Clarke, P. and O'Connor R.V. The situational factors that affect the software development process: Towards a comprehensive reference framework, Journal of Information Software and Technology, Vol. 54, Issue 5, May 2012. pp. 433-447.

The situational factors that affect the software development process: Towards a comprehensive reference framework

Paul Clarke [a], Rory V. O'Connor [b]

[a] Lero - Irish Software Engineering Research Centre, Dublin City University, Ireland,
[b] School of Computing, Dublin City University, Ireland
{pclarke, roconnor}@computing.dcu.ie

Abstract

Context
An optimal software development process is regarded as being dependent on the situational characteristics of individual software development settings. Such characteristics include the nature of the application(s) under development, team size, requirements volatility and personnel experience. However, no comprehensive reference framework of the situational factors affecting the software development process is presently available.

Objective
The absence of such a comprehensive reference framework of the situational factors affecting the software development process is problematic not just because it inhibits our ability to optimise the software development process, but perhaps more importantly, because it potentially undermines our capacity to ascertain the key constraints and characteristics of a software development setting.

Method
To address this deficiency, we have consolidated a substantial body of related research into an initial reference framework of the situational factors affecting the software development process. To support the data consolidation, we have applied rigorous data coding techniques from Grounded Theory and we believe that the resulting framework represents an important contribution to the software engineering field of knowledge.

Results
The resulting reference framework of situational factors consists of 8 classifications and 44 factors that inform the software process. We believe that the situational factor reference framework presented herein represents a sound initial reference framework for the key situational elements affecting the software process definition.

Conclusion
In addition to providing a useful reference listing for the research community and for committees engaged in the development of standards, the reference framework also provides support for practitioners who are challenged with defining and maintaining software development processes. Furthermore, this framework can be used to develop a profile of the situational characteristics of a software development setting, which in turn provides a sound foundation for software development process definition and optimisation.

Keywords: Software engineering process, process definition, process implementation and change.

1. INTRODUCTION

Over the past four decades, we have witnessed significant improvement in the software development field. The publication of Winston Royce's now famous waterfall approach [1] to software development in 1970 heralded the beginning of a new era for software development, an era which recognises that software development is a complex, human centric activity that is subject to many pitfalls if not appropriately organised.

Since Royce's waterfall, many alternative approaches to organising software development have been proposed. In the 1980s, the International Organization for Standardization introduced ISO 9000 [2] as a family of generic quality management standards that aimed to promote the quality of deliverables through managing the quality of the production process, and ISO 9000 offers many benefits for software development [3-5].

Despite the reported benefits of ISO 9000, it is sometimes criticised for being a general product quality approach that is not software development centric [3-7] and for lacking a self-improvement dimension [4], [8]. Process maturity reference frameworks, such as ISO/IEC 15504 [9] and the Capability Maturity Model Integrated (CMMI) [10] address some of the criticisms of ISO 9000 and these frameworks have also benefited software development enterprises [11-19]. Such process maturity reference frameworks recognise that the software development process can have various degrees of maturity, where increased process maturity results in increased predictability in relation to schedule, cost and quality levels.

ISO 9000, IS0/IEC 15504, CMMI, as well as a variety of additional related approaches, represent very significant contributions to the software development field. However, they have been criticised for being overly mechanistic [20] and for not adequately dealing with the dynamic nature of software development – wherein requirements are often difficult to identify and finalise [21], [22]. These criticisms have fuelled the case for increased flexibility in software development approaches, and this has given rise to agile software methods. By applying the principles of the agile manifesto [21], agile methods seek to support the sometimes dynamic nature of software development. Agile approaches such as eXtreme Programming (XP) [23], Scrum [24], and Feature Driven Development (FDD) [25], have gained a significant foothold in the software development field, with clear benefits reported in a number of studies [20], [26-29]. However, agile methods also suffer a number of criticisms, including that they may only be suited to small software development teams [30] and that they may require *premium* software engineers [30], [31].

The various approaches outlined above (ISO 9000, ISO/IEC 15504, CMMI and agile software development methods) offer guidance to the complex problem of software development - with the protagonists from each camp often locked in a bitter battle, each arguing that their particular approach is best. However, despite the significant reported benefits of these approaches, attempts to generalise a single software development process for a multitude of settings have met with considerable challenges and the evidence suggests that no single approach to software development "is universally deployed or even universally useful" [32]. An important question to ask is: why is no single approach universally useful? To answer this question, we must examine the basic requirement of a software development process.

According to [33], the basic requirement of a software development process is that it "should fit the needs of the project". The needs of the project are informed by the situational context wherein the project must operate and therefore, the most suitable software development process "is contingent on the context" [34]. Software development and process managers must "evaluate a wide range of contextual factors before deciding on the most appropriate process to adopt for any given project" [35], a view that is shared by [36], where it is stated that the chosen development approach should "best fit the conditions, product, talent, and goals of the markets and organisations". Kautz [37] also shares this view, stating that process improvement initiatives, especially in small companies, should

be “adjusted to their particular situation and… should not slavishly follow one of the comprehensive approaches”.

When it comes to defining a software development process, it therefore seems likely that the claim that “one size fits all” is in fact a myth [38] and that one of the central reasons accounting for this is the rich variation in situational contexts. Although individual situational circumstances are reported as a key consideration when constructing a software development process, there does not exist a general reference framework of the various dimensions of situational circumstance that affect the software development process. Therefore, we analytically analyse a selection of significant contributions from a broad spectrum of related research areas. Our analysis, which applies the rigorous data coding techniques from Grounded Theory [39], produces a comprehensive initial reference framework of the situational factors that affect the software development process.

In Section 2, we present the related research domains, with details of the data sources selected as the building blocks for the initial reference framework. Section 3 outlines the systematic approach to constructing the reference framework. Section 4 describes the application of the systematic approach to the development of the general reference framework of the situational factors that affect the software development process. Section 5 discusses the utilisation of the initial framework, along with suggestions for future work and an examination of the potential weaknesses of the framework. Finally, a conclusion is provided in Section 6.

2. RELATED WORK

Although no general comprehensive reference framework of the situational factors affecting the software process presently exists, there have been a number of related works. While individually helpful to some extent, none of these related works offers a comprehensive general reference framework of the situational factors affecting the software development process – nor was this their intended purpose. We therefore construct an initial reference framework of the situational factors affecting the software development process, incorporating a range of related domains. To avoid the omission of key considerations, we conducted a comprehensive literature review and identified seven distinct related domains that offer data sources that can contribute to the situational factors reference framework. By extensively analysing a selection of significant contributions from seven distinct but overlapping domains, we mitigate the risk of overlooking key considerations – and later independent reviewing steps further insure against unintended omissions. While the resulting labelling and classification that we present later may be open to debate, the rigorous approach to identifying and synthesising a broad range of significant related works ensures that the reference framework is systematically developed and comprehensive. The seven identified domains are summarised in Table 1, along with an explanation of why these domains were selected for inclusion. Within each of the selected domains identified in Table 1, some of the most influential contributions over recent decades are identified as data sources to be included in the analysis process. Primarily, the targeted contributions have had a significant impact on the research area. Furthermore, recent contributions, which don’t yet have a strong citation record but which are considered by the authors to offer comprehensive sources for the initial reference framework, are also included in the analysis process.

From the related domains identified in Table 1, a total of 22 individual works are selected for inclusion in the analysis process that will render an initial reference framework of the situational factors affecting the software development process. Although we have applied the selection criteria identified in Section 2.1, to some extent this initial selection does represent the personal choice of the researchers. However, the selected works represent contributions from some of the most influential sources/authors in software engineering, with sources including the IEEE, the ISO, the SEI, Barry Boehm and Alan Albrecht. Furthermore, they span a period of 40 years of research and have accumulated a total of over 5000 citations. While it is not practically possible to consult and integrate every single possible data source for the initial reference framework of the situational factors affecting the software development process, it is essential that a broad spectrum of sources is incorporated and that the selected contributions represent important and substantial contributions in the individual

research domains. Adopting this broad yet selective approach to data source identification enables the construction of an initial reference framework that is reasonably comprehensive, credible, and generally useful. Furthermore it provides a foundation upon which to build a more comprehensive reference framework of the situational factors that affect the software development process in the future.

Table 1
Related Research Domains

Domain	Reason For Selection
Software development models and standards	Although software development models and standards do not directly seek to identify the situational factors that affect the software development process, they sometimes describe the situational context in a general sense.
Risk factors for software development	The domain of risk factors for software development is concerned with identifying the dimensions of risk that are important considerations for software development projects. Anything that is considered to be a risk for software development is a concern that the software development process may need to take into consideration.
Software development cost estimation	The domain of software development cost estimation is concerned with projecting the costs of software development projects early in the development cycle. Anything that gives rise to a cost in a software project is potentially a consideration for the software development process.
Software development environmental factors	Some research has examined the software development environment and identified a set of characteristics that describe a software development setting. Such characteristics are related to the situational factors that affect the software development process.
Software process tailoring	The domain of software process tailoring is concerned with taking generalised software development approaches and tailoring them to specific software development settings. Consequently, this domain offers some description of the important considerations when tailoring the software process.
Degree of required software process agility	With the advent of agile software development, some research has been conducted that is aimed at identifying the extent to which the software development process should be agile in any given setting. The degree of required agility is an important characteristic for the software development process.
Software engineering body of knowledge	The software engineering body of knowledge (SWEBOK) provides a consensually-validated knowledge repository for software development, and as such, provides some guidance with respect to the factors of situational context that are important considerations for the software development process.

