Hybrid Agile Development and Software Quality

David Parsons[1], Ramesh Lal[2]
[1]Institute of Information and Mathematical Sciences, Massey University,
Auckland, New Zealand
d.p.parsons@massey.ac.nz

[2]Institute of Information and Mathematical Sciences, Massey University,
Auckland, New Zealand
r.lal@massey.ac.nz

Abstract

Agile methods have been increasingly adopted as a way to increase the speed and flexibility of software development whilst maintaining or improving quality. However, organisations with a heavy investment in, and emphasis on, more traditional software engineering approaches may regard the wholesale adoption of agile methods as being potentially risky. As a result, a number of experiments in hybrid approaches have been made, suggesting that a combination of agile methods and traditional software engineering can be a route to quality software development. Further, that component based software engineering has an important role to play. In this paper we review the key issues in this debate and propose that test related practises are the most significant enabler in providing quality assurance in hybrid systems.

1.0 Introduction

Despite the mission criticality of software for most businesses, and decades of experience and technological advancement, significant numbers of software development projects continue to fail, costing substantial amounts of money [1] [2] [3]. These software project failures happen with companies regardless of their size. Table 1 provides a historical perspective on project success and failure from 1980 to 2000. Projects identified as ‘challenged’ are those projects that were completed but were over their budget and time estimates, with fewer features and functions than required. Thus we can see that even though these projects did not fail outright, they were not of sufficient quality.

Year	Succeeded	Failed	Challenged
1980	5%	48%	47%
1994	16%	31%	53%
1996	27%	40%	33%
1998	26%	28%	46%
2000	28%	23%	49%

Sources: [4],[5]

Table 1: A historical perspective on project success and failure (1980-2000)

The statistics do suggest that there has been some improvement in success rates, and a significant decrease in the proportion of failed projects. However the percentage of completed projects that are of insufficient quality seems to have remained fairly constant.

Some writers assert that formal software development methodologies have a positive impact on project success [6],[7],[8]. The Standish Group 2001 report [5] states that the use of a formal methodology should increase the chance of project success by 16%. We need to be aware however that use of a methodology is not the only factor, and the available literature provides many perspectives on project failures that occur despite the significant progress made in development methodologies and tools [9],[10],[11]. These perspectives can be grouped into socio-organizational, technical, and economic factors [12].

Even where we might isolate the effect of methodologies from other factors, we have to be aware of the number and diversity of methods that might be applied, It has been estimated by Jayaratna [13] that there are over 1,000 software development methodologies world-wide, and these cover a range of approaches, including ad-hoc, prescriptive, agile and hybrid [14],[15]. Table 2 summarises some of the key methodologies and their evolution since the 1960s.

Whether despite, or because of, the plethora of methods developed in the 1980s and 1990s, several key problems have remained with software development, including how to develop systems quickly while accommodating requests for changes late in development process and how to maintain quality while controlling costs [16]. In response to these problems, a new set of informal analysis and design approaches emerged, initially known as lightweight methods but later renamed agile methods by their promoters due to the potentially derogatory use of the term ‘lightweight’ [17]. It appears that agile methods are becoming increasingly popular and many organisations are adopting this new way of developing software. A number of reasons have been suggested as to why these new methods are more appropriate for software development in a dynamic business environment such as the ability to; (a) move quickly and react to change, (b) accept and welcome change, (c) deviate from a plan and treat it as new information, (d) optimize communication among various stakeholders, and (e) learn from each agile project

[18]. However there seems to be more adoption on an ad-hoc basis than for strategic reasons, with adoption based on subjective accounts of how methods were used to design and develop software in a given organization [19].

