Practical Implications of Rapid Development Methodologies

Aurona Gerber
Meraka Institute, CSIR, Pretoria, South Africa
agerber@csir.co.za

Alta van der Merwe
School of Computing, University of South Africa, Pretoria, South Africa
vdmeraj@unisa.ac.za

Ronell Alberts
Meraka Institute, CSIR, Pretoria, South Africa
ralberts@csir.co.za

Abstract

Rapid development methodologies are popular approaches for the development of modern software systems. The goals of these methodologies are the inclusion of the client into the analysis, design and implementation activities, as well as the acceleration of the system development phases through an iterative construction approach. These methodologies also claim to manage the changing nature of requirements. However, during the development of large and complex systems by a small and technically competent development team, there is a danger that certain unforeseen practical implications are introduced into the development process by rapid development methodologies. In this paper we reflect on some observed practical implications of rapid development methodologies after the completion of two projects that involved the construction of large and complex software systems.

Keywords: Systems Engineering Methodologies, System Development Methodologies, Rapid Development Methodologies, Information Systems Development.

Introduction

The goal of rapid development methodologies is to address the perceived limitations of formal methodologies based on the traditional System Development Life Cycle (SDLC) such as long system development times, rigorous and inflexible requirements management and the separation of the client from the development process (Avison & Fitzgerald, 2006; Beck, Beedle, van Bennekum, Cockburn, Cunningham, et al., 2001). The intent of rapid development methodologies is for development to take precedence in the process, for the client to be part of the analysis, design and implementation activities, the acceleration of these phases through an iterative construction approach and the incorporation of changing requirements into the development process (Avison & Fitzgerald, 2006; Whitten, Bentley & Dittman, 2001).

Rapid development methodologies such as RAD or *Rapid Application Development* (Avison & Fitzgerald, 2006, p.469) , XP or *eXtreme Programming* (Beck, 2000) and the methodologies of the Agile Alliance (Agile Alliance, 2006) were

adopted by many software development organizations in the 1990s as an alternative to the rigorous prescribed traditional development methodologies (Ambler, 2002; Beck, 2000).

Proposed advantages of these rapid development methodologies include that the system being developed could accommodate the changing nature of system requirements (Avison & Fitzgerald, 2006) and that clients have earlier access to limited versions of the functional system (Pressman, 2005).

Despite the enthusiasm for these methodologies, several criticisms are voiced against the adoption thereof. For example, Reilly and Carmel (1995) argue that the RAD methodology will generally fail in Information Systems projects due to the incorrect selection of the team members, the poor understanding of the project by management and customers, and the lack of design and rigorous methodology processes.

Notwithstanding these criticisms, rapid development methodologies are popular approaches for the development of modern software systems (Fowler, 2005). However, during the development of *large and complex systems* by a *small and technically competent development team*, there is a danger that certain practical implications that were not foreseen are introduced into the development process.

In this paper we reflect on observed practical implications of rapid development methodologies after the completion of two projects that involved the construction of large and complex software systems.

An investigation into the practical implications of the modern rapid development methodologies in contrast with the traditional SDLC based methodologies (such as presented in this paper), would assist organizations to understand the pitfalls associated with rapid and ad-hoc approaches that have a strong focus on *the skill if the developers*. In the next section a discussion of systems development methodologies in general is presented, with the focus specifically on the history and the change in emphasis, followed by a section highlighting the differences between the methodologies. Furthermore, a discussion of rapid development methodologies in practice is presented that contains two case studies, followed by a discussion of the practical implications of development using these methodologies. The paper is concluded in the last section.

The Development of Methodologies

Since the introduction of *structured programming* by pioneers such as Parnas and Dijkstra, developers were formulating processes or *methodologies* for the development of software systems (Dijkstra, 1968, 2001; Parnas, 1972, 1978; Weiner, 1978). These pioneers derived techniques to model a system as consisting of different and related components, and the identification of these components necessitated the first structured information systems development methodologies.