2.1 Selected Data Sources

A review of the literature reveals that currently, there is no single general reference point for the situational factors that affect the software development process. However, as highlighted above, there are a number of different domains that have strong associations with the situational factors that influence the software development process – as outlined in Table 1. In selecting the data sources, the authors applied a multi-faceted approach. Firstly, we looked to sources that were highly cited in their particular domain. However, given that works in the domains under examination may only deal tangentally with situational factors that affect the software development process, we also identify papers for inclusion based on other criteria. Therefore, we consider the number of situational factors identified in the work (with a bias towards those contributions that yielded higher numbers of situational factors). We also consider the number of references provided in the data source, since this permits a more thorough examination of the meaning of different factors presented in the data source. In addition, rather than limiting the time period of works for inclusion, we decided that a data source from the more distant past could be selected for inclusion if it was significant in depth of detail and overall contribution (this was quite an important consideration as significant works in the cost estimation and risk factors for software development domains date from earlier research eras). In this section, we discuss the related research domains, explaining their relevance to the exercise of constructing an initial reference framework of the situational factors that affect the software development process. Furthermore, we identify the data sources for inclusion in the initial reference framework construction exercise.

2.1.1 Software Development Models and Standards

In relation to the first domain, software development models and standards, there have been many contributions over the past thirty years. For the purpose of the framework generation analysis process,

two of the more extensive and established models and standards are included as data sources: ISO/IEC 12207 [40] (the process reference model associated with ISO/IEC 15504 [9]) and CMMI [10]. These two sources have been selected as they represent the two most widely adopted software process maturity reference frameworks, and although they do not seek to identify the full spectrum of situational factors that affect the software process, they do give a high level indication of the general categories of factors that are believed to be important when defining a software development process.

ISO/IEC 12207 recognises that all software development projects are "conducted within the context of an organization" [40] and offers two broad classifications of the factors that affect the software development process: *Organisational process outcomes as factors affecting process* and *Project factors affecting the process definition*. These two broad classifications are further broken down into a set of factors that are included as data sources for the construction of the general reference framework. In common with ISO/IEC 12207 [40], CMMI [10] does not present a detailed listing of the situational factors that affect the software development process. However, CMMI does recognise that "an organization's processes operate in a business context that must be understood" [10]. Furthermore, CMMI states that "issues related to finance, technology, quality, human resources, and marketing are important process considerations" [10]. In addition to these important process considerations, CMMI also identifies a set of *derived requirements* that essentially represent the contextual requirements that may not be explicitly stated in customer requirements, but which are inferred (1) from contextual requirements (e.g. applicable standards, laws, policies, common practices, and management decisions), or (2) from requirements needed to specify a product component. These aspects of situational context described in ISO/IEC 12207 [40] and CMMI [10] are included as data sources for the reference framework data analysis.

2.1.2 Risk Factors for Software Development

The second related domain, risk factors for software development, is concerned with identifying the dimensions of risk that are considered important for software development projects. A risk factor "represents an uncertain attribute of a project or its contextual environment" [41] that can "adversely affect a project, unless project managers take appropriate countermeasures" [42]. Such countermeasures can include software development process changes and therefore risk factors are an important data source for consideration when constructing a general set of situational factors that affect the software process.

The area of software development risk is a large and quite distinct research domain that has grown over a number of decades and it is not the intention of this data analysis to consider each and every contribution to the domain. However, it is important that a comprehensive profile of the risk factors for software development is extracted from the existing literature – and the following paragraphs identify the contributions that this work includes so as to achieve such a comprehensive profile.

Casher [43] presents a project risk framework, identifying 21 risk factors associated with information systems projects. While [43] identifies a substantial listing of the risk factors associated with software development, there is no attempt to rank the individual risk factors in terms of their priority or to identify what might be considered to be the more important risk factors. This prioritisation of risk factors is an area that Barry Boehm has examined, and his research culminated in the publication of the *Top 10 Software Risk Items* [44]. Based on a survey of "several experienced project managers", the "top ten primary sources of risk on software projects" are identified [44].

Later research into the prioritisation of risk factors finds that Boehm's top ten listing of software risk items (along with much of the pre-existing risk-related literature) is focused on what are termed *execution* risks [45]. *Execution* risks are considered to be those risk factors over which the project manager has some control. However, [45] attempts to include a broader spectrum of risks than just the *execution* risks and consequently, the top eleven risk factors identified by [45] contain a number of risk factors that are absent from the earlier classification compiled by Boehm [44].

In an attempt to synthesise the best aspects of all pre-existing risk management techniques, [46] combines the benefits and wisdom of established software development risk management approaches into a single risk management model, a so called *socio-technical* model of risk management, which identifies 17 distinct risk items which are classified under five risk classifications. Ropponen and Lyytinen [47] subsequently carry out a survey of 83 project managers in order to determine the components of software development risk and to what extent these components are influenced by environmental factors. This later study by Ropponen and Lyytinen identifies six software risk components (broken down into 26 associated risk items) and four environmental factors that are acting on the risk components. Ropponen and Lyytinen [47] acknowledge that some of the environmental factors may be beyond the control of the project manager (e.g. setting project deadlines and making decisions relating to subcontracting). Nonetheless, the authors conclude that even these issues are "crucial issues to pay attention to and to take actions to reduce risks related to them" [47].

Building upon the work of Ropponen and Lyytinen [47] and others, Barki et al. [48] develop a framework of software development risks that encompasses a broader spectrum of risks than the earlier risk frameworks – including new risk items such as *interpersonal conflicts* and *user attitudes*. The framework includes 35 risk factors that are classified under 18 *underlying concepts*. Furthermore, some of the risk factors are further decomposed into a series of components. The broad risk factor framework introduced in [48] heralded the arrival of more extensive risk factor models. In 2004, a larger framework again [42] is introduced, this time with 53 risk factors (with considerable overlap with the risk factors identified earlier in [48]).

More recent work that seeks to develop a method for pricing the risk factors for software development has also identified a list of "several software development risk factors" [49]. These risk factors are set out in two publications, both from 2010 [41], [49]. While these two recent works provide some modern research on risk factors for software development, they do not offer new additional risk factors over the earlier risk factor related studies.

The software development risk contributions outlined above span a period of almost thirty years and the significant depth of information provided in these contributions is included in the reference framework data analysis.

2.1.3 Software Development Cost Estimation

The third related domain, software development cost estimation, is concerned with projecting the cost of software projects early in the development lifecycle. A number of approaches exist to support the early estimation of cost associated with a software development effort, with Delany and Cunningham [50] suggesting that these techniques broadly fall under three principal categories; 1) algorithmic models which predict estimates of effort and duration using parametric equations, 2) expert judgement involving predictions based on the skill and experience of one or more experts, and 3) analogy involving the comparison of one or more completed projects with details of a new project to predict cost and duration.

Expert judgement is considered to be "consulting with one or more experts, who use their experience and understanding of the proposed project to arrive at an estimate of its cost" [51], while estimation by *analogy* "involves reasoning by analogy with one or more completed projects to relate their actual costs to an estimate of the cost of a similar new project" [51]. *Algorithmic* models "use mathematical formulae to predict effort and duration as a function of a number of variables" [50]. Unlike *expert judgement* and *estimation by analogy*, which attempt to leverage the accumulated knowledge and information in individual settings, *algorithmic* models attempt to provide a generic model from which software development costs can be estimated in any setting. As such, *algorithmic* models capture some of the situational factors that can affect the software development cost in a general sense. These cost factors are contained within the *algorithmic* models and are potentially a valuable source of knowledge when constructing a comprehensive reference framework of the situational factors

affecting the software development process – since any factor that affects the cost of software development can have a direct influence on the software development process.

According to Kitchenham [52], there are two categories of *algorithmic* cost estimation models: empirical factor models which provide an estimate of the value of a cost parameter, and constraint models which demonstrate the relationship between the various cost factors. Of the empirical factor models, the Constructive Cost Model (COCOMO) [51] and its later revision COCOMO II [53] are the most widely accepted and used [54]. COCOMO II has 17 cost factors that are considered to represent a conclusive listing of the factors that can affect the cost of a software development effort. Furthermore, COCOMO II includes five additional scale factors that are incorporated into cost estimation: *Precedentedness*, *Flexibility*, *Design/Risk*, *Team Cohesion*, and *Process Maturity*. To some extent, these scale factors embody the potential risk and they are applied to the cost as a multiplier of the basic cost convention captured in the accumulated cost factors. Of the constraint models, the Software Lifecycle Methodology (SLIM) [55] is most commonly in use [50]. The SLIM model is expressed as two equations describing the relationship between the development effort and the schedule. Four basic SLIM parameters and four productivity factors are the essential components of the two equations.

Both the empirical factor models and the constraint models use estimates of the number of source lines of code as a size driver. However, it can be difficult to estimate the lines of code in advance of implementation and an alternative approach, which measures the size of the functionality of the software based on functional specifications, is consequently proposed as an alternative to empirical factor models and constraint models. *Function point analysis* (FPA) [56] is the first and among the most commonly used functional size measurement approach [57].

The FPA method is based on the idea of determining size based on functional requirements from the end user's viewpoint, taking into account only those elements in the application layer that are logically visible to the user and not the technology used [58]. Application elements, which include things such as external inputs and outputs, are assigned weighting factors which are summed together rendering an unadjusted function point rating. The unadjusted function point rating is scaled up or down based on an evaluation of the application's characteristics.