Period	Era	Methodology types
1960s and early 1970s	Pre-Methodology Era	Ad-hoc approach
Late 1970s - early 1980s	Early Methodology Era - prescriptive methodology	SDLC- waterfall model
Mid 1980s - late 1990s	Methodology Era – proliferation, software engineering, prescriptive methodologies	Structured- STRADIS, Yourdon Systems Method, SSADM, Jackson Systems Development; Data-oriented- IE, Prototyping-RAD, Unified Process, Object-Oriented Analysis, Participative-ETHICS, Strategic-ISP, Systems-ISAC,SSM, MULTIVIEW, Formal methods, Vienna Development Method
Late 1990s onwards	Post Methodology Era	Ad-hoc, Agile methods; Scrum, Dynamic Systems Development Method, Crystal Methods, Feature-Driven Development, Lean Development, Extreme Programming, Adaptive Software Development, Agile modelling, Internet-speed development

Sources: [8],[14],[20],[13]

Table 2: Some key methodologies used for information systems development since the 1960s

There are claims that the use of agile development methods enables software to be created without the overheads of prescriptive methods but it appears that no major academic research has been undertaken to verify these claims and practices, or to provide a better understanding of the mechanics of agile methods. Therefore there is a potential risk in organisations migrating to agile methods without an understanding of how quality assurance can be maintained in a less formal approach. In this paper we look at quality assurance in the adoption of agile methods by organisations with a traditional software engineering ethos. We begin, in the following section, by describing the key features of agile methods. We then move on to look at how these methods might be adopted by organisations that wish to retain significant aspects of their current software engineering infrastructure and review some reported practice. We then consider the role of component based software engineering practice in ensuring software quality and consider how this can be integrated with an agile approach. Finally we address the main contribution of this paper, where we propose that test related practises are the most significant

enabler in providing quality assurance in hybrid systems. We conclude with some suggestions for further work.

2.0 Agile Methods

In February 2001 a group of seventeen software experts got together in Snowbird ski resort in the Wasatch Mountains of Utah, to discuss the growing field of what used to be called lightweight methods [17]. They decided to use the term *agile* to describe these new methods and the agile software development manifesto was written, describing the values and principles of the agile movement [21]. Agile software development is seen as an alternative to software engineering driven development. Software engineering is often seen as a rigorous process that requires substantial planning, modelling, and creation of various artefacts. Table 3 lists methods that are part of the agile family, which are based on the belief that a better way of developing software is by actually creating the software itself rather than spending a considerable amount of time determining what is to be developed, planning the various activities of the software development, designing, and modelling the features of the software before embarking on any software construction work.

Method name	Year
Lean Development (LD)	1980s
Dynamic Systems Development Method (DSDM)	1995
Scrum	1995
Crystal Methods	1998
Extreme Programming (XP)	1999
Internet-speed development (ISD)	1999
Adaptive Software Development (ASD)	2000
Feature-Driven Development (FDD)	2002
Agile modelling (AM)	2002

Sources: [17], [19]

Table 3: Methods that are part of the agile family

The Manifesto for Agile Software Development states the following: “we are uncovering better ways of developing software by doing it and helping others to do it. Through this work we have come to value:

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

That is, while there is value in the items on the right, we value the items on the left more” ([17] p. xvii). These new methods attempt to provide balance between the items on right and the items on the left, rather than replacing the items on the right. According to Martin [22] it is a useful compromise between no process and too much process, providing just enough process to gain a reasonable payoff.

Agile software development is based on customer satisfaction, early incremental delivery of software, small and highly motivated teams, informal methods, and minimal software processes [23]. Before the emergence of agile methods, there was a belief in the software industry that successful software development would succeed only through careful project planning, formalization, quality assurance, the use of analysis and design methods supported by CASE tools, and controlled and rigorous software development processes [6]. However, this plan-based software development tied the developers down in following the processes, where a considerable amount of time is spent in planning, eliciting systems requirements, designing and modelling the requirements, and writing extensive documentation rather than creating the actual system. Another major concern was that the systems requirements tended to change even before any new system could be fully implemented. In addition, 'high ceremony' methods like this are more suited for large scale industrial and scientific software development and when applied to software development for small and medium-sized business, the cost is overwhelming [6].

The 'minimal' process that is used with an agile software development method is based on the belief that during systems development it cannot be determined in advance which requirements will remain static and which ones will change due to changing business conditions. The requirements are prioritized, and the agile process recognizes the fact that the requirements and requirement priorities may well change during the development period. With agile methods, any function or feature of the software is developed immediately after minimal design to test it out, hence becoming a test-driven methodology. The agile process is based on the idea that analysis, design, development, and testing activities cannot be predicted and planned in advance [21].