Before the advent of methodologies, the emphasis of systems development was on the skills associated with programming (Sommerville, 1982). However, as system requirements became more complex, programmers started to appreciate the skill of the system analyst and realized that there was a need for a repeatable process (Satzinger, Jackson & Burd, 2002). These realizations as well as a requirement to manage the cost associated with systems development, lead to the attempts to formulate the first methodologies (Pressman, 2005).

The methodology initiatives of the early-methodology era of the 1970s and early 1980s were characterized by attempts to identify system phases for the management and development of information systems (Avison & Fitzgerald, 2006). This original approach was refined and became known as the as the SDLC (Systems Development Life Cycle) or the *waterfall model*.

SDLC

Although there exist many variations, the SDLC or *waterfall model* generally has the following basic structure (Avison & Fitzgerald, 2006; Kassler, 1997; Sommerville, 1982):

- Feasibility study;
- Requirements definition/System investigation;
- Systems analysis;
- Systems design;
- Implementation;
- Testing; and
- Review and maintenance.

The emphasis of the SDLC is on *requirements definition* as indicated by **R** in Figure 1. The bars in Figure 1 depict the emphasis on the different phases as described by the methodology or process and the thick, bright red arrow indicates that the methodology is essentially driven from the *requirements definition* phase (Avison & Fitzgerald, 2006; Kassler, 1997). The brown arrow depicts the actual emphasis of the project team during development. In the case of the SDLC, the brown arrow follows the defined phases closely, implying that the actual effort of the project team follows the prescribed effort of the process. The deviation of the brown arrow from the phases is only at the end of the process where the methodology does not put a lot of emphasis on maintenance, but most project executions required a substantial effort during system maintenance (Avison & Fitzgerald, 2006; Sommerville, 1982).

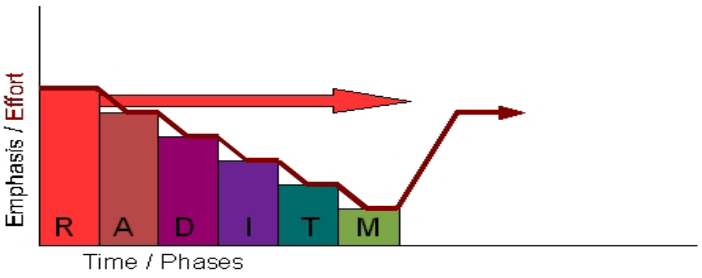

Figure 1: Emphasis of the SDLC is on requirement definition, and the process is driven from the requirement definition (**R**) phase. **A** indicates analysis, **D** design, **I** implementation, **T** test and **M** maintenance.

It is possible to argue that the traditional SDLC forms the basis of most information systems development methodologies of the *early-methodology* and *methodology eras*, as it is recognizable through the sequence and naming of its phases in almost any systems development project execution. Methodologies such as SSADM (Structured Systems Analysis and Design Methodology) (SSADM, 2003), Merise (Merise, 2006), IE (Information Engineering) (Martin, 1989), and even UP (Unified Process) (Jacobson, Booch & Rumbaugh, 1999) and RAD (Rapid Application Development) (Avison & Fitzgerald, 2006, p.469) generally aimed to refine the SDLC and address the identified limitations by focusing on aspects that were perceived to be neglected by the SDLC.

There are several advantages to following SDLC based approaches, including that it has been well tried and tested, and was found to assist with the process of systems development through the rigor prescribed by its phases (Sommerville, 1982). Following the approach allows at least some control and management of the development process. The SDLC specifies specific docu-

mentation standards and these documentation sets help to ensure that the requirements and system design are documented and traceable for maintenance purposes (Pressman, 2005).

However, there are also a number of identified limitations with SDLC based approaches such as the inflexibility to allow for changing requirements, the problems to keep the specified documentation standards updated, the instability of models of the prescribed system processes, development backlogs and systems maintenance overload (Ambler, 2002; Kassler, 1997).

