Standards and Agile Software Development

W.H. MORKEL THEUNISSEN, DERRICK G. KOURIE AND BRUCE W. WATSON
ESPRESSO Research Group
Department of Computer Science
University of Pretoria

This paper investigates the adaptability of agile methodologies to software development standards laid down by ISO. On the basis of this analysis, guidelines are proposed for developing software in such a way that the development process conforms to the standards required by the software acquirer, while still retaining its agile characteristics. Extreme Programming (XP) is used as the primary representative of the various agile methodologies.

Categories and Subject Descriptors: D.2.9 [**Software Engineering**]: Management—*Life cycle*

General Terms: Documentation, Standardization

Additional Key Words and Phrases: Agile methodologies, software development, extreme programming, ISO/IEC 12207:1995

1. INTRODUCTION

In recent times, *agile* software methodologies (also known as lightweight methodologies) have been used increasingly in projects and, one may even say, are being accepted increasingly by software developers. These methodologies have raised considerable debate between the big-design-upfront (BDUF) followers and the agile followers. The crux of the debate appears to be concern over the lack of documentation that is to be generated as part of an agile development process. Coupled with this lack of documentation is the question of proof of compliance with accepted software engineering standards. The fact is that some projects and/or customers require standards to be followed when developing software. This may be due to organizational, managerial or regulatory needs.

Compliance with standards usually entails the generation of documentation. This is in apparent contradiction to the agile principle of “working software over comprehensive documentation” [Fowler & Highsmith 2001; Agile Manifesto URL]. It should be noted that the agile movement is not against documentation *per se* but rather, against the overemphasis on writing documentation that provides no real value to a project’s main goal of delivering effective software.

Fortunately agile methodologies are, by definition, highly adaptable and are thus able to comply with standards when required. However, it would seem that there are almost no guidelines for incorporating into agile methodologies, processes that ensure their compliance with specified standards. This paper suggests a few such guidelines, based on an analysis of currently used ISO software standards. Since Extreme Programming (XP) [Beck 2000] is perhaps the best known and most widely used agile methodology, the discussion below will focus on it as a representative of the various agile methodologies. This means that whenever there is a need to refer to some specific instance of an agile feature or practice, then the way this feature is realised in XP will be cited. Other agile methodologies such as Crystal [Cockburn 2002] can be adapted in a similar manner.

Section 2 will highlight ISO standards that are of interest to software developers. A deeper investigation of some of these standards, and guidelines for using the relevant standards are provided in Section 3.

Authors Address: WHM Theunissen, ESPRESSO Research Group (http://espresso.cs.up.ac.za), Department of Computer Science, University of Pretoria, Pretoria, 0002, South Africa; mtheunis@cs.up.ac.za
DG Kourie, ESPRESSO Research Group (http://espresso.cs.up.ac.za), Department of Computer Science, University of Pretoria, Pretoria, 0002, South Africa; dkourie@cs.up.ac.za
BW Watson, ESPRESSO Research Group (http://espresso.cs.up.ac.za), Department of Computer Science, University of Pretoria, Pretoria, 0002, South Africa; watson@cs.up.ac.za

2. STANDARDS THAT ARE OF INTEREST

The most important ISO standards applicable to software development are ISO/IEC 12207:1995 and its replacement ISO/IEC 15288:2002, both referring to the *Software life cycle processes.* This paper focuses on the ISO/IEC 12207:1995 standard, since this is currently used in industry. The ISO/IEC 15288:2002 standard was only approved by ISO in October 2002 and is still under consideration by local standards bodies such as South African Bureau of Standards (SABS). At the time of writing, the standard was not generally available to the public.

Other standards that are also of interest to software development are ISO/IEC 15939:2002 (Software measurement process) and ISO/IEC 14143 (Software measurement - Functional size measurement). Although these standards are used in support of software development they are not directly relevant to the present discussion and will not be further considered here.