3.0 Prescriptive Methods- Quality Focus

According to Pressman [23], prescriptive methodologies can be classified broadly into four types; (a) the waterfall model, (b) incremental model-RAD (c) evolutionary development – prototyping, spiral model, concurrent development model and (d) specialized process models - component based software engineering. These software engineering models are process driven, where software process is seen as providing the necessary framework that enforces to perform the required tasks whereby a high quality software product is built. Process includes the approach to be taken for analysis, design and development of software.

Software engineering identifies the process layer (Figure1) as critical in enabling tools and methods to be used to develop quality software. According to Pressman, the process layer defines the framework activities - communication, planning, modelling, construction, and deployment, which must be established for effective delivery of the software. For software development projects, it provides for the following: (a) control activities, (b) work products, (c) milestones, (d) quality standards, and (e) change management for requirements.

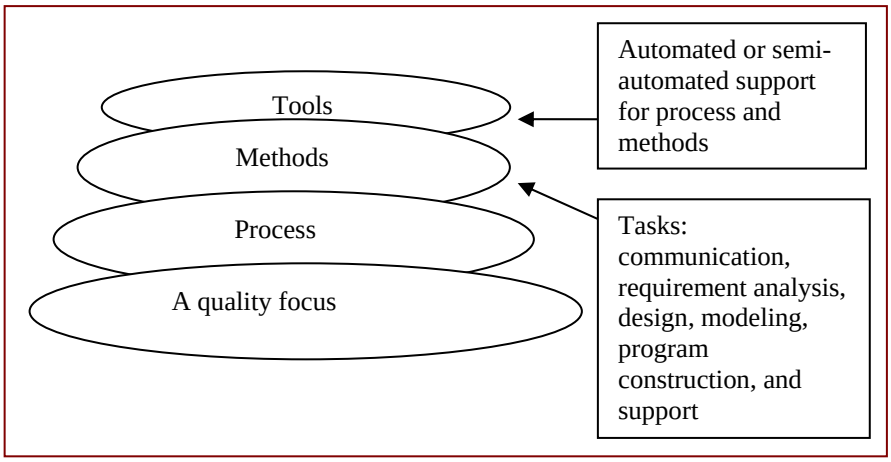

Figure 1: Software Engineering as a layered approach (from Pressman [23])

The software engineering perspective on quality is that accurate definition of requirements provides the necessary groundwork for ensuring quality of the software together with using quality metrics and employing rigorous testing. Software quality is also impacted by a combination of factors. According to Pressman these factors can be categorized into two types; those that can be measured directly (defects) and those that can only be measured indirectly (usability and maintainability).

In ensuring quality through this approach, a considerable amount of time is spent determining requirements before any software is written. A feasibility study, requirement elicitation and analysis, requirement validation, requirement reviews, and requirements management occur first. Various artefacts such as activity diagrams, class diagrams, and use cases are created to model systems requirements. System models; context models, behavioural methods, data models, and object models are developed based on requirements showing operational, functional, and behavioural characteristics of the system [6]. Before any coding begins, an architectural design is created showing the structure of data and program components and component level design is created to determine if software will work as planned [23]. This shows that with software engineering a considerable amount of design and modelling work happens, which may take months, before any attempt is made to create the software [6]. Software engineering views these detailed levels of activities, designs and models as critical for achieving quality whereas agilists believe creating this level of detailed artefacts in advance is unnecessary and time consuming [18],[22].

3.1 Comparing Agile and Prescriptive Methods

There is substantial literature available regarding agile methods, documenting conditions where these methods are more suited for software development than prescriptive methods. Agile methods claim to offer an improvement in quality of software, requirements management, customer satisfaction, and team satisfaction

[24] [25]. They suit software projects with uncertain requirements [21], which suggests that the agile practice of iteration as essential for software to be created. The rapid and iterative aspect of agile methods also enables frequent release of working products for customers to see throughout the entire development process [26],[27].

