Again, a specific project objective is identified and activities selected that will ensure the objective is met. However, there is no provision for examining multiple project objectives and the framework upon which the approach is based contains a number of process-related assumptions that constrain flexibility.

2.3 Discussion

The approaches described above all contribute in some way to the vision of providing project decision makers

with a mechanism for defining desired outcomes and selecting activities according to outcomes and project contexts. Some highlight the need to focus on a specific project objective and to select activities most likely to ensure the objective is met; some identify the need to consider multiple objectives in a holistic way; some support modelling at different levels of granularity; some acknowledge the need to consider human-related aspects in addition to technical ones. However, each is constrained in some way and the vision is not achieved for the general case. The constraints that characterise the models relate to assumptions about the kinds of activities that take place in software projects and a limiting of outcomes-of-interest to the standard project outcomes of cost, quality, etc. The result is an inability to easily mix activities from both traditional and agile worlds or to include objectives and activities that relate to people and product, as well as to project.

In the next section, we consider some properties for a model that addresses the constraints described above.

3. MODEL PROPERTIES

If we are to model software development processes in a flexible way, we require a model that allows us to capture any process model and supports comparison of processes in relation to desired objectives and construction of new processes. If we are to model software development processes in a flexible way, we need a model that allows us to: capture existing processes, provide comparison of processes, and construct new processes. As a first step towards creating an appropriate model, we present some considerations relevant to this intent and draw on these to suggest some properties we would like our model to exhibit. Our considerations are sourced from characteristics of existing processes, model limitations as identified in Section 2 and some observations of real-world situations.

3.1 Holistic approach

Holistic considerations relate to the idea that researchers and practitioners have an interest in many kinds of objectives and these cannot be considered in isolation.

• Traditional approaches to modelling the software development process focus on outcomes such as 'Cost' and 'Product Quality' and view knowledge as being held in documents. The more recent agile approaches are characterised by a greater focus on people, with concepts such as 'Shared Vision' and 'Team Memory'.

• Many models of the software development and management processes restrict outputs to single variables, for example, 'Number of Defects'. However, decision makers generally need to understand all of the consequences of a process decision. For example, when considering the effects of introducing an inspection into the process, the decision maker may be interested in all of defect reduction, additional cost and increase in subject-area knowledge [Kitchenham et al. 2002].

• Product line processes and management-by-projects require that the status of a delivered product and information about individuals are available as inputs to subsequent projects.

Property 1: Our model must have the capability of representing the effects of a process on a number of factors-of-interest, both product- and human-related.

3.2 Process definition

Process considerations relate to the idea that many kinds of process are possible.

• Different studies use the same term for a task to mean different things. For example, 'Design' might mean 'Create formal designs from requirements', 'Design the code based on discussions with the client', 'Create design documents to be formally inspected', and the like. This means we cannot easily compare studies and build cumulative knowledge.

• Some activities, for example, 'share knowledge', effect change to people only (not product).

• Models and studies involve processes of varying granularities for example, an XP process or a 'Design' task.

Property 2: Our model must have the capability of representing all existing and future processes in an unambiguous way and must allow processes of any granularity.

3.3 Policy support

Policy considerations relate to the need to support project managers who must decide, for example, when to commence a coding activity or release a product.

• Decision makers often want to specify that an activity can commence before its predecessor is complete, for example, 'commence coding when designs are 80 percent complete'.

• Release managers need to define 'readiness for delivery' in terms of product factors, for example 'deliver when all critical defects have been removed'.

Property 3: Our model must provide sufficient information for managers to make policy-based decisions about activity commencement and completion without dictating policy.

4. THE PROPOSED MODEL

In this Section, we present an overview of a proposed framework that exhibits the above properties. For reasons of pragmatism, we present a restricted overview only. A full description is available in [Kirk 2007].

4.1 Modelling paradigm and context

The framework presented in this paper is based on the concept of defining the objectives that are relevant for a project, representing each as a *system factor-of-interest (SFoI)*, operationalising each SFoI as one or more *system variables-of-interest (SVoI)* and transforming the values of these SVoI by application of activities. Objectives may

relate to any aspect of the software system, for example, to business-, product- or stakeholder-related objectives.