2.1 ISO/IEC 12207:1995

It should be noted that this section will rely on definitions in ISO/IEC 12207:1995 when referring to certain terms. Where needed the definition of a term will be given in a footnote. ISO/IEC 12207:1995 defines a *software product* as "The set of computer programs, procedures, and possibly associated documentation and data."; It defines a *software service* as the "Performance of activities, work, or duties connected with a software product, such as its development, maintenance, and operation.". A *system* is defined as "An integrated composite that consists of one or more of the processes, hardware, software, facilities and people, that provides a capability to satisfy a stated need or objective."

The ISO/IEC 12207:1995 standard defines a framework for software life cycle processes, spanning all stages from the initiation stage through to the retirement stage of software. The framework is partitioned into three *areas*: primary life cycle processes; supporting life cycle processes and organizational life cycle processes. Relevant *subprocesses* in each of these areas are identified and various *activities* are specified for each subprocess. The subprocesses and their associated activities are given in Tables I to III.

2.1.1 *Compliance*

Compliance with the ISO 12207:1995 standard "is defined as the performance of all the processes, activities, and tasks selected from this International Standard in the Tailoring Process ... for the software project." [ISO 12207:1995].

This so-called tailoring process is discussed in an annex to the standard itself and is to be used to customise ISO 12207:1995 to a specific project. The process starts off by identifying the characteristics of the project environment. These may include the team size, organizational policy and project criticality. Next, the process requires that all the affected stakeholders of the project should be consulted on the way in which the ISO 12207:1995 process should be tailored. Based on this consultation, the processes, activities and tasks that will be followed during the project should be selected. The selection should also take into consideration the processes, activities and tasks that are *not* specified in ISO 12207:1995 but which nevertheless form part of the contract. The selection activity should also document who will be responsible for each process, activity and task. Finally all the tailoring decisions should be documented, and explanations for the relevant decisions should be noted.

The foregoing implies that if an organization prescribes conformance to ISO 12207:1995 as a requirement for trade, then that organization holds the responsibility of specifying what will be considered as the minimum required in terms of processes, activities and tasks, in order to conform with the standard. What is to constitute compliance may be further refined and negotiated when defining the contract between the acquirer and supplier.

3. THE AGILE ANGLE

As previously stated, the focus here is on ISO 12207:1995 since it is the most relevant ISO standard in regard to software development. This present section motivates and provides guidelines for implementing the standard in an agile context. The discussion will focus on extreme programming (XP) as a typical example of the agile methodologies. The following question is addressed: "Can agile methodologies be implemented or extended in such a way that they conform to ISO 12207:1995 but still retain their agile characteristics?" To the authors's knowledge there is no literature that directly addresses this question. Nevertheless, we take the view that the question can be answered affirmatively. In support of this view, implementation guidelines are proposed that will ensure that an agile-based project conforms to ISO 12207:1995. These guidelines are derived from an analysis of the standard on the one hand, as well as from an analysis of the characteristics of the agile methodologies on the other.

The task of implementing an agile methodology in such a way that it conforms to ISO 12207:1995 can be viewed from two perspectives.

1. From the ISO standard's view the standard should first be *tailored* to meet the requirements of the project. This tailoring may involve all the parties but demands the special attention of the acquirer. The "tailored" standard should then be mapped to the development methods and processes that are used.

2. From the XP perspective, the methodology itself requires that its processes should be customised to comply with the project requirements. The same holds true for Crystal. This means the methodology inherently requires that it should be adapted to support the needs of the project. In the present context, this means that the methodology should be adapted to comply with the ISO standard. It should be noted that in both XP and Crystal, conforming to a standard is regarded as a system requirement specification in itself and is treated as such.

The discussion below refers to two of the three areas mentioned in the framework: the area dealing with primary life cycle processes (Table I); and the area dealing with supporting life cycle processes (Table II). Only the third of the various primary life cycle subprocess, namely the *development subprocess*, is relevant to the present discussion. Its activities are supplemented by the activities of each of eight supporting life cycle subprocesses in Table II. The standard itself contains various clauses that elaborate in greater detail than provided by the tables above on what should happen in order for these various subprocesses to be realised.