Agile methods emphasize that any process used should be effective and efficient and needs to change as an organization's needs change. Agile methods are based on the idea of 'barely sufficient process' [28],[29],[30], which agilists believe enables successful software development. Researchers also point out that moving away from extensive process based design enables development teams to meet customer demand quickly [31],[32]. Agile methods adopt the practice of simplicity, i.e. avoid any unnecessary work or tasks that do not add value to the project [18]. Communication and feedback enable developers to optimize various stakeholder involvements in agile projects [33]. Agile methods advocate a significant amount of interaction between the development team and the customer to create a positive working relationship. Hence, collaboration with customers is also regarded as one of the important agile software development practices [34],[24]. This practice emphasizes that all stakeholders must work together as a team throughout the development process.

To adopt agile methodologies, the software development team will need to be equipped with people-skills or soft-skills. According to Highsmith [17], agile methodologies place far more value on the interaction of talented individuals over the process and tools that are the key themes of prescriptive methodologies. Agile methods require team members to have appropriate communication skills for collaboration to happen in a team situation with customers and also have a friendly approach and talent to relate well with others [35]. Pressman [23] lists the following 'must have' traits of agile team members; (a) competence, (b) common focus, (c) collaboration, (d) decision-making ability, (e) fuzzy problem-solving ability, and (f) mutual trust and respect. It is a significant move away from technical skills as being the only important skills for any team member using a prescriptive methodology.

The pros and cons of agile methodologies are hotly debated and some critics of agile methodologies argue that agile concepts are simply the adoption of good practices from different prescriptive methodologies [36]. However Beck argues that this is a positive aspect, that agile methods are unique in identifying and unifying these practices [37]. However other pitfalls have been identified, such as; effective customer involvement is not possible, individuals have different personalities which can often be a barrier for forming an effective team, prioritizing requirements is difficult for large systems with many stakeholders, and simplicity requires extra work, which is difficult to carry out when working under pressure to meet deadlines [6].

4.0 Agile Practices and Quality Assurance

In this section we explore the relationship between agile practices and Quality Assurance (QA). There are a number of agile practices that to varying degrees impact on quality but it is in design and implementation that we see the most striking examples of a quality focus. Quality in design is achieved by a continuous process of refactoring that works against the traditional notion of software entropy. Whereas in the past, constant changes to software might cause it to 'decay', constant refactorings during development and maintenance actually enable the design to improve [38].

Unlike some earlier approaches to developing software quickly and flexibly such as Rapid Application Development, agile approaches emphasize quality of design as the essential prerequisite to maintaining agility [39]. Fundamental to the ability to refactor is the role of testing in an agile development process. Indeed an emphasis on testing might be regarded as the most important feature of the agile approach, perhaps most clearly in Extreme Programming [37]. Testing in an agile process covers a number of important practices, including unit testing, a test-first approach to coding, tests written by users and continuous integration (which assumes a regression testing cycle). The central role of testing in the support of design quality through refactoring means that quality is built into the process of development rather then being merely a supporting concept. Research suggests that such an approach not only improves the quality of code but may be the most important quality related practice of agile methods, since test driven development is essentially focused on quality of service. [40]

If agile practices, particularly those that relate to testing, are intrinsically quality assured, this changes the role of QA in organisations that undertake agile development. For QA professionals to be relevant within this type of environment their working practices need to integrate with agile development. This means working closely with other team members, understanding that agile development is an evolutionary process and developing a range of skills that is wider than those of 'traditional' QA [41]. Such integration of QA into the agile development lifecycle is not only possible but fully compatible with widely used quality focused processes such as Software Quality Assurance (SQA) and Verification and Validation (V&V) [42].

Some agile practices integrate both development and QA activities, meaning that some QA work will be done by developers. Examples of such practices include integrated code inspection through pair programming, refactoring, collective code ownership and coding standards. In fact the main difference between the QA role in traditional and agile methods tends to be in the balance between static and dynamic techniques. In traditional 'high ceremony' methods, QA is focused on the review of static materials such as design documents and 'completed' code. Agile practices such as continuous integration are more dynamic QA techniques.

In summary, we might conclude that QA practices may occur earlier and more often in an agile rather than in a more traditional approach, but that they are equally, if not more, relevant.

