1999

Towards an ontology of software maintenance

Kitchenham, Barbara A.; Travassos, Guilherme H.; Von Mayrhauser, Anneliese; Niessink, Frank; Schneidewind, Norman F.; Singer, Janice; Takada, Shingo; Vehvilainen, Risto; Yang, Hongji

Wiley

B.A. Kitchenham, et al., "Towards an ontology of software maintenance," Journal of Software Maintenance: Research and Practice, v.11, (1999), pp. 365-3898.
http://hdl.handle.net/10945/55140

JOURNAL OF SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE
J. Softw. Maint: Res. Pract. **11**, 365–389 (1999)

Research

Towards an Ontology of Software Maintenance

BARBARA A. KITCHENHAM[1]*, GUILHERME H. TRAVASSOS[2], ANNELIESE VON MAYRHAUSER[3], FRANK NIESSINK[4], NORMAN F. SCHNEIDEWIND[5], JANICE SINGER[6], SHINGO TAKADA[7], RISTO VEHVILAINEN[8] and HONGJI YANG[9]

[1]*Department of Computer Science, Keele University, Staffordshire, ST5 5BG, U.K.*
[2]*COPPE/UFRJ, BR and DCS/ESEG, University of Maryland, A. V. Williams Bldg., College Park MD 20742, U.S.A.*
[3]*Computer Science Department, Colorado State University, 601 S. Howes Lane, Fort Collins CO 80523-1873, U.S.A.*
[4]*Faculty of Sciences, Division of Mathematics and Computer Science, Vrije Universiteit, De Boelelaan 1018 A, 1081 HV Amsterdam, The Netherlands*
[5]*Naval Postgraduate School, 2822 Racoon Trail, Pebble Beach CA 93953, U.S.A.*
[6]*Institute for Information Technology, National Research Council, Ottawa ON K1A 0R6, Canada*
[7]*Dept of Information and Computer Science, Faculty of Science and Technology, Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Kanagawa 223-8522, Japan*
[8]*DPM Consulting Oy, Haukantie 21 A, 04320 Tuusula, Finland*
[9]*Computer Science Department, De Montfort University, Leicester, LE1 9BH, U.K.*

SUMMARY

We suggest that empirical studies of maintenance are difficult to understand unless the context of the study is fully defined. We developed a preliminary ontology to identify a number of factors that influence maintenance. The purpose of the ontology is to identify factors that would affect the results of empirical studies. We present the ontology in the form of a UML model. Using the maintenance factors included in the ontology, we define two common maintenance scenarios and consider the industrial issues associated with them. Copyright © 1999 John Wiley & Sons, Ltd.

KEY WORDS: empirical research; maintenance factors; maintenance scenarios; evolutionary maintenance; independent maintenance groups; maintenance ontology

1. INTRODUCTION

This paper arose from a discussion session held at the 3rd Annual Workshop on Empirical Studies of Software Maintenance (‘WESS ’98’). The task of the session was to consider the question ‘What are the differences between maintenance tools/methods/skills and those of development?’ From the point at which members of the group stated their preliminary positions, it was evident that we would find it difficult to give a single answer. The position statements ranged from what can be paraphrased as ‘Nothing much’ to ‘Lots of stuff.’

*Correspondence to: Dr. Barbara A. Kitchenham, Department of Computer Science, Keele University, Staffordshire ST5 5BG, U.K. Email: barbara@cs.keele.ac.uk

CCC 1040–550X/99/060365–25$17.50

Received 10 May 1999
Revised 23 September 1999

As the discussion continued, it became clear that our difficulties arose from our different views of what constituted 'maintenance'. We concluded that we could not answer any serious questions about maintenance methods, tools or skills until we had a description of maintenance rich enough to encompass all our different experiences of maintenance. We concluded that what we needed was an ontology of maintenance—that is, a specification of a conceptualisation (Gruber, 1995). This ontology should not be only a hierarchy of terms, but a framework talking about the maintenance domain and identifying the factors that affect maintenance, supported by a taxonomy describing the different factor levels.

We believe that such an ontology would have four major benefits for the maintenance research community. It would:

1. allow researchers to provide a context within which specific questions about maintenance can be investigated;
2. help to understand and resolve contradictory results observed in empirical studies;
3. provide a standard framework to assist the reporting of empirical studies in a manner such that they can be classified, understood and replicated; and
4. provide a framework for categorising empirical studies and organising them into a body of knowledge.

Furthermore, if we could report our research results in a systematic fashion, clarifying the context to which the results apply, it would also help industrial adoption of research results.

In Section 2, we present an overview of the ontology. In Section 3 we describe our proposed maintenance ontology in more detail. In Section 4, we look at two maintenance scenarios and consider how the ontology can be used to help characterise the difference between the scenarios.

2. OVERVIEW

de Almeida, de Menezes and da Rocha (1998) describe the process of constructing an ontology as involving the following activities:

- purpose identification and requirement specification;
- ontology capture and formalisation;
- integration of existing ontologies; and
- ontology evaluation and documentation.

Knowledge captured in an ontology is usually represented in a graphical notation. For instance, GLEO (Graphical Language for Expressing Ontologies) was used to describe a software process ontology (de Almeida, de Menezes and da Rocha, 1998).