While leading algorithmic cost estimation models such as COCOMO II, SLIM and FPA provide a mechanism for estimating the cost of software development efforts, there remains a debate as to the effectiveness of using these approaches [50]. This debate has given rise to some emerging cost estimation techniques, including the application of case based reasoning (CBR). CBR [59], [60] involves matching the current problem against similar problems that have already been encountered in the past and thereafter, using earlier solution strategies to inform engineering solutions to present problems. CBR is therefore a problem solving technique that is based on the reuse of past experiences. Delany & Cunningham [50] apply this problem solving technique to the software cost estimation problem, claiming that it is a more effective approach to software cost estimation than algorithmic cost estimation models. In order to apply CBR to software cost estimation, it is first necessary to identify the features of the software cost estimation case (or problem). Having conducted a literature review into related material on software cost estimation, Delany & Cunningham [50] develop a list of case features that are considered to capture the 26 key aspects of the software cost estimation case.

The software development cost estimation contributions outlined above span a period of over thirty years and the significant depth of information provided in these contributions is included in the reference framework data analysis.

2.1.4 Software Development Environmental Factors

The fourth related domain, software development environmental factors, is concerned with identifying a set of characteristics that can describe a software setting. Environmental factors are considered to be

the “factors that characterize a project and its environment” [61] and there have been a number of contributions in this area [61-64]. In the case of Xu and Ramesh [61], four primary categories of factors are identified. These categories encompass the environment of a software development process: *Project*, *Team*, *External Stakeholders*, and *Organisation*. These four categories are comprised of 20 distinct software factors, including such items as *project size*, *project type* and *turnover of personnel*. In addition to identifying these environmental factors, [61] also presents a model of software process tailoring that consists of four distinct dimensions: *Set Project Goals*, *Tailor Process*, *Assess/adjust Environmental Factors*, and *Evaluate Challenges*. In this process tailoring model, the four dimensions are constructed so as to be interdependent. Changes to the software process can give rise to changes in the environment, and changes in the environment can also give rise to changes in the software process. This bilateral relationship is an interesting concept, as it depicts the software development setting as being quite fluid, where changes in one dimension give rise to changes in another dimension, which may in turn give rise to changes in another dimension again (including changes in the dimension in which the initial change originated). This inter-relationship is considered to be representative of an *amethodical* process [65] rather than a *methodical* process, suggesting that software development processes are highly dynamic, that software development is an opportunistic process that needs to be adapted and tailored constantly during execution of the process.

The software development environmental factors identified by Xu and Ramesh [61] offer a significant depth of information which is included in the reference framework data analysis.

2.1.5 Software Process Tailoring

The fifth related domain, software process tailoring, is concerned with taking generalised software development approaches and tailoring them so specific software development settings. As outlined in the introduction, adjustments to standard software processes are necessary to make them suitable for specific environments [66]. Such adjustments are often referred to as process *tailoring*, an activity which has been described as “adjusting the definitions and/or particularizing the terms of general description to derive a description applicable to an alternate (less general) environment” [67]. Process tailoring factors are of relevance when constructing a reference framework of the situational factors affecting the software process, as identified by Coleman and O’Connor [68] who state that “contextual issues… [are among] the main inputs to the tailoring process” [68].

Cameron [69] explains that “the diversity of IT projects frustrates any direct attempt to systematize the processes used for their development. One size just won’t fit all… All too often, deviation from a standard methodology is seen as an imperfection, as an unwelcome compromise (despite the fact it always happens!)” [69]. Furthermore, Cameron [69] suggests that “the many factors that contribute to the variation among projects are exactly the factors that influence tailoring decisions” [69] and identifies a set of five process tailoring factors. Building on the work of Cameron [69] and others, Ferratt and Mai [70] identify a set of six factors affecting process tailoring decisions. The software process tailoring contributions outlined above present another rich data source for inclusion in the reference framework data analysis.

2.1.6 Degree of Software Process Agility

The sixth related domain, the degree of required software process agility, is concerned with identifying the extent to which the software development process should be agile in any given software development setting. Perhaps the most notable contribution in this domain is from Boehm and Turner [38], who present a model that can be applied in order to determine the extent of agility that is required in a software development setting. The framework identifies five key environmental characteristics that are important in determining the required agility. The basis of the Boehm-Turner model is that by determining the importance of the five key dimensions, it is possible to reach a decision in relation to the degree of agility required in a given project or setting. Included in these dimensions are factors such as *size* and *personnel* – both of which are considered important for Boehm and Turner in assessing the degree of required agility. The degree of agility contribution from

Boehm and Turner [38] outlined above offers further valuable source data for inclusion in the reference framework data analysis.

2.1.7 Software Engineering Body of Knowledge

The seventh and final related domain, the Software Engineering Body of Knowledge (SWEBOK), was created with the purpose of providing "a consensually-validated characterization of the bounds of the software engineering discipline and to provide a topical access to the Body of Knowledge supporting that discipline" [71]. It is designed to "serve as a compendium and guide to the body of knowledge that has been developing and evolving over the past four decades" [71] and recognises that "the context of the project and organization will determine the type of process definition that is most useful" [71]. SWEBOK [71] distinguishes between the factors that are important when considering the efficacy of the software process and the important variables affecting the software process definition, and provides a listing of the important factors for consideration when defining a software development process. As such, SWEBOK [71] provides an important data source for inclusion in the reference framework data analysis.

2.2 Summary

This section has outlined the research areas and contributions that represent strong sources of data for the data analysis associated with the development of an initial comprehensive reference framework of the situational factors that influence the software development process. The data sources identified represent some of the most significant contributions, from some of the most significant contributors over the past four decades. Furthermore, recent contributions that are considered to be of direct relevance but have not yet had the time to build up a strong citation record are also included in the data source listing.

In building an initial reference framework of the situational factors affecting the software development process, it is important to identify and integrate the collective wisdom from related research domains. However, given the large volume of contributions over the past forty-plus years, it is not practically feasible to consult every single possible source from every research domain – nor is it necessary (since there is often a strong overlap of data within individual research domains). It is, however, important that a broad set of significant and comprehensive contributions are selected from the related research domains – and this is the philosophy that this research has adopted.

This section has described the seven related research domains that were selected for inclusion in the data analysis associated with the reference framework construction. Applying the selection criteria identified at the start of this section, 22 individual contributions have been selected and examined. These 22 contributions have accumulated in the region of 5000 citations and have provided almost 400 individual data items that can now be used to construct an initial exploratory reference framework of the situational factors that influence the software process definition.

3. METHODOLOGY

Having identified a broad set of comprehensive works for use in constructing an initial general reference framework of the situational factors affecting the software process, next we must distil the data from the various sources into a core set of situational factors that affect the software process. While the data in the various sources is highly relevant to the situational factors framework construction, there is an inconsistent use of language, and there is a wide variety of content and classifications. We must therefore analyse the data sources and generate core concepts from what are disparate sources. Such an analysis requires that we systematically develop patterns of meaning in the data and that the data is gradually transformed into a coherent categorisation of situational factors. Grounded Theory [39], with its data coding and concept elaboration elements, offers an ideal framework for this type of analysis.

3.1 Data Coding in Grounded Theory

Grounded Theory is a general methodology of analysis linked with data collection that systematically applies a set of methods to generate an inductive theory about a substantive area [72]. The purpose of Grounded Theory is to "build theories from data about the social world such that theories are 'grounded' in people's everyday experiences and actions" [73]. Within the Grounded Theory family, there exists some philosophical disagreement with respect to the application of Grounded Theory in practice [74]. However, this research is concerned only with the data analysis methods used by grounded theorists which are common across all grounded theory approaches and therefore, the disputes in relation to the application of grounded theory are not relevant concerns in this instance.

Grounded Theory "emphasises the systematic approach to data collection, handling and analysis" [75]. This research seeks to adopt a similarly systematic approach to data handling and analysis and therefore borrows three key data management and analysis techniques from Grounded Theory; (1) coding, (2) constant comparison, and (3) memoing.

In Grounded Theory, data analysis involves what is commonly termed coding, "taking raw data and raising it to a conceptual level" [76]. Coding involves interacting with data using techniques such as asking about the data, making comparisons between the data, and in doing so, deriving concepts to stand for those data, then developing those concepts in terms of their properties and dimensions [76]. Therefore, essentially coding is "the process of defining what the data is about" [77], of "deriving and developing concepts from data" [76], whereby "codes capture patterns and themes and cluster them under an evocative title" [78]. For this research, the initial data codes are the individual identifiers and classifications that each of the data sources uses to identify the key atoms of data and their associated concepts.

In Grounded Theory, constant comparison refers to "the analytic process of comparing different pieces of data for similarities and differences" [76]. This method of analysis "generates successively more abstract concepts and theories through inductive processes of comparing data with data, data with category, category with category, and category with concept" [77]. This type of comparison is considered essential to all analysis because it allows the researcher to differentiate one category or theme from another and to identify properties and dimensions specific to that category or theme" [76]. In the case of this research, the various data codes provided by the various data sources must be compared for similarities and differences through constant comparison until a final rendering of core factors and categories emerges – this constant comparison is in effect a distillation process with successive iterations producing a more perfected integration of the underlying component data parts. Data coding using constant comparison becomes quite complex, since "simultaneously many categories and their properties may be emerging at different levels of conceptualization and different ways of being related by theoretical codes" [79]. Glaser [79] recommends that this build-up of complexity is kept track of by the *memoing* process.