5.0 Hybrid approaches to Agile Software Engineering

If we are convinced by the previous arguments that moving to an agile approach does not necessarily compromise quality, we might usefully investigate why certain types of organisation and development team may nevertheless not be prepared to adopt a fully agile approach. Perhaps the key issue is the philosophy of agile methods, whereby we replace the traditional all encompassing methodology with a more flexible and less prescriptive approach. It may be that while there is an acceptance that agile methods can be useful in certain types of organisation or software project, that they are not sufficient for every case. We can see, for example, that agile methods provide a generative rule set which is a minimum set of things that apply to all projects, whilst being aware that organizations themselves are complex and adaptive systems, where individuals are self organising and results may be innovative and emergent [43]. As a consequence of this, some organisations may perceive agile methods to be inadequate in both detail and coverage to provide a fully viable methodology. In an industry where significant investment has been made in engineering approaches such as ISO 9000, the Capability Maturity Model (CMM), and a number of industry-led initiatives such as the Unified Modeling Language and the Unified Process, there is sure to be a reluctance to accept wholesale change.

One problem that has reinforced this perception is that there are many different methodologies purporting to be ‘agile’, so that there seems to be an emphasis on the quantity of different methods rather than the quality of the methodology. Different methods tend to focus on different aspects of the software development lifecycle, meaning that a single method may not provide sufficient support for all aspect of the lifecycle. Further, many methods seem to include a large number of abstract principles rather than concrete guidance. Indeed, one analysis showed that five out of nine agile software development methods that were analysed emphasised abstract principles rather than concrete guidance [19]. This issue is complicated further where abstract principles are not clearly understood. The concept of metaphor, for example, has proved both confusing in principle and of little utility in practice [44]. Faced with methods that in some cases could be regarded as replacing rigor with smoke and mirrors, organisations might feel justified in fearing that replacing traditional prescriptive methods with much less formal ones might compromise quality. Because of this, we have seen a number of examples of agile practices being used in a complementary manner with more formal processes.

5.1 Hybrid Approaches in Practice

Manhart and Schneider [45] report on an organisation where the agile principles of unit testing and test-first development were integrated into an existing formalised process. The organisation did not feel that agile methods could be adopted indiscriminately, but rather that certain principles could be adapted to deal with specific issues related to developing embedded systems. Although the change to a test-first approach was somewhat radical from the programmer perspective,

changing from an implement-document-test mindset to a test-implement-document mindset, it was felt to be beneficial in terms of software quality.

Such hybrid approaches underline that fact that the broad goals of engineering and agile development are already the same: to develop the required software at an appropriate cost and meet the customer's quality requirements. Thus there is no conflict of interest between agile and more formal approaches, rather a change of emphasis. For example, quality certification assumes that process quality will equate to product quality, but the agile perspective puts more emphasis on product quality, though reflective practice rather than a restrictive process. Traditional quality processes are also not entirely incompatible with an agile approach. For example, Capability Maturity Model Integration (CMMI) has many process areas and practices that are compatible with agile practices [46]. Lycett et al [47] suggest a framework based on activities, artefacts and patterns that embraces agile practices while still providing an audit trail. Some agile methods also stress certain artefacts and activities as a minimum level of ceremony, such as ICONIX [48]

A further hybrid approach is described by Armitage [49], which overlays an agile process with higher level design approaches. The intent here is to avoid the potential fragmentation of the overall design by developing low fidelity redesigns of existing builds to assist in refactoring. In addition, a further 'project vision' level of design is suggested. This hybrid approaches attempts to balance the empirical approach of high fidelity low level components with low fidelity, high level components, to provide the overall vision of the product. Again, design quality is seen as the key driver.

In all of the approaches that attempt to balance agile practices with more traditional engineering approaches, there is an attempt to get the benefits of both discipline and agility. The benefits can be that none of the existing value of the current methodology is lost, but that new value may be added. This may include the development of individual skills, such as the ability to plan through the revising of decisions in iterations and refactorings [50]

6.0 Components and the Economics of Quality

In our previous discussion we have reviewed a range of research material that suggests that it is possible to integrate agile practices within a more formal engineering focused software development process to enhance quality. However there is a further argument regarding software quality and agile development that we need to address, which is that agile development alone does not lead to total quality because of its minimalist approach. The economics of agile development assume that once a piece of software meets its requirements within the current development that it is good enough. Further enhancement of the software is therefore economically unjustifiable, in other words, total quality is economically bad. In contrast, in the development of software components that are designed for reuse, perfectionism is economically good, thus the concept of 'trusted components' [51]. For this argument we might infer that to enhance software

quality as much as possible we should adopt features from both agile development and Component Based Software Development (CBSD). At first glance it may appear that the low ceremony agile approach and the high ceremony component engineering approaches are incompatible. However it is also true that agile development is very appropriate for the development of discrete high fidelity software parts, which is one way of defining a component. Again, therefore, we can see the potential for a hybrid approach that integrates features of component development and agile practice [52].