Sections 1 to 4 of ISO 12207:1995 mearly describes the standard and the document itself, whereas sections 5, 6 and 7 provides the prescriptions of the standard and are summarised in Tables I, II and III respectively.

General comments that are broadly applicable to multiple clauses of the standard are first given below (Section 3.1). Then, Section 3.2 considers specific clauses and proposes guidelines to meet their requirements. Only clauses that relate to the development subprocess of Table I, or that relate to relevant subprocesses in Table II are discussed. Finally, Section 3.3 addresses the matter of incremental documentation.

3.1 General comments

One broad approach for ensuring that an agile development team conforms to the ISO:1995 standard is to assign the task of providing that assurance to one or more individuals. In essence, then, this person enables the team to produce the necessary artifacts in the manner required of the standard.

To ensure that the developers, the programmers specifically, are isolated from the burden of documentation and administrative tasks that are needed to conform to the standard, it is suggested that an organizational model similar to the one proposed by Brooks should be followed[Brooks 1995]. This model of a so-called *Surgical Team* consists of a surgeon (an expert in performing the design); a copilot (who follows the design and knows the alternatives); an administrator (who manages all the administrative issues such as resources and legalities); an editor ('translates' the surgeon's documentation for general usage); an administrator's secretary; an editor's secretary; a program clerk(who maintains changing artifacts through version and configuration control); a tool smith (who is an expert on development tools); a tester; and a language lawyer (who is an expert on programming language usage and other specifications).

The above model is for the organization of a development team and was proposed in the 1960s. Although it might seem inappropriate for some of the contemporary software development projects, it does suggest a few useful ideas. The particular point worth considering is the notion that the programmers should be isolated from administrative tasks and from generating documentation. In an agile project this means that the individuals assigned to ensure compliance with the selected ISO standards should generate the necessary documentation as required by these standards without burdening the developers. Thus the documentation sub-team should solicit the required information and data in a manner that is as non-intrusive as possible. As an example, when implementing this proposed model in an XP context, it is suggested that a 'standard-conformance' sub-team should be assigned as part of the development team. The members of this sub-team should be co-located in the same room as the programmers and the on-site customers in order to easily solicit information that is required for documentation. They may do so informally through normal everyday communication, and also more formally through attending XP required interactions such as regular white-board discussion sessions. Information should also be gathered from the test-cases developed and through the use of software tools that can extract relevant information directly from the source code that has been developed. Where possible, the documentation should be incorporated as part the source code to conform with the XP principles stating that the source code should be the main source of information regarding the development effort.

These personnel arrangements represent a general way to enable an agile development team to act in compliance with the ISO standards. However, also in general sense, an agile development team will quite naturally comply with a substantial part of the ISO standard's requirements, merely by virtue of following the agile methodology itself. In particular, consider the eight supporting life cycle subprocesses mentioned in Table II. The next section will discuss a number of specific clauses relating to the first three subprocesses (documentation, configuration

Primary Life Cycle Processes				
Acquisition subprocess	Supply subprocess	Development subprocess	Operation subprocess	Maintenance subprocess
responsibility of the *acquirer*[a] .	responsibility of the *supplier*[b] .	done by the *developer*[c] .	done by the *operator*.	responsibility of the *maintainer*.

Activities:
—initiation;
—request-for-proposal [-tender] preparation;
—contract preparation and update;
—supplier monitoring;
—acceptance and completion.
	Activities:
	—initiation;
	—preparation of response;
	—contract;
	—planning;
	—execution and control;
	—review and evaluation;
	—delivery and completion.
		Activities:
		—process implementation;
		—system requirement analysis;
		—system architectural design;
		—software requirements analysis;
		—software architectural design;
		—software detailed design;
		—software coding and testing;
		—software integration;
		—software qualification testing;
		—system integration;
		—system qualification testing;
		—software installation;
		—software acceptance support.
			Activities:
			—process implementation;
			—operational testing;
			—system operation;
			—user support.
				Activities:
				—process implementation;
				—problem and modification analysis;
				—modification implementation;
				—maintenance review/acceptance;
				—migration;
				—software retirement.

