Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 81 (2008) 1899–1919
The Journal of Systems and Software
www.elsevier.com/locate/jss

A framework to support the evaluation, adoption and improvement of agile methods in practice

A. Qumer, B. Henderson-Sellers *

Faculty of Information Technology, University of Technology, Sydney, P.O. Box 123, Broadway 2007, Australia

Received 7 May 2007; received in revised form 12 December 2007; accepted 27 December 2007
Available online 12 January 2008

Abstract

Agile methods are often seen as providing ways to avoid overheads typically perceived as being imposed by traditional software development environments. However, few organizations are psychologically or technically able to take on an agile approach rapidly and effectively. Here, we describe a number of approaches to assist in such a transition. The Agile Software Solution Framework (ASSF) provides an overall context for the exploration of agile methods, knowledge and governance and contains an Agile Toolkit for quantifying part of the agile process. These link to the business aspects of software development so that the business value and agile process are well aligned. Finally, we describe how these theories are applied in practice with two industry case studies using the Agile Adoption and Improvement Model (AAIM).
© 2008 Elsevier Inc. All rights reserved.

Keywords: Agile methodologies; Framework; Transition to agile; Industry case studies

1. Introduction

Agile methods are often welcomed by both managers and programmers as providing a much needed release from the overheads typically perceived as being imposed by traditional software development approaches. Created in the context of small, greenfield projects, agile methods are often seen as unable to scale to larger situations (Greenfield and Short, 2004, page 123). Their adoption seems to need an all-or-nothing approach, suggesting that "being agile" is binary.

In practice, few organizations are able, psychologically or technically, to take on agile development approaches immediately and adopt them successfully over a short period – a full transition often taking a few years. Furthermore, it may be inappropriate for them to be fully agile in all aspects of development, perhaps retaining well-known and trusted elements of a more traditional approach within an overall agile project. One way to do this is by the use of situational method engineering (Henderson-Sellers, 2002, 2003). But even then, the method engineer and the software development manager may be unsure how to identify how to adopt agile methods incrementally, which bits to choose as most appropriate for their situation, how to engender enthusiasm in team members (Syed-Abdullah et al., 2007), how to ensure that their adopted method can mature and grow as the development team's skills mature and how to ensure that the whole of the development team don't succumb to the inherent desire of humankind to "resist change" (Henderson-Sellers and Serour, 2005).

In this paper, we introduce a complete framework to assist managers in assessing the degree of agility they require and how to identify appropriate ways to introduce this agility into their organization, illustrated with some industry case studies. Section 2 describes the Agile Software Solution Framework (ASSF). A major element of the ASSF is the Agile Toolkit, which is discussed in Section 3. A second, and as yet unexplored, element of the ASSF is governance – the topic of Section 4. Section 5 discusses issues relating to the adoption of an agile process in

* Corresponding author. Tel.: +61 2 9514 1687; fax: +61 2 9514 4533.
E-mail address: brian@it.uts.edu.au (B. Henderson-Sellers).

0164-1212/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2007.12.806

industry situations and how the ASSF can help, linking the approach to software process improvement concepts. In Section 6, we illustrate how the ideas propounded in Section 5 regarding the Agile Adoption and Improvement Model (AAIM) are enacted on two real industry case studies; before concluding in Section 7.

2. Agile software solution framework

With the above questions in mind, we have developed an Agile Software Solution Framework (ASSF). Fig. 1, at a very abstract level, represents the components of the ASSF and their relationships. The elements of the ASSF can be classified in terms of agile conceptual aspect model (agile as a characterizing-noun) and tools. The agile conceptual aspect model represents the aspects of knowledge, governance and method core; these three being linked to business via a business-agile alignment bridge or business value. This bridge has an impact on governance, which in turn shapes an agile software development method (construction and application), in terms of the business value it delivers. Business represents the software development organization. The remaining 'abstraction' element of the ASSF represents the type of abstraction in use (e.g. object, agent, service) or the associated software technology. The Agile Toolkit (Qumer and Henderson-Sellers, 2007c) provides an application of these ideas in practice (construction or tailoring of software processes), whereas the embedded analytical tool (4-DAT; four-dimensional analysis tool) is specially used as a quality evaluation measure to evaluate the degree of agility in software development practices. The 4-DAT (as a quality evaluation measure) has already been tested, documented and published, and the details can be found in Qumer and Henderson-Sellers (2007a).

Furthermore, in Fig. 1, Method Core represents the different but related core aspects of an agile method (a.k.a. methodology): agility, people, process, product and tools (agile workspace), which can be combined by using a method engineering approach for the construction of agile situation-specific methods to achieve the desired business value. The purpose of the ASSF (a vision-guiding model) is to guide the behaviour of self-organizing and empowered agile teams with a cohesive set of shared information in large and complex project development environments. The ASSF is also a first step towards the future development of a meta-model for agile software development methodologies – although there are a number of meta-models (OPF: Firesmith and Henderson-Sellers, 2002; Standards Australia, 2004; ISO/IEC, 2007) for traditional approaches to software development, they do not explicitly discuss the new aspect of agility, which is an essential aspect for an agile software development method. According to our survey and focus groups, it is also found that existing standards do not explicitly discuss the aspect of abstraction either – a critical aspect of any software development methodology (Qumer and Henderson-Sellers, 2006d). The following sub-sections precisely describe the main parts of the ASSF, two of which (the toolkit and governance), being the least well documented in the literature, are described in more detail in Sections 3 and 4.

2.1. Method core

The absence of a shared or common vision is one of the main factors of software project failures. As noted above, the "method core" and abstraction elements of ASSF (Fig. 1) represent six aspects of an agile software development methodology: agility, process, people, product, tools and abstraction. This set of aspects attempts to provide a

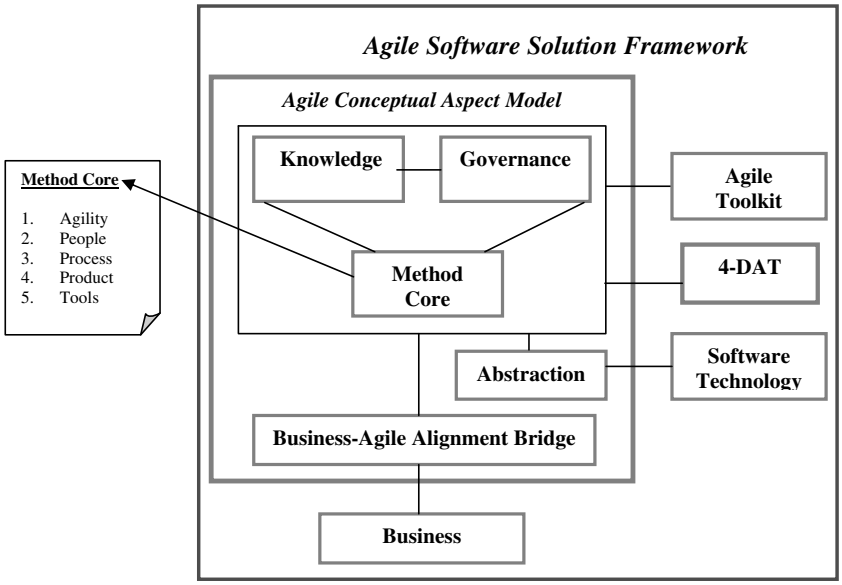

Fig. 1. The main components of the agile software solution framework.

vision-guiding or mental-model for an agile methodology. To date, the abstraction and agility aspects of a software development methodology have not been a factor in the choice of software development processes (Qumer and Henderson-Sellers, 2006c); we propose that they should be. These six method core "cells" provides the core elements of a software development methodology. These cells can be used to link a situational method engineering approach, a feedback mechanism and a meta-model such as ISO/IEC (2007) to produce a software development methodology/process(es).

2.2. Knowledge

The nature of software development has changed with the industrial economy transitioning to a knowledge economy (Augustine, 2005). Agile software development is a knowledge-intensive process, where knowledge is created and shared, when different aspects of a methodology (concepts, products, tools, process, people etc.) interact with each other. For example, people cooperate and communicate in order to create and share knowledge in a software development organization. The Knowledge Cell is used to engineer and manage the knowledge related to agile software development (for example, knowledge regarding different process fragments and processes, such as agile process fragments, agent process fragments, object process fragments, service-oriented process fragments etc.). According to Rus and Lindvall (2002), as well as the feedback from our own industrial survey, most software development-related knowledge is tacit and resides in the brains of people (McBride, 2006). Therefore, we argue here that an agile knowledge engineering and management approach (Auer and Herre, 2006) should be integrated with an agile software development approach, in order to capture and manage related knowledge and to use it for performance improvement, learning and decision making in an agile software development environment.

2.3. Agile Toolkit

The Agile Toolkit (discussed fully in Section 3) facilitates the construction and evaluation of multi-abstraction or m-abstraction (mix of different abstraction-mechanisms: object-oriented, agent-oriented, service-oriented etc.) situation-specific process fragments or processes for the development of complex software development projects that may involve more than one abstraction mechanisms.

2.4. A 4-Dimensional analytical tool

Considering a test-first approach, a 4-Dimensional Analytical Tool (4-DAT) has been developed, tested, published (Qumer and Henderson-Sellers, 2006a,b, 2007a) and then included in the ASSF for the assessment and analysis of the constructed or to-be constructed process. 4-DAT will facilitate the examination of agile methods from four perspectives or dimensions: method scope characterization, agility characterization, agile values (agile manifesto) characterization and software process characterization. Although there are currently four dimensions evaluated in the 4-DAT approach, it is in fact extensible – we can add or remove dimensions or items from the dimensions of 4-DAT, if found necessary in the future.

2.5. Governance

Governance or, more specifically, IT governance (Weill and Ross, 2004; Webb et al., 2006) provides a mechanism for a strategic IT-business alignment to acquire maximum business value delivered by the consumption of IT resources, but it has not been discussed in the context of agile software development organizations.

We therefore developed a model for responsibility, accountability and business value governance in the context of agile development (Qumer, 2007) that introduces sufficient control, discipline and rationale to scale up agile software development methods for large and complex projects (Fig. 2). Although governance sounds bureaucratic, we nevertheless need to integrate governance practices or models with agile software development approaches. Indeed, it has been reported that a better governance (IT) mechanism can earn a 20% higher return on assets than an organization with an average governance (Weill and Ross, 2004). Based on the novelty of this cell, we devote Section 4 to a fuller description of governance appropriate to agile development.

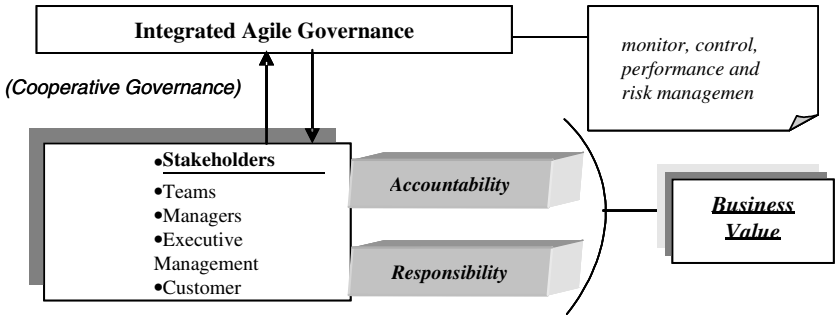

Fig. 2. Agile responsibility, accountability and business value governance model (after Qumer, 2007).