For such an approach to be successful, there needs to be an acknowledgment that certain parts of our software development process will be more high ceremony than others, that we may use a more structured and formal process to develop components that we do the develop the systems that reuse them. [53]. By combining these approaches we can develop product line frameworks that provide multiple applications from a common framework and components

7.0 Test Related Practices as the Key to Quality Assurance

In all of the hybrid approaches we have discussed in this papers. The most important features that bridge the different development philosophies are quality assurance practices embedded in testing related activities. Thus in any hybrid development approach the constant factor should be a comprehensive testing framework and set of practices. In turn, a robust testing strategy enables safe refactoring to ensure design quality. Pair programming in both component and product areas ensures a further quality assurance role for developers. Figure 2, adapted from Wills [53], shows how these test related quality assurance strategies can be applied in an integrated component based and agile development environment. Here, a higher ceremony approach is used to build components, including rigorous testing and a search for perfection, justified by the economics of reuse. A lower ceremony agile approach is used to build products, which are built to meet customer requirements. The integration of components and products is achieved by the use of product lines, families of products that enable the reuse of components between multiple products. Unifying these hybrid approaches is an integrated test framework that enables the quality assurance activities of refactoring and pair programming to be effective.

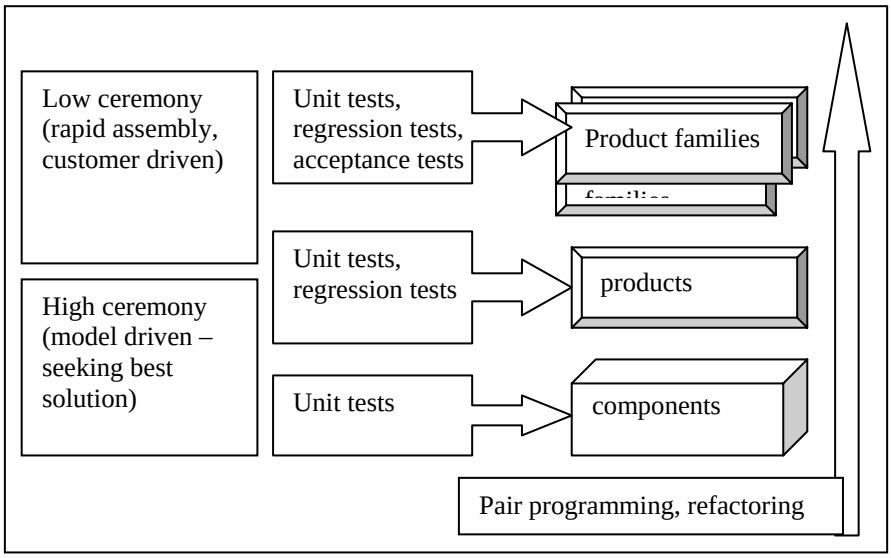

Figure 2: Test-related quality assurance strategies in an integrated component-based and agile development environment

8.0 Conclusions and Further Work

In this paper we have explored both agile development methods and their potential use with hybrid development approaches alongside more prescriptive, high ceremony methods. Previous research indicates that test based practices are perhaps the most easily accepted and immediacy useful of agile practises that can be integrated into existing formal approaches. Other authors have suggested that a combination of agile and component based software development may be good way to overcome the economic constraints on quality assurance in an agile development environment. In this paper we have further proposed that test related practices are the most significant enabler in providing quality assurance in hybrid systems.

Research into combined component based and agile development has so far been limited. Field studies are required to explore the issues raised in this paper regarding the relationship between quality and the economics of agile development, and the pivotal role of testing in improving project success and overall software quality.