In this paper, we consider only a part of the ontology construction process. We consider only purpose identification and requirement specification and ontology capture. Moreover, since we do not intend to provide a formal description, we present our ontology in a subset of UML (Unified Modelling Language) notation (Fowler and Scott, 1997) instead of GLEO. UML has been used by other researchers to describe knowledge. For example, Hasselbring (1999) used UML to describe knowledge concerned with health care information systems. Since UML is a standard object-oriented notation, we believe it will make our ideas more accessible to software engineering and

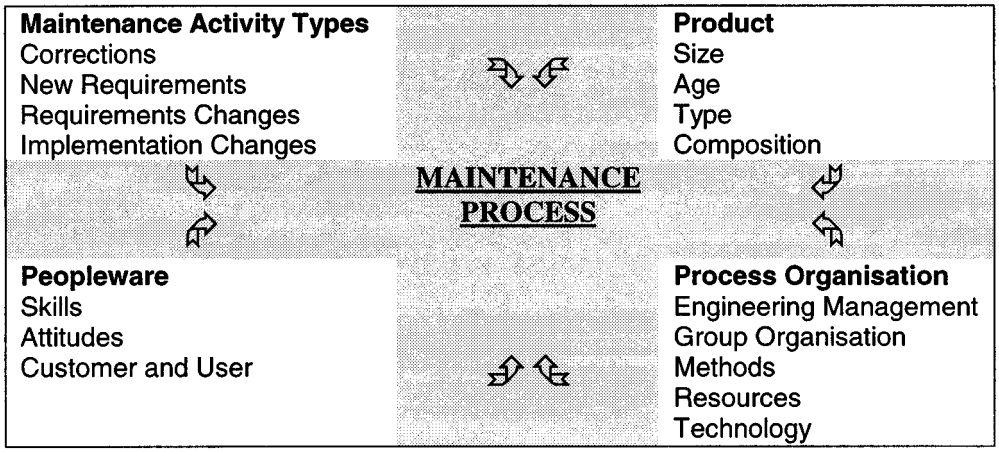


Figure 1. Overview of domain factors affecting software maintenance

software maintenance researchers. Furthermore, it is possible to improve the representation of the ontology at a later date by inserting the axioms needed to formalise the whole model.

As a result of our discussions at the WESS 98 workshop, we identified a number of domain factors that we believe influence the maintenance process. Figure 1 shows these factors and how they can be classified. Figure 1 was the starting point for our ontology, which is described in more detail in Section 3. In order to describe empirical maintenance research, we believe that the maintenance factors must be specified. This will allow researchers to better understand the maintenance context and to plan the research needed to investigate the relationships among these factors and the maintenance context. A better understanding of the relationships that exist between factors and context should lead both to improvements in the maintenance process and to the development of new research topics.

The maintenance process describes how to organise maintenance activities. It is similar to the software development process, but the focus is on product correction and adaptation, not just on the transformation of requirements to software functionality. We take the same viewpoint when considering methods and tools. It is not usually necessary to define new methods or tools to accomplish maintenance activities; conventional software development tools are usually sufficient. However, the maintenance process defines how these methods and tools should be applied to maintenance activities, and which skills and roles are necessary to carry out the activities. Previous research work has considered the definition of methods (Karam and Casselman, 1993), process description (Pfleeger, 1998), software environment ontology (de Almeida, de Menezes and da Rocha, 1998), and tool classification (Pressman, 1997). Although these research results considered the software development process as the basic framework, they are also useful in the context of the maintenance process.

In order to understand the relationships among maintenance domain factors, we need to specify each factor and define the impact that it has on maintenance activities. Next, the relationships themselves can be captured and validated. Validation usually requires empirical studies and experiments.

Figure 1 has some similarities with the framework for software maintenance suggested by Haworth, Sharpe and Hale (1992). They defined a framework based on four entities: programmer, source code, maintenance requirement and environment. They suggested that each of these basic entities in the framework interacted to a degree with the other entities. Each of the entities and each combination of possible interactions contribute to a research area and define the type of attributes that can be manipulated. For example, one area of research is source code attributes, and another is the interaction between source code attributes and programmer attributes. They use the areas to classify existing research and discuss the way in which experiments aimed at considering interactions could be designed. In our ontology, we have generalised the concepts of maintenance requirement, source code and programmer to maintenance activity, product, and maintenance engineer respectively. We have also introduced another concept: the maintenance organisation process. We have omitted an environment entity because our more generalised concepts include environmental considerations. The main difference between the Haworth, Sharpe and Hale framework and our ontology is that they are concerned with the structure of empirical experiments. So, they are not concerned with the nature of the attributes attached to each of their entities, whereas our main concern is the attributes and the way in which they define the context of empirical research.

3. THE MAINTENANCE ONTOLOGY

3.1. Purpose specification and requirements specification

Before discussing our conceptualisation of the maintenance domain, we need to consider the first stage of ontology development, which is purpose specification and requirements specification. de Almeida, de Menezes and da Rocha (1998) define the activity of purpose specification to be 'to clearly define its purpose and intended uses, that is, the competence of the ontology'. The competency of the ontology identifies the questions the ontology is meant to answer.