Structured and Evolutionary Methodologies

The methodology era of the 1980s and 1990s was characterized by attempts to address the identified limitations of the traditional SDLC by specifying enhancements to the basic SDLC process. Noteworthy methodologies of the time such as IE (Martin, 1989), SSADM (2003) and Merise (2006) all incorporated the SDLC and have, as significant characteristics, aspects such as a *rigorous development process, well defined deliverables,* and *substantial documentation sets.*

The emphasis of structured methodologies is generally also on the requirements definition phase as indicated by **R** in Figure 1 because these methodologies are SDLC based. However, one of the themes that emerged during the development of formal SDLC based methodologies is evolutionary development. Methodologies including some form of evolutionary development address the maintenance overload caused by the SDLC as depicted in Figure 2. Evolutionary development thus introduces a feedback mechanism into the SDLC, which implies that the development phases are repeated. Evolutionary development is an incremental approach that periodically delivers a functional system that is increasingly complete (Avison & Fitzgerald, 2006).

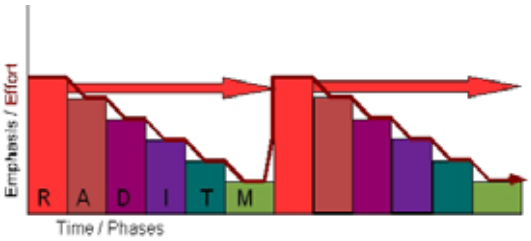

Figure 2: Emphasis of evolutionary approaches is also on *requirement definition*, the process is driven from the *requirement definition* (**R**) phase but the whole process is repeated. **A** indicates *analysis*, **D** *design*, **I** *implementation*, **T** *test* and **M** *maintenance*.

As before, the bars in Figure 2 depict the emphasis on the different phases as described by the methodology and the thick, bright red arrows indicate that these methodologies are driven from the *requirements definition phase* (Avison & Fitzgerald, 2006). The brown arrow depicts the actual emphasis of the project team during development, and as in the case of the SDLC, the brown arrow follows the defined phases closely, implying that the actual effort of the project team follows the prescribed effort of the process (Avison & Fitzgerald, 2006).

During this era of formal methodology development, the term *methodology* was coined to describe systems development approaches and processes (Avison & Fitzgerald, 2006), and a definition of the term was established as:

A collection of procedures, techniques, tools and documentation aids which will help the system developers in their efforts to implement a new information system. A methodology will consist of phases, themselves consisting of sub phases, which will guide the systems developers in their choice of the techniques that might be appropriate at each stage of the project and also help them plan, man age, control, and evaluate information systems projects. (Avison & Fitzgerald, 2006, p.24)

This definition attempts to capture the nature and comprehensiveness of software development activities by grouping it together into a *software development methodology*.

However, in spite of all the refinements to the SDLC by the formal methodologies of the time, there were still significant failures in systems developed using these methodologies. In order to address the perceived failure of SDLC based methodologies to sufficiently address significant information systems development issues such as productivity, inflexibility and complexity, methodologies with a substantial different approach were subsequently developed during the so-called era of methodology reassessment (Avison & Fitzgerald, 2006).

Rapid Development Methodologies

The late 1990s and onwards are characterized by the appraisal of methodologies because of their failure to sufficiently addresses information systems development issues such as the mentioned productivity, inflexibility and complexity (Avison & Fitzgerald, 2006). During this era several alternative methodologies, such as UP (Jacobson et al., 1999) and XP (Beck, 2000), that deviate from the traditional SDLC approach were formulated. In addition, the evolutionary approach was refined and included as *incremental development* into several modern methodologies.

Even more significant, this era includes the rejection of methodology use by some organizations altogether, signifying a return to *ad hoc software development* where no formalized methodology is followed (Avison & Fitzgerald, 2006; Introna & Whitley, 1997). In these cases the adopted approach is the one developers understand and feel works for them. It is driven by, and relies heavily on, the *skills and experience* of developers and could be compared to the pre-methodology era and the advent of structured programming.

Rapid development methodologies all emphasize the *skills and experience* of developers and focus on the *implementation phase*. In general, rapid development methodologies also involve the client at all levels and reduce the development time of systems through an incremental approach and by eliminating perceived methodology overload such as unnecessary documentation. For example, XP (Extreme Programming) adopts five basic principles (Beck, 2000, p.37):

- Rapid feedback;
- Assume simplicity;
- Incremental change;
- Embracing change; and
- Quality work.

