Standards Compliant Software Development*

Wolfgang Emmerich, Anthony Finkelstein
Interoperable Systems Research Centre
City University
London EC1V 0HB, UK
{emmerich | acwf}@cs.city.ac.uk

Carlo Montangero
Dip. di Informatica
University of Pisa
56125 Pisa, Italy
monta@di.unipi.it

Richard Stevens
QSS Ltd.
Oxford Science Park
Oxford OX4 4GA, UK
100010.3303@compuserve.com

Abstract

Software engineering standards determine practices that 'compliant' software processes shall follow. Standards generally define practices in terms of constraints that must hold for documents or the information in these documents. The document types identified by standards include not only typical development products, such as user requirements, but also process-oriented documents, such as progress reviews and management reports. The degree of standards compliance can be established by checking these documents against the constraints. It is is neither practical nor desirable to enforce compliance at all points in the development process, thus compliance must be managed rather than imposed. In this paper we define a model of standards and compliance. We then outline a lightweight implementation of the model built on a generic document management system. Finally, we discuss the broader implications of our work for process modelling and the management of inconsistent information.

1 Compliance

In this section we outline the general problem of managing standards compliance in software development and motivate the development of automated support for this activity.

There is intense interest in adopting standards in software development. This interest arises for a number of reasons: standards have been a means of transferring 'good practice' in software engineering; standards have been demanded by clients or procurement agencies; the demands of SPI initiatives, ISO 9000 and SPICE certifications; product certification requirements.

In each case once a standard has been adopted it is important to manage compliance with the standard. By compliance we mean the extent to which software developers have acted in accordance with the practices set down in the standard. More narrowly we can think of this as consistency between the actual development process and the normative models embedded in the standard.

Existing, well established standards, such as ISO 12207 [ISO/IEC, 1995] and PSS05 [Mazza et al., 1994] set down the properties that both the process and the product must possess at given points in the development. Such standards are both large and complex, they are often incomplete and ambiguous. Determining compliance, particularly as development progresses and the information can be used to support correction, is thus a challenging task.

In our approach, described below, we take advantage of an important feature of the standards we have examined. They tend to express the requirements of the standard in the form of 'practices' which in turn are usually expressed as constraints on the structure or content of documents.

*This work was funded through the Teaching Company Directorate through Scheme No. 1884 and performed while Carlo Montangero was an EPSRC Visiting Fellow at City University funded through Grant No. GR/L54561

We adopt what might be termed a 'tolerant' approach in which developers are free to organise for themselves the way they reach the goals set by project management. They are provided with ways to assess where they are with respect to their duties to conform to the practices. Policies set down the points at which different sorts of compliance should be established. Such policies can however be overridden by an appropriately authorised developer who can postpone or even renounce compliance.

In our preliminary work on compliance we have focussed on requirements management. We have done so because it is a document-intensive activity of critical importance in software development. More importantly, because requirements processes cross organisational boundaries, common standards and compliance play a particularly significant role.

In section 2 we describe the model of standards and compliance which we have adopted and illustrate this with a simple example. In section 3 we show how this model integrates with an industrial-strength document management environment. Section 4 sets out some important pieces of related work. In section 5 we outline a research agenda suggested by our initial work. We conclude with some general observations on process modelling and the management of inconsistent information.

2 Model

This section outlines the model of standards and compliance underlying our approach. Figure 1 shows an entity-relationship diagram which summarises the principal elements. There are three parts to the model: the first (top part of the figure) shows the domain in which we want to intervene; the other two parts show the support for compliance management. We separate user-driven support for compliance, in which checks – mechanisms to assess the current state of compliance – are introduced but the responsibility to exploit these mechanisms is left to the developers, from policy-driven support in which policies govern the application of the checks and determine the actions in the face of potential non-compliance.