In our case, the purpose of our ontology is to identify contextual factors that influence the results of empirical studies of maintenance. For example, suppose a researcher were investigating the impact on productivity of new maintenance tools but did not specify the experience of the tool users. In this case, it would be difficult for other researchers to replicate the study, or for practitioners to know whether or not the results were likely to apply in their own situation. Furthermore, it is not just the experience of tool users that is likely to affect the study's results and their interpretation. Other factors that need to be specified include the type of product being maintained, and the type of maintenance tasks being performed.

In observational studies of maintenance, researchers measure maintenance performance characteristics such as the quality of maintained products, or the productivity or efficiency of the maintenance process for different products or different maintenance activities, in order to identify how and why these performance characteristics vary. In controlled experiments, researchers investigate the impact of one or more factors that they believe affect maintenance quality or productivity by varying the factors in a systematic fashion, while controlling other factors.

Thus, in order to support empirical studies of both kinds, each factor in our ontology needs to answer the following competency question:

> *Would variations in this factor (i.e., concept) influence empirical studies of maintenance productivity, quality or efficiency?*

For the purposes of ontology capture, we do not believe it is necessary to identify every possible interaction between maintenance factors and maintenance performance. However, we do need to present a reasoned argument explaining at least one interaction for each factor. This can also be regarded as a contribution to ontology evaluation. Any such explanation would depend on being able to identify the way in which each element can vary in different circumstances. This implies a second competency question:

> What is the nature of the variations in this factor?

This second question leads to preliminary taxonomies of maintenance elements. The taxonomy is also intended to help practitioners identify whether or not empirical results are likely to be relevant to their specific maintenance situation. The two competency questions already identified are sufficient to represent the viewpoint of practitioners as well as researchers.

Finally, we hoped that our taxonomy would also cast some light on our original workshop goal, which was to consider the differences between maintenance and development from the viewpoint of skill, tools and methods. This leads to a third and final competency question:

> To what extent do maintenance methods/tools/skills differ from those of development?

To address this question fully, we would need a software process ontology as well as a maintenance ontology. Thus, we have not addressed this competency question fully. We do, however, point out some of the differences we found between our maintenance ontology and the de Almeida, de Menezes and da Rocha software process ontology, and identify some concepts that are of relevance only to maintenance.

The following sections define our ontology. Because the domain is very complex, we describe each main dimension shown in Figure 1 separately, with the final integrated ontology shown later in Figure 7. In the next sections we present our ontology of software maintenance with definitions of all the main concepts (i.e., maintenance factors). Where possible, we make use of definitions and concepts used by de Almeida, de Menezes and da Rocha (1998) in their software process ontology. We also consider the different properties of the maintenance factors that impact the maintenance process and can thus affect the results of empirical studies.

3.2. Maintained product

3.2.1. Overview

Figure 2 shows our product ontology. Table 1 defines the concepts used in the ontology. Characteristics of these elements that affect maintenance performance are discussed in the following sections. Note that in their software process ontology, de Almeida, de Menezes and da Rocha do not consider the relationship between the total product and its composite artefacts.

3.2.2. Product size

The size of the product affects the number and organisation of the staff needed to maintain it. Table 2 suggests a coarse-grain size measure for classification purposes. There are relationships

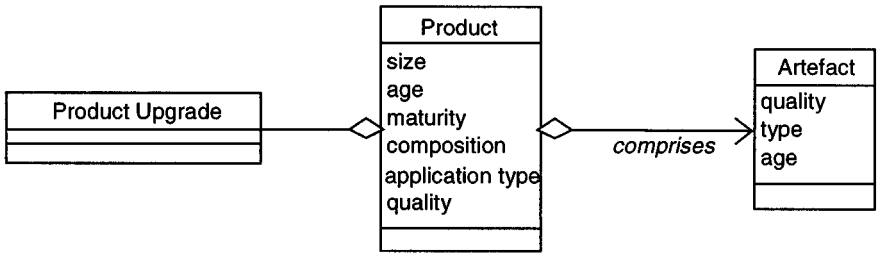


Figure 2. The maintained product ontology

Table 1. The maintained product ontology definitions

Product	The product is the software application, product or package that is undergoing modification. A product is a conglomerate of a number of different artefacts.
Product upgrade	A change to the baseline product that implements or documents a maintenance activity. An upgrade may be a new version of the product, an object code patch, or a restriction notice.
Artefact	Artefacts that together correspond to a software product can be of the following types: documents that can be subdivided into textual and graphical documents, COTS products, and object code components. Textual documents include source code listings, plans, design and requirements specifications.

Table 2. Product size

Product size	Maintenance team size
Small	1 person
Medium	1 team
Large	Multiple teams

between the size measure and maintenance team organisation. For example, geographically distributed maintenance teams usually maintain large products. The size of the enhancements and the size of the product are likely to affect maintenance productivity. The larger the product the more likely it is that product knowledge will be spread unevenly among the maintenance staff, making it more difficult to diagnose the cause of some problems and identify all the modifications needed to support a large enhancement. In addition, when many people are working together on a large enhancement, there are more opportunities for misunderstandings that can lead to quality problems. Thus, maintenance activities on large products may be less productive than maintenance activities on small products.