The coding of data in Grounded Theory occurs "in conjunction with analysis through a process of *conceptual memoing*, capturing the theorist's ideation of the emerging theory" [77]. Memos can be considered to be "written records of analysis" and during the process of memoing, a certain degree of analysis occurs [76]. The very act of memoing "forces the analyst to think about the data and it is in thinking that analysis actually occurs" [76].

Grounded Theory's tandem process of coding and memoing helps to alleviate the pressure of uncertainty by challenging the researcher to stop and capture, in the moment, their conceptual ideas about the codes that they are finding [77]. As coding and memoing progress, patterns begin to emerge.

3.2 Framework Development Technique

The task of developing common factors and classifications for the disparate data sources identified earlier shares many of the challenges of a Grounded Theory research effort. Therefore, the core

Grounded Theory methods of constant comparison and memoing are employed to ensure a rigorous data consolidation technique.

In an earlier section, several distinct areas within the software development domain were explored in order to determine the type of data that they offered up for inclusion in an initial general reference framework of the situational factors that can affect the software development process. These areas include risk factors for software development, software cost estimation, and factors affecting software development process tailoring. This section now describes the technique that is adopted in order to synthesise the data from these different domains, with an overview provided in Figure 1.

3.2.1 Phase 1 – Identification of Data Units

The data sources are examined so as to identify the key details (henceforth referred to as *data units*) that are important considerations when developing a reference framework of the situational factors affecting the software development process. In the case of some of the data sources, this will involve extracting all of the factors in a framework or classification – for example in the case where there is a listing of cost estimation factors or risk factors. Where frameworks and classifications are not explicitly presented in a data source, this will involve analysing a data source to carefully identify and extract any data that is considered to offer a potential situational factor that affects the software development process. In the first instance, all the data units from all of the data sources are compiled into a master table. This master table represents the data universe upon which the constant comparison and memoing techniques from Grounded Theory will be deployed in order to derive the core factors and classifications. The master list contains all data units from all data sources, and identifies the original source for each data unit. This originating source linkage is retained throughout the constant comparison process so that the relationship between the eventual factor and classification reference framework and the original sources will be clearly identifiable.

3.2.2 Phase 2 – Constant Comparison and Memoing

The first phase of the constant comparison seeks to identify those data units for which there are explicit duplications present elsewhere in the data universe. This part of the process involves scanning the data universe to identify instances of duplication. Since there is a one-to-one mapping between the individual data units that are consolidated in this comparison, there is no need to record memos that reflect the thinking that has influenced the consolidation. Where data units that are consolidated owing to duplication, care is taken to retain the source information. Therefore, if two separate data units have the same textual description and meaning, they are merged into a single unit – however, both sources for the data unit will be evident in the consolidated data unit record in the master table.

Following the initial consolidation of duplicated data units, the constant comparison and memoing effort now seeks to identify conceptual duplications or similarities in the data units. At this stage, it is possible to consolidate two data units which do not have the same textual label, but which are considered to embody the same essential meaning. Memos are used to record and develop the thought process and, as with the initial consolidation phase, where data items are consolidated, the resulting consolidated data unit retains clear links back to all originating data sources. The resulting consolidated master list of data items can now be considered to be starting to evolve into a systematically consolidated reference framework of the situational factors that affect the software development process.

The next step in the constant comparison involves the construction of an initial classification for the emerging data units. Again, memos record the rationale that has informed the initial classification – so that there is a consistent and traceable vein of thought development present in the master listing. In many cases, the data sources provided a native classification listing for factors, and these classifications are useful for organising and classifying the broader list of factors now contained in the emerging framework. Following the initial classification, the individual data units are consolidated

into core factor groupings, each of which is assigned a factor grouping label. These factor groupings are in effect data codes that embody the essential meaning of the individual consolidated data units. As with the earlier stages, memos record the rationale employed in classifying and labelling.

Equipped with a basic form of the factors and associated macro-classifications, the constant comparison effort now revisits the factor groupings and constituent data units in order to evaluate the accuracy of the factor grouping labels and the consistency of the data units that constitute factor groupings. The factor grouping labels are renamed as appropriate and individual data units may be transferred to alternate factors or to entirely new factors – as deemed appropriate with respect to the emerging factors and classification. Memos are once again used as descriptions that help to clarify the thought process and to offer a historical record of the actions taken. As with all previous iterations of the constant comparison, the original sources for data units – many of which are by now consolidated into factor groupings – are retained so that an overview of the master list composition and sources is readily evident. This helps in visualising the sources of factors and classifications, and permits an evaluation of the impact of individual data sources on the overall master list as it emerges.

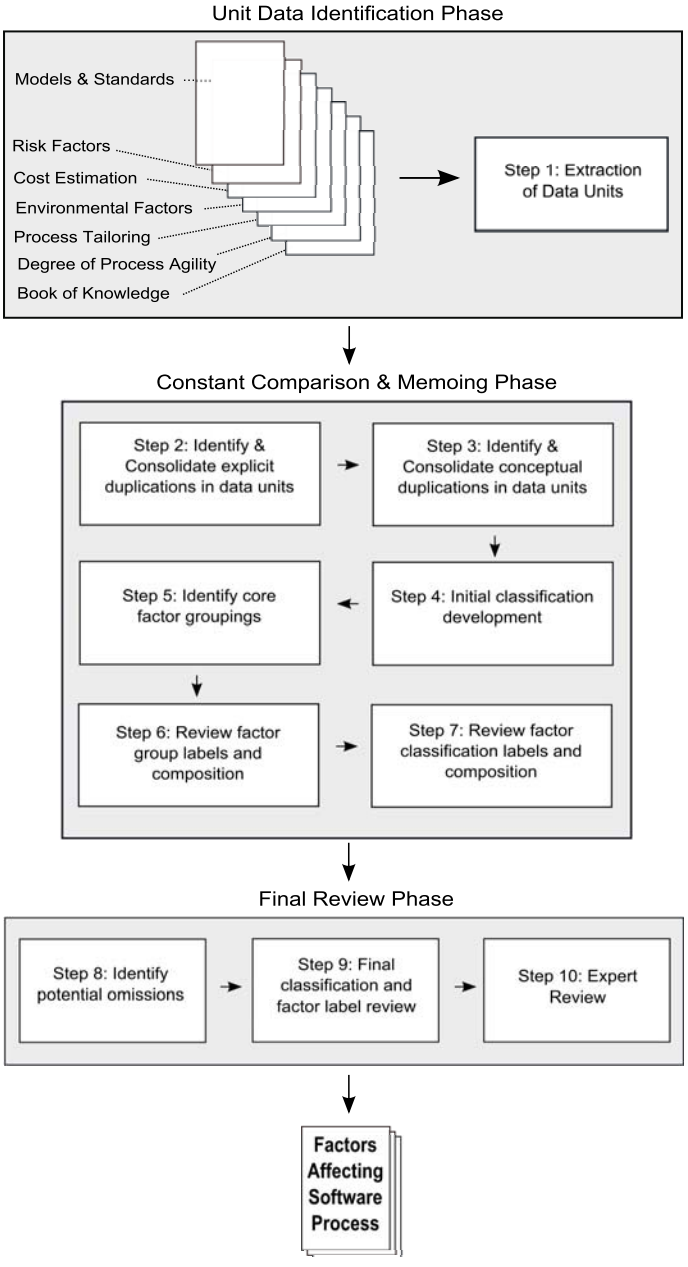

Fig. 1. Systematic Framework Development Technique

The classification labels and corresponding factor groupings are now evaluated to determine if the labels are representative of the classification component factors, and also to check that the factors are appropriately classified relative to the emerging master listing. In this iteration, some classifications may be renamed or even consolidated (or broken out) as deemed appropriate. Upon completion of this iteration, the master list is considered to be in a highly emerged state, where the classification and factors are becoming settled in a configuration that is reflective of the emerging themes.

3.2.3 Phase 3 – Final Review Phase

The master list is now examined for any possible omissions – items that for some reason have not emerged from the underlying data sources (perhaps due to their absence in the underlying data altogether). While every effort is taken to ensure that a broad and accurate set of data sources is considered in building the set of situational factors that can affect the software development process, it is not possible to guarantee that each and every eventuality is catered for – equally, there remains the possibility that there are situational factors that are important but which have never before been considered for inclusion in the underlying data sources. Again, memos are used to clarify thinking and to act as an historical record of the master list development.

A final factor and classification label review is now undertaken – so as to ensure that each and every item is accurately named. The resulting reference framework of the situational factors that affect the software development process is now presented to knowledgeable reviewers (with a minimum of 15 years' experience in the software development domain) for feedback on the content and structure – so as to raise the overall quality of the reference framework and to bring greater certainty to the product as a whole.

4. APPLICATION OF SYSTEMATIC APPROACH

This section contains details of the application of the technique outlined in Section 3. Much of the memo and constant comparison data associated with the application of the technique is verbose in nature and is therefore not presented herein. However, the core details are presented here as a reference - so as to facilitate an understanding of the systematic and rigorous approach applied in the development of the reference framework. The key output, a reference framework containing the situational factors affecting the software development process is also presented (refer to Table 2, Table 3 and Fig. 2).

4.1 Phase 1 – Identification of Data Units

During this phase, 397 data units are extracted from the 22 separate data sources identified in Section 2. Each of these units represents a factor that can affect the software development process definition. Furthermore, any classification provided in the data source is also extracted during this phase, such that the baseline raw data listing contains all factors and classifications from all data sources. However, this raw data must be examined for data duplication and analysed for core concept development. In order to support the development of the core concepts and themes, the constant comparison and memoing techniques are adopted.