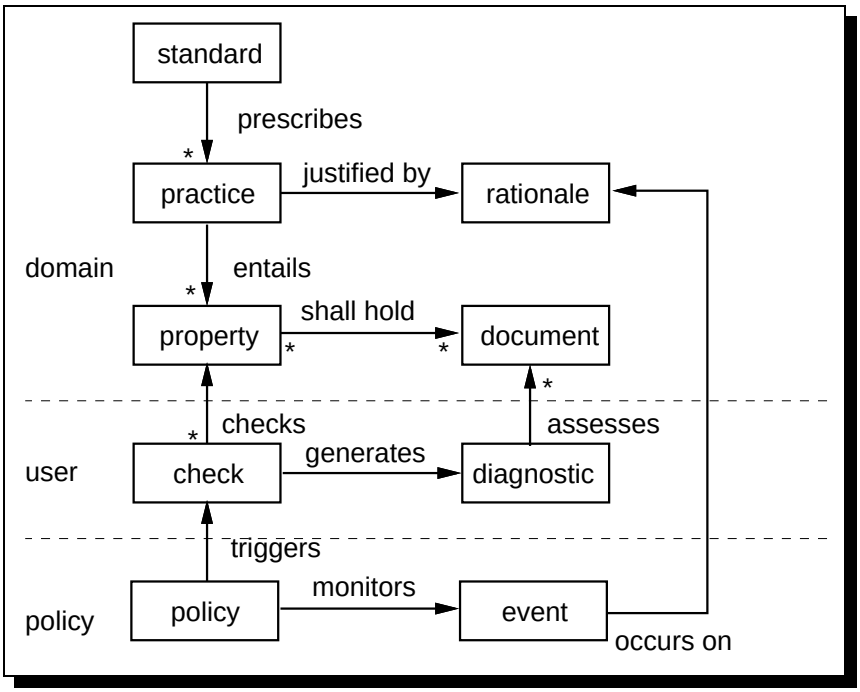

Figure 1: Standards and Compliance Model

As discussed above, in order to express their requirements on the development process, software development standards tend to prescribe a number of practices to be followed - in this section, word in this font

denote entities or relationships in Figure 1. Since standards do not aim to define processes precisely, they usually leave ample room for tailoring of the actual processes, within the constraints they lay down. The distinction between mandatory and recommended practices, common to most standards, is one way of supporting this tailoring. For our purposes the distinction is irrelevant: we want to handle all the practices that the process owner demands compliance with.

PSS05, for example, lists almost 200 practices, counting only mandatory practices. A typical practice, taken from PSS05 is the following:

> UR04 - For incremental delivery, each user requirement shall include a measure of priority so that the developer can decide the production schedule.

Aside from the identifier - UR04 - it is easy to recognise two parts to the practice: a rationale - so that the developer can decide the production schedule - and a compliance requirement - for incremental delivery, each user requirement shall include a measure of priority.

Practices do not always have an explicit rationale, many standards tend to exclude rationale in favour of conciseness. Obviously, however, practices get into standards only after they have been proven effective. We think it important, for user guidance, that rationale is available to motivate compliance, though users may choose not to view it.

A compliance requirement is an intrinsic part of any practice, and in many cases, as in UR04, it entails that a given predicate on the product of the process shall hold at some point. We highlight the static facet of a practice in the model, the property of interest, since this is the basis for defining checks. We are less interested in modelling the dynamics of the process, the *shall* part. So, in our example, we have the property:

> For incremental delivery, each user requirement *includes* a measure of priority

We aim to provide support to the user to assess the current state of compliance with respect to this property. Some careful reading of the standard allows us to discover that the property entailed by UR04 shall hold for a specified document, namely the User Requirement Document (URD). Generally, properties of the state of the product are associated with one or more documents, so the shall hold relation allows us to limit the domain of applicability of the checks.

It should be noted that not all the practices obviously define compliance requirements on the product. For instance, UR10 states:

> UR10 - An output of the User Requirements phase shall be the URD.

This is, on the face of it, a constraint on the process. We believe that these constraints can be readily expressed as constraints on the product, by considering with more care those management documents, such as project plans and progress reports, that capture the essential features of the dynamics of the process. These documents, which actually constitute a large proportion of the the documents produced during software development, have up to now received little attention in research on software process support. As an example, UR10 might entail the following property:

> The Software Management Plan for the User Requirements phase includes a task or work-package for the construction of URD.