3.2.3. Application domain

Many researchers (e.g., Maxwell, van Wassenhove and Dutta, 1966) have observed major

productivity differences between products from different application domains. We believe such differences apply to maintenance activities as well as development activities. In addition, the application domain (e.g., finance, telecommunications, command and control, etc.) places domain knowledge requirements on maintenance human resources. It also places constraints on the maintenance artefacts and product. For example, safety critical system maintenance must, at all cost, preserve software reliability requirements, whereas in the telecommunications world there is more emphasis on fast upgrades to software in order to minimise time to market. These different constraints mean that different aspects of maintenance performance are optimised.

3.2.4. Product age

The age of a product (i.e., the age in years since first release) can affect maintenance in different ways:

- If the development technology is very old, it may be difficult to find maintenance human resources with skills in the old technology (hence, the practice of 'grey-sourcing' the maintenance of some products by bringing older programmers out of retirement). In addition, it may be difficult to find support tools, such as compilers and static analysers, and support for the tools.
- If the product is old, it may be difficult to access the original developers or the original development documentation. This can lead to products or parts of products that no one understands well enough to change.

Thus, in general we expect maintenance performance to be better for younger than older products.

3.2.5. Product maturity

Product maturity is different from product age. It concerns the life cycle of a product after initial release. The basic phases in the life of a product and their relationship with maintenance tasks and user population are summarised in Table 3, which is similar to the life cycle described by Kung and Hsu (1998). The maintenance life cycle starts at first release and ends when a product is withdrawn from use. It is important to note that large enhancements cause mini-cycles, where a product can be forced back into periods of infancy and adolescence as a result of poor quality product releases. Table 3 suggests that the type of maintenance tasks undertaken by an organisation is related to the maturity of a product, as is the size of its user population. Note that a consideration of user population is irrelevant for some custom-built products that have a single client-single mission profile.

3.2.6. Product composition

The level of abstraction of the component artefacts of a product affects the skills required by maintenance engineers and the tools they need to support them. If products are generated from designs, maintenance engineers need access to the code generation tools. If the product is composed of black box components (e.g., a COTS product), maintenance engineers need integration skills rather than coding skills.

Table 3. Maintenance life cycle

Life cycle stage	Maintenance task prevalence	User population
Infancy—after release, initial users start reporting defects.	Corrections	Small
Adolescence—as the user population grows, defect reports still predominate but there may be changes to amend the system behaviour.	Corrections, requirement changes	Growing
Adulthood—the product is relatively defect free, but if it is accepted by a wide user population there will be requests for new functionality. In addition, as change accumulates there will be a need to restructure parts of the system to avoid design decay, so implementations changes to improve code structure may be required.	New requirements, implementation changes	Maximum
Senility (legacy)—there are newer products available and only a few users remain to be supported. Usually only corrective maintenance and workarounds are provided.	Corrections	Declining

3.2.7. Product and artefact quality

The original software development process and the quality of the product it delivered place constraints on the subsequent maintenance process. In our experience it is easier to maintain a good quality product than a poor quality product, where 'quality' includes issues such as product structure, documentation, and the quality of individual artefacts. Furthermore, the less contact a maintenance organisation has with the original software developers, the more it is dependent on the availability of good quality documentation, bearing in mind that there are many different forms of documentation associated with a software product. In terms of defining the impact of document quality on maintenance activities, we need to assess the extent to which documentation is:

- complete,
- accurate, and
- readable.

For old products, documentation is often poor or non-existent. In such cases, maintenance engineers need specialised tools such as re-engineering tools. Thus, comparisons of maintenance performance across different products will be of limited value unless it is clear that the maintenance tool requirements of each product have been met to an equivalent degree, and that the quality of the component artefacts is comparable.

3.3. Maintenance activities

Figure 3 shows our maintenance activity ontology, which is derived from de Almeida, de Menezes and da Rocha's software development activity ontology. We have amended that ontology to consider maintenance activities rather than software construction activities, and have omitted elements that