2.6. Business value and agility

Business value (Elssamadisy, 2006) and agile alignment (business-agile) is another factor of choice that has not been investigated and highlighted to any great extent by the agile community (Barnett, 2007). Here, we argue that it has a great impact on the decision to adopt or discard any agile practice or set of agile practices. The value to a business of using an agile approach can be categorized as a business value delivered through agile software development as well as a business value delivered through appropriate governance. A cost-benefit analysis metric would be helpful for the appraisal of business value delivered through working in an agile development environment.

The business-agile alignment bridge is an issue that has not been investigated to any great extent by the agile community. Here, we propose that it should be, because it has an impact on both the construction and application of agile methods in terms of the business value delivered (Elssamadisy, 2006) to customer, team, process, workspace and product (Fig. 3). This bridge aligns business goals and agile software development goals.

2.7. Business

Business, here, refers to a software development organization. The policies, strategies, business goals (at a higher level) and the existing culture of a software development organization have a potentially large impact on the adoption, construction, execution and governance of any agile software development method. We cannot ignore this fact and, therefore, it is included in ASSF.

2.8. Agile method choreography

The six software solutions cells are used and combined to construct an agile software development method/methodology by using a situational method engineering (SME) approach (e.g. Kumar and Welke, 1992) for a specific organizational context. ASSF includes templates and models ("Shared Guiding Vision") to keep the development and teams aligned and directed with a shared mental model, which is clearly a distinguishing feature of ASSF in comparison with traditional agile methods. According to Augustine (2005), the guiding vision is created and refined through release planning in each iteration. The agile software development method created with the help of the ASSF will contain aspects of governance and knowledge that have not previously been the focus of traditional agile methods. In summary:

$$\text{AgileMethod} = f(\text{agility, abstraction, people, processes, product, tools, knowledge, governance}) \quad (1)$$

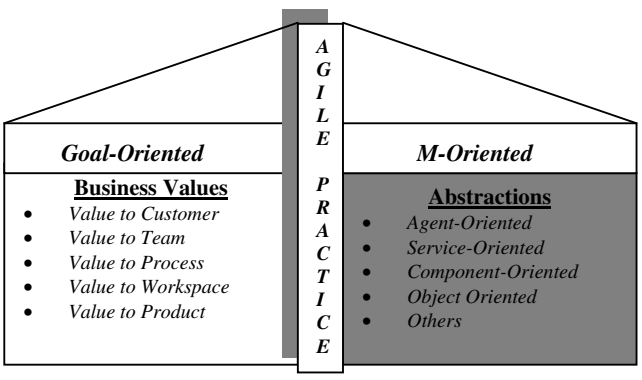

Fig. 3. The two-dimensional view of an agile practice (after Qumer and Henderson-Sellers, 2007c).

2.9. Validation of a framework

A number of assessment and validation methods have been applied to carry out the tests on this framework. In order to check the ability of ASSF to represent the domain of interest, we first conducted content and construct validity tests and created an agile process with this framework and, furthermore, we tested the created agile process with our 4-dimensional analytical tool (Qumer and Henderson-Sellers, 2006a). We also used the opinions of domain experts, administered individual interviews, a focus group and two industrial case studies to empirically validate this framework (Qumer, 2007; Qumer and Henderson-Sellers, 2007c). We selected two software development organizations (one medium-sized and one large-sized) for study (industry-based empirical work). These software development companies mainly provide software solutions (from medium to large size projects) in the areas of e-commerce systems, embedded systems, network solutions, print solutions and multimedia production. The software development organizations showed a strong interest in the adoption of agile methods. We conducted workshops on the framework (in software development organizations) for the purpose of understanding and initial feedback, and then, together with personnel from the organization, constructed agile processes for the organizations by using this agile software solution framework. We used ASSF for the construction of new agile processes for a service-oriented e-health project case study (empirical testing) for Exclusive Logic Design (coded name) and also constructed a new agile product enhancement process for Printing and Streaming Technologies (coded name).

Yin (1994, 2003) defines a case study as "an empirical inquiry that investigates a contemporary phenomena within its real-life context, especially when the boundaries between phenomena and context are not clearly evident". Gummesson (2000) and Yin (2003) come up with two and three overlapped types, respectively, giving an overall group of five types of case study research methodology; such as theory generation, initiation of change, exploratory, descriptive and explanatory to give a holistic view of the process in this context. A case study could be interest

from two different perspectives: firstly, it involves the derivation of general conclusions from a limited number of cases, whereas, secondly, a single case may be used as a basis for the building of a specific conclusion(s) (Gummesson, 2000).

However, there are still several concerns about our two empirical studies. There were only two empirical case studies in this research (because of the time constraint and other limitations); this is of course not sufficient for drawing broadly appropriate conclusions. Therefore, we can only claim that the findings of the case studies are local to the case study organisations and cannot be generalised; yet at the same time providing a datum that can be incorporated into future studies (either by us or by other researchers). The results of the case studies (Qumer, 2007; Qumer and Henderson-Sellers, 2007c) are summarized here in Section 6.

3. Agile Toolkit

An important component of the ASSF is the Agile Toolkit (Fig. 4). The Agile Toolkit described here encapsulates seven main components: (1) knowledge-base, (2) process fragment and process composer, (3) publisher, (4) registry, (5) agility calculator, (6) knowledge-transformer and (7) visualizer. The knowledge-base provides the basic components (agile, abstraction and business value) for the construction of agile process fragments and an agile process. Within this knowledge base, an agile component contains knowledge regarding agility (Qumer and Henderson-Sellers, 2006c), agile values and principles (Agile Manifesto, 2001; Qumer and Henderson-Sellers, 2006d) and agile practices (software development, governance and knowledge management for agile approaches). Abstraction refers to knowledge regarding various abstraction mechanisms (object, agent, service etc.). The business value component contains knowledge related to possible business values (value to customer, product, team, workspace etc.) that could be expected from an agile process fragment or an agile process.

A process fragment represents an individual agile or non-agile practice. A constructed agile process contains a collection of agile process fragments i.e. a composition of agile practices by using a software development meta-model (for example, those of the OPF: Firesmith and Henderson-Sellers, 2002; Standards Australia, 2004; ISO/IEC, 2007). The services for this composition of process fragments and processes are provided by the composer. The agility calculator (Qumer and Henderson-Sellers, 2006a) provides the services for the calculation of the degree of agility of either a composed process fragment or process. Publisher provides the services to transform the composed process fragments and process into an XML format and then exports that to the registry. The registry (a shared resource) contains process fragments and processes (with the description of process fragments), which are made

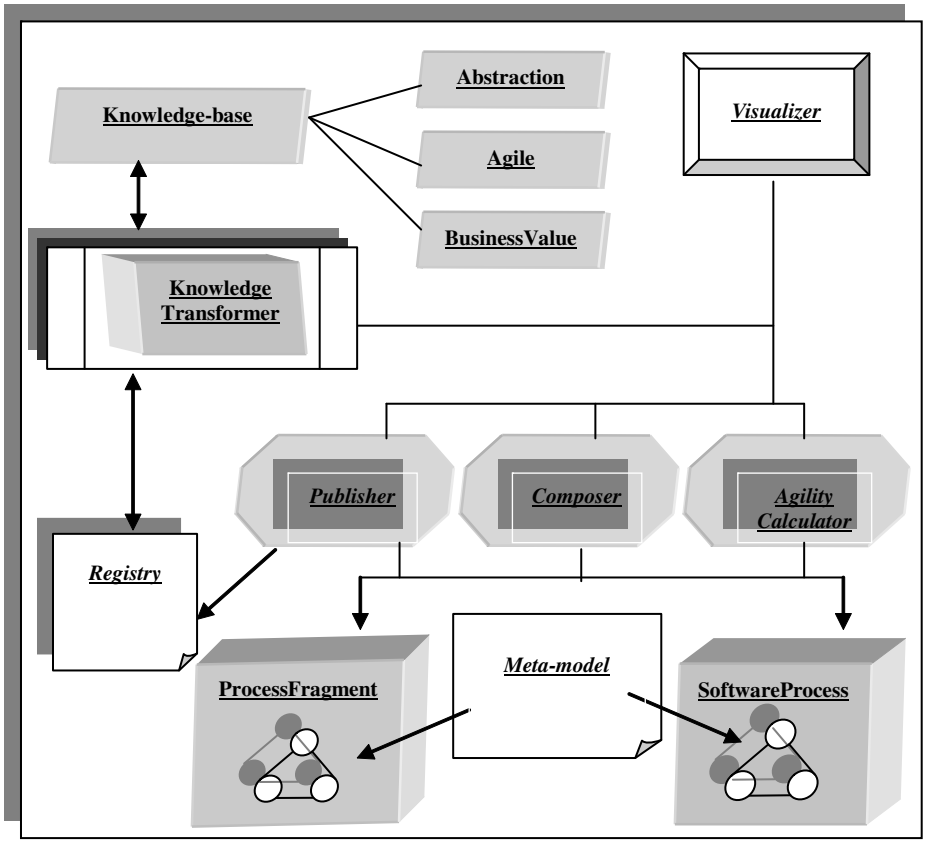

Fig. 4. Components of the Agile Toolkit (after Qumer and Henderson-Sellers, 2007c).

accessible to developers. The registry exports and imports process fragments to/from the knowledge-transformer. The knowledge-transformer transforms the related knowledge (process fragments etc.) to a useable format between the knowledge-base and other Agile Toolkit components. Finally, the visualizer provides an interactive interface to the users of the toolkit.

Perhaps the most important element of the ASSF/Agile-Toolkit is the Agility Calculator known as the 4-DAT (four-dimensional analysis tool) (Qumer and Henderson-Sellers, 2006a,b), which crystallizes and measures the key attributes of agility: flexibility, speed, leanness, learning and responsiveness. The approach of 4-DAT is partly qualitative and partly quantitative. However, since the tool is generic, it may be used to analyze and quantitatively measure the agility (dimension 2 of 4-DAT) of any other software development methods or approaches including governance, knowledge engineering and management (Qumer and Henderson-Sellers, 2007a). 4-DAT examines software methods from four perspectives or "dimensions" (see Table 1 for details): method scope, agility characterization (with its 5 key attributes), characterization of agile values (based on those proposed in the Agile Manifesto, 2001) and software process characterization (Jalote, 1997). A report generated with the help of 4-DAT will assist organizations in making decisions about the selection or adoption of an agile method or method fragments.

The only currently quantified dimension of the 4-DAT is the second (Table 1), which permits a numerical evaluation of the degree of agility at both the large scale (called here process level) and the small scale (called here the method practices level). To calculate agility values for this dimension, a table is constructed from a supplied template in which cell values of 0 or 1 are entered for each phase (for a high level assessment) or, at a more detailed level, for each individual practice (Table 2). The five agility features of flexibility (FY), speed (SD), leanness (LS), learning (LG) and responsiveness (RS) for each method fragment are considered and then the overall method total (and hence average degree of agility) can be calculated. Some results of typical agile methods are compared in Fig. 5 with each other and with two traditional, non-agile methods. Based on such quantitative evaluations, we can offer as a ballpark figure a threshold value of around 0.5–0.6 for any constructed agile method to have sufficient measured agility to qualify for *consideration* as an agile method.