For our framework, we apply the term *RealisedProcesses,(RP)* to represent a software development process [Kirk 2007]. The standard use of the term *process* generally refers to a description of a set of technical or management tasks and does not include any non-technical factors, for example, relating to humans. Our definition as *transformation on SVoI* means that all aspects of the transformation are included. If we consider an inspection that transforms 'Discovered defects' and 'Effort', we understand, for example, that two actual inspections will effect different sizes of transformation according to the experience of the participants.

A software development process RP is modelled as:

$RP = \{Values, V_{RP}, Activities_{RP}, s_0, T\}$

where VALUES is the set of all possible *values*, V_{RP} is a set of *system variables of interest (SVoI)*, $Activities_{RP}$ is a set of *activities*, s_0 is the initial state and T is the set of states representing the target of the process. This model of a process can be treated as a (potentially infinite) state machine, with states implied by V_{RP} and the transformation function given by $Activities_{RP}$.

VALUES is the set of all possible values. There is no restriction on what constitutes a value (e.g., tuples, sets, and other structures are allowed). This formulation means that the set of values is the same across all processes modelled.

A *state* is a set of VALUES. Different *RealisedProcesses* may care about different parts of the state.

We attach meaning to any given value by associating it with a *system variable* (usually abbreviated to variable). If a set of states have values associated with the same variable, and those states are considered to be in some order, then the interpretation is that the variable (potentially) changes value across the states. A system variable then represents a part of the world that we are interested in modelling.

We can describe a state as a set of variables that have values. We decide what variables we are interested in (the SVoI), and that dictates the states that are relevant to us.

An *activity* models the actions carried out by developers, such as 'software inspection', 'create requirements', and so on. An activity (potentially) causes a change in state, and so can be regarded as *transforming system variables*. Any given activity will probably only transform a few variables. For example, a 'software inspection' activity may change the 'number of known defects' value but not the 'number of requirements' value, whereas a 'create requirements' activity may not change the 'number of known defects' value. For an activity *a*, we denote the system variables it transforms by $Vars_a$. This gives us, for an activity *a*, $a : States_{Varsa} \rightarrow States_{Varsa}$

An activity is defined by the variables it transforms and how it transforms them. If it is allowed that new requirements identified in the 'software inspection' activity are to be recorded, then that activity could also change the 'number of requirements' value, and it would be considered to be a *different* activity to the one that ignores any requirements identified. Thus our framework allows us to explicitly model variations on activities that are based on the desire to model different outcomes.

A RealisedProcess RP is determined by the set of SVoI V_{RP} and a set of activities $Activities_{RP}$ that transform some subset of V_{RP}. The set of activities in the process together make up the transformation function of the state machine that is the process. Finally, s_0 is then a state in $StatesV_{RP}$ representing the initial state for the process, and $T \leq StatesV_{RP}$ is the states indicating the process has reached the target values for the SVoI.

4.1.1 Illustration

As illustration of the basic model, in Figure 1, we depict three activities, 'Code', 'Unit test' and 'Fix defects', changing SVoI 'Effort' and 'Defects'. In this illustration, the 'Code' activity increases 'Effort' and a number of 'Defects' are injected. 'Unit Test' also increases 'Effort' but no change is effected to 'Defects' (although some defects may be uncovered). 'Fix Defects' increments 'Effort' and reduces 'Defects' as existing defects are resolved and a smaller number are injected as a result of the activity.

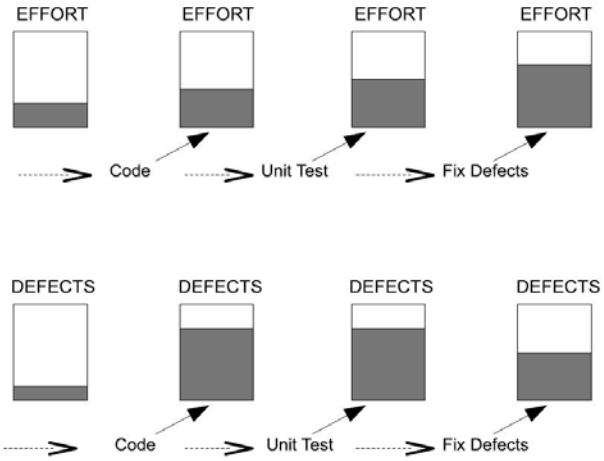

Figure 1. Activities changing Effort and Defects

Note that we are transforming variables, not process outputs, such as documents. In our model, documents may contain variable-related content, for example, the quality-related aspects of the process may be captured in a number of documents. However, the SVoI for the process represents an abstraction of quality in which we have some interest i.e. represents how we operationalise quality.