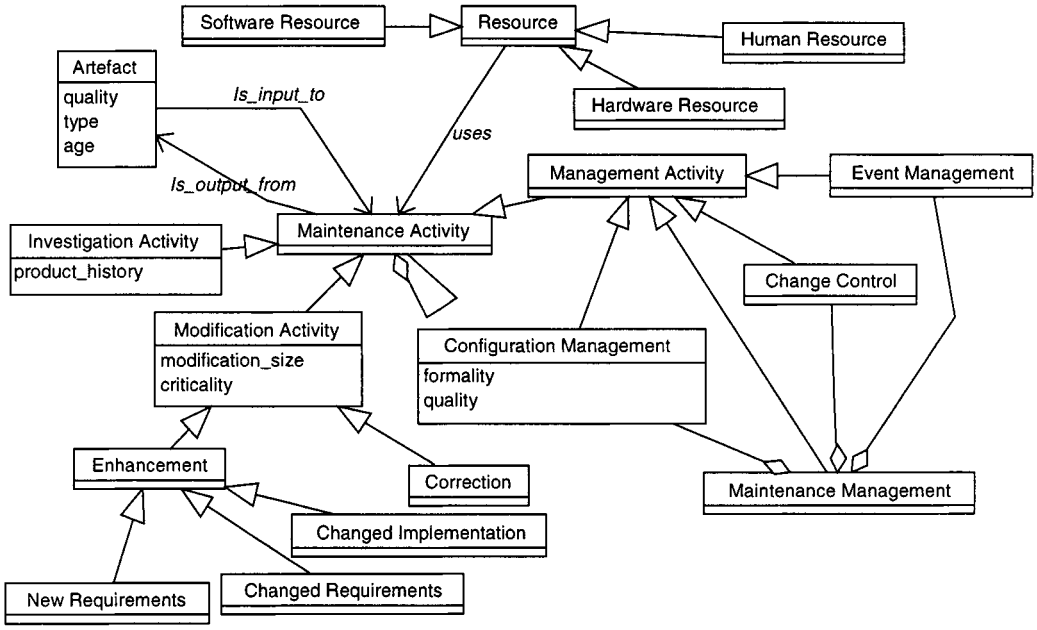


Figure 3. The maintenance activity ontology

do not have any major impact on maintenance performance. In particular, we have added the concept of an investigation activity, and, instead of having a construction activity, we have a maintenance activity. Furthermore, we have identified configuration management as one of the types of management activity. We have also included the resource concept in this ontology, whereas de Almeida, de Menezes and da Rocha (1998) had a separate resource ontology. Definitions of the elements in the ontology are given in Table 4. A discussion of the impact of the elements on maintenance performance follows.

In our view, one of the major differences between software development and software maintenance is that development is requirement-driven and maintenance is event-driven. This means that the stimuli (i.e., the inputs) that initiate a maintenance activity are unscheduled (random) events.

Input events usually originate from the users (or client or customer) of the software application, but may also originate from maintenance human resource (engineers or managers). Thus, the first activity needed by a maintenance process (after the administrative process of logging the event) is an investigation activity, whereby a maintenance engineer is assigned to assess the nature of event, which can be either a problem report or change request. On completion of an investigation activity, maintenance managers must decide whether or not to proceed with a maintenance modification. This is discussed in more detail in Section 3.4.3.

Maintenance modifications are often referred to as corrective, adaptive or perfective following Swanson's typology (Swanson and Chapin, 1995). However, since identifying a modification as an adaptive or a perfective maintenance activity depends on the reason for the change, and not on an objective characteristic of the change, we have used the following definition for types of

Table 4. Maintenance activity ontology definitions

Activity	An action of one of the following types: an investigation activity, a modification activity, a management activity, or a quality assurance activity. An activity may be made up of a number of sub-activities. Usually, it takes as input one, or more existing artifacts and outputs zero, one or many new or modified artifacts.
Investigation activity	An activity that assesses the impact of undertaking a modification arising from a change request or problem report.
Modification activity	An activity that takes one or more input artefacts and produces one or more output artefacts that, when incorporated into an existing system, change its behaviour or implementation.
Management activity	An activity related to the management of the maintenance process or to the configuration control of the maintained product (see Figure 5 and Table 6).
Quality assurance activity	An activity aimed at ensuring that a modification activity does not damage the integrity of the product being maintained. Quality assurance activities may be classified as testing or certification activities (entity omitted from Figures 3 and 7).
Resource	Everything that is used to perform an activity. Resources may be hardware, software or human resources.

maintenance changes:

- *Corrections* that correct a defect—i.e., a discrepancy between the required behaviour of a product/application and the observed behaviour.
- *Enhancements* that implement a change to the system that changes the behaviour or implementation of the system. We subdivide enhancements into three types:
    - enhancements that change existing requirements,
    - enhancements that add new system requirements, and
    - enhancements that change the implementation but not the requirements.

Broadly speaking, enhancements that are necessary to change existing requirements can be equated to Swanson's perfective maintenance changes. Those that are necessary to add new requirements to a system can be equated to adaptive maintenance. Changes that do not affect requirements but only affect implementation might be referred to as preventive maintenance (by analogy to what happens when you have your car serviced). Note that corrections may result in similar types of product modifications, but we do not feel that it is necessary to define correction subtypes.

There is not a one-to-one relationship between problem reports and corrective maintenance. Sometimes, the 'problems' noted by users are requests for behaviours that were not originally required. In such cases, the problem report leads to an enhancement rather than a correction. It is important to determine whether maintenance work is a correction or an enhancement because the activities are often budgeted separately. In fact, many of the disputes between the customer/client and maintainers revolve around whether a change is a correction or an enhancement. If the customer/client did not fully and unambiguously define the required behaviour, it is often difficult to decide whether a modification is a correction or an enhancement.

Characteristics of maintenance activities that affect the productivity and efficiency of maintenance activities include the size of the modification and the criticality of the modification. Large enhancements, particularly large enhancements of large products, are likely to require effort from several different maintenance engineers, and will thus incur coordination and communication overheads. Smaller enhancements that can be performed within schedule by one maintenance engineer are usually more productive. The criticality of an enhancement or correction impacts the elapsed time it takes for the modification to be delivered to users, since the scheduling of the modification will be determined mainly by its criticality.

To accomplish the different maintenance activities, maintenance engineers require different degrees of product understanding and different types of development tools. A corrective activity may require only the ability to locate faulty code and make localised changes, whereas an enhancement activity may require a broad understanding of a large part of the product (Singer, 1998). In the first case, a maintainer will require testing or simulation tools to recreate the problem and debugging tools to step through suspect code. In the second case, a maintainer's tool requirements will depend on the quality of the development documentation, and the availability of the development environment. If the maintainer has poor documentation and little of the original development environment, he/she may require re-engineering tools and/or code navigation and cross-referencing tools.