Table 1
4-DAT's four dimensions (after Qumer and Henderson-Sellers, 2006a)

Dimension 1 (method scope)	
Scope	Description
1. Project size	Does the method specify support for small, medium or large projects (business or other)?
2. Team size	Does the method support for small or large teams (single or multiple teams)?
3. Development style	Which development style (iterative, rapid) does the method cover?
4. Code style	Does the method specify code style (simple or complex)?
5. Technology environment	Which technology environment (tools, compilers) does the method specify?
6. Physical environment	Which physical environment (co-located or distributed) does the method specify?
7. Business culture	What type of business culture (collaborative, cooperative or non-collaborative) does the method specify?
8. Abstraction mechanism	Does the method specify an abstraction mechanism (object-oriented, agent-oriented)?
Dimension 2 (Agility Characterization)	
Features	Description
1. Flexibility	Does the method accommodate expected or unexpected changes?
2. Speed	Does the method produce results quickly?
3. Leanness	Does the method follow the shortest time span, use economical, simple and quality instruments for production?
4. Learning	Does the method apply updated prior knowledge and experience to create a learning environment?
5. Responsiveness	Does the method exhibit sensitiveness?
Dimension 3 (Agile Value Characterization)	
Agile values	Description
1. Individuals and interactions over processes and tools	Which practices value people and interaction over processes and tools?
2. Working software over comprehensive documentation	Which practices value working software over comprehensive documentation?
3. Customer collaboration over contract negotiation	Which practices value customer collaboration over contract negotiation?
4. Responding to change over following a plan	Which practices value responding to change over following a plan?
5. Keeping the process agile	Which practices help in keeping the process agile?
6. Keeping the process cost effective	Which practices help in keeping the process cost effective?
Dimension 4 (software process characterization)	
Process	Description
1. Development process	Which practices cover the main life cycle process and testing (quality assurance)?
2. Project management process	Which practices cover the overall management of the project?
3. Software configuration control process/support process	Which practices cover the process that enable configuration management?
4. Process management process	Which practices cover the process that is required to manage the process itself?

4. Governance

Based on a grounded theory research methodology, Qumer (2007) reviewed and analyzed an extensive number of proposed definitions for IT governance. As well as the seven elements thus identified (Table 3), we also reviewed seven proposed IT governance frameworks (Lainhart, 2000; Sisco, 2002; Hammer, 2002; Meijer, 2003; Behr et al., 2004; OGC, 2005; CALDER-MOIR, 2006), summarized in Table 4. However, the analysis suggested that most

Table 2
Schematic table for the calculation of the agility of x individual phases and y practices (a.k.a. techniques) and the overall agility of the constructed method (after Qumer and Henderson-Sellers, 2007a)

	Agility features FY	SD	LS	LG	RS	Total
(i) Phases						
Phase 1	0 or 1	0 or 1	0 or 1	0 or 1	0 or 1	0 – 5
Phase 2	0 or 1	0 or 1	0 or 1	0 or 1	0 or 1	0 – 5
Phase 3	0 or 1	0 or 1	0 or 1	0 or 1	0 or 1	0 – 5
etc.	0 or 1	0 or 1	0 or 1	0 or 1	0 or 1	0 – 5
Total	$(0-x)$	$(0-x)$	$(0-x)$	$(0-x)$	$(0-x)$	$0-5^{*}x$
Degree of agility (high level)	$(0-x)/x$	$(0-x)/x$	$(0-x)/x$	$(0-x)/x$	$(0-x)/x$	Total divided by number of cells in table
(ii) Practices						
Practice 1	0 or 1	0 or 1	0 or 1	0 or 1	0 or 1	0 – 5
Practice 2	0 or 1	0 or 1	0 or 1	0 or 1	0 or 1	0 – 5
etc.	0 or 1	0 or 1	0 or 1	0 or 1	0 or 1	0 – 5
Total	$(0-y)$	$(0-y)$	$(0-y)$	$(0-y)$	$(0-y)$	$0-5^{*}y$
Degree of agility (low level)	$(0-y)/y$	$(0-y)/y$	$(0-y)/y$	$(0-y)/$ y	$(0-y)/y$	Total divided by number of cells in table

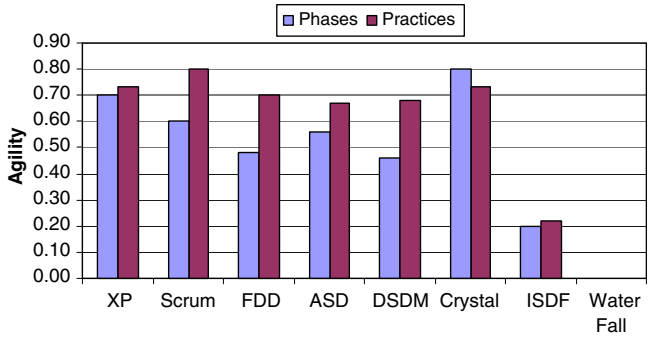

Fig. 5. Comparison of the degree of agility derived from 4-DAT evaluations for the two traditional methods (spiral and waterfall) as compared to values derived earlier for agile methods – this time grouped according to methodology rather than phase/practice (after Qumer and Henderson-Sellers, 2007a).

of these governance frameworks are too bureaucratic and labour-intensive for direct application to agile environments. These frameworks are seldom sufficient to grasp the emergence and dynamic nature, and the needs of agile environments. Therefore, a model of IT governance is required that does not incur unnecessary overhead and brings necessary control and discipline in large agile software development arrangements for optimum results. The main characteristics of governance in the context of IT-enabled organisations are discipline and control, accountability, alignment of business-IT goals, performance management and risk management. The intended fundamental outcome of the governance is to maximize the business value from the investments that have been made in IT-enabled projects or services, whereas the governance with an agility factor is aimed at providing a lightweight governance approach in order to reduce the unnecessary overhead and bureaucracy in comparison with a traditional governance approach. However, this research does not focus on the method of "business value measurement" and, therefore, does not explain or investigate the measurement of business value.

Table 3
IT governance concepts and description

IT Governance Concepts	Description
Leadership and decision making *structure*	Leadership and decision making in IT governance is facilitated by providing an organizational structure. IT governance itself, is not a structure; rather, it uses a structure
Responsibility and *accountability framework*	IT governance involves and is the responsibility of senior executive management. An accountability framework is provided to monitor and performance assessment
Processes supported by structure	IT governance related processes are applied with the help of a supported structure
Maximize Business value- or goals achievement through IT, *Business-IT alignment*	IT resources are used to maximize the business value Business goals and IT goals are aligned and synchronized that support business-IT alignment
Sub-part of a corporate governance	IT governance is more focused on IT whereas corporate governance is more business focused. But IT governance is considered as a part of corporate governance
Risk management	The management of IT related risks but not business related risks
Performance Management	The effective utilization of IT resources to get optimal performance

4.1. Agility and the need for governance

Notwithstanding the implications of current agile software development models, methods, frameworks and theories, as well as the interest they have generated in industry, there are still concerns about the ability of these methods to be used in large and complex projects and development environments. The main reason behind such an inability can be argued to be the lack of governance practices for

Table 4
Categorization of IT governance key concepts (after Qumer, 2007)

IT Governance frameworks	Key concepts of IT governance
Control Objectives for Information and Related Technology (COBIT)	• Performance management • Critical success factors (processes) • Capability maturity models
IT audit	• Technology risks, capacity, security, scalability and stability issues • Human resource (expertise) • Project management, change management, IT policies, procedures, software licenses and performance management
Six sigma	• Measure, control and improve performance • Metrics and measures for staff, product requirements, design and continuous improvement
Application Service Library (ASL)	• Functional, technical and application controls
Information Technology Infrastructure Library (ITIL)	• Disciplined practices for service delivery and service management
Projects IN Control Environments (PRINCE)	• Project organization, management and control practices
The Calder-Moir IT governance framework	• Information technology, risk management, strategy, intellectual property, business design, project management, compliance

agile software development environments since an IT organization's turnover and performance are directly influenced by IT governance practices (Weill and Broadbent, 1998; Weill, 2004). Indeed, it has already been reported that a better governance (IT) mechanism can earn 20% higher return on assets than an organization with average governance (Weill and Ross, 2004).

Governance in an agile environment aims to identify and implement the initiatives to systematically determine the personnel in empowered teams (agile teams) that have the right to make decisions and that have the right to give input to decisions and to hold responsible those personnel for the decisions that they make in order to achieve desired business value – this imperative fact of governance and business value has been clearly recognized by both Pettit (2006c) and Barnett (2007).

It is therefore proposed to integrate results-oriented, lightweight governance practices or models into agile software development methods in order to increase the comfort of large software development organizations, customers and business partners (in the case of sub-contract and outsourcing). Empowered agile teams make decisions or give input to decisions in agile environments that seem fine in small or medium organizations and developments but, in order to scale up an agile process for application to large and complex project developments, an integrated governance approach or model is indispensable.

In order to evaluate our hypothesis that governance can be beneficial in agile approaches as well as traditional ones, we also conducted a focus group, survey and interviews (people from large software organizations) to identify the impact and importance of governance for the successful operation of large agile arrangements. This fact was recognized by 77% (27 + 38 + 12) of the participants who reported (Fig. 6) that the agile approach is productive in small and medium developments but, with the additional element of governance, it could be scaled up for large and complex projects. It was also found that in both business and agile software developments, this strongly impacts the way software development is organized, resourced and managed; how the risks are identified and mitigated; and what targets and measures are set in the context. Hence, to effectively inject governance practices into agile environments, organization-wide leadership and collaboration efforts are required, and not only for executive-level management. Cooperative and collaborative evaluation mechanisms should encourage and support self-assessment and self-improvement of the agile teams (self-organized self-evaluation and empowered). IT governance in emergent organizations (such as agile environments) require governance practices to flow down through the organization in order to embrace these advances (Patel, 2002).

4.2. Defining integrated agile governance

In this section, we define the concept of integrated agile governance in the light of the above analysis, discussion and the concepts of agility, agile methods and agile values (Agile Manifesto, 2001; Qumer and Henderson-Sellers, 2006b,c). We may define Integrated IT Governance (Qumer, 2007) as

"an integrated agile governance involves lightweight, collaborative, communication-oriented, economical and evolving effective accountability framework, controls, processes, structures to maximize agile business value, by the strategic alignment of business-agile goals, performance and risk management".

This proposed definition highlights the five key perspectives of governance for agile processes: lightweight, collaborative, communication-oriented, economical and evolving. The lightweight governance (less labour intensive) factor stresses governance but without unnecessarily burdening agile software development environments. Rather, it will support and help strengthen the roots of an agile mindset.

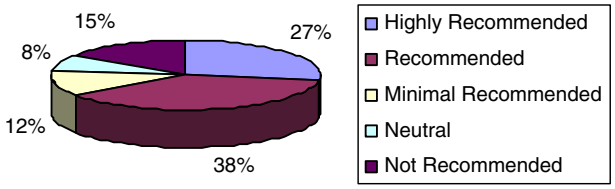

Fig. 6. Importance of IT governance in large agile software development projects (after Qumer, 2007).