Similarly, James Martin's RAD (Martin, 1991) adopts an evolutionary approach with four phases namely (1) joint requirements planning (JRP), (2) joint application design (JAD), (3) construction and (4) cut over. The JAD phase places a lot of emphasis on prototyping.

Furthermore, several agile development approaches were defined that have as a starting point the perceived inadequacies of the SDLC. The agile school believes that requirements are so difficult to define that systems should evolve in collaboration with the client (Fowler, 2005). The agile movement defines their philosophy in the Agile Manifesto (Beck et al., 2001) as:

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation; and
- Responding to change over following a plan.

The common principles adhered to by rapid development methodologies are that the development process should be able to incorporate change through incremental development and rapid feed-

back (Beck, 2000), as well as by eliminating prescribed rigor and development phases. In addition, communication with the client takes the form of active participation and is encapsulated by the developers into the implementation of the system. These methodologies in general shy away from documentation artifacts as deliverables (Ambler, 2002; Fowler, 2005).

As stated, emphasis of rapid development methodologies is generally on the *implementation* phase as indicated by **I** in Figure 3. Even though the SDLC phases of *requirements, analysis, design, implementation* and *test* are still recognizable, the emphasis upon and sequence of the phases deviates completely from previously defined methodologies (Avison & Fitzgerald, 2006).

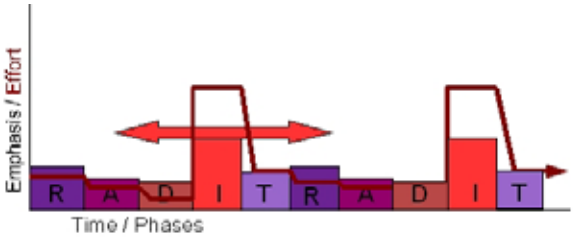

Figure 3: Emphasis of the rapid methodologies is on *implementation* indicated by **I**, and the process is driven from the *implementation* (**I**) phase. **R** indicates *requirements definition*, **A** *analysis*, **D** *design* and **T** *test*.

As before, the bars in Figure 3 depict the emphasis on the different phases as described by the methodology and the thick, bright red arrows indicate that rapid methodologies are driven from the *implementation* phase (Avison & Fitzgerald, 2006). The brown arrow depicts the actual emphasis of the project team during development, and in this case, developers tend to spend even less time than prescribed on phases other than *implementation*, and more on *implementation* itself (Avison & Fitzgerald, 2006) as indicated by the brown arrow. *Design* in particular tends to be omitted in favor of *implementation* (Fowler, 2004). This observation is strengthened by the case study descriptions presented in this paper.

The Fundamental Differences between the Approaches

From the previous discussion, the differences between the methodological approaches are reflected upon in this section.

The traditional SDLC based methodologies generally prescribe rigorous processes and activities within the project phases that follow strictly on one another. Each phase prescribes specific sets of deliverables, usually several documentation sets along with the working system, and some even define document templates in support of these deliverables. Within these methodologies, the roles of individual team members are usually well-defined. Communication between all stakeholders is formally agreed upon and captured into the system documentation (Avison & Fitzgerald, 2006; Pressman, 2005).

In contrast, rapid development methodologies shy away from rigour and formally prescribed processes. These methodologies acknowledge development phases, but generally move through these phases in an ad-hoc and incremental manner. The main delivery focus is a working system, delivered in working increments in collaboration with the client. These methodologies prescribe minimal documentation artifacts, and communication with the client is verbal and through active participation of the client in the development process (Avison & Fitzgerald, 2006; Fowler, 2004; Pressman, 2005).

Rapid Development Methodologies in Practice

The research approach reflected on in this paper is based on a case study approach combined with ethnographic research where the researcher is an active participant in the research activity (Myers, 1999). The systems that are described in the case studies, were developed by a small but very competent development team (varying between five and ten members), who adopted a rapid development methodology. This approach was adopted after the claimed success stories of these approaches and in order to keep up with significant trends within the IS (Information Systems) domain. The team investigated rapid development methodologies and adopted an agile approach mainly because of the promises of reduced cost and development time.