Table I. Primary Life Cycle Processes

[a] "An organization that acquires or procures a system, software product or software service from a supplier." [ISO 12207:1995]

[b] "An organization that enters into a contract with the acquirer for the supply of a system, software product or software service under the terms of the contract." [ISO 12207:1995]

[c] "An organization that performs development activities (including requirements analysis, design, testing through acceptance) during the software life cycle process." [ISO 12207:1995]

Supporting Life Cycle Processes							
Documentation subprocess	Configuration management subprocess	Quality assurance subprocess	Verification subprocess	Validation subprocess	Joint review subprocess	Audit subprocess	Problem resolution subprocess
Specifies how information generated by the life cycle process should be recorded.	Addresses the administrative and technical procedures for the life cycle.	Ensures that the software conforms to the specifications as defined by the plans.	Used to confirm that the software products satisfy the requirements defined.	Confirms that the final system satisfy the intended use.	Used to assess the activities in a project.	Ensures conformity to the requirements, plans and contract for the system.	Used to analyze and resolving problems that are encountered during any process.
Activities: —process implementation; —design and development; —production; —maintenance.	**Activities:** —process implementation; —configuration identification; —configuration control; —configuration status accounting; —configuration evaluation; —release management and delivery.	**Activities:** —process implementation; —product assurance; —process assurance; —assurance of quality systems.	**Activities:** —process implementation; —verification.	**Activities:** —process implementation; —validation.	**Activities:** —process implementation; —project management reviews; —technical reviews.	**Activities:** —process implementation; —audit.	**Activities:** —process implementation; —problem resolution.

Table II. Supporting Life Cycle Processes

Organizational Life Cycle Processes			
Management subprocess	Infrastructure subprocess	Improvement subprocess	Training subprocess
The activities that may be used by any party on any process to manage it.	Used to set and maintain the infrastructure required for the other processes.	Enables the ability of improving a software life cycle process.	Sustaining trained personnel in all parties.

Activities:
—initiation and scope definition; planning;
—execution and control;
—review and evaluation;
—closure.
	Activities:
	—process implementation;
	—establishment of the infrastructure;
	—maintenance of the infrastructure.
		Activities:
		—process establishment;
		—process assessment;
		—process improvement.
			Activities:
			—process implementation;
			—training material development;
			—training plan implementation.

Table III. Organizational Life Cycle Processes

management and quality assurance) as well as to the last subprocess (problem resolution). But a cursory examination of the remaining four subprocesses in Table II will reveal that the issues which they address are, by and large, inherently built into the very substance of the agile methodology itself.

Thus, for example, *verification*[1] and *validation*[2] is inherently addressed by the agile practice of writing code specifically to meet test cases – i.e. test cases are set up prior to coding and code is not accepted until it has been verified against the test cases. Usually verification is done through unit test-cases and validation through functional test-cases.

The notion of *joint reviews* of project activity is strongly built into XP by virtue of requirements that enforce regular planning sessions, that insist on pair programming, that demand the rotation of coding tasks (such that code belongs to no particular person but is constantly reviewed by fresh pairs of developers), that ensure the availability of an on-site customer representative, etc. These kinds of arrangements ensure that continuous joint reviews of various levels of activity and at various levels of detail take place on an ongoing basis within an XP project as a natural outflow of the methodology itself.

In a broad sense, *auditing* can be regarded as a process whereby some independent agent reviews the activities of the audited party, reporting on the findings in a fairly formal fashion. The need for auditing typically arises in a context where activities are carried out in an independent and/or private fashion. But in an XP context, this is the very antithesis of the way in which code is developed. Instead, code is seen as belonging to the entire development team. Everyone has access to all code and the practice of rotating coding tasks ensures that different team members are constantly exposed to, and indeed becoming intimately familiar with, code developed by others. Of course, if the client has additional auditing requirements (e.g. stemming from concerns about possible collusion in the development team), then there is nothing to prevent arrangements amongst the various parties to have code independently audited at various times, or indeed, to periodically insert a code auditor into the development team at different stages of the development.