Table 5
Categorization of IT governance key concepts

Integrated agile governance elements		
Agile perspectives	Governance toolkit	Agile governance goals and operations
1. Lightweight	1. Framework	1. Control
2. Collaborative	2. Processes	2. Accountability
3. Communication-oriented	3. Structures	3. Maximize business value
4. Economical		4. Strategic alignment of Business and agile goals
5. Evolving		5. Performance and risk management

It has been reported that most of the available IT governance frameworks, such as COBIT and ITIL, are too detailed for effective adoption and seem to be labour intensive, being criticized by many users (Dahlberg and Kivijärv, 2006). Governance should be designed and communicated across the organization in a collaborative manner (multi-level collaboration among the stakeholders: customer, board and executive management, manager and agile teams). According to Meyer (2004), an organization should focus on communication, teaching, convincing, refining and measuring the success of IT governance. The economical and evolving factors stress the encouragement of practices such as self-assessment, self-improvement, self-discipline and self-accountability among the empowered agile teams to minimize the governance cost. Governance practices for agile development evolve to handle day to day issues which make them different from fixed and detailed governance arrangements. Table 5 presents the key elements of governance as integrated into an agile environment.

5. Agile adoption

While the AgileToolkit and, specifically, the 4-DAT provide new and highly useful information regarding the agility characteristics of process elements, neither offer any support for the process adoption process (see also Henderson-Sellers and Serour, 2000). To fill this gap, we have recently devised, based on industry analysis and a grounded theory research methodology, an Agile Adoption and Improvement Model (AAIM). The AAIM has been constructed to take into account existing knowledge of management of agile teams (Anderson, 2004), agile software development organizations (Chau and Maurer, 2004), the use of an agile approach in a large organization (Lindvall et al., 2004), the concepts of people-orientation (Cockburn et al., 2001), an "agile" way of documenting software (Dickerson, 2004) and the concepts of agile rationalization (Agile Manifesto, 2001; Qumer and Henderson-Sellers, 2006d).

Data have been collected from different sources such as the existing agile frameworks (e.g. Beck, 2000; Baskerville and Pries-Heje, 2001; Schwaber and Beedle, 2002; Auer and Miller, 2002; DSDM, 2003a,b; Aydin et al., 2004; Koch, 2005), industrial agile adoption case studies (e.g. Boehm and Turner, 2003; Leffingwell and Muirhead, 2004; Nielsen and McMunn, 2005; Leffingwell and Smits, 2006; McMunn and Nielsen, 2005; Smits, 2006; Elssamadisy, 2006; Pettit, 2006a,b; Lawrence and Yslas, 2006; Sliger, 2006; Gat and Martens, 2006; Ambler, 2006; Barnett, 2006; Meadows and Hanly, 2006; Qumer and Henderson-Sellers, 2007b), and agile process assessment (Qumer and Henderson-Sellers, 2006a,b).

The "Open Coding" and "Theoretical Coding" techniques of Glaser (1978) have been applied iteratively to identify different categories and their properties; and then to establish the relationship among them (identified categories). The main identified categories are flexibility, speed, responsiveness, communication-orientation, people-orientation, executable-artefacts, learning and leanness. These are then used to define the agile blocks and AAIM levels (AAIML), influenced in part by concepts underpinning software process improvement (SPI), as follows (Fig. 7):

Three agile blocks, from basic to advanced: prompt, crux and apex. In each block, the degree of agility of an agile process is measured quantitatively by using the agility measurement modelling approach (the 4-DAT tool).

Six agile stages. These stages are embedded in the three agile blocks. Each block and stage have a name and specify the agile practices to follow in order to achieve the particular AAIM level (AAIML). The prompt has the AAIML 1: agile infancy. The crux consists of the core of the AAIM levels, which are AAIML 2: agile initial, AAIML 3: agile realization and AAIML 4: agile value. Finally, the apex block presents the AAIML 5: agile smart and the AAIML 6: agile progress. Each level establishes the agile practices in the agile software development process/method, which in turn enable the organization to achieve the desired AAIML over the period of a time.

5.1. Agile block: prompt

The first block, called Prompt, is a base starting point for an organization commencing an agile software development. It consists of a single level, AAIML 1, called agile infancy.

5.1.1. AAIML 1: Agile infancy

At this level, a software development organization does not apply an agile method off-the-shelf; rather, the focus of this level is only to introduce and establish the basic agile properties (speed, flexibility and responsiveness) in a software development process/method in practice. Speed enables the quick development of a quality useable software product, a situation-specific emergent software process (which may combine agile practices from various agile methods), a situation-specific emergent development team(s) and a situation-specific emergent development environment (tools). It also incorporates planning (both release planning and iteration planning) by using an

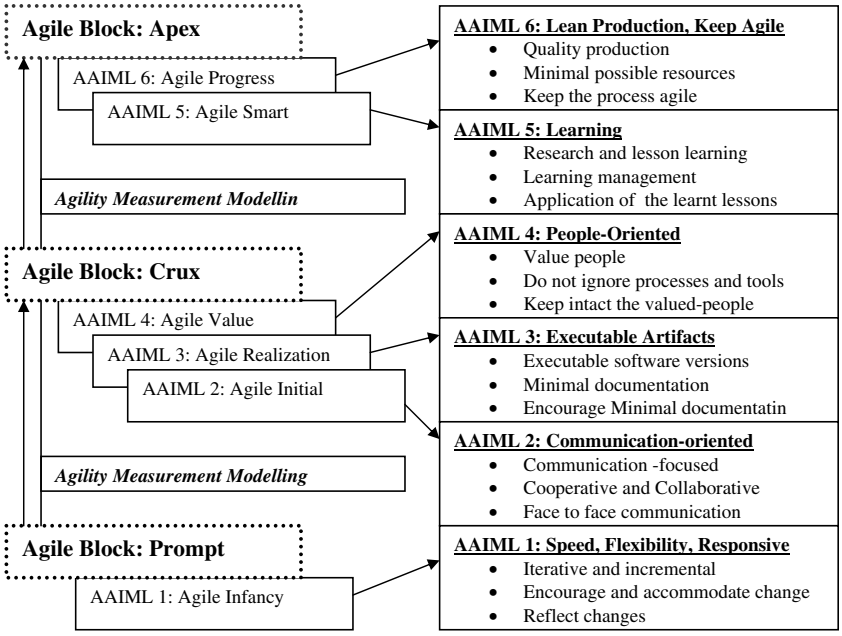

Fig. 7. Agile adoption and improvement model (after Qumer et al., 2007).

iterative and an incremental test-driven approach (test first). Flexibility encourages the acceptance and accommodation of changes (generated from an internal environment or by a customer) in a software product, process, plan, development team(s) and development environment – new tools and technology, tools configuration changes. Finally, responsiveness refers to the fact that not only does it become easy to accept the changes but the changes must be reflected and visible. These are the three properties that establish a foundation for achieving the rest of the agile levels since they cannot be achieved in a single endeavour.

5.2. Agile block: Crux

The crux block (core of the AAIM) consists of three levels. The focus of this block is on the establishment of the key agile practices and properties in a software process/method, which differentiate an agile process from a traditional software development approach. The AAIM levels (in this block) are presented in the following sub-sections.

5.2.1. AAIML 2: agile initial

At this level of the AAIM, the focus is to enable the communication and collaboration (communication-oriented) among the people by establishing good communication and cooperation protocols within the organization (communication among/within the development teams) and outside the organization (communication with customers and with relevant organizations/stakeholders). It has been noticed that communication and cooperation is very important for working with co-workers and establishing accurate requirements and feedback from customers.

5.2.2. AAIML 3: agile realization

This level emphasizes the production of the executable artifacts with a minimal and reduced documentation. The software documentation (non-executable) is used for communication purposes and can be reduced by using other means of communications (verbal or face-to-face communication) and tools. It has been observed that the documentation (non-executable artifacts) can be reduced if there is a well-established communication-oriented culture in the organization (as suggested by the AAIML 2). The AAIML 1, 2 and 3 establish a platform to achieve AAIML 4.

5.2.3. AAIML 4: agile value

At this level of AAIM, the practices are established and focused to value the people (people-oriented) both within the organization (developers) and outside the organization (customers) without ignoring the importance of the software development tools and processes. We have observed and noticed that, in an agile team, highly skilled people should be indulged (as the agile developers are not only the developers but also the decision makers and they are allowed to do whatever they want to do to achieve a desired business value).

5.3. Agile Block: Apex

The apex block consists of the AAIML 5 and 6. The focus is on the establishment of a learning and quality production environment while consuming minimal possible resources with overall continuous progress in the establishment of an agile environment. The following are the details of these two levels. The consideration of the quality factor does not mean that the rest of the levels do not care about the quality but, here, the stress is to further reduce the

production cost while maintaining or improving the production quality (quality should not be compromised while reducing the production cost).

5.3.1. AAIML 5: agile Smart

At this level, the focus is on the establishment of a learning environment. The learning of the people (involve in a software development), software process (before, during and after the execution of a software process), product (before, during and after the production) and tools (the new tools and a technology) lead toward overall organization learning and improvement.

5.3.2. AAIML 6: agile progress

At this level, the practices are focused on the establishment of a lean production environment (the quality production with minimal resources and within a minimum timeframe) and to keep the process agile.

6. AAIM enactment

The adoption and improvement of agile practices is a continuous and evolutionary process (e.g. Salo and Abrahamsson, 2007) and takes time, depending upon various factors. The AAIM is a method-independent model that can guide a software development organization to adopt and improve agile practices for a specific situation or project. We have applied AAIM on one of our industrial pilot projects (Qumer and Henderson-Sellers, 2007d) (a large software development organization) and found that the success of the transition to an agile environment substantially depends on the leading role of the CIO and executive management using AAIM. Such a person should champion the adoption of agile methods and take the responsibility for eliminating any impediments to effective development and delivery of business value through agility. Agile practices could be adopted in different ways by different organizations. The AAIM lays the groundwork for the implementation of agility and the software development organization may tailor or customize AAIM according to their local organizational structure, culture, size and development environment. The transition to an agile software development practice is challenging and, therefore, it is a good idea to introduce an agile approach into a traditional software development environment/organization gradually.

The results of various agile adoption case studies have been analysed (e.g. Henderson-Sellers and Serour, 2005; Bajec et al., 2007) and it has been noticed that an agile approach requires a different mindset, process, people, environment and tools (Qumer and Henderson-Sellers, 2006d) for its successful implementation. The AAIM will help to establish such an environment to successfully follow an agile approach. The AAIM can be used for the assessment of a particular level of agility adoption and to advance to the next level. If the assessment is positive, an organization may proceed with the next level. Otherwise, it should stay at the same level for a specified period of a time as suggested by an assessor. An IT consultant, agility coach, software engineering director, process manager or process quality manager may use this model to introduce and assess an agile process/method for a particular situation. The following are the key features of this model.

- The AAIM helps to assess how and how well an agile process/method is being followed within a software development organization in practice.
- The AAIM helps to assess the current agile level of an organization.
- The AAIM helps to measure and assess (quantitatively) the degree of agility of a software development process.
- The AAIM provides a roadmap for the establishment of a systematic agile software development environment and the systematic use of agile practices within that environment.
- The AAIM combines the concepts from both theory and practice (data and feedback from both researchers and software industry).

6.1. First case study

Our first industrial application of AAIM was to an organization who required a new and formal process/method to support their core business of product enhancement. This section clearly outlines the newly engineered agile product-enhancement process for the case study industry (Fig. 8) by using the ASSF and a method engineering approach. The 'process' aspect of the ASSF had been used as a vision-guiding model for the construction of the agile product-enhancement process (APEP). The abstraction model (object-oriented and service-oriented), product model (the product that was subject to enhancement) and agile team choreography model (describing a self-organizing, empowered and co-located team) had also been outlined in order to direct the behaviour of the APEP agile team, including senior developers, responsible for the identification and the modelling of the new product-enhancement features by using this newly created agile approach; the product manager, responsible for the prioritization of the product features for development and its success in the market; and traditional developers, responsible for the detailed implementation of the identified features by using a formal incremental approach.