4.2 Phase 2 – Constant Comparison and Memoing

The six consecutive steps involved in this phase are designed to support the development of common factors and associated factor classifications. The 397 factors produced in the data unit identification phase are gradually and systematically distilled to a set of 48 factors and 11 associated classifications (throughout this distillation process, extensive notes are retained in the form of memos). The naming and classification of factors is heavily influenced by the classifications and terminology that was present in the data sources. By incorporating the factors and classifications directly from each data source, the framework development is guided in a direction that is consistent with commonly accepted

terminology and classifications. For example, 10 of the data sources contained native classifications for personnel or team considerations, while seven of the data sources had native classifications relating to application considerations. Therefore, as the constant comparison and memoing progressed, both of these areas emerged as strong clusters for distinct classification. Consequently, two of the factor classifications in the initial reference framework are dedicated to these areas: *Personnel* and *Application*. Other classifications arose from the recurring existence of individual factors in the data sources. For example, although just two of the data sources have native classifications related to requirements characteristics, 17 of the data sources contain factors that reflect the importance of requirements characteristics. As a result, the initial reference framework provides for a *Requirements* factor classification. The remaining classifications emerged as a result of applying the Grounded Theory principle of constantly comparing until the major themes become evident.

Table 2
Classifications, Factors and Data Sources

Classification	Factors	Ferratt & Mai (2010)	Cameron (2002)	Benaroch & Appari (2010)	Appari & Benaroch (2010)	Wallace & Keil (2004)	Wallace, Keil & Rai (2004)	Ropponen & Lyytinen (2000)	Lyytinen, Mathiassen & Rop (1998)	Keil et al. (1998)	Barki, Rivard & Talbot (1993 & 2001)	Boehm (1991)	Casher (1984)	Boehm CoCoMo (2000)	Albrecht FPA (1984)	Putnam SLIM (1978)	Delany & Cunningham (2008)	Ropponen & Lyytinen (2000)	Xu & Ramesh (2007)	SWEBOK (IEEE 2004)	ISO-12207(2008)	CMMI(2006)	Boehm & Turner(2003)	Factors identified in review phase
		Data Sources																						
Personnel	Turnover, Team Size	•				•					•		•	•					•	•				
	Culture	•																	•				•	
	Experience, Cohesion, Skill, Productivity	•	•	•	•	•	•	•		•	•		•	•		•	•		•	•		•	•	•
	Commitment					•																		
	Disharmony					•					•													
Requirements	Changeability, Feasibility	•	•		•	•	•		•	•		•	•						•		•		•	
	Standard, Rigidity		•		•	•	•	•	•	•	•	•	•	•		•	•				•			
Application	Degree of Risk	•	•			•					•		•	•										
	Performance	•		•				•				•		•	•		•							
	Complexity, Type, Size, Predictability, Connectivity			•		•	•	•	•		•			•	•	•	•	•	•	•	•		•	
	Reuse													•	•									•
	Development Phase																			•				
	Deployment Profile																							•
	Quality																			•				
Technology	Knowledge			•	•	•					•			•			•			•				
	Emergent				•	•				•	•		•	•			•							
Organisation	Maturity			•	•			•			•	•		•		•	•	•	•	•				
	Management Commitment					•				•	•						•		•					
	Stability					•	•				•								•					•
	Structure								•															
	Facilities								•												•			
	Size																			•				
Operation	End-Users					•	•			•	•		•				•							
	Prerequisites														•		•					•		
Management	Expertise					•	•	•	•	•	•	•	•	•			•	•	•			•		
	Accomplishment					•					•						•							
	Continuity					•	•										•							
Business	External Dependencies					•		•	•		•	•	•	•			•		•					
	Business Drivers																		•			•		
	Time to Market																			•				
	Customer Satisfaction																			•				
	Payment Arrangements																							•
	Opportunities																				•			
	Magnitude of Potential Loss										•		•											•

4.3 Phase 3 – Final Review Phase

In this final phase, the set of 48 factors and 11 associated classifications that were produced in the constant comparison and memoing phase are examined for possible omissions. In theory, possible omissions can include sub-factors, factors or entire classifications. Therefore, as the constant comparison progressed, part of the memoing exercise involved making notes of factors that were possibly absent. Following the completion of the constant comparison, a detailed examination of these memos revealed that there are six areas that would benefit from extended coverage. These areas are identified in the column labelled *Factors identified in review phase* in Table 2, and they are outlined in more detail in the following paragraphs.

Firstly, in relation to the *Application* classification, the *deployment profile* is added as a factor. Furthermore, two additional sub-factors are included with this new factor: *number of deployed versions of applications* and *number of deployed applications*. This addition is considered important as the number of applications and application versions under deployment can have a significant impact on the testing, support and maintenance requirements for a company (as well as the supporting infrastructure, including configuration management and documentation).

Table 3
Classifications, Factors and Sub-Factors

		Description
Classifications	Personnel	Constitution and characteristics of the non-managerial personnel involved in the softare development efforts
	Requirements	Characteristics of the requirements
	Application	Characteristics of the application(s) under development
	Technology	Profile of the technology being used for the software development effort
	Organisation	Profile of the organisation
	Operation	Operational considerations and constraints
	Managment	Constitution and characteristics of the software development management team
	Business	Strategic and tactical business consideration
		Sub-Factors
Personnel	Turnover	Turnover of personnel
	Team Size	(Relative) team size
	Culture	Team culture / Resistance to change
	Experience	General experience / Team experience / Team diversity / Team Ability to understand the human implications of a new information system / Team Ability to work with top management / Application experience / Analyst experience / Programmer experience / Tester experience / Experience with development methodology / Platform experience
	Cohesion	General cohesion / Team members who have not worked for you / Team not having worked together in the past / Team ability to successfully complete a task / Team ability to work with undefined elements and uncertain objectives / Overdependence on team members / Distributed team / Team geographically distant
	Skill	Operational knowledge / Team expertise (task) / Team Ability to work with undefined elements and uncertain objectives / Training development team members / Expectation of personnel's abilities / Analyst capability / Programmer capability / Tester capability / Team understanding of application
	Productivity	Team ability to carry out tasks quickly / General productivity
	Commitment	Commitment to the project among team members
	Disharmony	Interpersonal conflicts
Req's	Changeability	Scope creep / Continually changing system requirements / Ill-defined project goals / Gold plating / Unclear system requirements
	Feasibility	Straining computer-science capabilities
	Standard	General quality of input and output requirements / Conflicting system requirements / Incorrect system requirements / Misunderstanding of the requirements / Engagement of end-users in requirements capture / User understanding of requirements
	Rigidity	Rigidity of compliance to requirements
Application	Degree of Risk	Number of people and departments that the project affects / Thoroughness of design and risk resolution / Application causes major changes to the way end-users work
	Performance	Evaluation of performance requirements / Real-time performance shortfalls / Required reliability / Estimation of hardware/software capabilities
	Complexity	Product complexity / Hardware architecture / Task complexity
	Type	Application type / Application domain / Application criticality / Architecture type / Configuration demands / Back up and recovery demands
	Application Size	Hardware / Software / Required storage / Relative project size and duration
	Predictability	Extent of recent changes / Platform volatility
	Connectivity	Number of links to existing systems / Number of links to future systems
	Reuse	Required reuse / Extent of utilisation of externally-sourced components
	Development Phase	Development / Maintenance / Other phases (e.g. end of life)
	Deployment Profile	Number of deployed versions of applications / Number of deployed applications
	Quality	Required product quality / Maintainability
Tech.	Knowledge	Language experience / Tools experience / Experience with (general) technology / Project involves use of technology that has not been used in prior projects / Introduction of new technology (to general solutions base) / Need for new hardware/software
	Emergent	Emerging technology (the technology itself is emergent)
Organisation	Maturity	Maturity of the organisation / Use of modern programming practices / Availability of technical support
	Management Commitment	Senior management commitment to project / Lack or loss of organisational commitment to project
	Stability	Resources shifting from the project due to changes in organizational priorities / Unstable organizational environment / Affect of corporate politics on projects / Organization undergoing restructuring during the projects / Rate of organisational change (growth or decline)
	Structure	Organisational structure
	Facilities	Physical working arrangements / Facilities to house the project
	Organisation Size	Size of the organisation
Oper.	End-Users	Users resistant to change / End-user commitment / Degree of user engagement with development team / Conflict between users / Conflict between users/departments / Number of users outside the organisation / Number of users in organisation / Number of hierarchical levels occupied by end-users / User turnover / End-user experience with the activities to be supported by the future application / End-user familiarity with application type / End-user understanding of system capabilities and limitations
	Prerequisites	Applicable standards / Applicable laws / Organisational policies / Common practices / Operational ease / Installation ease
Management	Expertise	Effectiveness of project management methodology / Project planning capability / Project management systems / Experience with project management tools / Efficiency of governance structure / Appropriateness of rewards / Project sizing capability / Achievability of schedules and budgets / Estimation capability with respect to the personnel needs of projects / Degree of people skills in project leadership / Progress control capability / Effectiveness of work flow and coordination / Definition of project milestones / Effectiveness of project managers / Management communication skills / Manager familiarity with team / Effective understanding of responsibilities / User expectation management capability
	Accomplishment	Project management experience / Operational knowledge of leader
	Continuity	Changes in organisational management
Business	External Dependencies	Dependency on outside suppliers / Number of hardware suppliers / Number of software suppliers / Multiple implementers / Multisite development / Number of involved parties / Reliance on other projects or processing systems / Number of (external) stakeholders / Stakeholders' background / Access to Stakeholders
	Business Drivers	Project drivers / Finance considerations / Marketing activites / Maximise profit/turnover / Minimise costs
	Time to Market	Time to Market
	Customer Satisfaction	Customer satisfaction / Satisfaction with user interface
	Payment Arrangements	Time and Materials / Fixed price / Non-conventional payment arrangement
	Opportunities	Project opportunities
	Magnitude of Potential Loss	Customer relations / Financial health / Competitive position / Organisational reputation/survival / Market share / Loss of human Life