The rapid development approach adopted can be described as a lightweight, agile method. However, due to practical constraints and distributed responsibility allocations, the client (as a single entity) could not be such an active participant into the development process as prescribed. Therefore, there was an emphasis to involve the client through use-case scenario development and mock-up prototypes.

In this section we present a brief overview of two case studies of projects developed using rapid development methodologies.

Case Study 1

Case study 1 entails a project for a government health department to support and integrate the processes spanning three different operational units to compensate mine workers in the event of occupational related illnesses.

The requirements elicitation process included several site visits by the whole team to *live the process,* familiarize themselves with the environment and the associated problems of the customer. The initial requirements elicitation activities consisted of a definition of the scope as well as the deliverables of each project increment. Formal analysis techniques were used to analyze customer business processes, as well as all existing documentation and forms.

The project team adopted an agile iterative and incremental development methodology. Detailed requirements were documented for each identified iteration. Initial design activities included an entity-relationship data model with a data dictionary, as well as a high-level architectural design documented in an informal manner.

After these initial activities, the developers commenced with implementation of the first system iteration. The technical architecture and design were not formally documented but were encapsulated in the development platform environment. Test cases were developed based on the detailed requirements and each system increment was tested accordingly.

After deployment the development team performed support and maintenance for 18 months to enable the client to find a company to do the ongoing support and maintenance of the system.

Case Study 2

The second case study entails a project for the development of an information and communication portal for persons with disabilities. One of the most significant aspects of this project is an absent client since the project was developed to support a national priority of government as stated by the office of the deputy president (OSDP, 1997).

The requirements elicitation process consisted of literature reviews, site visits to organisations supporting persons with disabilities, workshops, mailing lists and the inclusion of Disabled Per-

sons Organizations in the project team. Since there was no specific client, the requirements were defined based on what was learned through the activities mentioned.

Due to the nature of the user base, usability and accessibility were identified as essential non-functional requirements the system must adhere to (Alberts, van der Merwe & Pretorius, 2006).

The project team again adopted a rapid iterative and incremental methodology. The features to be implemented in the system iterations were identified by the development team. Detailed requirement specifications were developed for each feature. The detailed requirements specifications were used by the development team for development as well as the testers to develop test cases.

The design of the system consisted of an entity diagram as well as GUI information architecture designs depicting page flow and layout. The architectural design was heavily influenced by the technology chosen for implementation. Detailed design was not formally documented and was verbally communicated in the development team. Testing was performed based on the test cases developed from the detailed requirements. After a system iteration was completed it was deployed as a Web-based system and access was provided to the user community. Feedback received on the deployed system was used to plan the next system iterations.

In general, the incremental approach included in the rapid development approach proved to be effective within both case studies in order to manage changing requirements. The short feedback cycle also proved to be satisfactory for developers as they could often get positive feedback on their development efforts. It must also be noted that all the developers experienced the rapid development approach as positive. They felt empowered and even though they often complained about client demands, the direct contact they had with clients were regarded useful.

Practical Implications of Rapid Development Methodologies

During and after the use of a rapid development approach, a number of observations were made on the changing nature of development activities, some of which are supported in literature (Ambler, 2002; Avison & Fitzgerald, 2006). Based on these observations, the following four significant implications were extracted namely the confusion with regard to the changing roles and responsibilities within the development team, ineffective communication, requirements prioritization and the omission of design activities. We provide a discussion of each of these in the following subsections.

Confusion with Regard to the Roles and Responsibilities of Team Members

One of the most significant observations regarding the practical implications of rapid development methodologies is the change in roles assigned to the members of the development team.

Roles and responsibilities in traditional methodologies

Traditionally, *analysts* would perform the requirements elicitation and analysis activities, and would rigorously and formally document their findings. One of the skills of an analyst is the ability to communicate with clients in their domain language and translate these requirements into technical terms for the development team to interpret (Avison & Fitzgerald, 2006; Pressman, 2005).