The APEP was defined as a mix of agile and traditional incremental software development practices (Jalote, 1997). It includes three main phases: product-enhancement features generation (agile approach), product development (traditional incremental approach) and post-mortem; and ten main practices: (1) communication-cooperation protocol development, (2) product client(s) selection, (3) product-enhancement features and business value identification (4) product-enhancement features model development, (5) product-enhancement model testing and feedback, (6)

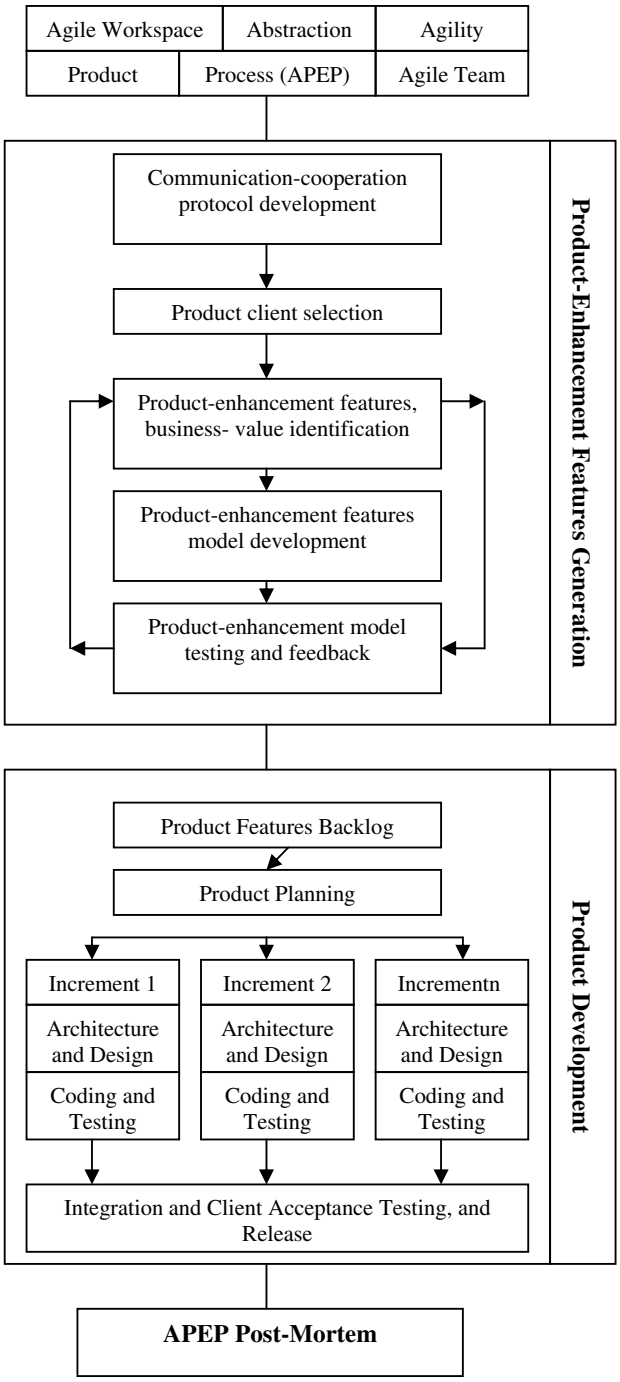

Fig. 8. Agile product-enhancement process (after Qumer and Henderson-Sellers, 2007b).

product (enhancement) features backlog (7) product planning, monitoring and control, (8) product architecture and design, (9) implementation, testing and release and (10) APEP post-mortem. The product-enhancement features generation phase contains agile practices for iterative generation of the product-enhancement features list (product backlog) and executable product-enhancement features model in small increments in a communication-oriented, cooperative and people-focused environment. Once the desired number of features had been identified, in the product development phase, a formal incremental approach for development is followed for the detailed description of the features identified in the requirements specification document, for architecture and design, implementation, testing and release of the enhanced product with its new features. Finally, the post-mortem phase is executed after the end of each iteration, phase and process for the purpose of future learning and decision making; a method engineering approach here enables the team to bring in changes in the APEP in order to implement the decisions and to reflect the lessons learned. The software company have accepted the APEP and is using it for product enhancement including any necessary changes, as required.

6.1.1. The communication-cooperation protocol development

The communication-cooperation protocol had been developed to specify and enable an effective face-to-face communication among the empowered, self-organizing and cross-functional agile team (senior developers of the company) and their clients. The purpose of this protocol was to reduce the documentation and waste while focusing on the development of the new features, rather than the documentation. This practice clearly highlighted the usability of three agile values "reduced documentation", "individuals and interactions", and "keeping the process cost effective" (Agile Manifesto, 2001; Qumer and Henderson-Sellers, 2006a) to maximize the business value.

6.1.2. The product client(s) selection

A group of the existing product clients had been selected for the construction, verification and validation of customer-oriented features list (according to the client desires, needs and perspectives), rather than developer-oriented (features developed from the developers perspectives). In the past, the company developed a state-of-the-art product with many useful features (from the perspectives of developers) but soon that product was shelved, since most of the clients rejected the features of the product. The purpose of the formation of this group was to identify and include only those features that met the needs of the client, while cooperating and communicating with the clients. This practice clearly highlighted the usability of agile value "customer collaboration" (Agile Manifesto, 2001) to maximize the business value.

6.1.3. The product-enhancement features and business value identification

The product-enhancement features with their business value had been identified. The execution of this practice involved market research, client-feedback, the different patterns of the previous requests (by the clients for the bug fixation and for new features), assessment of the features of the available similar products for the identification of the product-enhancement features and their business value.

6.1.4. The product-enhancement features model development

An internal high-level (with minimum details) executable model (essentially a rough prototype created using techniques useful for agile development, with prioritized features) had been iteratively developed, in order to document the currently identified features. The purpose of this executable model was to avoid the unnecessary waste of the paper documentation since it was easy to verify and validate the executable code or documentation. This practice clearly highlighted the usability of an agile value "working software over comprehensive documentation" (Agile Manifesto, 2001) to maximize the business value.

6.1.5. The product-enhancement model testing and feedback

In each iteration, both internal (automated) and client testing were performed for the verification and the validation of the executable-model for the purpose of the feedback and features generation. This practice clearly highlighted the usability of an agile value "customer collaboration" (Agile Manifesto, 2001) to maximize the business value.

The test reports and client-feedback had been used and the changes were directly realized in the code to document the new or changed feature. This practice clearly highlighted the usability of two agile values "customer collaboration" and "responding to changes" (Agile Manifesto, 2001) to maximize business value.

6.1.6. Product features backlog

The desired number of identified features (through the product-enhancement model) had been stored in the product features backlog; a formal (existing) approach for requirements specification was followed for the detailed description of the identified features. A formal requirements management approach was followed at this stage to approve, baseline, track and bring about any necessary change in the product requirements or features through a formal request. The change was assessed and approved by the change evaluation team. This practice is a traditional requirements engineering approach.

6.1.7. Project planning, monitoring and control

The planning, monitoring, and control artefacts were outlined and documented in detail and fixed before the commencement of the development of the product in small increments. The responsibilities and accountabilities were assigned to project stakeholders at this level. A formal procedure was used for the management of the project plan.

6.1.8. Product architecture and design

Once the requirements were described, detailed planning for the development of the product increments (increments for the product enhancement) was undertaken. A formal approach was used for the definition of the upfront product architecture and design in detail by using product backlog and product-enhancement features model (developed in the first phase), before implementation. The overall product components with their individual implementation routines and guidelines were specified.

6.1.9. Implementation, testing and release

Finally, the already developed product-enhancement model, detailed product specifications, architecture and design artefacts were used for the implementation of the product features. The test cases were also developed at this stage for the purpose of quality assurance of the product features. The enhanced version of the product was released after final client acceptance (a group of client organizations). The released product was baselined and handed over to the configuration management (CM) team, which became responsible and accountable for the integrity and change management of the product. Any change in the any of the artefacts or work products was subject to configuration management, which followed a formal approach to handle the situation, when required.

6.1.10. The post-mortem of the APEP

The post-mortem of the APEP had been done for the purpose of learning and modification in the APEP. This practice clearly highlighted the applicability of method engineering and the usability of an agile value "keeping the process agile" (Qumer and Henderson-Sellers, 2006a) to maximize the business value.

6.1.11. Case study results and analysis

This case study has underlined the observation that the success of the agile transition substantially depends on the leading role of the CIO and executive management, who should take responsibility for eliminating the impediments to effective development and delivery of business value through agility. The construction of an appropriate agile process was itself a challenge but the most important challenge was the development and encouragement of an agile culture and mindset within the organization. We had laid the groundwork for an agile process compatible with a formal approach by defining a set of hybrid agile practices by using the ASSF and method engineering. The case study company understood the potential advantages of agile for their organization. This transition also helped them to train the team of developers for future agile ventures and this team could serve as a change agent for the future and more extensive adoption of an agile method. Furthermore, it had been proven that agile practices would allow the organization to develop or invent new features for their products in a formal and large development environment where a complex product could be developed by using an agile approach. From the perspective of the company clients and product acceptance in the market, the stakeholders of the product realized the importance of being involved throughout the product development. The case study organization development teams learned that being collectively responsible for product features' generation and product scope enhancement, they could work creatively while communicating and cooperating with each other to meet the

budget to satisfy their most important business needs. The other developers within the organization also showed interest in an agile approach and realized that agile practices can work within large and complex technical development arrangements. The case study organization felt well-placed and found itself to be in a very comfortable position for the adoption of agile practices within the organization on a wider scale in the future. The results of this case study clearly demonstrate two things: first, the applicability of method engineering to agile and the second, the appropriateness of agile practices for large and complex projects. Agile and traditional phases had been combined in the constructed process (APEP) for the case study organization.

In the agile phase (first phase) of APEP, the empowered agile team made their decisions for delivering the business value (new features). They learned to identify the changes and reflect those changes quickly by helping themselves using a face-to-face communication and feedback approach. At the same time, they were encouraged to organise and manage themselves but were held accountable and responsible for the required deliverables (new features and executable model). They found documenting the newly identified features in the features executable model to be very productive. The product manager worked as a facilitator rather than as a manager and led the team by facilitating and governing them not only as a manager. However, the responsibilities were shifted more on to the developers in this phase. The transition from a traditional approach and mindset takes time and the product manager helped the developers at all levels and led them to make a successful transition. The product manager and self-motivated developers found the agile adoption exciting and rewarding.

In the second phase of the APEP, the traditional development team took the newly identified features and executable product-enhancement features model as an input for the detailed implementation and product release, under the supervision of formal development environment and recognized the quality of the deliverable of the agile team. They found that working with the executable model is much easier than with requirements on paper only. The developed model served them as a vision-guiding model for the later development. In this case study, it was also found how successfully an agile team communicated the developed artifacts to a non-agile team in the form of an executable model.

The last phase of APEP (post mortem), which was not actually a terminal phase, is an umbrella activity that is done all the time during the product-enhancement process life cycle for effective learning and decision making for the purpose of continuous process improvement and optimized business value delivery.