Secondly, in relation to the *Organisation* classification, the new sub-factor *Rate of organisational change* (*growth* or *decline*) is added to the *Stability factor*. The rate of growth or decline in an organisation can have a significant impact on the software development process – for example, if a

company doubles the headcount from 100 to 200 software engineers over a 12 month period, it is likely that this will have an impact on the required software development process (certainly it will have an impact on the potential economy of scale of the process). Equally, if an organisation reduces its headcount from 200 to 100 engineers, it is probable that there will be an impact on the software development process

Thirdly, in relation to the *Business* classification, the new factor *Payment Arrangements* is added. The payment terms for both on-going and new projects can have an impact on the process definition. If the payment terms are rigidly fixed against contractual requirements and deadlines, then an agile development process (where the requirements are not well understood at the outset) may not represent the best choice of development approach. Equally, if customers want to innovate a new solution with uncertain requirements and a time and materials approach to payment, then a strict traditional waterfall approach may not be well suited to the needs of the organisation. Therefore, the *payment terms* are an important situational factor affecting the process definition.

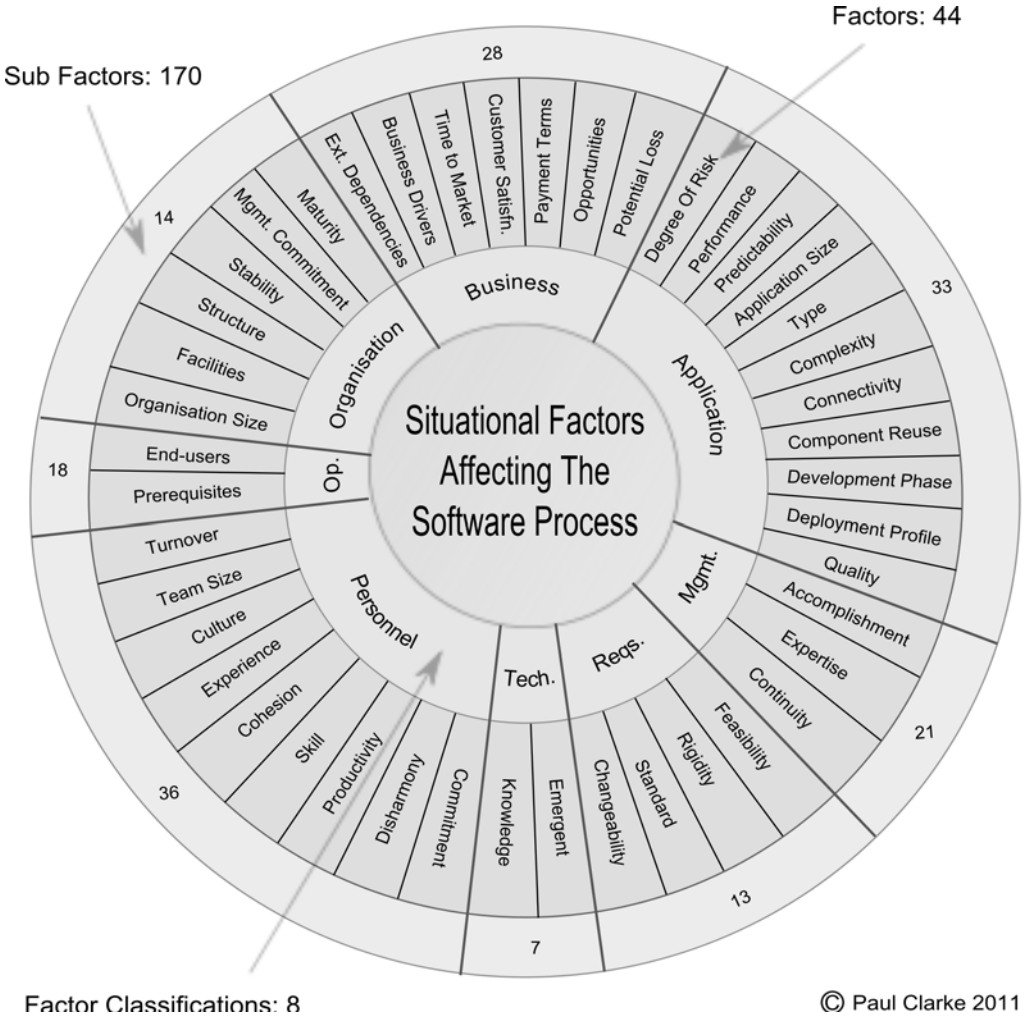

Fig 2. Situational Factors Affecting the Software Process

Fourthly, in relation to the *cohesion* factor, categorised under the *Personnel* classification, two additional sub-factors were added: *Distributed team* and *Team geographically distant*. Although these two new sub-factors are to some extent implicitly covered by other sub-factors such as *Team ability to successfully complete a task*, it is the view of the authors that the sometimes distributed nature of software development (especially where this is international) is significant enough to be given explicit mention. Where software development teams are distributed beyond a single office, new challenges

are introduced, particularly in relation to communication and co-operation. Therefore, these two new sub-factors have been added.

Fifthly, in relation to the *Component Reuse* factor, categorised under the *Application* classification, a new sub-factor was added: *Extent of utilisation of externally-sourced components*. For example, with the emergence of the open source development community over recent years, and the increasing probability that cloud computing may be the future for some software development efforts, the use of externally developed software and the management of the supply and maintenance of that software is something that could have an impact on the software development process. A second example of an externally-sourced component includes Commercial Off-The-Shelf (COTS) software.

Finally, an additional sub-factor, *Loss of human life*, is added to the *Magnitude of potential loss* factor associated with the *Business* classification. While the criticality of the applications under development does consider the importance of the criticality of application in the factor *Type*, it is important to explicitly list the potential for loss of life as a factor for the business to consider.

Having added the six new items listed above, a final classification and factor label review is undertaken to ensure that the overall concepts that emerged in the distillation are accurately and succinctly represented in the final reference framework. This involves a further consolidation of some of the classifications and factors, the results of which are brought forward to an expert review.

The earlier review initiatives have helped to ensure that the reference framework is internally consistent and valid. However, in order to provide an extra level of certainty regarding the complete and consistent nature of the reference framework, an expert review is undertaken. During the expert review, the draft reference framework is presented to two experienced software development professionals, each with over 15 years of experience in the software development domain across the full spectrum of software development lifecycle activities. The reviewers made a number of recommendations which were applied to the draft framework. Perhaps the most valuable feedback from the reviewers was a recommendation that each of the 44 factors be given unique labels. In the draft framework, several factor names were reused. For example, there was a *volatility* factor under both the *Requirements* classification and the *Management* classification. Following the expert review, these factor labels were renamed to *Changeability* (in the case of *Requirements*) and *Continuity* (in the case of *Management*). There were a number of other incidents of factor name reuse, all of which were addressed through similar relabelling. The introduction of unique factor labels improves the understandability and usability of the reference framework. The reviewers also recommended that some sub-factors could be further consolidated. For example, the *Potential loss* factor had originally included a *Possibility of lost funds due to system failure* sub-factor which was recommended for consolidation into the existing *Financial health* sub-factor. The reviewers highlighted other instances where such consolidation was recommended and subsequently, the draft framework sub-factors were further consolidated in line with the reviewers' feedback. The reviewers also suggested that broad community-level collaboration would be a requirement if producing a consensually-agreed general reference framework of the factors affecting the software development process (a point which we discuss later in Section 5.2). However, the reviewers reported that the initial reference framework was in their view comprehensive and useful in its present stage of development.

Following the completion of the final review phase, the initial reference framework of situational factors consists of 8 classifications and 44 factors that inform the software process. This final iteration, along with the association back to the data sources, is summarised in Table 2. The situational factors outlined in Table 2 have a broader listing of 170 sub-factors – these are the components of the factors themselves. These sub-factors have been carefully and diligently carried forward (and consolidated) from the data sources so that a substantial level of detail is available within the reference framework. Along with a description of the eight classifications, a complete listing of the sub-factors is presented in Table 3. In order to provide an easily digestible view of the contents outlined in Table 2 and Table 3, the essential components of the reference framework of the

situational factors that affect the software process definition are summarised in a consolidated form in Fig. 2.

5. DISCUSSION

In this section, we discuss how the framework can be useful for both the research and practicing communities. We also identify some possible areas for future work. Finally, we outline some of the limitations of our initial reference framework.

5.1 Utilising the Initial Reference Framework

We believe that the initial framework presented herein is currently the most comprehensive reference framework of the situational factors that affect the software development process. As such, the framework holds significant value for researchers and practitioners. Researchers will have access to a broad, systematically developed initial framework (replete with 170 sub-factors), which can be used as a reference for the situational factors affecting the software development process. This is an important reference for researchers in the software development domain in general, and in the software process and software process improvement (SPI) domains in particular. Since software processes and SPI actions should be designed so as to best support each individual software development setting, it is important that we are able to profile the important aspects of software development settings.