Analysts therefore document the requirements and perform initial high-level design, documented in the form of analysis diagrams that depict entities within the domain with the relationships be-

tween them (Sommerville, 1982). During the next phase, *system architects* and *developers* design the system, and would document the design and design decisions. Usually this design is reviewed by the whole development team including the analysts and even the client. After these activities were performed, the development team commences with implementation with the support of the analysts.

In contrast, rapid development methodologies only assign a clear role to the developer and negate the role of system analysts.

The "Developer as King" phenomenon

Developers rather than *analysts* interact directly with the clients about their requirements and the role of analyst therefore become uncertain and even redundant in rapid development methodologies. Analysts often loose their authority or decision making power with regards to the development process. This leads to the phenomenon where the system *developers*, because of their technical skill and the emphasis on the rapid construction of a working system, become the main decision makers within the process with regard to design and implementation choices.

This shift in roles and the associated responsibilities often lead to conflict within the team, and when a decision needs to be made in the team on behalf of the client rather than what technology prescribes (traditionally done by an analyst), the developer makes the decision based on technology preference. Developers become the *drivers* of the systems development process. In these cases developers often cannot be persuaded to implement the system in a way they do not agree with since they are the primary decision makers.

In addition, developers have a natural (and healthy) tendency to focus on new and exciting technologies, and design decisions are therefore often technology driven rather than project, client or requirements driven.

In case study 1 the client was not a single entity, but the the department of health as organization and the system spanned the work flow across various operating units. It was also not practically feasible for the client to be as involved with the process as required. The development team was not able to translate the perceived needs into technical requirements and in response reverted back into the traditional role structure where analysts extracted the requirements and translated it to requirements that were implemented by the development team.

In case study 2 this phenomenon was even more prevalent as the client as entity was absent and the system analysts, which were part of the team, had to enforce decisions based on their acquired knowledge of the domain. This was refuted on occasion by the technical team and since the analysts did not have authority due to role transitions, certain essential functional and non-functional requirements were not implemented.

Ineffective Communication

Rapid development methodologies generally emphasize the interaction of developers directly with the client and active involvement of the client in the process.

However, because of their technical orientation, developers tend to use technical terms when communicating with clients, which often leads to misunderstanding and confusion. Developers tend to speak a different language than that used by the general client, and this often results in developers deciding that clients *do not know what they want*. The *analyst*, who played an interpretation role in traditional methodologies, is bypassed in the rapid development methodologies.

In these cases developers may make decisions on behalf of the client without taking into account the operating domain of the client, resulting in systems that are not optimal solutions for the cli-

ent. This is especially true in the case of large systems where the 'client' is not embodied in a single person, but spans organizational processes, as was the case in case study 1.

In case study 2 the client was absent and the system analysts analyzed, studied and interpreted the domain and compiled the requirements on behalf of these absent clients. However, the defined requirements were often questioned or changed because of the assumption that the decisions of one team member (the analyst) was as good as those of another (the developer). This was mainly due to the fact the inter-team roles and communication was not clearly established.

Requirements Prioritisation

The rapid development methodologies prescribe a process that is less rigid than that of more formal methodologies. This process does not include a formal design review by all stakeholders, but rather prefer continuous feedback by the client.

This process with the additional emphasis on *rapid* delivery often leads to functional requirements implemented in a way that is the 'easiest' for the developer. Developers make the final design decisions in this case, often to the detriment of important functional and non-functional requirements.

In addition, the developers prioritize the requirements to be implemented in rapid development methodologies, and where a conflict arises in the implementation of these requirements, decisions are often made in favour of requirements that are important to the developers, rather than those important to the overall project.

Within case study 1, it is noteworthy that the development team decided to adhere to existing skills when a technology platform was chosen because it sped up delivery of the initial system increments. The developers chose to prioritize speedy delivery and a known skills base above the non-functional requirement of the client to outsource maintenance and support of the system. When the system was delivered, the technology platform was regarded as outdated, and hence maintenance intensive. This provided an obstacle with regards to the outsourcing of the ongoing support and maintenance of the system.