The case study company is using APEP and had already made several changes in the initial version of the APEP by introducing a few more agile practices in the formal phase of the APEP. They have introduced fixed duration iterations within each increment, two levels of planning (release planning and iteration planning) and a test-first (the design of the test-case) approach. However, the formal phase of the APEP is documentation-driven and change has not been welcomed overall so that a formal change control mechanism is used to avoid the change, i.e. the developers are not allowed to make their decisions and a formal change management mechanism is used. However, the developers may give input to the decisions. In future, the company is willing to establish a communication-corporation and a less document-oriented environment at a large scale, which will enable them to focus on the quick delivery of the working software rather on documentation.

6.2. Second case study

In a second industry application of the Agility Toolkit, it was used to assist an industry to adopt XP (Beck, 2000) for the development of a service-oriented application (e-health). Initially, in the first phase of this project, instead of a full scale agile process, we envisaged an experimental scenario and constructed only three agile process fragments: enhanced pair programming (EPP), pair review (PR) and on-site developer (OSD) for a service-oriented e-health project to familiarise the case study organization with an agile approach. Once the successful results of these practices had been recognized, the case study organization decided to engineer a full scale ASOP by using the ASSF, AAIM and a method engineering approach for their e-health consultancy services (EHCS) project.

6.2.1. First phase

Initially, in our facilitation of XP into this organization, we decided to use only a single XP practice "pair programming" (PP) and analysed it thoroughly before implementation (Elssamadisy and Schalliol, 2002; Qumer and Henderson-Sellers, 2006a,b), finding several issues that did not allow us to use "pair programming" off the shelf. Therefore, based on this industry experience and evaluation, we decided to customize it by using the Agile Toolkit to make it useable for the case study e-health development project. We found issues in PP – for example, since we cannot hold responsible a single person in a team, the team size is constrained to always be an even number. In EPP (Table 6), we decided to use a minimum of two developers (one senior and one junior developer) with an option to add another extra developer, if required. EPP developers had to work on individual computers on more than one service (software components) of the e-health project, rather than two on one computer and one component. In the EPP, one programmer should be senior (leader) and with other one or two junior (new to agile or less experience) developers. The senior developer led the development and was responsible for the design and the implementation of the overall service component. The senior developer decided (with the collaboration of other junior developer) which functions of the services would be developed by whom and agreed to help and cooperate with each other

Table 6
The agile process fragment (Enhanced Pair Programming) specifications: (after Qumer and Henderson-Sellers, 2007c)

Agile process fragment	Description
ID and name	Enhanced Pair Programming (EPP)
Description and purpose	A self-organizing pair of two developers will work on more than one independent components of a project but they will exchange the development of the components. The components will be owned by the pair not by the individuals, i.e. pair-ownership for the component
Abstraction	Service-oriented (tested)
Tools and people	Tools should allow for developing and sharing the work. People should be able to communicate and cooperate, self-organize, and have the necessary skills for iterative development with minimal documentation
Development style	Iterative and incremental
Physical environment	Preferably co-located
Pre and post conditions	A high-level design and high-level test case design should be available with a high level description of the project components (services). During and after the each iteration of EPP, the design, test cases, requirements and product features will emerge; the product shall be in a stable state with new features or modified features and will be available for testing
Constraints and risks	Only two or three co-located people in one set of EPP and one must be senior. Social risks and personality conflict, human resource risks (one person from the EPP may leave or get sick etc. which may affect the development)
Degree of agility	1.0
Business value	Reduced production cost Reduced duration Reduced documentation Improved product quality Trained team member Trust but sufficient discipline, control and accountability
Alerts	This approach may not work if all members of the EPP are junior

Table 7
The agile process fragment (pair review) specifications (after Qumer and Henderson-Sellers, 2007c)

Agile process fragment	Description
ID and name	Pair review (PR)
Description and purpose	In the pair review practice, a self-organizing pair of two developers will be used that utilizes the self-testing and exchange-testing techniques for unit testing and integration testing
Abstraction	Service-oriented (tested)
Tools and people	Testing tools should allow for the testing and sharing of the work. People should be able to communicate and cooperate, self-organize, and have the necessary skills for iterative testing with minimal documentation
Development style	Iterative and incremental
Physical environment	Preferably co-located
Pre and post conditions	An executable-module with necessary test cases should be available. During and after the each iteration of PR, the design, test cases, requirements and product features will emerge
Constraints and risks	Only two or three co-located people in one set of PR and one must be senior. Social risks and personality conflict, human resource risks (one person from the PR may leave or get sick etc. which may affect the development)
Degree of agility	1.0
Business value	Reduced production cost Reduced duration Reduced documentation Improved product quality Trained team member Trust but sufficient discipline, control and, accountability
Alerts	This approach may not work if all members of the PR are junior

for the development of assigned services (software components). Developers in EPP organized themselves by conducting their own small meetings. In-program documentation and face-to-face communication were used to reduce unnecessary documentation and overhead. The senior developer designed the component implementation (code the overall component – skeleton code) and handed it over to the junior for the detailed implementation of one of the specified functions of a service (which they decided mutually). Meanwhile, the senior developer designed the implementation skeleton for the second component; the junior developed a specified function with the help of a senior and then asked for further directions regarding the development of the component. In this way, by using an exchanged development strategy, they iteratively developed the services (components).

In PR (Table 7), the developers (two developers with an option to add another extra developer) had to test the services by themselves (self-testing), exchange testing (testing the functions/services of each others), and then finally had to perform the integration testing together. The senior developer led the PR and was responsible for the overall quality of the developed service component. The developers organized themselves for PR by conducting their own small meetings. The developers used a collaborative and communication-oriented (face-to-face) approach to reduce the unnecessary documentation and waste.

The developers found EPP and PR very productive in comparison with a traditional pair programming approach. EPP and PR both helped to improve the quality of the services (components) before user acceptance testing and very few bugs were reported during user acceptance testing. We also trained one junior developer for an agile development environment. The junior developer reported that he was well motivated by the senior and he learned many new programming and testing techniques. It has been observed that the junior developer worked very well with the motivation of the senior as well as being self-motivated. In order to bring sufficient control and discipline to the EPP and PR trusted-team, we embedded the factor of accountability. We allowed and empowered them to take their decisions but they had to justify whatever they decided.

The third practice developed within this project case study was "on-site developer" (Table 8) i.e. a developer at the customer site during the start of the project and the prototype development, in order to get the necessary quick feedback for any clarification of requirements.

In this case study, we not only developed these three new practices but also tested them by adding them to the industry-customized version of XP. The results of these practices in terms of their business value are reduced production cost, reduced duration, reduced documentation, improved product quality, a trained team member and a disciplined team with responsibility and accountability.

The old pair programming practice had been assessed (Qumer and Henderson-Sellers, 2006b) and the degree of agility recorded as 0.60 the range of value of the degree of agility is [0.0 (Min.)–1.0 (Max.)]. We then assessed the newly developed practices EPP, PR and OSD and the degree of agility was calculated by using 4-DAT (Qumer and Henderson-Sellers, 2006a) as 1.0.

6.2.2. Second phase

In the second part of this case study, we constructed a full-scale, agile, service-oriented process (ASOP – Fig. 9) to help in the development of the e-health consultancy services (EHCS) application. The EHCS provides a remote clinic interface and services to patients and consultants and enables them to interact. The main identified services in the domain of EHCS are patient service, quality service (security and reliability service), consultant service, treatment service, administration service, a clinic service and a report service that include business operations. The EHCS also provides a patient-consultant forum service, messaging service and query service. A service is a logical view or an abstraction of a business process such as a program or database for carrying out business-level operations.

Table 8
The agile process fragment (On-site Developer) specifications: (after Qumer and Henderson-Sellers, 2007c)

Agile process fragment	Description
ID and name	On-site Developer (OSD)
Description and purpose	Developer on the customer site during the start of the project and the prototype development, in order to get quick feedback for clarification of the requirements
Abstraction	Service-oriented (tested)
Tools and people	Tools (portable) should allow rapid requirement gathering, reporting and prototype development. A self-organizing team needs the ability to demonstrate and communicate, and should be able to produce a prototype and work to ensure that the requirements are identified and addressed within the prototype with a minimal documentation
Development style	Iterative and incremental
Physical environment	Off-site (at the customer site)
Pre and post conditions	Customer consent and approval for the requirements gathering. During and after the each iteration of OSD, the design, test cases, requirements and product features will emerge
Constraints and risks	The workspace at customer site may not be suitable for developers. Customer or customer representative may not be collaborative
Degree of agility	1.0
Business value	Reduced requirements gathering cost Reduced duration Reduced documentation
Alerts	This approach will not work if the customer does not allow the OSD team to work on their site

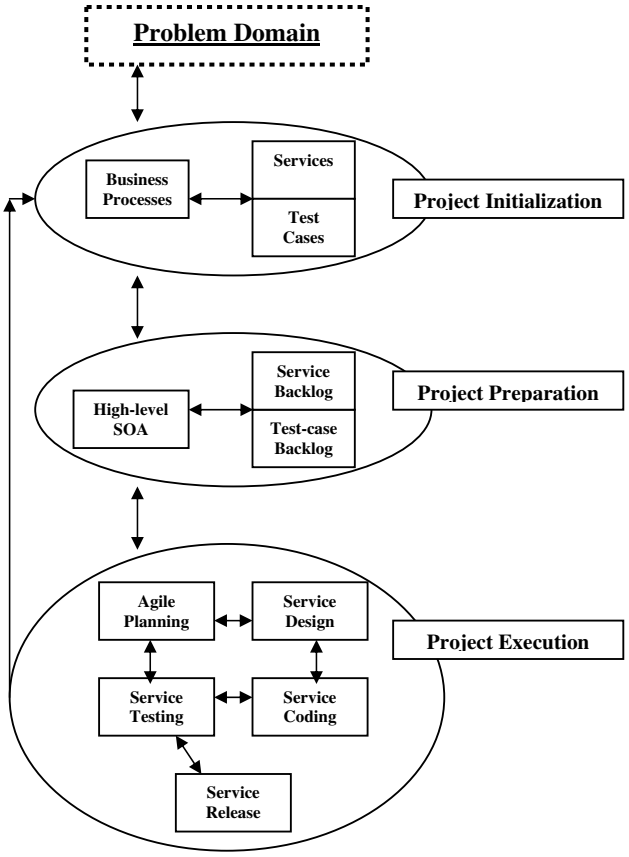

Fig. 9. Agile service-oriented process – ASOP (after Qumer and Henderson-Sellers, 2007d).