9.0 References

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

[2] Keil M & Robey D (2001). Blowing the whistle on troubled software projects, *Communications of the ACM*, 44, 2, 87-93.

[3] Mahaney R C & Lederer A L, Runaway information systems projects and escalating commitment, proceedings of ACM SIGCPR conference on computer personnel research, pp291 - 296, New Orleans, United States, 1999

[4] US Government Accounting Office, *Report FGMSD-80-4: Contracting for Computer Software development: Serious problems require management attention to avoid wasting additional millions.* 1979, US Government Accounting Office.

[5] *Extreme CHAOS*, Standish Group, Standish Group International Inc., http://www.standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf, (visited January 2006)

[6] Sommerville I, *Software Engineering (7th edition)*. Addison-Wesley 2004, ISBN 0-321-21026-3

[7] Whyte G & Bytherway A (1996). Factors affecting information systems' success, *International Journal of Service Industry Management*, 7, 1, 74-93.

[8] Avison D & Fitzgerald G, *Information systems development: methodologies, techniques and tools (2nd ed.)*. McGraw-Hill 1995, ISBN 0-632-01644-2

[9] de Abreu A & Conrath D, The role of stakeholders' expectations in predicting information systems implementation outcomes., proceedings of Conference on Computer Personnel Research, pp408-415, St Louis, Missouri, United States, 1993

[10] Ewusi-Mensah K (1997). Critical Issues in Abandoned Information Systems Development Projects, *Communications of the ACM*, 40, 9, 74-80.

[11] Dhillon G, Interpreting Key Issues in IS/IT Benefits Management., proceedings of 33rd Hawaii International Conference on System Science, pp1-9, Hawaii, 2000

[12] Ewusi-Mensah K, *Software Development Failures*. MIT Press 2003, ISBN 0-262-05072-2

[13] Jayaratna N, *Understanding and evaluating methodologies NIMSAD: A systemic framework*. McGraw-Hill 1994,

[14] Avison D & Fitzgerald G (2003). Where Now for Development?, *Communications of the ACM*, 46, 1, 79-82.

[15] Taylor H (2001). Information Systems Development Practice in New Zealand, *New Zealand Journal of Applied Computing and Information Technology*, 5, 2, 80-84.

[16] Baird S, *Teach Yourself Extreme Programming in 24 Hours*. Sams Publishing 2003, ISBN 0-672-32441-5

[17] Highsmith J, *Agile Software Development Ecosystem.* Addison-Wesley 2002, ISBN 0-201-76043-6

[18] Koch A S, *Agile Software Development: Evaluating The Methods For Your Organization*. Artech house 2005, ISBN 1-5805-3842-8

[19] Abrahamsson P, Warsta J, Siponen M,Ronkainen J, New Directions on Agile Methods: A Comparative Analysis, proceedings of 25th International Conference on Software Engineering, pp244 - 254, Portland,