The efficiency and quality of investigation activities depends on the maintenance engineer knowing the current status of patches and planned modifications that apply to the part of the product involved with the new problem report or change request. The availability of such information depends on the effectiveness of the product configuration control and change control process. A good configuration control process is necessary to identify the status of each product component, including information such as the currently applied patches. A formal change control process might slow down the rate at which the maintenance process responds to input stimuli, but may improve the ability of the change control and maintenance processes to preserve the integrity of the product under maintenance and its constituent artefacts.

3.4. Software maintenance process

3.4.1. Two processes

Within a software maintenance department, there are two different maintenance processes:

- the maintenance process used by individual maintenance engineers to implement a specific modification request, and
- the organisation level process that manages the stream of maintenance requests from customers/clients, users and maintenance engineers.

We consider both types of process separately. In order to use terminology similar to that used by de Almeida, de Menezes and da Rocha (1998), we refer to our definition of the first process as the software maintenance procedure ontology (see Figure 4). de Almeida has no equivalent to the second process in his ontology. We refer to the second process as the maintenance organisation process (see Figure 5).

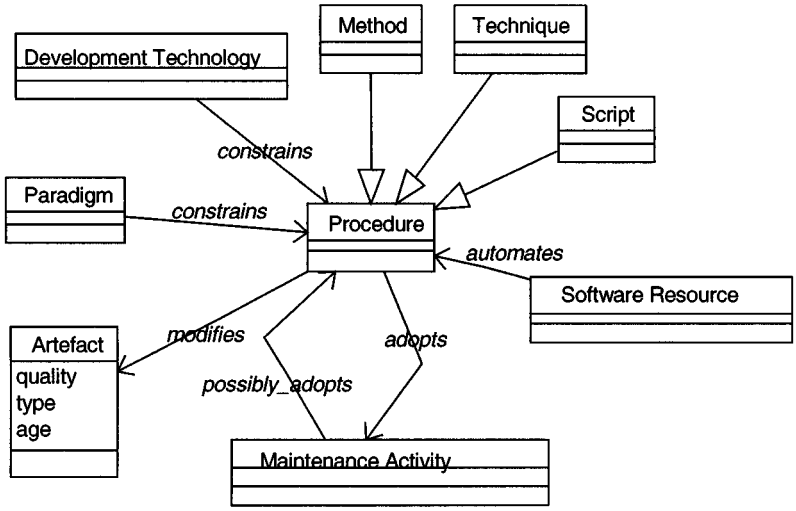


Figure 4. The maintenance procedure ontology

Table 5. Maintenance procedure ontology definitions

Development technology	The technology used when the product and its constituent artefacts were originally constructed, for example, knowledge-based system technology, conventional data processing technology. The original development technology constrains the possible maintenance procedures.
Paradigm	The philosophy adopted during the original construction of the maintained product, for example, the object-oriented paradigm or procedural paradigm. The original paradigm constrains the possible maintenance procedures.
Procedure	The conduct followed to perform an activity. A procedure may be classified as a method, technique or script. A procedure may be adopted to perform a specific activity from a set of possible procedures.
Method	A systematic procedure defining steps and heuristics to permit the accomplishment of one or more activities.
Script	A guideline for constructing/amending a specific type of document.
Technique	A procedure used to accomplish an activity that is less rigorously defined than a method.

3.4.2. Software maintenance procedure

The software maintenance procedure ontology shown in Figure 4 is used to modify one or more artefacts in order to implement a required software modification. The concepts shown in Figure 4 are defined in Table 5. The definitions have been adapted from de Almeida, de Menezes and da Rocha's definitions.

Artefacts are not solely source and object code items. They comprise documents, system representations and plans, etc., constructed throughout the software development process, and

modified during maintenance. A variety of different scripts, methods and techniques are used to construct and modify such artefacts, and they are usually available to support maintenance activities.

Maintenance activity performance will be affected by the choice of software development technology and development paradigm. It will also be affected by the extent to which procedures are automated. In general, development technologies such as the development language and the development paradigm place constraints on maintenance activities, and skill requirements on maintenance human resources. The ISO/IEC 12207 Standard defines an 'activity' as a life cycle phase and a 'task' as something done as part of an activity. Here we are using only the term 'activity', but an activity can be decomposed into smaller activities, therefore capturing the ISO/IEC definitions.

In addition, the chosen development technology may present a significant risk to product maintainability. A software product cannot continue to be maintained if its development environment is not available to its maintainers. For products with a long lifetime it is necessary to ensure that technologies such as compilers, code generators and CASE tools will themselves be supported throughout the estimated lifetime of the product.

3.4.3. Maintenance organisation processes

Figure 5 shows the maintenance organisation process. Table 6 briefly defines the concepts used in the model.

A maintenance organisation must handle a stream of maintenance requests from users, customer and maintainers. Thus, a major element of a maintenance organisation is event management (Niessink and van Vliet, 1998). Another major element of a maintenance organisation is configuration management. Configuration management is the process responsible for releasing new system versions and system amendments to users. In addition, configuration control systems need to protect the integrity of the product when it is being modified. In particular, they need to ensure that maintenance engineers know the current repair status of the product and product components. If the configuration control system is inadequate, maintenance activities will be less efficient and there is a danger that product quality will be compromised.