As can be seen, our model imposes very few limitations or, indeed, meaning. The meaning of variables and activities is determined by the modeller's interpretation of the values and transformation of variables. We see this as a benefit. For example, 'Design' could mean many things. The different variations would be modelled as different sets of variables. For example, 'Create formal designs from requirements' implies SVoI relating to 'Requirements', 'Requirements designed' and 'Formality of designs'. 'Design the code based on discussions with the client' implies SVoI relating to 'Features' and

'Features coded'. As these activities change different SVoI, they are different activities.

Even when the SVoI are the same, the transformations may be different in different contexts. Organisation A may have a group of developers all experienced in describing designs, whereas Organisation B's group may be less experienced. The models for the two organisations may have two activities 'Create designs from requirements' with the same SVoI 'Number of requirements designed' but the size of the transformations will be different reflecting the different levels of experience.

4.2 Model properties

Although the model described above is simple, we submit that the change in perspective from 'defined activities' to 'system variables being transformed' has a number of advantages relating to the properties discussed in Section 3.

Property 1: Holistic approach. As there is no constraint on what aspect of the software system can be represented as an SVoI, the framework supports the modelling of the 'traditional' objectives, such as quality and cost, the human-related objectives, for example, 'developer satisfaction', of interest to the agile community, and any other objectives, for example, relating to economic value. As we are dealing with system state, the current state, including aspects such as 'developer experience', may be made available for new projects.

Property 2: Process definition. As an activity is anything that transforms SVoI, activity content and granularity are unconstrained. This means we may model tasks such as 'pair programming' and 'team meeting' and whole processes, such as Waterfall. As our definition of an activity is unambiguous, we have a powerful way to compare techniques and tools applied during activities. In a set of studies often stated as concerning 'Pair Programming', we find that there is lack of real definition about the technique studied and about what are the input to, and outputs from, the process [Kirk 2007]. If we do not know how techniques change the environment, any comparisons we make must inevitably be superficial. Our model forces definition of what is changed and so paves the way to more robust investigations.

Property 3: System 'readiness'. As our framework is based on system state, policy decisions, for example, relating to when to start coding or deliver to a client, may be based on state, for example 'when designs are 80 percent complete' rather than activity completion, for example, 'when testing is finished'.

5. CLAIMED BENEFITS AND EVIDENCE

We claim that our proposed lift in focus from 'choosing activities' to 'identifying project objectives and selecting activities to meet those objectives' facilitates the representation and comparison of software processes in a way that encourages modellers to make transparent any assumptions relating to objectives and contexts. We suggest that the ability to compare supports:

Planning Planners must first consider objectives (encouraging systems thinking), how to operationalise these as SVoI and then may select activities that best meet objectives by carrying out 'what if' analysis [Kirk and MacDonell 2009b].

Software process improvement (SPI) We have used the approach to support a model for SPI based on an analogy of 'human health' [Kirk and MacDonell 2009a].

Project management Our approach is consistent with the recent interest in a more holistic viewpoint of the project management function [Sauer and Reich 2009].

Software research meta-analysis We hypothesise that our model will provide a framework to support meta-analysis of formal research studies, for example, pair programming studies [Kirk 2007].