The existing service-oriented analysis and design process had been a traditional phased approach and lacked the attributes of agility. Therefore, it took months or years to show real business value in terms of executable working software services, whereas agile provides a value-driven and result-oriented approach that attempts to increase the business value throughput through the rapid release of a working software in small iterations of 2–3 weeks. Consequently, ASOP was engineered (by focusing on AAIM level 1 agile practices) to iteratively carry out a systematic test-driven development for the EHCS project. The ASOP includes ten main practices: identification of the business processes and service-mapping, service backlog, test-cases backlog, high-level service-oriented architecture, release planning, iteration planning, software services-components mapping and design, enhanced pair programming,

pair review and user acceptance testing, service release and final documentation. These are summarized as follows:

- *Identification of the business processes and service-mapping*: the business processes (a sequence of activities with specific business goals) in the problem domain are iteratively identified and conceptualized in terms of services. A business process is a sequence of activities or business level operations with specific business goals in the given problem domain or client requirements. The client requirements or business domain are iteratively analysed, defined and specified in terms of currently known and utilized business processes. It is often impossible to get the complete requirements list at the start of a project from customers; therefore an iterative approach is recommended. UML business activity and use case diagrams are frequently adopted as the key elements of a business process model. The currently known business processes are then further iteratively identified and conceptualized in terms of services, called service-mapping. The mapping is done by using business process and a service mapping matrix. Service-mapping is a method to verify whether the services correctly refer to the business processes or business requirements. A service is a logical view or an abstraction of a business process such as a program or database for carrying out business-level operations. Face-to-face communication with a client is preferred here in this phase of ASOP; however, other means of communications (email, fax, teleconferencing, video conferencing, telephone etc.) may be used, if appropriate.
- *Service backlog*: the service backlog is a repository for the currently known and identified services, development. It formally documents and prioritizes the currently known and identified services, and is updated as new services are iteratively identified during the course of an agile development. The intensity of the details of the documentation varies from project to project and problem to problem. There are various factors that can be considered contextually for deciding upon the documentation details such as the ability of the business analyst of a customer to convey requirements, availability of the tools and the skill of the developers.
- *Test-cases backlog*: a repository for test-cases (associated with currently known services) that are identified with the help of a customer. The associated test-cases, with currently known services, are also identified and specified by the interaction of a developer and client. The developers directly communicate with a client or client representative (business analyst) to develop, prioritise (from a criticality point of view) the service test-cases and put into the test-case backlog.
- *High-level service-oriented architecture*: instead of a detailed up-front design, a high-level service-oriented architecture is developed, which is updated after each iteration as the development progresses. The architecture includes two types of architectures: enterprise system architecture, which specifies hardware, and software infrastructure to support transactions, services distribution and clustering, messaging, security, session management; and application architecture (which, in this case study, refers to the EHCS application that is built on top of the enterprise system architecture using J2EE – Weblogic). Instead of a detailed up-front design of architecture, a high-level service-oriented architecture is developed, which is updated after each iteration as the development progresses. The focus and preference should be given to the development of an executable high-level architecture and should be used as a feedback tool from a client, rather than simply detailed documentation, although necessary documentation cannot be avoided and is likely to depend upon the project.
- *Release planning*: at a high level, release planning is done by the manager to estimate (at a high level) the number of iterations, service functionality, cost and duration. Burndown charts are also used here for project tracking and monitoring. A detailed plan is not possible at the start of the project, since a wrongly estimated plan may lead toward the wrong decision. Therefore, planning is done at two levels: release and iteration planning (see below). At a high level, release planning is done by the manager to estimate (at a high level) the number of iterations, service functionality, cost and duration with burndown charts being used for project tracking and monitoring. As agile development focuses on the frequent delivery of executables, the release duration should be between 2–3 months. At a low level, the developers are responsible for providing the estimate (low-level) for the current iteration in hand.
- *Iteration planning*: currently, the software organization is managing the release duration between 2–3 months. By using iteration planning, the developers provide estimates (low-level) for the current iteration in hand. The duration of each iteration is 3 weeks. The software company is looking forward to reducing it to 2 weeks.
- *Software services-components mapping and design*: the mapping and design of the services are iteratively done by modelling services in terms of a configured set of software components and objects (a further low level abstraction) before implementation. Client feedback and a developed high-level architecture is used for the iterative mapping and design of the currently known services (modelling services) in terms of software components and objects configuration (a further low level abstraction) before implementation.
- *Enhanced pair programming*: the service components are implemented using J2EE. The services that are ready (according to the priority order of the service-backlog) for implementation are fetched from the service backlog together with their test-cases from the test-case backlog and are implemented iteratively by using an enhanced pair programming (EPP [24]) approach (J2EE or. NET specific implementation for the services). In enhanced pair programming, as we have noted above,

two or more developers may work together and are responsible for the service that is allocated to them for development. One of the developers should be senior and the other developer(s) may be junior. It is the responsibility of the developer to make sure that the developed service should work according to the specified test-cases. If there are any changes that are required in the service-backlog or test case-backlog, these must be reported and realised in the affected artefacts.

- *Pair review and user acceptance testing*: allows service-testing to be performed in each iteration and release. The developed services are individually tested by the developers concerned and then further testing is done in pairs (pair-reviews) before integration test are undertaken in each iteration. Finally, after internal testing of the developed service, user acceptance testing is done with the collaboration of the client and developer in each iteration and before each release.
- *Service release and documentation*: Once the service has passed the user acceptance test, the service is prepared for the release with the necessary user documentation and then shipped to the client.

On the basis of an agile implementation experience-estimate, it is believed that the current version of the ASOP (AAIML 1) will decrease the development cost by about 20% in comparison with the existing traditional approach. In future, as the software organization gains more experience with the current version of the ASOP agile process, new agile practices will be added to ASOP to achieve higher AAIM levels. A full transition to agile may take 3–5 years.

The focus of this case study research was to empirically validate the usability of the ASSF to support method engineering and adoption of an agile approach. Therefore, firstly, process engineering workshops (Fig. 10) were conducted to discuss and iteratively define the processes by involving participants from the concerned organisations. The process engineering workshops involved the definition of process engineering resources, selection of framework (ASSF), process fragment specifications, interaction specifications, verification and validation (process peer reviews) before process release and adoption.

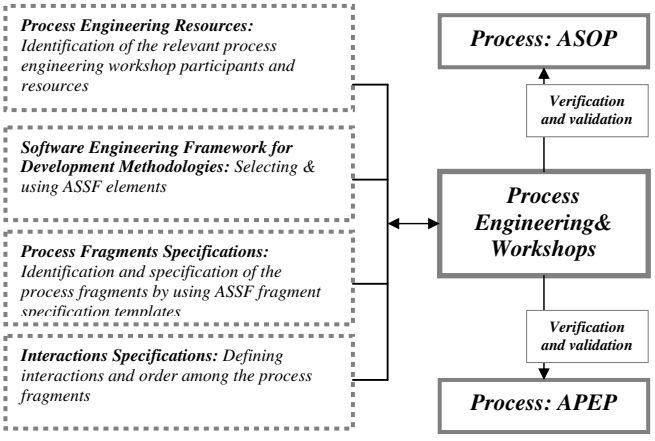

Fig. 10. Agile software process engineering – case studies.

7. Summary and conclusions

We have described here a new approach to extending and customizing the agile approach to software development by means of an Agile Software Solution Framework, focusing on its Agile Toolkit and governance. These ideas have found practical application in the Agile Adoption and Improvement Model (AAIM).

The Agile Software Solution Framework (ASSF) can be used to create, modify or tailor situation-specific agile software processes based on any one of the several contemporary abstraction viewpoints (e.g. agent-oriented, aspect-oriented, service-oriented etc. – collectively called here m-oriented) by using a situational method engineering approach, feedback and a standard meta-model. We have embedded a number of new models and processes in ASSF; such as an agility measurement model and process, an agile adoption and improvement model and process, an agile software solution framework knowledge-base engineering and management process, an agile workspace and an Agile Toolkit.

An important element of the ASSF is its agile tookit. This toolkit facilitates the construction, modification or tailoring of situation-specific agile m-oriented process fragments and then supports the combination of those fragments into agile software processes by using a method engineering approach, feedback and a standard meta-model.

A second important component of the ASSF, and one not previously discussed in the literature, is governance in the context of an agile software development approach. Here, we have highlighted and explained the essential properties of governance integrated into an agile mindset and proposed an appropriate model. We have pointed out the relevant factors of governance such as strategic alignment of business and agile, evaluation of business value delivery through agile (benefit-cost), monitoring of resources, risks and management, monitoring of agile development performance, control and accountability, as well as a governance framework, policies, procedures and structure. We stressed the issue of maximizing the business value (return on investments) by focusing on decision making, accountability and assessment frameworks for performance and risk management in the context of agile environments. The purpose of this effort is to uncover the potential and important elements of governance that will facilitate the transition from small-medium scale to large and complex scale agile methods. In future, more research is required towards the design, implementation and evaluation of governance frameworks, processes and structure to support governance in the context of agile development.

To assist managers in transition to an agile development environment, we have introduced here the Agile Adoption

and Improvement Model (AAIM). This has been developed to aid the introduction, assessment and improvement of the agile software development approach (processes or methods) in a software development organization. We have tested this model on two pilot projects in industry. We applied this model to transform a large non-agile software development organization to adopt an agile approach. In the first pilot project (reported in the first case study), the current state of the case study software company was first assessed and then an agile product-enhancement process (APEP: hybrid agile practices) was engineered rather than a whole methodology. Secondly, the engineered process (APEP) was adopted by using AAIM. Currently, the case study organization is operating at AAIML 1 and is successfully using agile practices to gain the desired business value. In the second pilot project, we first assessed the current state of the software organization and then envisaged an experimental scenario and constructed only three agile process practices. On the basis of the results and the business value gained through these practices, a full scale agile process (ASOP) has been engineered in the third case study by using the ASSF, AAIM and a method engineering approach for their e-health consultancy services (EHCS) project. In future, both case study companies are passionate to establish a communication-corporation and a less document-oriented environment on a large scale, which will enable them to achieve AAIM levels 2 and 3.The results of the case studies highlight two things: firstly, a step-by-step approach may be considered reasonable for a gradual, successful transition or adoption of agile ideas, rather than all at once, which may pose several risks and problems; and, secondly, the appropriateness of agile practices for large and complex projects for different abstraction mechanisms (both object-oriented and service-oriented). We intend to improve the ASSF and AAIM as we further proceed in our research and get feedback from the software community.

Acknowledgements

We wish to thank the Australian Research Council for financial support. This is Contribution Number 07/09 of the Centre for Object Technology Applications and Research (COTAR). We are also thankful to the people from both the research community and the software industry who helped us with their valuable feedback and experience.

References

Agile Manifesto, 2001. Manifesto for Agile Software Development.

Ambler, S.W., 2006. Scaling agile development via architecture, Agile Journal. <www.agilejournal.com>.

Anderson, D.J., 2004. Agile Management for Software Engineering. Pearson Education Inc..

Auer, S.O., Herre, H., 2006. RapidOWL – an agile knowledge engineering methodology. In: Proceedings of the Sixth International Andrei Ershov Memorial Conference on Perspectives of System Informatics, Russia.

Auer, K., Miller, R., 2002. Extreme Programming Applied: Playing to Win. Addison Wesley Pearson Education, Indianapolis.

Augustine, S., 2005. Managing Agile Projects. Pearson Education Inc., USA.

Aydin, M.N., Harmsen, F., Van Slooten, K., Stegwee, R.A., 2004. An agile information systems development method in use. Turkish Journal of Electrical Engineering & Computer Sciences 12, 127–138.

Bajec, M., Vavpotič, D., Krisper, M., 2007. Practice-driven approach for creating project-specific software development methods. Information and Software Technology 49, 345–365.

Barnett, L., 2006. Learning from others best practices for large-scale agile development, Agile Journal. <www.agilejournal.com>.

Barnett, L., 2007. Agile projects must measure business value, Agile Journal. <www.agilejournal.com>.

Baskerville, R., Pries-Heje, J., 2001. How Internet is redefining information systems development methodology? In: Realigning Research and Practices in IS Development, pp. 49–68.