3.2 Clause specific proposals

This section provides guidelines for ensuring that agile teams adhere to specific clauses from ISO 12207:1995.

The clauses that will be examined are from sections 5.3 and 6 of ISO 12207:1995. The former section relates to the development subprocess of the primary life cycle processes) and section 6 deals with various supporting life cycle processes. Clauses in ISO 12207:1995 that are not specifically mentioned here mainly relate to the business and organizational activities – for example, the various activities implied in Table III. They fall beyond the scope of the development activities discussed here.

The various clauses that are to be examined are shown as framed inserts. In each case the insert is followed by a discussion of the clause and suggested guidelines for bringing an agile methodology into line with the clause. Note that the clauses are examined from the development organization's perspective – i.e. from the perspective of the suppliers of the software product. Also note that whenever appropriate, documentation in regard to the results from and outputs of each of these clauses should be produced on the basis of the model proposed in Section 3.1.

5.3.4.2 The developer shall evaluate the software requirements considering the criteria listed below. The results of the evaluations shall be documented.

a) Traceability to system requirements and system design;
b) External consistency with system requirements;
c) Internal consistency;
d) Testability;
e) Feasibility of software design;
f) Feasibility of operation and maintenance.

XP explicitly takes the above into consideration during the *planning game*[3] with the aid of *spikes*[4] and in

[1] "Confirmation by examination and provision of objective evidence that specified requirements have been fulfilled" [ISO 12207:1995]. Thus, verification checks that the system executes correctly.

[2] "Confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use are fulfilled" [ISO 12207:1995]. In essence, validation checks that the system provide the functionality required to fullfill the needs of the acquirer.

[3] XP's jargon to describe the process of planning and their planning sessions.

[4] This is a quick programming or literature exploration of an issue that developers are unsure of.

collaboration with the *on-site customers*[5]. During the *planning game* the development team verifies the *user-stories* (the system requirements), ensuring not only their mutual consistency but also their overall feasibility. If the developers are unsure about a *user-story*'s characteristics, then they use the XP process called a *spike* to discover in greater detail what the user-story entails. The feasibility of the software requirements may therefore also be determined through spikes.

During implementation, the practice of *pair programming* obliges the non-typing programmer to evaluate the above mentioned criteria. In fact, one of the explicit roles of the non-typing programmer is to think strategically. By its very nature, such strategic thinking will invariably relate the current coding task to the overall system's requirements in terms of the foregoing criteria.

Furthermore, testability is addressed explicitly in terms of the test-driven development principle that is applied in XP.

Any other issues to emerge during development and that relates to the above clauses (e.g. in respect of traceability, external consistency etc.) should be discussed in the team context and, if necessary written as comments in the source code for other members to read and/or to be extracted by software tools used by the documentation sub-team.

5.3.5 Software architectural design. For each software item (or software configuration item, if identified), this activity consists of the following tasks:

5.3.5.1 The developer shall transform the requirements for the software item into an architecture that describes its top-level structure and identifies the software components. It shall be ensured that all the requirements for the software item are allocated to its software components and further refined to facilitate detailed design. The architecture of the software item shall be documented.

5.3.5.2 The developer shall develop and document a top-level design for the interfaces external to the software item and between the software components of the software item.

XP relies on the use of *metaphors*[6] to facilitate the architectural visioning of the system. Using metaphors helps the developers and customers to bridge the gap of understanding between the technical jargon of the developers and the business jargon of the customer. It does this by providing a medium to describe an unknown domain using a known domain as comparison, thus using a known domain's architecture as reference. For more concrete architectural design information, the documenter should capture information from the *planning game* and from *white-board* discussions.

Given that XP propounds the idea that the source code should be the documentation, it is natural that it would be well-disposed towards the use of technologies that extract user friendly documentation from source code. Examples of such technologies include Javadoc and model extraction from code. Of course, the deployment of such technologies will usually impose certain coding standards on the development team. For example, to use Javadoc effectively means to reach some agreement about where Javadoc comments will be inserted into the code (eg before all public methods), what the nature of those comments should be (eg a statement of all preconditions in terms of public methods and public instance fields), the Javadoc parameters to be provided, etc.

