Towards an Information Driven Software Development Life Cycle

Dr. Ernest Cachia, Mr. Mark Micallef
Department of Computer Science, Faculty of ICT, University of Malta, Msida, Malta

Abstract - *Abstract--- Although software engineering has matured greatly over the years, a large number of ICT projects continue to fail[1][2] . Studies continue to identify non-technical issues such as poor communication, shifting requirements and poor executive involvement as the main causes of these failures. This paper identifies such well known causes and poses the question as to why currently available software development life cycles fall short of dealing with them. Drawing on results from a research exercise carried out by the authors, a link is made between the quality of information used throughout the development life cycle and the quality of the resultant product. Consequently, it is proposed that organisations knowingly or unknowingly maintain a knowledge context and the quality of this knowledge context has direct impact on product quality. Furthermore, it is proposed that a software development life cycle be developed in which participants do not focus explicitly on the traditional phases of software development. Rather, a conscious decision is made to focus instead on information which is being created, manipulated and utilised throughout the lifetime of a product. If a link can be established between the quality of the knowledge context and the quality of a finished product, then it is sound to argue that if one nurtures a high quality knowledge context, a high-quality product will naturally follow.*

Keywords: Quality Assurance, Software Development Life Cycles, Software Engineering

1 Introduction

It could be said that the research area of development life cycles is indeed mature. Since the early days of software engineering, this area has seen the development of a number of models and methodologies ranging from the generic waterfall model [3] to the more recent agile techniques [4][5]. Different approaches function to varying degrees of success depending on the scenario at hand. However, given that ICT projects persistently continue to be late and even of insufficient quality [1], one is compelled to consider the possibility that the software engineering community may have taken a wrong turn at some point. One must explore the possibility of developing an altogether different way of thinking by which high quality systems could be engineered within budget and on time.

Traditionally, a software development life cycle has been perceived as a structured process imposed on the development of a product. In so doing, the development process focuses explicitly on the product thus putting it through a number of phases before finally delivering it in its finished form. At its core, a particular life cycle differs from others in the way it guides a product through transitions between these different phases. Throughout this paper, such life cycles will be referred to as product-oriented life cycles. Due to the fact that the primary goal is usually that of delivering a product, the thinking behind product-oriented life cycles inherently seems to make sense. However, perceiving software engineering as simply being all about the product may be misleading. Software is after all, an intangible artifact conceived entirely from knowledge and at its core, exists solely to facilitate the use of information and knowledge. Furthermore, the nature of modern software engineering environments gives rise to a whole new genre of problems which directly or indirectly affect product quality and project timeliness. Due to issues such as high expectations of software, constricting time lines, increased staff turnover, engineers' intra-project mobility and the dynamic nature of all information related to a product, problems such as cognitive overload, information anxiety and duplication of effort amongst others have been observed. These problems are discussed further in section 2 but are being mentioned here to highlight a problem which is not explicitly dealt with by product-oriented development life cycles.

In this paper, it is being proposed and hypothesized that every organization, knowingly or unknowingly maintains a *knowledge context*. We define this knowledge context as being *the knowledge, technical or otherwise, held by any of the organization's stakeholders at a particular point in time*. It is being proposed that the quality of a product is directly related to the quality of the knowledge context used to create it. Consequently, this knowledge context should be nurtured and maintained so as to ensure the timely delivery of high quality products. Finally, it is being proposed that a new type of software development life cycle be developed whereby the focal point is the development and maintenance of a high quality knowledge context. If the proposals put forward here are true, it is felt that a high quality product will naturally follow.

2 Modern Software Engineering Environments

This section describes typical characteristics of the modern work place which give rise to problems effecting project timeliness and/or product quality. The problems described here tend to go beyond the problem domains handled by traditional development life cycles and serve to illustrate the benefits of maintaining a high quality knowledge context within an organisation.