A similar argument applies to the conditional clause in UR04 - for incremental delivery. This condition on the state of the process, which relates to the overall strategy of the project in PSS05, can be transformed into a condition on the product, in a straightforward manner: the general description of the project, in the Software Management Plan, shall include a 'project mode' attribute, which may take as value, among others, 'incremental delivery'. The property in UR04 then becomes:

> If the project mode in the project general description of the Software Management Plan is 'incremental delivery', each user requirement shall include a measure of priority so that the developer can decide the production schedule.

Before moving to the bottom part of Figure 1, to consider the mechanisms for user-driven support, we recall that not all the relations in Figure 1 are one-to-one: a standard usually recommends many practices, and some practices may entail several properties which in turn may refer to several documents. Obviously, a document may be affected by more than one practice, and therefore required to satisfy many properties.

The basic mechanism to support the user is the check, which has two goals: firstly, to evaluate the property it relates to in the current state, and, second, in the case that the property does not hold, to generate some diagnostic information, to help the user assess the extent of compliance of the document(s) of interest.

For example, the check for UR04 might produce the diagnosis 'not relevant' where the property trivially holds (because the current project is not incremental delivery), or the list of the requirements for which priority is undefined, and the percentage of non-compliant requirements. This information would allow the user to assess the importance and the difficulty of making the URD compliant. At the same time, the percentage of compliance of this and other similar properties, gives the manager a measure of the status of the project.

Even the best motivated user may fail to apply all the checks that are needed before some performing some sensitive action, such as baselining. Also, given the scale and complexity of the practices they may be uncertain of the best points to establish compliance. To ensure that no unintended breach of compliance occurs, we introduce policies, that trigger checks whenever some events occur on some documents. An event is essentially an attempt to perform an action on a document. We have identified three modes for policies:

- the *error* mode, in which failure of the check prevents the action from being performed, in which case the problem can be fixed using the diagnosis as support;
- the *warning* mode, that provides the user with the diagnosis but allows the user to perform the action and knowingly become non-compliant;
- the *guideline* mode that simply warns the user that it is advisable to perform a check but allows the user to perform the action.

The most useful mode, given our tolerant approach, is the warning mode. The others open the door to more varied conformance management: for example, besides providing strict compliance enforcement, the error mode might be useful when the fix is so simple that there is no point in letting the breach occur, and the guideline mode allows the introduction of discretionary practices and delivers fine grain guidance.

We close this section with some requirements on the language(s) for expressing the checks, that is properties and diagnostics, and the policies. It is important to understand that expressing the compliance requirements with the properties, the measures of discrepancy with the diagnostics, and introducing policies, is the responsibility of the compliance manager or process engineer. All the other people who are parties to the process, developers, managers and QA engineers, need not see this description. They may however be presented with some parts of it, notably properties, so there is a strong requirement for simplicity. To allow the compliance manager to construct or adapt the practices efficiently, the language should be as declarative as possible, strongly typed, and easily compiled into the extension language of the underlining support. The potential for success in achieving this goal lies in the restricted nature of the domain we are tackling; we are dealing with documents with a simple hierarchical structure, and with a limited number of meaningful events.

3 Support Environment

Standards, such as PSS05 and ISO 12207, are large and complex. Checks as to the compliance of actual processes with the prescribed practices are difficult to perform manually. A support environment is needed that checks compliance and delivers appropriate diagnoses.

Rather than developing an environment from scratch we are using and extending an existing system. We have chosen DOORS (Dynamic Object Oriented Requirements System), a generic document management system. DOORS is widely used in industry to manage requirements and management documents that are produced during system engineering processes. The choice of DOORS derives partly from the fact that requirements engineering is the domain of most interest to us. Also the DOORS extension facilities support the construction of a lightweight standards compliance layer without having to rebuild many of the underlying document storage and query mechanisms. Most significantly DOORS has a large user base with an expressed interest in problems of compliance – we hope to build something that will actually be used!