Architectural documentation may be complemented by the information generated from source code whereas design documentation can be mostly derived from the source code. One of the advantages of generating documentation from the code is that the documentation is up to date and a true reflection of the system.

[5]A customer representative that actively partake in the development effort. This representative is co-located with the development team.
[6]"A story that everyone - customers, programmers and managers - can tell about how the system works" [Beck 2000]

6.1 **Documentation process**
The Documentation Process is a process for recording information produced by a life cycle process or activity. The process contains the set of activities, which plan, design, develop, produce, edit, distribute, and maintain those documents needed by all concerned such as managers, engineers, and users of the system or software product.
List of activities. This process consists of the following activities:

a) Process implementation;
b) Design and development;
c) Production;
d) Maintenance.

The standard "only tasks the development process to invoke the documentation process"[IEEE 12207.2-1997]. This means that each of the organizations (the acquirers; suppliers and/ or other 3rd parties) should decide on how to use and implement the documentation process. Being able to define how to implement the process enables the development organization (supplier) to use agile methodologies and adapt them as needed.

The documentation needed is agreed upon and defined during the tailoring process. What is to be documented is therefore contracted between the acquirer and supplier. This aquirer-required documentation, together with documentation that the development organization requires for internal purposes, jointly constitute the set of documentation that is to be generated.

6.2.3 Configuration control. This activity consists of the following task:

6.2.3.1 The following shall be performed: identification and recording of change requests; analysis and evaluation of the changes; approval or disapproval of the request; and implementation, verification, and release of the modified software item. An audit trail shall exist, whereby each modification, the reason for the modification, and authorization of the modification can be traced. Control and audit of all accesses to the controlled software items that handle safety or security critical functions shall be performed.

Nowadays there is a diverse range of configuration management software tools that enable development teams to meet the standard. They include Rational's ClearCase® and Visible's Razor. The full list of available products is too extensive to state here. Configuration control is an integral part of most development organizations and is a widely accepted practice. It should be noted that using these tools does not automatically ensure conformance to ISO 12207:1995 – the tool should be used in such a way that the outcome specifically conforms to the requirements of ISO 12207:1995.

6.3 Quality assurance process

The Quality Assurance Process is a process for providing adequate assurance that the software products and processes in the project life cycle conform to their specified requirements and adhere to their established plans. To be unbiased, quality assurance needs to have organizational freedom and authority from persons directly responsible for developing the software product or executing the process in the project. Quality assurance may be internal or external depending on whether evidence of product or process quality is demonstrated to the management of the supplier or the acquirer. Quality assurance may make use of the results of other supporting processes, such as Verification, Validation, Joint Reviews, Audits, and Problem Resolution.

List of activities. This process consists of the following activities:

a) Process implementation;
b) Product assurance;
c) Process assurance;
d) Assurance of quality systems.

Quality Assurance (QA) is built into XP through the use of functional-, acceptance- and unit tests as well as through the presence of an on-site customer. It is explicitly part of the role description of the on-site customer to ensure that the development team delivers software that meets the requirements of the acquirer by defining and carrying out acceptance tests and by doing reviews during development.

It would appear, therefore, that XP already does much to ensure quality. Although the developers on the team could hardly be regarded as unbiased (and therefore as agents for formally carrying out QA), it may sometimes be appropriate to consider the on-site customer as an appropriate "external" authority. In some contexts, the extent to which on-site customer may be considered unbiased may however be limited. For example, it would be quite natural for a congenial relationship between the on-site customer and the development team to be built up over time. In as much as it may be considered necessary to do more than merely provide for an on-site customer in order to conform to the standard, a person or sub-team could be assigned to coordinate the QA in conjunction with the on-site customer. For an even greater assurance of impartiality, a totally independent team may be assigned to verify the quality, without any reference at all to the on-site customer.