Over the years, expectations of ICT systems have gone from storage and retrieval of data to complex functionality which automates and complements business processes in an attempt to gain a competitive edge. Due to market pressures, this increasing functionality is being demanded in shorter spans of time [6]. Whereas in the past it may have been common to have software development projects go on for over a year, today delivery dates of between four to twelve weeks are more common place [6]. Compounding this increased complexity and time restrictions, modern systems are also highly susceptible to an onslaught of external factors manifested in the form of changing requirements, conflicting decisions, changing directions, experimental technologies, and so on. In essence, the software engineering process no longer exists in a convenient bubble which enables engineers to ignore an evolving world whilst engineering a product which caters for a freeze-frame of that dynamic world.

Putting technical merits aside for the time being, this constant onslaught of new or changing information in a diversity of formats from across the spectrum of quality has lead to the observation of cognitive overload in the work place [7]. A study amongst Fortune 1000 workers indicates that workers now work in environments of increased complexity, saturated with multi-tasking, interruption, and profound information overload [8]. A number of studies claim that consequences of such environments include information anxiety, social tension, job dissatisfaction, ill health, increased staff turnover, and consequently poor quality of work [8].

Another characteristic of the software engineering environment resulting from all this is the increase in inter-project mobility. An engineer can expect to be shifted between projects on a regular basis depending on a number of factors such as customer priorities, project schedules, funding and so on. When an engineer switches projects in this way, she needs time to adjust to the new context. This may involve familiarising herself with the project, technologies being used, design architectures being utilised, decisions which were taken, and so on. During this adjustment period, the engineer may also distract other employees from their work because of her need to ask questions and understand project-specific issues. All being said, one realises that there is a certain amount of time after a switch during which the engineer is minimally productive at best or counter productive to the team's efforts at worst. Given the shortening project schedules, this is not a desirable situation.

A somewhat related concept refers to staff turnover, a recurring concern with ICT companies where annual turnover rates can rise above 10% [9]. With this regular flow of staff leaving and new staff joining, one's challenge is two-fold. Firstly, one must somehow retain the knowledge held by departing staff for use in current and future projects. Secondly, one needs a strategy for transferring all required knowledge to new staff as quickly and effectively as possible so as to enable them to be productive.

Finally, we examine a situation stemming from the independant way in which teams within the same company seem to operate. It is not uncommon for a development team to spend a considerable amount of time (typically days) solving a problem with (for example) a third-party component only to realise a few weeks later that the same problem had already been solved by another team in the same company. This discovery would understandably result in frustration on the engineers' side for having wasted time reinventing the wheel, as well as on the management's side due to the waste in time and money that unnecessary duplication of efforts causes.

With all this information and knowledge being created, modified, used, and retired on a daily basis, one needs to develop ways to effectively manage this information and focus it towards achieving the goals at hand. It is the opinion of the authors of this paper that the formalisation of the concept of a knowledge context would be a concrete first step in dealing with these situations. Take the example whereby engineers are likely to be shifted around projects regularly. In this situation, the organisation in question would do well to somehow ensure that all engineers had a certain minimal knowledge about most (if not all) ongoing projects in the company. If this was achieved, switching engineers between projects would be smoother. Similarly, the concept of duplication of work would be virtually eliminated if an engineer could be notified that the problem which he is currently working on has already been solved and was somehow pointed to the solution.

When considering the whole concept of knowledge context and how it may be used, one is undoubtedly inundated with questions about how a number of issues would be solved. For example, in the case of duplicated effort, one must certainly be aware of the difficulties inherent in keeping everyone informed about everything all the time. This would surely only compound the problem of cognitive overload. At this point, the scope is to put forward the concept of the knowledge context and the benefits which its formalisation would bring. It is beyond the immediate scope to delve into the details of how to actually build, maintain and use such a context.