Oregon, 2003
[20] Lewis P J, *Information Systems Development: Systems Thinking in the Field of IS*. Pitman 1994, ISBN 0-273-03107-4
[21] *The New Methodology*, Fowler M, MartinFowler.com, http://www.martinfowler/articles/newMethodology.html, (visited January 2006)
[22] Martin R C, *Agile Software Development: Principles, Patterns and Practices*. Prentice Hall 2003, ISBN 0-135-97444-5
[23] Pressman S, *Software Engineering: A Practitioner's Approach (6th edition)*. McGraw-Hill 2005, 0-07-285318-2
[24] Ceschi M, Sillitti A, Succi G,Panfilis S D (2005). Project Management in Plan-Based and Agile Companies, *IEEE Software*, 22, 3, 21-27.
[25] Patton J, Hitting The Target: Adding Interaction Design to Agile Software Development, proceedings of Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA), pp1 – ff, Seattle, Washington, 2002
[26] Schatz B & Abdelshafi I (2005). Primavera Gets Agile: A Successful Transition to Agile Development, *IEEE Software*, 22, 3, 36-42.
[27] Johnson J, *Turning Chaos into Success*, in *Software Magazine*. 1999. p.30-39.
[28] Little T (2005). Context-Adaptive Agility: Managing Complexity and Uncertainty, *IEEE Software*, 22, 3, 28-35.
[29] Grossman F, Bergin J, Leip D, Merritt S,Gotel O, One XP Experience: Introducing Agile (XP) Software Development Into a Culture That is Willing But Not Ready., proceedings of 2004 Conference of the Centre for Advanced Studies on Collaborative Research, pp242-254, Markham, Ontario, Canada, 2004
[30] Cockburn A, *Agile Software Development*. Pearson Education, Inc 2001, ISBN 0-201-69969-9
[31] *Caterpillar Digs into Agile Development*, DePauw T, http://www.computerworld.com/printthis/2002/0,4814,67147,00.html, (visited January 2006)
[32] *Users Warm Up To Agile Programming*, Sliwa C, ComputerWorld, http://www.computerworld.com/printthis/2002/0,4814,69182,00.html, (visited January 2006)
[33] Lindstrom L & Jeffries R (2004). Extreme Programming and Agile Software Development Methodologies, *Information Systems Management*, 21, 3, 41-52.
[34] Thomas D (2005). Agile Programming: Design to Accommodate Change, *IEEE Software*, 22, 3, 14-16.
[35] Cockburn A & Highsmith J (2001). Agile Software Development: The People Factor, *IEEE Computer*, 34, 11, 131-133.
[36] Stephens M & Rosenberg D, *Extreme Programming Refactored: The Case Against XP*. Apress 2003, ISBN 1-59-059096-1
[37] Beck K, *Extreme Programming Explained: Embrace Change*. Addison-Wesley 1999, ISBN 0-201-61641-6
[38] Fowler M, *Refactoring: Improving the Design of Existing Code*. Addison-

Wesley 1999,
[39] Fowler M & Highsmith J (2001). The Agile Manifesto, *Software Development*.
[40] Melnik G & Maurer F, Introducing Agile Methods: Three Years of Experience, proceedings of 30th EUROMICRO Conference (EUROMICRO'04), 2004
[41] Ambler S, *The Object Primer: Agile Model-Driven Development with UML 2.0 (3rd edition)*. Cambridge University Press 2004, 0-521-54018-6
[42] Huo M, Verner J, Zhu L,Babar M A, Software Quality and Agile Methods, proceedings of 28th Annual International Computer Software and Applications Conference (COMPSAC'04), 2004
[43] Highsmith J & Cockburn A (2001). Agile Software Development: The Business of Innovation, *IEEE Computer*, 34, 9, 120-122.
[44] Tomayko J & Herbsleb J, *How Useful Is the Metaphor Component of Agile Methods? A Preliminary Study*. 2003, School of Computer Science, Carnegie Mellon.
[45] Manhart P & Schneider K, Breaking the Ice for Agile Development of Embedded Software: An Industry Experience Report, proceedings of 26th International Conference on Software Engineering (ICSE'04), pp378-386, Edinburgh, Scotland, 2004
[46] Turner R & Jain A, Agile Meets CMMI: Culture Clash or Common Cause?, proceedings of Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile Universe 2002, pp153-165, Chicago, IL, USA, 2002
[47] Lycett M, Macredie R, Patel C,Paul R (2003). Migrating Agile Methods to Standardized Development Practice, *IEEE Computer*, 36, 6, 79-85.
[48] Rosenberg D, Stephens M,Collins-Cope M, *Agile Development with ICONIX Process*. Apress 2005, ISBN 1-59059-464-9
[49] Armitage J (2004). Are Agile Methods Good for Design?, *Interactions*, 11, 1, 14-23.
[50] DeMarco T & Boehm B (2002). The Agile Methods Fray, *IEEE Computer*, 35, 6, 90-92.
[51] Meyer B, The Grand Challenge of Trusted Components, proceedings of 25th International Conference on Software Engineering, pp660 - 667, Portland, Oregon, 2003
[52] Radinger W & Goeschka K M, Agile Software Development for Component Based Software Engineering, proceedings of Conference on Object Oriented Programming Systems Languages and Applications, pp300 - 301, Anaheim, CA, USA, 2003
[53] *Agile Components: Scaling XP*, Wills A C, Trireme International Ltd, http://www.trireme.com/whitepapers/process/xp-uml/agilePLA.pdf, (visited January 2006)