By consulting the initial reference framework that we have developed, software development practitioners can access a check list of the important considerations for their software development process. The framework includes areas that are of both a technical and a non-technical nature, and practitioners can examine the various items in their setting when making software process decisions. These include considerations such as the size and complexity of the application(s) under development, the payment terms for the project(s), the volatility of the requirements, the complexity of the application(s), the experience and culture of the development team, the location of the development team, as well as any applicable laws and standards. Therefore, the initial framework is a useful reference checklist for practitioners when profiling the characteristics of individual software development settings. Furthermore, the initial framework can be harnessed in order to examine the degree of situational change occurring in a setting, an important consideration for SPI. Rather than simply taking a snapshot of the situational factors at a point in time, the situational framework could be applied to task of examining the amount of situational change over a period of time. The task of examining the amount of situational change could be performed in a single engagement – by examining the change in each of the situational factors over a defined time period. Alternatively, the amount of situational change could be profiled by conducting two separate situational factor investigations at two distinct points in time, and thereafter performing a finite difference analysis on the two profiles.

5.2 Future work

While the initial reference framework presents a broad and useful checklist for use in examining and discussing software development settings, it does not identify the relationship between the situational factors and aspects of the software development process. Therefore, important future work in this area should seek to provide an associative mapping between the reference framework and components of the software development process. This associative mapping would be of very significant value to both the research and practicing communities, since it would indicate exactly what process areas are affected by the various situational factors. We should caution however, that future work to determine such associations is a substantial undertaking which we feel would require a large body of resourcing and funding (though, in our view, such future work is sufficiently important to justify a large investment).

Further future work should also be carried out with respect to gaining consensual agreement on a general reference framework of the situational factors affecting the software development process. While the authors have expended great effort in bringing the reference framework to its present position, much community-wide collaboration is required before a generally-agreed reference framework of the factors affecting the software development process can be published. This collaboration would ideally incorporate expertise from a broad range of software engineering disciplines, and from a broad international base.

5.3 Limitations

Although the initial reference framework that we present herein is broader in scope and depth of information that any existing reference framework of the situational factors affecting the software development process, there are a number of limitations that we would like to highlight. Firstly, while great care has been applied in developing the initial reference framework, the scope of the framework is inherently limited to the data sources included in the framework development. The individual data sources rendered almost 400 distinct data units for inclusion in the initial reference framework, however, we must acknowledge that many additional potential data sources exist and that the scope of the initial reference framework would certainly not have been diminished had additional data sources been included in the data analysis and framework construction. Additional domains could be incorporated into a later evolution of the framework. For example, there has been some interesting work on architectural patterns [80], while other efforts have investigated the sequencing of tasks for software development [81]. Similarly, work in the area of defect prediction [82] and software metrics [83] present further interesting domains for examination.

A second limitation of the framework relates to the absence of broad community involvement in the construction and validation of the framework. It is primarily for this reason that we refer the framework as being *initial.* As outlined in Section 5.2, the task of gaining broad consensual agreement for a framework of the situational factors affecting the software development process would require an international set of experts from a broad range of software domains, and a managed forum within which all parties could gradually develop and agree on the constituent factors. The participation of such a broad spectrum of expertise would also ensure that the adopted terminology would be generally acceptable to the broad software development community.

6. CONCLUSION

In this paper, we have shown that many contributors to the software development field believe that the situational context of a business is instrumental in determining the most appropriate software process definition for that business. However, to date there is no general comprehensive reference framework of the situational factors that affect the software development process. The absence of such a reference framework presents a significant problem in that it undermines our ability to identify the important characteristics of a software development setting – and a failure to correctly understand the characteristics of the setting may result in an inappropriate and inefficient software development process. Consequently, by drawing on the insight afforded by a variety of closely related domains, we have applied the data coding and analysis techniques from Grounded Theory [39] to the construction of an initial reference framework of the situational factors that affect the software development process.

Naturally, any effort to generally capture the situational factors that inform the software development process will inevitably meet with some degree of criticism. Such criticism can be centred on the granularity offered by the factor listing, on the categorisation adopted, on the chosen descriptive labels, or on possible omissions. To address these criticisms would require broad international community collaboration on a general reference framework, something that is not presently in existence. In the absence of such a broad international effort, and in order to maximise the completeness of the our initial factor framework, we have drawn on some of the most influential related works in recent decades, in areas such as software development risk, software development

cost estimation and software development process models and standards. Furthermore, we have adopted a rigorous approach to analysing and integrating the data contained in this rich variety of sources. Future work that holds potentially high value would be the identification of the relationship between specific situational factors and specific aspects of the software development process. Such a resource would be of great benefit to software development practitioners seeking to understand the likely impact of a changing situation on the incumbent software development process.

The variety and diversity of factors that have been identified in our initial reference framework serves as a reminder of the level of complexity involved in software development. Indeed, it is likely that a lack of appreciation for this inherent complexity is itself a factor that contributes to the significant difficulties that are witnessed in software development projects. Software development is at once a technological, a business and a human endeavour; and professional software development is a continual innovation that must withstand ever-present practical business considerations such as financial viability, product quality and delivery schedules. We believe that the situational factor reference framework that we have presented herein represents a sound initial reference framework for the key situational elements affecting the software process definition. In addition to providing a useful reference listing for the research community and for committees engaged in the development of standards, the reference framework also provides support for practitioners who are challenged with defining and maintaining software development processes.

Acknowledgment
This work is supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero, the Irish Software Engineering Research Centre (www.lero.ie).

References

[1] W. Royce, Managing the development of large software systems: Concepts and techniques, IN: Western Electric show and Convention Technical Papers, 1970.

[2] ISO, Quality Systems: Model for Quality Assurance in Design/Development, Production, Installation and Servicing, ISO 9001, ISO, Geneva, Switzerland, 1987.

[3] D. Stelzer, W. Mellis, G. Herzwurm, Software Process Improvement via ISO 9000? Results of Two Surveys among European Software Houses, Software Process: Improvement and Practice, 2 (3) (1996) 197-210.

[4] F. Coallier, How ISO 9001 fits into the software world, IEEE Software, 11 (1) (1994) 98-100.

[5] D. Stelzer, W. Mellis, G. Herzwurm, A critical look at ISO 9000 for software quality management, Software Quality Journal, 6 (2) (1997) 65-79.

[6] O. Oskarsson, R. L. Glass, An ISO 9000 Approach to Building Quality Software, Prentice Hall, New Jersey, USA, 1996.

[7] Y. H. Yang, Software quality management and ISO-9000 implementation, Industrial Management & Data Systems, 101 (7) (2001) 329-338.

[8] R. Grady, Successful Software Process Improvement, Prentice Hall, Upper Saddle River, New Jersey, USA, 1997.

[9] ISO/IEC, ISO/IEC Technical Report 15504, ISO, Geneva, Switzerland, 1998.

[10] CMMI Product Team, CMMI for Development, Version 1.2, Software Engineering Institute, CMU/SEI-2006-TR-008. Pittsburgh, PA, USA, 2006.

[11] J. Herbsleb, A. Carleton, J. Rozum, J. Siegel, D. Zubrow, Benefits of CMM-based software process improvement: Initial results, CMU/SEI-94-TR-013, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 1994.

[12] J. Herbsleb, D. Goldenson, A systematic survey of CMM experience and results, IN: Proceedings of the 18th International Conference on Software Engineering (ICSE 1996), pp. 323-330, 1996.

[13] P. Lawlis, R. Flowe, J. Thordahl, A Correlational Study of the CMM and Software Development Performance, Crosstalk, the Journal of Defense Software Engineering, 8 (9) (1995) 21-25.

[14] K. Hyde, D. Wilson, Intangible benefits of CMM-based software process improvement, Software Process: Improvement and Practice, 9 (4) (2004) 217-228.

[15] D. Gibson, D. Goldenson, K. Kost, Performance results of CMMI-based process improvement, CMU/SEI-2006-TR-004, Software Engineering Institute, Carnegie Mellon University, CMU/SEI-2006-TR-004. Pittsburgh, Pennsylvania, USA, 2006.

[16] K. Lebsanft, Process improvement in turbulent times - is CMM still an answer? IN: Proceedings of the 3rd International Conference on Product Focused Software Process Improvement, pp. 78-85, 2001.

[17] H. Wegelius, M. Johansson, Practical experiences on using SPICE for SPI in an insurance company, IN: EuroSPI 2007 Industrial Proceedings, pp. 2.27-2.34, 2007.

[18] K. El Emam, A. Birk, Validating the ISO/IEC 15504 measures of software development process capability, Journal of Systems and Software, 51 (2) (2000) 119-149.

[19] A. Cater-Steel, Process improvement in four small software companies, IN: 13th Australian Software Engineering Conference (ASWEC 2001), pp. 262-272, 2001.

[20] P. Abrahamsson, O. Salo, J. Ronkainen, J. Warsta, Agile Software Development Methods - Review and Analysis, VTT Technical Research Centre of Finland, VTT Publications Number 478. Finland, 2002.

[21] M. Fowler, J. Highsmith, The Agile Manifesto, Software Development, pp. 28-32, August, 2001.

[22] J. Highsmith, A. Cockburn, Agile software development: the business of innovation, IEEE Computer, 34 (9) (2001) 120-127.

[23] K. Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley, Reading, Massachusetts, USA, 1999.

[24] K. Schwaber, SCRUM development process, IN: Business Object Design and Implementation Workshop at the 10th Annual Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 1995), 1995.

[25] S. R. Palmer, J. Felsing, A Practical Guide to Feature-Driven Development, Prentice Hall, Upper Saddle River, New Jersey, USA, 2002.