In addition, there needs to be a management process for authorising or rejecting modification activities after initial investigation of the trigger event. This is usually the responsibility of a change control board. The authorisation process may also include a process of negotiation with the client about contractual arrangements for implementing a required modification (e.g., budgets/price and time-scales). Only after a proposed modification activity is approved by the change control board and any necessary contractual arrangements are agreed with the client (which, for applications like operating systems or self-standing products, may be the marketing department), will the proposed modification activity be scheduled. A change control board can be organised as a formal process involving meetings between users and customers/clients and maintenance managers, or as a simple working procedure. The level of formality can affect quality and efficiency. Formal change control boards are likely to slow the maintenance process but are better able to protect the integrity of the product being maintained.

The efficiency of maintenance management activities is affected by the use of support tools. Most organisations have configuration control tools. There are also many tools to assist event management. For example, many maintenance organisations use 'help' desk tools, which allow

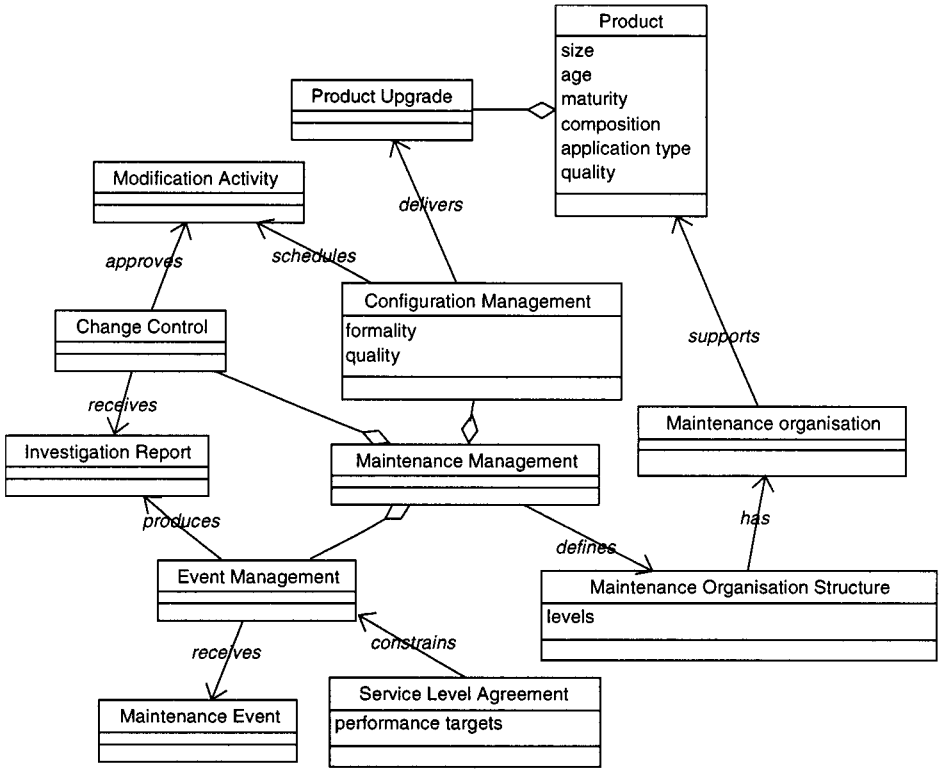


Figure 5. The maintenance organisation process ontology

events to be logged into an organisation and their progress tracked though the various maintenance tasks needed to resolve the event. Another type of tool that supports the interface between the user population and a maintenance organisation is a 'known error log', which identifies all currently known errors and their workarounds or fixes.

The volume and type of maintenance requests affect the performance of the maintenance organisation. For example, if there are a large number of defects reported, there may be insufficient resources to undertake perfective or preventive modifications.

Service level agreements define the maintenance organisation's performance targets. Differences in achieved performance level may, therefore, be due to different performance targets. Maintenance organisations must be engineered to meet their service level agreements. This is often done by separating various support activities into well-defined roles that can be performed by staff with specialised skills. For example, many maintenance organisations use the concept of support levels to separate staff, whose main concern is to support the user population and those concerned with correcting or enhancing software.

At its simplest there may just be two support levels:

- Level 1—this level provides the personnel who staff the help desk.
- Level 2—this level provides the personnel who make changes to software.

Table 6. Maintenance organisation process ontology definitions

Service level agreement	An agreement between the providers of a maintenance service and the customers of a maintenance service that specifies the performance targets for the maintenance service.
Maintenance management	The process used to manage the maintenance service (as opposed to the procedure used to manage individual maintenance requests). The organisation process is established and maintained by senior maintenance managers. It is responsible for defining the structure of the maintenance organisation such that it can fulfill its service level agreement. Maintenance management has three main concerns other than the normal concerns of quality assurance and project management: event management, configuration control, change control.
Event management	Event management is the process responsible for handling the stream of events received by the maintenance organisation.
Change control	Change control is the process responsible for evaluating the results of maintenance event investigations and deciding whether or not to approve a product modification.
Configuration management	Configuration management is responsible for maintaining the integrity of the product in terms of its version and modification status. It is also responsible for the production of product upgrades.
Maintenance organisation structure	The roles undertaken by maintenance human resources in a maintenance organisation in order to perform the required administrative procedures.
Maintenance event	A problem report, or change request originating from a customer or user of the maintained product or a member of the maintenance organisation.
Investigation report	The outcome of investigating the cause and implications of a maintenance event.