Beck, K., 2000. Extreme Programming Explained. Addison-Wesley Pearson Education, Boston.

Behr, K., Kim, G., Spafford, G., 2004. The Visible Ops Handbook: Starting ITIL in 4 Practical Steps. Information Technology Process Institute.

Boehm, B., Turner, R., 2003. Observations on balancing discipline and agility. In: Proceedings of the Agile Development Conference. IEEE Computer Society, pp. 32–39.

CALDER-MOIR, 2006. The Calder-Moir IT Governance Framework. <http://www.itgovernance.co.uk>.

Chau, T., Maurer, F., 2004. Tool support for inter-team learning in agile software organizations. In: Proceedings of the Workshop on Learning Software Organizations, Banff. Springer, pp. 98–109.

Cockburn, A., Highsmith, J., Boehm, B., 2001. Agile software development: The people factor. Computer 34, 131.

Dahlberg, T., Kivijärv, H., 2006. Integrated framework for IT governance and the development and validation of an assessment instrument. In: Proceedings of the 39th Hawaii International Conference on System Sciences, USA, IEEE Computer Society, Track 8, p. 194.

Dickerson, C., 2004. Documentation dilemma, InfoWorld 26. InfoWorld Publishing Group.

DSDM, 2003a. DSDM Consortium, Dynamic Systems Development Method Ltd.

DSDM, 2003b. Guidelines For Introducing DSDM into an Organisation Evolving to a DSDM Culture, DSDM Consortium.

Elssamadisy, A., 2006. Getting beyond "It Depends!" being specific but not prescriptive about agile practice adoption, Agile Journal. <www.agilejournal.com>.

Elssamadisy, A., Schalliol, G., 2002. Recognizing and responding to "bad smells" in Extreme Programming. In: Proceedings of the ICSE'02, ACM, Orlando, Florida, USA, pp. 617–622.

Firesmith, D.G., Henderson-Sellers, B., 2002. The OPEN Process Framework. Pearson Education, London.

Gat, I., Martens, R., 2006. CASE STUDY: how BMC is scaling agile development, Agile Journal. <www.agilejournal.com>.

Glaser, B.G., 1978. Theoretical Sensitivity: Advances in the Methodology of Grounded Theory. Sociology Press, Mill Valley, CA.

Greenfield, J., Short, K., 2004. Software Factories. Wiley Publishing, Inc., Indianapolis, IN, USA, p. 666.

Gummesson, E., 2000. Qualitative Methods in Management Research, second ed. Sage, London.

Hammer, M., 2002. Process management and the future of six sigma. MIT Sloan Management Review 43 (2), 26–32.

Henderson-Sellers, B., 2002. Agile or rigorous OO methodologies – getting the best of both worlds. Cutter IT Journal 15 (1), 25–33.

Henderson-Sellers, B., 2003. Method engineering for OO system development. Communication of the ACM 46 (10), 73–78.

Henderson-Sellers, B., Serour, M., 2000. Creating a process for transitioning to object technology. In: Proceedings Seventh Asia-Pacific Software Engineering Conference. APSEC 2000, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 436–440.

Henderson-Sellers, B., Serour, M.K., 2005. Creating a dual agility method – the value of method engineering. Journal of Database Management 16 (4), 1–24.
ISO/IEC, 2007. Software Engineering – Metamodel for Development Methodologies. ISO/IEC Standard 24744.
Jalote, P., 1997. An Integrated Approach to Software Engineering, second ed. Springer-Verlag, New York.
Koch, A.S., 2005. Agile Software Development: Evaluating the Methods for Your Organization. Artech House, Inc., London.
Kumar, K., Welke, R.J., 1992. Methodology engineering: a proposal for situation-specific methodology construction in challenges and strategies for research in systems development. In: Cotterman, W.W., Senn, J.A. (Eds.). John Wiley & Sons, Chichester, UK, pp. 257–269.
Lainhart, J.W., 2000. COBIT: a methodology for managing and controlling information technology risks and vulnerabilities. Journal of Information Systems vol. 1 (14), 21–25.
Lawrence, R., Yslas, B., 2006. Three-way cultural change: introducing agile within two non-agile companies and a non-agile methodology. In: Proceedings of AGILE 2006 Conference Minneapolis. IEEE Computer Society, USA, pp. 255–262.
Leffingwell, D., Muirhead, D., 2004. Tactical Management of Agile Development: Achieving Competitive Advantage. Rally Software Development Corporation.
Leffingwell, D., Smits, H., 2006. A CIO's playbook for adopting the scrum method of achieving software agility, Rally Software Development Corporation and Ken Schwaber-Scrum Alliance.
Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., May, J., Kahkonen, T., 2004. Agile software development in large organizations. Computer 37, 26–34.
McBride, T., 2006. The use of project management mechanisms in software development and their relationship to organizational distance: an empirical investigation, Ph.D. Thesis, University of Technology, Sydney.
McMunn, D., Nielsen, J., 2005. Showcasing agile development: a powerful tool for "Crossing the Chasm", Digital Focus. <www.digitalfocus.com>.
Meadows, L., Hanly, S., 2006. Agile coaching in British Telecom, Agile Journal. <www.agilejournal.com>.
Meijer, M., 2003. Application service library (ASL) and CMM. The Journal of IT Alignment and Business IT Alignment 1 (1), 21–26.
Meyer, N.D., 2004. Systematic IS Governance: an Introduction. Information Systems Management.
Nielsen, J., McMunn, D., 2005. The agile journey: adopting XP in a large financial services organization. In: Proceedings of XP2005. Springer-Verlag, Berlin Heidelberg, pp. 28–37.
OGC, 2005. Managing Successful Projects with PRINCE, Office of Government Conference.
Patel, N.V., 2002. Emergent forms of IT governance to support global e-business models. Journal of Information Technology Theory and Application 4 (2), 33–49.
Pettit, R., 2006a. An "agile maturity model?" Agile Journal. <www.agilejournal.com>.
Pettit, R., 2006b. Scaling up: an approach to organizational agile adoption, Agile Journal. <www.agilejournal.com>.
Pettit, R., 2006c. An agile approach to IT governance, Agile Journal. <www.agilejournal.com>.
Qumer, A., 2007. Defining An Integrated Agile Governance for Large Agile Software Development Environments XP, 2007. Springer LNCS.
Qumer, A., Henderson-Sellers, B., 2006a. Measuring agility and adoptability of agile methods: a 4-dimensional analytical tool. In: Guimarães, N., Isaias, P., Goikoetxea, A. (Eds.), Proceedings of the IADIS International Conference Applied Computing. IADIS Press, pp. 503–507.
Qumer, A., Henderson-Sellers, B., 2006b. Comparative evaluation of XP and Scrum using the 4D Analytical Tool (4-DAT). In: Irani, Z., Sarikas, O.D., Llopis, J., Gonzalez, R., Gasco, J. (Eds.), Proceedings of the European and Mediterranean Conference on Information Systems 2006 (EMCIS2006). CD, Brunel University, West London.
Qumer, A., Henderson-Sellers, B., 2006c. Crystallization of agility: Back to basics. Proceedings of the First International Conference on Software and Data Technologies, vol. 2. INSTICC Press, pp. 121–126.
Qumer, A., Henderson-Sellers, B., 2006d. A framework to support non-fragile agile agent-oriented software development. In: Fujita, H., Mejri, M. (Eds.), Frontiers in Artificial Intelligence and Applications. New Trends in Software Methodologies, Tools and Techniques. In: Proceedings of the Fifth SoMeT_06, IOS Press, pp. 84–100.
Qumer, A., Henderson-Sellers, B., 2007a. An evaluation of the degree of agility in six agile methods and its applicability for method engineering. Inf. Software Technol. doi:10.1016/j.infsof.2007.02.002.
Qumer, A., Henderson-Sellers, B., 2007b. Construction of an agile software product enhancement process by using an agile software solution framework (ASSF) and situational method engineering. In: Ceballos, S. (Ed.), Proceedings of the 31st Annual International Computer Software and Applications Conference, Beijing, China, 23–27 July 2007, vol. 1. IEEE Computer Society, pp. 539–542.
Qumer, A., Henderson-Sellers, B., 2007c. An agile toolkit to support agent-oriented and service-oriented computing mechanisms. In: Munch, J., Abrahamsson, P. (Eds.), Product-Focused Software Process Improvement 8th International Conference, Profes 2007, Riga, Latvia, July 2007 Proceedings, LNCS 4589. Springer-Verlag, Berlin, Germany, pp. 222–236.
Qumer, A., Henderson-Sellers, B., 2007d. An Agile Service-Oriented Process. In: Proceedings of the SOMET 2007.
Qumer, A., Henderson-Sellers, B., McBride, T., 2007. Agile adoption and improvement model. In: Proceedings of the EMCIS2007. The Information Institute, Brunel University.
Rus, I., Lindvall, M., 2002. Knowledge management in software engineering. IEEE Software 19 (3), 26–38.
Salo, O., Abrahamsson, P., 2007. An iterative improvement process for agile software development. Software Process Improvement and Practice 12, 81–100.
Schwaber, K., Beedle, M., 2002. Agile Software Development with SCRUM. Prentice Hall.
Sisco, M., 2002. Technology Review is the Core of IT Assessment. TechRepublic.
Sliger, M., 2006. A project manager's survival guide to going agile. Rally Software Development Corporation.
Smits, H., 2006. 5 Levels of Agile Planning: from Enterprise Product Vision to Team Stand-up. Rally Software Development Corporation.
Standards Australia, 2004. Standard metamodel for software development methodologies, AS 4651-2004. <www.standards.com.au/>.
Syed-Abdullah, S., Holcombe, M., Gheorge, M., 2007. The impact of an agile methodology on the well being of development teams. Empirical Software Engineering 11, 145–169.
Webb, P., Pollard, C., Ridley, G., 2006. Attempting to define IT governance: wisdom or folly? In: Proceedings of the 39th Hawaii International Conference on System Sciences, IEEE, USA, Track 8, p. 194a.
Weill, P., 2004. Don't just lead, govern: How top-performing firms govern IT. MIS Quarterly Executive 3 (1).
Weill, P., Broadbent, M., 1998. Leveraging the new infrastructure: how market leaders capitalize on IT. Harvard Business School Press, Boston.
Weill, P., Ross, J.W., 2004. IT Governance on One Page, CISR WP, p. 349.
Yin, R., 1994. Case Study Research: Design and Methods, second ed. Sage Publishing, Beverly Hills, CA.
Yin, R.K., 2003. Case Study Research, Design and Methods, third ed. Sage Publications, Newbury Park.

Brian Henderson-Sellers is Director of the Centre for Object Technology Applications and Research and Professor of Information Systems at the University of Technology, Sydney (UTS). He is author or editor of 28 technical books, the co-editor of the ISO standard 24744 ("SE Metamodel

for Development Methodologies") and is Editor of the *International Journal of Agent-Oriented Software Engineering* and on several editorial boards. In July 2001, Professor Henderson-Sellers was awarded a Doctor of Science (DSc) from the University of London for his research contributions in object-oriented methodologies.

Asif Qumer is a Ph.D., Student (Software Engineering) in the Centre for Object Technology Applications and Research, Faculty of Information Technology, University of Technology, Sydney, studying agile and agent methods in the context of method engineering and quality assessment. He has been working in the software industry for more than five years.