3 The Knowledge Context

In section 1, a knowledge context is defined as being *the knowledge, technical or otherwise, held by any of the organisation's stakeholders at a particular point in time*. This definition, although concise, illustrates the importance of three particular issues. Firstly, it puts forward the concept that all relevant knowledge, be it technical or not, is important to a project's success. That is to say that although sound technical knowledge (specifications, design, programming language knowledge, etc) is essential when delivering quality software, non-technical information is just as essential. Examples of non-technical knowledge include things such as the client's future business aspirations, legislation relating to the product being developed, staff vacation plans and so on. Secondly, the definition makes reference to *all stakeholders* of the company. This is important because communication problems have been shown to considerably influence the success of a project [10]. Therefore, there needs to be a constant flow of relevant information between all levels of the organisation's hierarchy as well as any external stakeholders. Finally, the definition makes reference to the temporal aspect of knowledge and information. Different information may be required by the same person at different points in time. The temporal information requirements may be as obvious as the engineer needing specifications during the design phase and needing design documents during the development phase. However, it is often the case that one may need access to the same knowledge albeit it from a different perspective or maybe using information with different characteristics (finer granularity, different media, etc). People will accumulate a certain level of knowledge over time and placing the right information in the right peoples' hands at the right time will facilitate better product quality in all its aspects.

At this point, it is useful to explicitly distinguish between knowledge and information. This is necessary because these two terms are sometimes used interchangeably and the difference between the two is key to the concepts presented in this paper. Knowledge refers to one's acquaintance with facts, principles, concepts, theories and so on. Information on the other hand, refers to the transfer of knowledge in some way, shape or form.

4 Knowledge used in Software Engineering

In order to delve deeper into the abstract concept of a knowledge context, the authors of this paper carried out a research exercise with the participation of development professionals, management professionals and entrepreneurs who have had experience commissioning ICT systems. The scope of this exercise was to identify information which is used throughout software development, classify it into a number of manageable knowledge areas and discuss the impact which the quality of this information would have on a finished product.

4.1 Research Methods

The research exercise consisted of a number of group discussion sessions with participants followed up by a research questionnaire resulting from the sessions. The questionnaire was deemed important because it merged the ideas resulting from the separate group sessions and also helped lay the foundations for future development of metrics and measures related to the subject at hand.

The face-to-face sessions involved the participation of five groups, each consisting of four participants. The four participants consisted of two development professionals, one management professional and one entrepreneur. The reason for having two development professionals in a group hinges on the fact that such professionals are likely to specialise in different areas of software development and may require different information. For example, a technical architect would probably require and use different information than that which a test engineer might utilise. Each group session lasted around one hour and consisted of three parts. The first part consisted of an introduction to the research being carried out and how the session would proceed. Following that, a brain storming session was held in order to identify what knowledge is required during a project's lifetime. Finally, a discussion was held in order to identify what effect particular items of knowledge or information have on the quality of a developed product.

Following the face-to-face sessions, results were analysed and a questionnaire was put together for participants to answer. This questionnaire consisted of questions targeted at identifying what effect (if any) the quality of particular information would have on the quality of the resulting product.

4.2 Research Results

The group discussions identified thirty-two items of knowledge which participants claimed influence a product's development. It is acknowledged that some of these items may overlap and that this number may fluctuate from one organisation to another. It was also noted that some of the items mentioned were relevant only in particular development methodologies. For example, burndown charts are used in the Scrum development process. Nevertheless, the scope of this research was to obtain an adequate sample of different knowledge items which influence a product's development. The items are listed below in alphabetical order.

Table 1 – Information used in Software Engineering

Bug Reports	Relevant Legislation
Burndown Charts	Requirements
Business Studies	Risk Assessments
Client Profiles	Source Code
Company Goals	Specifications

Company Policies	Spring Backlog
Decisions	Staff Morale
Designs	Staff Profiles
Feasibility Studies	Staff Project Allocation
Hardware Allocation	Staff Sickness Tendencies
Peer Review Results	Staff Vacation Plans
Product Backlog	Static Code Analysis Results
Project Budgets	System Architecture
Project Status	Technical Issues and Solutions
Project Timelines	Test Plans
Quality Metric Readings	Training Needs

After further discussion and analysis, it resulted that these knowledge items could each be placed in one of three categories. The first category is the *Technical Knowledge* category. This refers to knowledge which is related to the technical aspect of building a software product. Examples from this category include product requirements, architectural designs, test plans, metrics readings, and solutions to past technical problems. Thirteen (41%) of the items identified fell into this category.