We now consider the issue of accumulating evidence to support the proposed framework. Our evidence rests on the ability to capture and compare many different kinds of process and process models [Kirk 2007].We have chosen techniques that have been applied in the domain of safety-critical software and which provide a powerful way of organising and evaluating evidence in the wider field of evidence-based software engineering [Kitchenham et al. 2005, Weaver et al. 2005]. The techniques involve defining objectives (in our case, Represent/Compare/Combine any software process or process model) and accumulating in a tree structure different kinds of evidence, for example, case studies, surveys, expert opinion, anecdotes and controlled experiments, to show that objectives are met. One strength of the approach is transparency as it is clear what 'evidence objectives' have been considered and what kinds of evidence are available and missing. As illustration, we have shown our part of our evidence tree in Figure 2 [Kirk 2007].

Our evidence thus far includes representing and comparing typical waterfall and XP processes, exposing assumptions in some 'pair programming' studies, representing simulation models based on different modelling paradigms and identifying risks inherent in XP projects [Kirk and Tempero 2006].

5.1 Model limitations

A key limitation in the model overviewed in this paper is that at present it does not deal with uncertainty. It is widely acknowledged that the human-intensive nature of software development introduces uncertainty into the processes applied to create software-intensive products [Connor 2007, Kitchenham and Linkman 1997, Rao et al. 2008, Yang et al. 2007]. It is not generally possible to say exactly how many defects will be injected by an activity or exactly how much effort will be required. The handling of uncertainty is being addressed in future works.

A second limitation lies in the inability of the model to include SVoI whose value depends upon whether or not activities are overlapped. For example, if we consider an activity A with a duration of 5 days to complete and activity B with duration 3 days, we have no idea what is the resultant value of the 'duration' variable, because this will depend upon how activities are scheduled i.e. on the

degree of overlap. This is an inherent characteristic of the model and would require some kind of 'implementation overlay' for its solution. Allowed SVoI include only those which are incremented or scaled in a way that is independent of other transformations.

A final limitation, at least for implementation of the described framework, relates to the current lack of evidential data within the industry. Although we suggest the framework may be used to help with evidence accumulation, there exists at the present time very little data to characterise how activities change SVoI or how best to operationalise system factors-of-interest. This means that the immediate use of the framework for decision support relies on the existence of organisational datasets or expert opinion.