DOORS documents are hierarchically composed from objects. Objects are used to store information needed for sections, subsections, down to individual paragraphs and titles. Every object in a DOORS document has the same attributes. Users can attach attributes to objects by defining a name and a type during editing sessions and then create and/or display values of these attributes. For the implementation of compliance checks, we would be able to use attributes already available for objects or, if necessary, attach new attributes to document objects in order to enable users to provide information requested by the standard. We would, for instance, attach an attribute for storing a value indicating the importance of a requirement in order to implement UR04. DOORS also supports the concepts of links that can be used to relate one object to another and stored in a separate object. Links are used, for instance, to capture requirements traceability information.

DOORS has a Dynamic eXtension Language (DXL) that can be used to automate tasks. DXL is an interpreted language. It includes imperative and rule-based language concepts. DXL functions can be attached to user interface components, such as pull-down menus. Functions are used to create template documents, whose structure and attributes correspond to those prescribed by certain standards. We intend to use functions to implement compliance checks. For the implementation of these checks, control flow primitives, such as iterations, attribute accesses and traversal across links are available.

DXL also provides the concept of triggers. Triggers are associations between events and actions. Triggers can be used to react to the occurrence of the event (post triggers), or to guard the event possibly in order to prevent it (pre triggers). DXL triggers seem to be an appropriate and flexible mechanism for the implementation of different compliance policies.

In order to implement compliance policies, both DXL functions and triggers will be used. Policies in which users initiate checks are implemented as functions that are included in the DOORS command and menu structure. Policies that require the execution of checks in a way unsolicited by users are implemented as triggers. A trigger is defined for events, such as closing a document or baselining the document and the function associated with the trigger would then be executed as soon as the event occurs. Some policies require a check before the event occurs. In these cases pre triggers can be used conveniently as an implementation. Post triggers may be used to introduce some 'forward chaining', that is to propagate the consequences of an event. The utility of this well known mechanism in our setting has yet to be assessed.

Many standards define document type structures and demand certain sections to be part of the document contents. Although inclusion of contents can be supported by functions that generate document templates in DOORS, users have the freedom to modify the template structure, hence violating the compliance to the standard. Likewise, in an interpreted and dynamically typed setting, users could detach attributes from objects and render any compliance checks meaningless. We would, therefore, need to include a number of meta checks into the compliance layer. These meta checks would ensure that attributes are available before they are accessed in compliance checks and that documents comply to the structure prescribed by standards.

4 Related Work

Our work draws on a number of intertwined strands of research. The problem of compliance as we have treated it is closely related to inconsistency management in specification. Key contributions in this area are [Finkelstein et al., 1994], [Easterbrook et al., 1994] and [Finkelstein et al., 1996]. The use of process modelling techniques to control the application of consistency checks have been explored in [Finkelstein et al., 1994] and [Leonhardt et al., 1995]. The problem of process divergence has been analysed by [Cugola et al., 1995] and [Cugola et al., 1996]. Our approach to representing policies is strongly influenced by work on Oikos [Montangero and Semini, 1996] and by interesting work on policy modelling in distributed systems management [Lupu and Sloman, 1997]. Other relevant work includes policy driven event monitoring [Fickas and Feather, 1995].

5 Research Agenda

Our immediate research agenda is defined by the preceding discussion. We must make good on our statements of intent. There are, however, some broader issues which remain to be tackled.

In addition to the practices considered above, standards incorporate high-level goals. The question of how we can establish that the practices correctly implement these high-level goals is one which needs an answer. Some preliminary work on such correctness problems has been developed in [Montangero and Semini, 1996].

Given a large project and a complex set of standards it is important for the compliance manager to be able to get a rapid, high-level view of the position with respect to compliance. Such a high level view might support planning and allocation of resources, identifying 'hot spots', and so on. Visualising compliance opens some potentially interesting research issues.

We would hope that the ideas on which our work is based can be fed back into the standards process itself and might assist in the formulation of new systems engineering standards, for example we are working on [ISO/IEC, 1997].

We are party to the shared research aim of building a better formal understanding of inconsistency, a contribution to this is [Spanoudakis and Finkelstein, 1997]. In particular we would hope that such work would yield a better understanding of diagnosis and opportunities to share tools and techniques.