The second category is the *Resource Knowledge* category and refers to knowledge related to the resources required to carry out a project. This includes knowledge such as staff training needs, staff project allocation, staff vacation plans, hardware availability, staff tendency to be sick, and so on. Ten (31%) of the items identified were classified as being in this category.

Finally, a third category emerged and was named the Constraining Knowledge category. As the name suggests, knowledge in this category would lead to stakeholders having to make decisions and take actions within certain boundaries, even if this sometimes means going against sound technical principles. Some examples of knowledge in this category include company goals and policies, decisions, time lines, market status, legislation and project budgets. Nine (28%) of the identified items were deemed to be in this category.

5 Information and product quality

One of the original goals of the research leading up to this paper was that of establishing a link between the quality of information used throughout product development and the quality of the resulting product. Results from the research exercise discussed in section 4.1 indicate that this is indeed the case. At this point, our research is only concerned with linking information quality to product quality. Although a future research goal would involve quantifying what aspects of product quality are influenced by particular aspects of information quality, this is not yet within our scope. As such, instead of analysing each individual knowledge item and the information associated with it, it suffices to analyse the three knowledge categories identified in section 4.2. This section categorised all knowledge information as being technical, resource-related or constricting. Each of these categories is examined in turn below.

Participants in our research exercise claimed the quality of the *technical information* used throughout a development process was paramount to the resulting solution. It may sound obvious that, for example, creating code based on a design which was in turn based on conflicting and inaccurate specifications will result in a product of questionable quality. However, participants highlighted a number of interesting situations which may not seem so obvious. One such example involved a team encountering a problem with a third-party library used to develop a product. This problem was a show-stopper and took three days to solve. Considering that the team was working within a twenty day iteration, this resulted in the loss of 15% of the total iteration time. During a postmortem meeting, it was frustrating for the team to discover that one of the other teams said they had encountered and solved the same problem in a previous project. Had there been adequate knowledge transfer between teams, the 15% of iteration time spent fixing the problem would have instead gone towards adding more functionality and/or improving overall quality.

With regards to *resource-related* information, opinions initially varied as to the actual impact this had on product quality. Beyond staff-project allocation, participants seemed to be used to a fire-fighting approach when it came to resources. If someone took some unplanned days off or was out sick, the other team members would cover for him or the person involved would work late nights to make the deadlines upon returning to work. The same approach seems to be applied to hardware availability. If for example an important test server develops a fault, participants claimed they simply do the best with whatever resources were left until the server was fixed. These arguments seem to indicate that human resourcefulness and sheer effort makes the need for high-quality resource-related information unnecessary. However, further discussion revealed otherwise. It transpired that in the case of the sick engineer who worked late nights in order to make up for lost time, the resulting module for which that engineer was responsible for a large number of problems discovered by the testing team. Similarly, in the case of a test server failing, this sometimes resulted in a product release being delayed or products being released without adequate testing. Eventually, participants agreed that having high-quality resource-related information at hand would facilitate better project planning which in turn would have a positive impact on product quality.

Finally, issues related to *constraining information* are analysed. In this regard, participants acknowledged that not having the right information at hand in this area would affect product quality although there seemed to be a certain aura of helplessness in the discussion and scenarios put forward. One participant complained that he had worked for a company which kept changing the priority of projects which were worked upon. Consequently, she was forced to switch between projects on a very regular basis. Project priorities are a result of company goals and company policies, both of which were identified as being types of constricting information. This is because even though on a technical or project management level, it makes more sense to finish an item of work before moving on to the next, if project priorities change you may be constricted to do otherwise. Another participant described a scenario where a product had to be considerably restructured because of a change in financial legislation. It turns out that this change in legislation had been announced more than a year before it actually came into effect. Had this knowledge been available to engineers, the product would have been done right the first time round.