In case study 2 one notable situation enforces this observation. People living with disabilities require usable and accessible systems and these requirements were captured as high-priority by the system analysts. However, the chosen technology platform placed a high priority on maintainability which often conflicted with the usability requirements. In general, the developers made the final decisions, which resulted in the implemented system not fully adhering to the accessibility and usability requirements and therefore failing to sufficiently address the needs of the intended audience.

The Omission of Design

The emphasis of rapid development methodologies on rapid delivery and an adjustable process often leads to the omission of the design phase. The design phase might be included in the first system development increments, but further system iterations are often developed by just entering the *implementation* phase again without executing a preceding *design* phase.

In addition, there is no emphasis on documentation by rapid development methodologies and therefore design documentation are often lacking. In traditional methodologies a design document had to be generated before implementation could commence and this forced developers to do design.

Generally, requirements documentation is still generated because most clients still require some agreement that is captured formally on paper. There is, however, no pressure to do design or ar-

chitectural documentation in rapid development methodologies and in most cases the architectural and systems design reside only in the heads of the development team. The development team often do not regard this as a problem because tasks to update documentation 'wastes time' and impacts on *rapid* delivery. This phenomenon was also discussed by Fowler (2004).

Within case study 1, it is noteworthy that one of the reasons the client experienced problems with regards to the outsourcing of the ongoing support and maintenance of the system was a lack of design documentation. When this became apparent, the development team attempted a reverse engineering of the system. This proved to be difficult because the design decisions, especially with regards to systems architecture, were not documented at design time and was hence lost.

Within case study 2, the initial design of the system conflicted with the system requirements and due to the fact that the design was not formally documented and reviewed, the discrepancy was only discovered after the implementation phase. This situation caused conflict between developers and analysts and in the end necessitated a redesign effort which put unnecessary pressure due to time constraints and limited resources on the whole development team.

To illustrate that this pitfall is a general phenomenon, the following quote appeared on a discussion forum in response to a request on how to capture the systems architecture in UML:

Hehe [sic], outside of university courses and some -extremely- backward consulting companies, designing software using UML is a definite anti-pattern :-) (JBoss.com, 2006).

Conclusion

In this paper we presented some unforeseen practical implications that were observed when development teams adopted rapid development methodologies for the development of large and complex systems. We observed that when organizations adopt these rapid development methodologies, care must be taken to avoid role and responsibility confusion and communication breakdown within the development team, and between the team and the client. In addition, especially in cases where the client is absent or not able to participate with authority in the development process, the system analyst should be endowed with this authority on behalf of the client to ensure appropriate prioritisation of non-functional requirements. Lastly, no increment of the system should be developed without a thorough and formally documented design phase.

Acknowledgments

The authors want to acknowledge the contributions of the development teams that were responsible for the development and successful deployment of the systems described in the two case studies.

References

Agile Alliance. (2006). *Agile Alliance Home*. Agile Alliance Web site. Retrieved 10/12/2006 from http://www.agilealliance.org/

Alberts, R., van der Merwe, A. & Pretorius, H. (2006). Using quality requirements to systematically develop a national accessibility portal. In *Proceedings of the International Science and Technology Conference, Vanderbijlpark, South-Africa* (2006).

Ambler, S. (2002). *Agile modeling: Effective practices for eXtreme programming and the unified process*. Wiley Computer Publishing. ISBN 0-471-20282-7.

Avison, D. & Fitzgerald, G. (2006). *Information systems development: Methodologies, techniques and tools* (4th ed.). UK: McGraw-Hill Education. ISBN 13-978-0-07-711417-6.

Beck, K. (2000). *eXtreme programming explained: Embrace change*. Addison-Wesley. ISBN 201-61641-6.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A.,Cunningham, W., et al. (2001). *Manifesto for agile software development*. Agile Manifesto Web site. Retrieved 10/12/2006 from http://agilemanifesto.org/

Dijkstra, E.W. (1968). *The structure of the THE-multiprogramming system*. *Communications of the ACM*, *11*(5), 341-346. ISSN 0001-0782. doi: rmhttp://doi.acm.org/10.1145/363095.363143.