6 Conclusions

There is a long tradition of research on software process technology, where the goal is to support software development through process-centred software engineering environments (PSEEs). However, most process modelling approaches tend to produce environments which are far too constraining, and which engineers experience as too prescriptive. The tendency to enforce compliance in an eager manner may be one of the reasons why PSEEs have experienced difficulties in being to put to practical use. It is widely accepted that software development requires more flexibility than business processes, where workflow tools have been instead largely accepted. Our work attempts to provide this flexibility, introducing lightweight mechanisms that support compliance management, rather than enforcement.

Previous work which attempts to introduce flexible guidance for example [Leonhardt et al., 1995], while delivering the basic fine-grain mechanisms, still lack adequate means of expressing global.

process requirements. We think that by focusing on standards, we are providing firm ground for bridging the gap. This is by no means a silver bullet. However, we anticipate significant benefits by providing many simple, highly syntactic compliance checks and providing mechanisms for managing their application.

References

[Cugola et al., 1995] Cugola, G., Di Nitto, E., Ghezzi, C., and Mantione, M. (1995). How To Deal With Deviations During Process Model Enactment. In *Proc. 17th Int. Conf. on Software Engineering, Seattle*, pages 265–273. IEEE Computer Society Press.

[Cugola et al., 1996] Cugola, G. P., Di Nitto, E., Fuggetta, A., and Ghezzi, C. (1996). A Framework for Formalizing Inconsistencies in Human-Centred Systems. *TOSEM*, 5(3).

[Easterbrook et al., 1994] Easterbrook, S., Finkelstein, A., Kramer, J., and Nuseibeh, B. (1994). Coordinating Distributed ViewPoints: The Anatomy of a Consistency Check. *Int. Journal of Concurrent Engineering: Research & Applications*, 2(3):209–222.

[Fickas and Feather, 1995] Fickas, S. and Feather, M. (1995). Requirements Monitoring in Dynamic Environments. In *Proc. of the 2nd IEEE Int. Symposium on Requirements Engineering, York*, pages 140–147. IEEE Computer Society Press.

[Finkelstein et al., 1994] Finkelstein, A., Gabbay, D., Hunter, H., Kramer, J., and Nuseibeh, B. (1994). Inconsistency Handling in Multi-Perspective Specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578.

[Finkelstein et al., 1996] Finkelstein, A., Spanoudakis, G., and Till, D. (1996). Managing Interference. In Vidal, L., Finkelstein, A., Spanoudakis, G., and Wolf, A. L., editors, *Joint Proc. of the SIGSOFT '96 Workshops*, pages 172–174. ACM Press.

[ISO/IEC, 1995] ISO/IEC (1995). *International Standard, Information Technology Software Life Cycle Process. 12207.*

[ISO/IEC, 1997] ISO/IEC (1997). *Draft Systems Engineering Standard. 15288.* To appear.

[Leonhardt et al., 1995] Leonhardt, U., Finkelstein, A., Kramer, J., and Nuseibeh, B. (1995). Decentralised Process Enactment in a Multi-Perspective Development Environment. In *Proc. of the 17th Int. Conf. on Software Engineering*, pages 255–264. IEEE Computer Society Press.

[Lupu and Sloman, 1997] Lupu, E. and Sloman, M. (1997). Conflict Analysis for Management Policies. In *5th IFIP/IEEE International Symposium on Integrated Network Management IM'97.* Chapman & Hall Publishers. To appear.

[Mazza et al., 1994] Mazza, C., Fairclough, J., Melton, B., De Pablo, D., Scheffer, A., and Stevens, R. (1994). *Software Engineering Standards.* Prentice Hall.

[Montangero and Semini, 1996] Montangero, C. and Semini, L. (1996). Applying Refinement Calculi to Software Process Modelling. In *Proc. of the 4th Int. Conf. on the Software Process, Brighton, UK*, pages 63–74. IEEE Computer Society Press.

[Spanoudakis and Finkelstein, 1997] Spanoudakis, G. and Finkelstein, A. (1997). Overlaps among Requirements Specifications. Submitted to this workshop.