From the research exercise carried out, it is clear that the presence or absence of required information with the required level of quality will impact the quality of the finished product. Hence a development process should ensure that all stakeholders have the all the information, with the right characteristics (quantity, representation, and so on) at the right time. The following section identifies a number of challenges involved when maintaining a knowledge context in this regard.

6 Challenges involved when maintaining a Knowledge Context

Having shown the need for development processes to maintain a knowledge context within an organisation, it is worth exploring what challenges one is likely to face when attempting this. Seven key challenges where identified and are discussed in this section. Given that systems grow increasingly larger in terms of the functionality they offer, the amount of information associated with such systems is also bound to grow. These circumstances, along with the temporal properties of information which were discussed in section 2, leads to the natural conclusion that electronic tool support would be needed when it comes to maintaining a knowledge context. This immediately gives rise to the challenges of how one would *capture* and *manage* increasing amounts of dynamic information relating to a project and the organisation as a whole. Typically, a chunk of information would need to be captured, associated or related to other information and somehow tagged with attributes so that it can be easily accessed in future.

Even if one develops a way of capturing and managing information, there is still a matter of quality which needs to be addressed. Before one allows a chunk of information to somehow influence the development of a product, one needs to be sufficiently sure of its quality. Lee et al [11] identified fourteen attributes relating to the quality of information. The third challenge surfaces here. How does one *evaluate the quality* of information in a reliable manner without being overly intrusive? Although knowing the quality of information is important, one must strike a balance whereby information quality can ascertained with a reasonable degree of certainty without being counter productive to development effort. On a related note, it would be desirable to *quantitatively link information quality to product quality*. That is to say, by evaluating the quality of an organisation's knowledge context one would be able to reason about, or even measure the quality of a product which is being developed at a particular point in time. The establishment of such a link would enable an organisation to take corrective measures from a knowledge perspective should the product quality not be desirable.

Finally we identify three challenges related to the temporal and dynamic quality of information. Over time, certain events will occur which will result in one or more people needing particular information. Such events may include a particular milestone being reached, a change in requirements, a change in relevant legislation, someone leaving the company, and so on. The challenges here refer to knowing *when a particular information asset is needed*, knowing *who needs it* and knowing *what characteristics it needs to exhibit*. The latter requirement is important because the same body of knowledge may be represented in different ways. Furthermore, the chosen representation has the potential to positively or negatively influence the effective use of that knowledge. Representations of a body of knowledge may differ in a number of ways such as format, level of abstraction, type, and so on.

7 Product Focused Models

The reasoning behind product focused development life cycles is indeed logical and, at face value, completely correct. In such models, the emphasis is on building a product of a certain level of quality, usually within a stipulated time frame. Typically, a product would go through a number of phases (specification, design, development, testing, etc) before finally being delivered. The main difference between different main stream development life cycles is the way the product transitions between these phases. In fact, existing life cycles are classified into four groups: sequential, incremental, evolutionary, and agile. The naming of these classifications illustrates the way in which a product will be built if it were to be developed using a life cycle in a particular group. Initially, this makes perfect sense. A software development team is in fact meant to develop software products. Hence it should follow that such teams follow a process which is focused on delivering products. However, having identified the need for a knowledge context to be maintained by an organisation, how well do existing development life cycles actually cope with the challenges identified in section 6. Having conducted research into a number of development life cycles, the authors of this paper conclude that these challenges are not comprehensively addressed by mainstream models. Even though more recent life cycles such as extreme programming (XP) [4], Scrum [5] and DSDM [12] cushion the effect changing information by introducing iterations or sprints, they still do not address most of the challenges identified in section 6. Of all existing life cycles, XP comes closest to achieving what we are looking for. It acknowledges the existence of institutional knowledge and promotes communication in feedback so as to facilitate its dissemination among members of a development team [4]. However, the development team by no means constitutes all stake holders of an organisation. Also, although the constant communication and feedback loop will likely have a positive effect on maintaining a knowledge context, it does not protect stakeholders against cognitive overload and other pitfalls identified in section 2. That being said, one is compelled to explore the possibility of there being a better way to handle the challenges presented when maintaining an information context.