6.8 Problem resolution process

The Problem Resolution Process is a process for analyzing and resolving the problems (including non-conformances), whatever their nature or source, that are discovered during the execution of development, operation, maintenance, or other processes. The objective is to provide a timely, responsible, and documented means to ensure that all discovered problems are analyzed and resolved and trends are recognized.

List of activities. This process consists of the following activities:

a) Process implementation;
b) Problem resolution.

Although XP does not have a specific process labelled "Problem Resolution", its short iterative development cycles and its high-intensity team interaction processes naturally lead to early problem detection and resolution. Furthermore, it is a natural feature of XP to support change, including changes required to resolve problems. For example, accepting changes to the requirements is part of the *steering* phase of the planning game. However, XP does not make explicit provision for documenting the problems encountered nor for documenting the changes made to resolve those problems. Such documentation, if required by the acquirer, should be referred to the documentation sub-team as proposed in Section 3.1.

3.3 Incremental Documentation

The guidelines [IEEE 12207.1-1997; IEEE 12207.2-1997] to ISO 12207:1995 state that when development is incremental or evolutionary the documentation generation may also be incremental or evolutionary. This statement is important to agile methodologies in general, and to XP in particular, since the system evolves over the duration of development without the big upfront design that many other methodologies use. Generating documentation incrementally also supports the idea of generating the documentation from the source code, since the source code itself is generated incrementally. Thus, the use of tools to generate documentation from source code has the benefit of ensuring that the documentation is up to date and of therefore allowing the documentation to reflect the true nature of the software.

4. CONCLUSION

The guidelines provided in this paper are not intended to be exhaustive. Rather, they provide a starting point for agile developers who are required to comply with ISO 12207:1995. The guidelines should of course be tested in practice and should be further customised for specific project needs.

It has been argued above that agile methodologies can indeed be adapted to ensure compliance with a standard such as ISO 12207:1995. In doing so, care should be taken to ensure that the development organization abides by the agile manifesto principle of "working software over comprehensive documentation". The previous two statements might seem to contradict one another, but herein lies the heart of the tension that under discussion. The agile principle of stressing the development of working software rather than huge volumes of documentation need not be a rejection of producing documentation *per se*. Rather, it represents a shift of focus to what software development really is all about – the production of software. In situations where documentation is required, this requirement then becomes a system requirement and should be treated as such. The acquirer needs to understand that extensive documentation increases resource utilisation. This translates in turn into higher cost and perhaps slower delivery of the system. Where possible, the development team should use tools that automate the generation of documentation to reduce the resource utilisation. These tools should rely on source code as input, because source code is the most accurate representation of the software product. However, it should be realised that the effective use of such tools may well impose certain coding standards upon the development team, and these standards will have to be agreed upon and followed.

ACKNOWLEDGMENT

This research is funded by Technology and Human Resources for Industry Programme (THRIP).

REFERENCES

BECK, K., 2000, *Extreme Programming Explained*, Addison-Wesley.

BROOKS, JR., F. P., 1995, *The Mythical Man-Month: Essays on Software Engineering*, Anniversary Edition, Addison-Wesley.

COCKBURN, A., 2002, *Agile Software Development*, Pearson Education Inc.

FOWLER M. & HIGHSMITH J., August 2001, *The Agile Manifesto*, Software Development Magazine.

NATIONAL COMMITTEE TC 71.1 (INFORMATION TECHNOLOGY), 1995/ 08/01, *SABS ISO/IEC 12207:1995, Information technology - Software life cycle processes.*

IEEE & EIA , April1998, *IEEE/EIA 12207.1-1997, IEEE/EIA Guide: Industry Implementation of International Standard ISO/IEC 12207:1995.*

IEEE & EIA , April 1998, *IEEE/EIA 12207.2-1997, IEEE/EIA Guide: Industry Implementation of International Standard ISO/IEC 12207:1995.*

Agile Alliance, 2001, *Manifesto for Agile Software Development*, http://www.agilemanifesto.org (2003/02/03)