Dijkstra, E.W. (2001). *The end of computing science? Communications of the ACM*, *44*(3), 92. ISSN 0001-0782. doi: rmhttp://doi.acm.org/10.1145/365181.365217.

Fowler, M. (2004). *Is design dead?* Retrieved 4/12/2006 from http://www.martinfowler.com/articles/designDead.html

Fowler, M. (2005). *The new methodology*. Retrieved 10/12/2006 from http://www.martinfowler.com/articles/newMethodology.html

Introna, L.D. & Whitley, E.A. (1997). Against method-ism: Exploring the limits of method. *Information Technology and People*,*10*(1), 235-245.

Jacobson, I., Booch, G. & Rumbaugh, J. (1999). *The unified software development process*. Addison-Wesley.

JBoss.com. (2006). *JBoss discussion forum - Seam UML diagrams thread*. Retrieved 6/12/2006 from http://www.jboss.com/index.html?module=bb&op=viewtopic&t=96447

Kassler, J. (1997). What do you mean, you can't tell me how much of my project has been completed? Presented at the *7th Annual International Symposium of the International Council of Systems Engineering (INCOSE)*. Retrieved 5/12/2006 from http://www.unisa.edu.au/seec/people/Jk/Pubs/crip.pdf

Martin, J. (1989). *Information engineering* (3 volumes). Prentice-Hall.

Martin, J. (1991). *Rapid application development*. Macmillan. ISBN 0023767758.

Merise. (2006). *Merise: Initiation al la conception*. Retrieved 10/12/2006 from http://www.commentcamarche.net/merise/concintro.php3

Myers, M.D. (1999). Investigating information systems with ethnographic research. *Communication of the AIS*, *2*(Article 23), 1-20.

OSDP. (1997). *Integrated national disability strategy white paper*. Retrieved 10/12/2006 from http://www.polity.org.za/html/govdocs/white_papers/disability1.html

Parnas, D. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, *15,* 1053 - 1058.

Parnas, D.L. (1978). Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd International Conference on Software Engineering* (pp. 264-277). IEEE Press.

Pressman, R.S. (2005). *Software engineering: A practitioner's approach* (6th ed). McGraw-Hill International. ISBN 0-07-285318-2.

Satzinger, J.W., Jackson, R.B., & Burd, S.D. *Systems analysis and design in a changing world* (2nd ed.). Course Technology: Thomson Learning, ISBN 0-619-06309-2.

Sommerville, I. (1982). *Software engineering* (2nd ed.). Addison-Wesley Publishing Company. ISBN 0-201-14229-5.

SSADM. (2003). *Structured systems analysis and design methodology (SSADM)*. Retrieved 10/12/2006 from http://www.ogcio.gov.hk/eng/prodev/essadm.htm

Weiner, L.H. (1978). The roots of structured programming. In *P8apers of the SIGCSE/CSA Technical Symposium on Computer Science Education* (243-254). New York, NY: ACM Press. doi: rmhttp://doi.acm.org/10.1145/990555.990636.

Whitten, J.L., Bentley, L.D. & Dittman, K.C. (2001). *Systems analysis and design methods* (5th ed.). McGraw-Hill Higher Education.

Biographies

Aurona Gerber is at present a Senior researcher in the Inclusive environments group within the Meraka Institute. She has approximately 7 years teaching and research experience, and was active as developer before that for more than 10 years doing software system design, development and project management.

Alta van der Merwe is an associate professor at the University of South Africa. She completed her Ph.D in 2005 in the requirements elicitation of generic process model structures. Her current research interest is in ontology engineering, with a special interest in the combination of ontologies and process engineering.

Ronell Alberts is a senior researcher at the Meraka Institute in the "Intelligent Environments for Independent Living" research group. She is actively involved in research and development activities and is responsible for analysis activities in the research group. Her research interests include software development methodologies and software engineering.