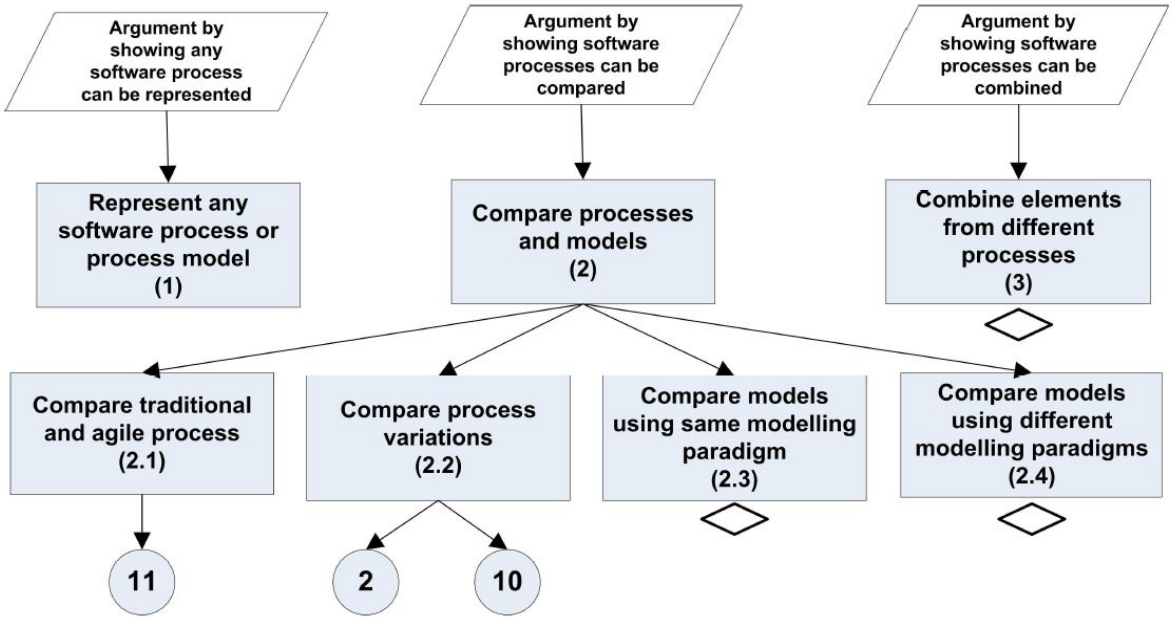

Figure 2. Evidence illustration

6. SUMMARY AND FUTURE WORK

In this paper, we suggest that existing software process models are characterised by unstated assumptions and this means that we cannot easily compare models or transfer data from one model to another. A consequence is that software planners have no mechanism for selecting process activities that are best suited to individual projects. We believe that the existence of so many disparate models is a consequence of our lack of a real understanding of the software process and suggest that, if we are to progress our understanding in any real way, we must first attempt to create a more general model of the process that can be applied and validated in a local way, and that forces a consideration and declaration of the specific factors that characterise the local application.

We consider the form and properties of such a model and propose a framework we believe satisfies these properties. We present some evidence to support this belief. Finally, we discuss some current limitations of our proposed framework.

Our next steps are to create an implementation of the framework and apply this in the areas of planning and software process improvement. To address the limitation relating to uncertainty in activity outputs, our current research includes study of probabilistic and fuzzy transformation mechanisms.

7. REFERENCES

Victor R. Basili and H. Dieter Rombach. Tailoring the Software Process to Project Goals and Environments. In *Proceedings of the Ninth International Conference on Software Engineering*. IEEE, IEEE Computer Society Press, 1987.

Jesal Bhuta, Barry Boehm, and Steven Meyers. Process Elements: Components of Software Process Architectures. In M. Li, B. Boehm, and L.J. Osterweil, editors, *SPW 2005, Lecture Notes in Computer Science (LNCS)*, volume 3840, pages 332–346, Berlin, Heidelberg, 2005. Springer-Verlag.

Barry Boehm. *Software Engineering Economics*. Prentice-Hall, Inc., 1981. ISBN 0138221227.

Barry W. Boehm. A Spiral Model of Software Development and Enhancement. *IEEE Computer*, May(11), 1988.

Lionel C. Briand, Khaled El Emam, Dagmar Surmann, Isabella Wieczorek, and Katrina D. Maxwell. An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques. In *Proceedings of the 1999 Conference on Software Engineering*. IEEE Computer Society Press, 1999.

A.M. Connor. Probabilistic estimation of software project duration. *New Zealand Journal of Applied Computing and Information Technology*, 11(1):11–22, 2007.

A. Drappa and J. Ludewig. Quantitative modeling for the interaction simulation of software projects. *Journal of Systems and Software*, 46(2/3), 1999.

Liguo Huang, Hao Hu, Jidong Ge, Barry Boehm, and Jian Lu. Tailor the Value-Based Software Quality Achievement Process to Project Business Case. In Q. Wang et. al., editor, *SPW/ProSim 2006, Lecture Notes in Computer Science (LNCS)*, volume 3966, pages 56–63, Berlin, Heidelberg, 2006. Springer-Verlag.

Taghi M. Khoshgoftaar, Edward B. Allen, Kalai S. Kalaichelvan, and Nishith Goel. Early Quality Prediction: A Case Study in Telecommunications. *IEEE Software*, January, 1996.

Diana Kirk. *A Flexible Software Process Model*. PhD thesis, University of Auckland, Auckland, New Zealand, 2007.

Diana Kirk and Stephen MacDonell. A Systems Approach to Software Process Improvement in Small Organisations. *In Proceedings of the 16th European Systems and Software Process Improvement and Innovation Conference (EuroSPI 2009)*, 2009a.

Diana Kirk and Stephen MacDonell. A Simulation Framework to Support Software Project (Re)Planning. *In Proceedings of the 35th Euromicro Conference on Software Engineering Advanced Applications (Euromicro SEAA 2009))*, 2009b.

Diana Kirk and Ewan Tempero. Identifying Risks in XP Projects through Process Modelling. In *Proceedings of the Australian Software Engineering Conference (ASWEC'06)*, pages 411–420, Sydney, Australia, 2006. IEEE Computer Society Press. ISBN 0-7695-2551-2.