8 The Information-Driven Approach

It is being proposed that an information driven software development life cycle be developed. This life cycle should effectively tackle the challenges identified in section 6 and produce a high-quality knowledge context as well as a high quality finished product. Broadly speaking, the life cycle should provide capabilities in two areas: *Knowledge capture and evaluation* and *Knowledge utilisation*. The knowledge capture and evaluation capabilities involve capturing real-world knowledge, relating it to other knowledge, evaluating its quality and storing it for later use. This aspect of an information driven life cycle has the potential of being tedious and prone to error so care must be taken to devise techniques which utilise automation as much as possible and minimise the risk of human error. Once knowledge of known quality is stored, the life cycle will utilise it over time to achieve the development of a high quality product. This will involve disseminating the information to people who need at the time they need it, monitoring the quality of the information as it changes over time, allowing querying of information. Finally it would be useful to be able to predict product quality based on the quality of the information being used by the development process. It is therefore perceived that the life cycle consist of the components shown in the figure 1.

It is the intention of the authors of this paper to carry out further research in this area so as to develop such a life cycle. Preliminary work carried out in this area suggests that such a life cycle will require interdisciplinary contributions from areas such as computer science, psychology and educational theory.

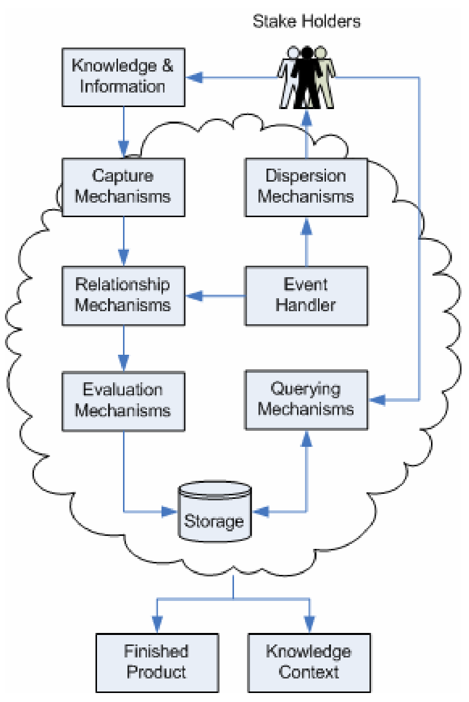

Figure 1 - Proposed Components of Life Cycle

9 References

[1] Standish Group International, The Chaos Report, 1994

[2] Jones A., Williams L., Public Services and ICT - Where next for the transformational government?, Reasearch Report for The Work Foundation, 2006

[3] Royce W.W., Managing the Development of Large Software Systems: Concepts and Techniques, Proc. WESCON, 1970

[4] Beck K., Extreme Programming Explained: Embrace Changee, Addisson-Wesley, 1999

[5] Takeuchi H., Takeuchi I., The New New Product Development Game, Harvard Business Review, January 1986

[6] Cachia E., Micallef M., Towards a RAD Framework for e-commerce systems, Proceedings of the International Conference on Web Information Systems, 2006

[7] Ho J., Tang R., Towards an optimal resolution of information overload: an infomediary approach, Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work, 2001

[8] Kirsh, D. A Few Thoughts on Cognitive Overload. Intellectica, 2000

[9] Ang S., Slaughter S., Turnover of information technology professionals: the effects of internal labor market strategies, ACM SIGMIS Database, Volume 35 Issue 3, 2004

[10] DeMarco T., Lister T., Peopleware - Productive Projects And Teams, 1999

[11] Lee et. al., AIMQ: a methodology for information quality assessment, Information and Management Journal (Issue 40), Elsevier, 2002

[12] Stapleton J., DSDM: Business Focused Development, Pearson Education, 2003