[26] D. J. Reifer, How Good Are Agile Methods? IEEE Software, 19 (4) (2002) 16-18.

[27] B. Fitzgerald, G. Hartnett, K. Conboy, Customising agile methods to software practices at Intel Shannon, European Journal of Information Systems, 15 (2) (2006) 200-213.

[28] V. Kettunen, J. Kasurinen, O. Taipale, K. Smolander, A study on agility and testing processes in software organizations, IN: ISSTA '10: Proceedings of the 19th International Symposium on Software Testing and Analysis, pp. 231-240, 2010.

[29] E. Whitworth, R. Biddle, The social nature of agile teams, IN: Proceedings of the AGILE 2007, pp. 26-36, 2007.

[30] L. Constantine, Methodological agility, http://www.ddj.com/architect/184414743, Accessed: 20 October/2011.

[31] B. Boehm, Get ready for agile methods, with care, IEEE Computer, 35 (1) (2002) 64-69.

[32] C. Jones, Development Practices for Small Software Applications, CrossTalk, the Journal of Defense Software Engineering, 21 (2) (2008) 9-13.

[33] P. Feiler, W. Humphrey, Software Process Development and Enactment: Concepts and Definitions, Software Engineering Institute, Carnegie Mellon University, CMU/SEI-92-TR-004. Pittsburgh, Pennsylvania, USA, 1992.

[34] O. Benediktsson, D. Dalcher, H. Thorbergsson, Comparison of software development life cycles: a multiproject experiment, IEE Proceedings - Software, 153 (3) (2006) 87-101.

[35] A. MacCormack, R. Verganti, Managing the Sources of Uncertainty: Matching Process and Context in Software Development, Journal of Product Innovation Management, 20 (3) (2003) 217-232.

[36] G. H. Subramanian, G. Klein, J. J. Jiang, C. Chan, Balancing four factors in system development projects, Communications of the ACM, 52 (10) (2009) 118-121.

[37] K. Kautz, Software process improvement in very small enterprises: does it pay off? Software Process: Improvement and Practice, 4 (4) (1998) 209-226.

[38] B. Boehm, R. Turner, Balancing Agility and Discipline - A Guide for the Perplexed, Pearson Education Limited, Boston, Massachusetts, USA, 2003.

[39] B. Glaser, A. Strauss, The Discovery of Grounded Theory: Strategies for Qualitative Research, Aldine de Gruyter, Hawthorne, NY, USA, 1976.

[40] ISO/IEC, Amendment to ISO/IEC 12207-2008 - Systems and Software Engineering – Software Life Cycle Processes, ISO, Geneva, Switzerland, 2008.

[41] M. Benaroch, A. Appari, Financial Pricing of Software Development Risk Factors, IEEE Software, 27 (5) (2010) 65-73.

[42] L. Wallace, M. Keil, Software project risks and their effect on outcomes, Communications of the ACM, 47 (4) (2004) 68-73.

[43] J. Casher, How to control project risk and effectively reduce the chance of failure. Management Review, 73 (6) (1984) 50-54.

[44] B. Boehm, Software Risk Management: Principles and Practices, IEEE Software, 8 (1) (1991) 32-41.

[45] M. Keil, P. E. Cule, K. Lyytinen, R. C. Schmidt, A framework for identifying software project risks, Communications of the ACM, 41 (11) (1998) 76-83.

[46] K. Lyytinen, L. Mathiassen, J. Ropponen, Attention shaping and software risk-A categorical analysis of four classical risk management approaches, Information Systems Research, 9 (3) (1998) 233-255.

[47] J. Ropponen, K. Lyytinen, Components of software development risk: how to address them? A project manager survey, IEEE Transactions on Software Engineering, 26 (2) (2000) 98-112.

[48] H. Barki, S. Rivard, J. Talbot, An Integrative Contingency Model of Software Project Risk Management. Journal of Management Information Systems, 17 (4) (2001) 37-69.

[49] A. Appari, M. Benaroch, Monetary pricing of software development risks: A method and empirical illustration, Journal of Systems and Software, 83 (11) (2010) 2098-2107.

[50] S. J. Delany, P. Cunningham, The application of case-based reasoning to early software project cost estimation and risk assessment, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.42.9932&rep=rep1&type=pdf, Accessed: 11/15/2010.

[51] B. Boehm, Software Engineering Economics, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[52] B. Kitchenham, Making process predictions, in Software Metrics: A Rigorous Approach, N. E. Fenton, Ed., Chapman & Hall, London, United Kingdom, 1991.

[53] B. Boehm, B. Clark, E. Horowitz, A. Brown, D. Reifer, S. Chulani, R. Madachy, B. Steece, Software Cost Estimation with Cocomo II, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[54] P. Heejun, B. Seung, An empirical validation of a neural network model for software effort estimation, Expert Systems with Applications, 35 (3) (2008) 929-937.

[55] L. Putnam, A General Empirical Solution to the Macro Software Sizing and Estimating Problem, IEEE Transactions on Software Engineering, 4 (4) (1978) 345-361.

[56] A. J. Albrecht, Measuring application development productivity, IN: Proceedings of the IBM Applications Development Symposium, pp. 83, 1979.

[57] T. W. Koh, M. H. Selamat, A. Ghani, Exponential Effort Estimation Model Using Unadjusted Function Points, Information Technology Journal, 7 (6) (2008) 830-839.

[58] A. J. Albrecht, J. E. Gaffney, Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation, IEEE Transactions on Software Engineering, 9 (6) (1983) 639-648.

[59] J. Kolodner, Case-Based Reasoning, Morgan Kaufmann, California, 1993.

[60] I. Watson, F. Marir, Case-Based Reasoning: A Review, The Knowledge Engineering Review, 9 (4) (1994) 327-354.

[61] P. Xu, B. Ramesh, Software Process Tailoring: An Empirical Investigation, Journal of Management Information Systems, 24 (2) (2007) 293-328.

[62] K. Petersen, C. Wohlin, Context in industrial software engineering research, IN: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement, pp. 401-404, 2009.

[63] B. Dede, I. Lioufko, Situational Factors Affecting Software Development Process Selection, Master of Science, University of Gothenburg, 2010, http://gupea.ub.gu.se/bitstream/2077/23498/1/gupea_2077_23498_1.pdf, Accessed: 10/19/2011.

[64] W. Bekkers, I. van de Weerd, S. Brinkkemper, A. Mahieu, The influence of situational factors in software product management: An empirical study, IN: Proceedings of the Second International Workshop on Software Product Management (IWSPM '08), pp. 41-48, 2008.

[65] D. Truex, R. Baskerville, J. Travis, Amethodical systems development: the deferred meaning of systems development methods, Accounting, Management and Information Technologies, 10 (1) (2000) 53-79.

[66] R. Baskerville, J. Stage, Accommodating emergent work practices: Ethnographic choice of method fragments, in Realigning Research and Practice in IS Development: The Social and Organisational Perspective, B. Fitzgerald, N. Russo, J. DeGross, Eds., Kluwer Academic Publishers, New York, NY, USA, pp. 12-28, 2001.

[67] M. P. Ginsberg, L. H. Quinn, Process tailoring and the software capability maturity model (CMU/SEI-94-TR-024), Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 1995.

[68] G. Coleman, R. O'Connor, Investigating software process in practice: A grounded theory perspective, Journal of Systems and Software, 81 (5) (2008) 772-784.

[69] J. Cameron, Configurable development processes, Communications of the ACM, 45 (3) (2002) 72-77.

[70] T. Ferratt, B. Mai, Tailoring software development, IN: SIGMIS-CPR '10: Proceedings of the 2010 Special Interest Group on Management Information System's 48th Annual Conference on Computer Personnel Research on Computer Personnel Research, pp. 165-170, 2010.

[71] IEEE, Guide to the Software Engineering Book of Knowledge (SWEBOK), IEEE Computer Society, Los Alamitos, CA, USA, 2004.

[72] B. Glaser, Basics of Grounded Theory Analysis, Sociology Press, California, USA, 1992.

[73] L. Knigge, M. Cope, Grounded visualization: integrating the analysis of qualitative and quantitative data through grounded theory and visualization, Environment and Planning, 38 (11) (2006) 2021-2037.

[74] J. Duchscher, D. Morgan, Grounded theory: reflections on the emergence vs. forcing debate, Journal of Advanced Nursing, 48 (6) (2004) 605-612.

[75] D. Douglas, Grounded theories of management: a methodological review, Management Research News, 26 (5) (2003) 44-52.

[76] J. Corbin, A. Strauss, Basics of Qualitative Research, Sage Publications Limited, Thousand Oaks, CA, USA, 2008.

[77] A. Bryant, K. Charmaz, The SAGE Handbook of Grounded Theory, SAGE, Thousand Oaks, CA, USA, 2007.

[78] L. Lempert, Asking questions of the data: Memo writing in the grounded theory tradition, in The SAGE Handbook of Grounded Theory, A. Bryant, K. Charmaz, Eds., Sage, Thousand Oaks, CA, USA, pp. 245-264, 2007.

[79] B. Glaser, Doing Grounded Theory: Issues and Discussions, Sociology Press, Mill Valley, CA, USA, 1998.

[80] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[81] A. Wang, E. Arisholm, The effect of task order on the maintainability of object-oriented software, Information and Software Technology, 51 (2) (2009) 293-305.

[82] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, Cross-project defect prediction, IN: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), pp. 91-100, 2009.

[83] B. Kitchenham, What's up with software metrics? – A preliminary mapping study, Journal of Systems and Software, 83 (1) (2010) 37-51.