Barbara Kitchenham and Roland Cairn. Research and Practice: Software Design methods and Tools. In J.-M. Hoc, T.R.G. Green, R. Samurcay, and D.J. Gilmore, editors, *Psychology of Programming*, pages 271–284. Academic Press Ltd., London, U.K., 1990. ISBN 0-12-350772-3.

Barbara Kitchenham and Stephen Linkman. Estimates, Uncertainty and Risk. *IEEE Software*, May/June, 1997.

Barbara Kitchenham, David Budgen, Pearl Brereton, and Stephen Linkman. Realising Evidence-Based Software Engineering. In *Realising Evidence-Based Software Engineering Workshop 2005, Workshop co-located with ICSE 2005*, St. Louis, Missouri, 2005. Keele University.

Barbara A. Kitchenham, Shari Lawrence Pfleeger, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering*, 28(8), 2002.

Philippe Kruchten. *The Rational Unified Process: An Introduction, Second Edition*. Addison-Wesley, United States of America, 2000. ISBN 0-201-70710-1.

Peter B. Lakey. A Hybrid Software Process Simulation Model for Project Management. In *Proceedings of the 2003 International Workshop on Software Process Simulation and Modeling (ProSim'03)*, Portland, Oregan, U.S.A., 2003.

Filippo Lanubile and Giuuseppe Visaggio. Evaluating Predictive Quality Models Derived from Software Measures: Lessons Learned. *Journal of Systems and Software*, 38, 1997.

Bernard Londeix. *Cost Estimation for Software Development*. Addison-Wesley, Cornwall, UK, 1987. ISBN 0-201-17451-0.

Jurgen Munch. Goal-oriented Composition of Software Process Patterns. In *Proceedings of the 6th International Workshop on Software Process Simulation and Modeling (ProSim'05)*, pages 164–168, St. Louis, Missouri, 2005. Fraunhofer IRB. ISBN 3-8167-6761-3.

David Raffo, Umanatha Nayak, and Wayne Wakeland. Implementing Generalized Process Simulation Models. In *Proceedings of the 6th International Workshop on Software Process Simulation and Modeling (ProSim'05)*, pages 139–143, St. Louis, Missouri, 2005. Fraunhofer IRB. ISBN 3-8167-6761-3.

David Raffo. System and method for simulating product design and development. United States Patent Application 20050160103, July 2005.

Uma Sudhaker Rao, Srikanth Kestur, and Chinmay Pradhan. Stochastic Optimization and Modeling and Quantitative Project Management. *IEEE Software*, May/June: 29–36, 2008.

C. Sauer and B.H. Reich. Rethinking IT project management: Evidence of a new mindset and its implications. *International Journal of Project Management*, 27:182–193, 2009.

Harald Storrle. Making Agile Processes Scalable. In *Proceedings of the 2003 International Workshop on Software Process Simulation and Modeling (ProSim'03)*, 2003.

Rob Weaver, Georgios Despotou, Tim Kelly, and John McDermid. Combining Software Evidence - Arguments and Assurance. In *Realising Evidence-Based Software Engineering Workshop 2005, Workshop co-located with ICSE 2005*, St. Louis, Missouri, 2005. Keele University.

Da Yang, Barry Boehm, Ye Yang, Qing Wang, and Mingshu Li. Coping with the Cone of Uncertainty: An Empirical Study of the SAIV Process Model. In Q. Wang, D. Pfahl, and D.M. Raffo, editors, *ICSP 2007, Lecture Notes in Computer Science (LNCS)*, volume 4470, pages 37–48, Berlin, Heidelberg, 2007. Springer-Verlag.

He Zhang, Barbara Kitchenham, and Dietmar Pfahl. Reflections on 10 Years of Software Process Simulation Modeling: A Systematic Review. In Q. Wang, D. Pfahl, and D.M. Raffo, editors, *ICSP 2008, Lecture Notes in Computer Science (LNCS)*, volume 5007, pages 345–356, Berlin, Heidelberg, 2008. Springer-Verlag.

Xuemei Zhang and Hoang Pham. The analysis of factors affecting software reliability. *Journal of Systems and Software*, 50, 2000.