However, at least three support levels is the more common situation:

- Level 1—the help desk staff are non-technical, and are responsible for logging problems and identifying the technical support person most likely to be able to assist a user.
- Level 2—the technical support personnel know how to communicate with users and understand their problems, and they can advise on workarounds and quick fixes.
- Level 3—the maintenance engineers are authorised to make changes to the product.

The separation of maintenance services across different service levels makes it clear that not all maintenance work results in product modification. Users may simply require advice about how to use the product or how to circumvent a known problem with the product. The number of levels and the specific roles they support affect the performance of the maintenance service. For example, if there are too many levels there may be an unacceptable delay in responding to certain types of maintenance request.

The other main role for a maintenance organisation is the planning and scheduling of maintenance releases. This involves identifying the content of difference releases and a release cycle that is appropriate to customer requirements. Factors such as the interval between scheduled maintenance releases and the extent of change permitted to a product can have a significant impact on the quality

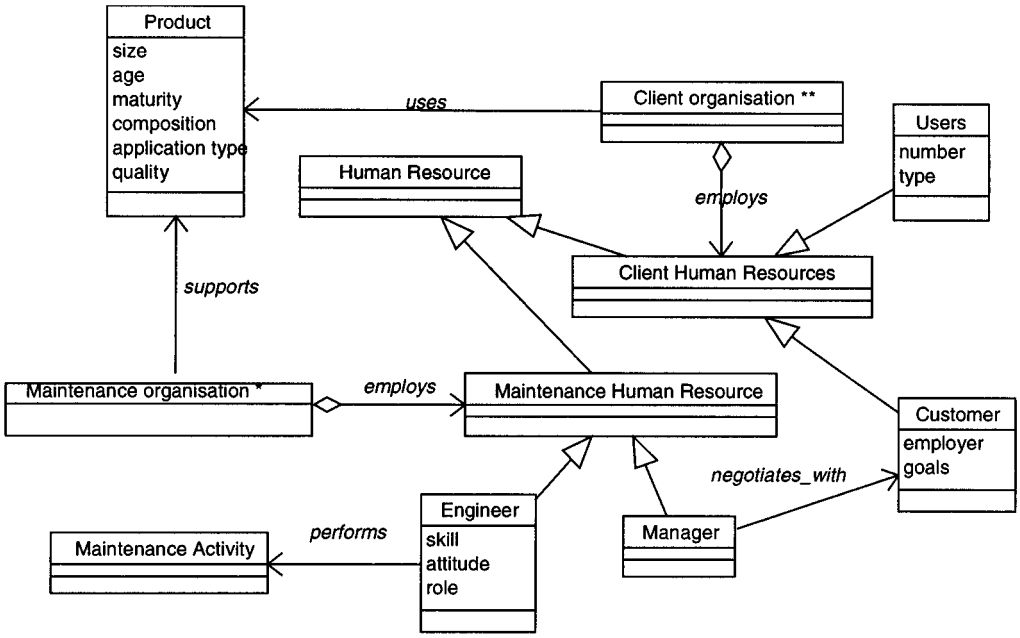


Figure 6. The peopleware ontology

of the maintained product (Lehman, Perry and Ramil, 1998). The procedures for releasing object code fixes (for example, fix on fail, or periodic collated updates) can also affect product quality (Mellor, 1983).

3.5. Peopleware

3.5.1. Two groups

Software production and maintenance are human intensive activities. Furthermore, they involve people working together in teams, which are in turn part of larger organisations. Thus, no complete description of factors affecting maintenance can ignore the human and social elements. There are two types of staff involved in a maintenance process: the staff in the maintenance organisation, and the staff in the customer/client organisation. Figure 6 shows our initial model of these factors. The definition of peopleware concepts is given in Table 7.

3.5.2. Maintenance organisation staff

3.5.2.1. Staff attitudes. Staff attitudes and motivation are generally agreed to impact on the quality of any activity. In the area of software maintenance, problems with motivation are expected because software maintenance is often perceived to be of less importance and less well-rewarded than development.

Table 7. Peopleware ontology definitions

Client organisation	The organisation or organisations that use the maintained product and have a defined relationship with the maintenance organisation.
Maintenance organisation	The organisation that maintains the product or products.
Human resource	Employees of the maintenance or client organisation. Maintenance organisation staff can be classified as managers or engineers. (For simplicity we have omitted specialised QA staff who may be considered a special class of engineer.) Employees of the client organisation can be classified as users or customers. Managers in the maintenance organisation negotiate with customers to determine service level agreements and costs and scheduling of requirement enhancements.

Management often compounds attitude problems by:

- making maintenance work equivalent to a punishment, and
- assigning novices to maintenance work.

This factor seems difficult to characterise, but is likely to have a major impact on the productivity and quality of maintenance activities and the extent to which the maintenance staff is receptive to process change.

3.5.2.2. Staff responsibilities. One area that seems to have a major impact on the entire maintenance culture of an organisation is whether or not there is a strict separation between staff responsible for software development and those responsible for software maintenance.

At one extreme, there is no real separation between development and maintenance. This seems to be associated with a particular type of product, i.e., a product undergoing continual evolution that is released periodically to clients and users. The software developers incorporate corrective, perfective and preventive maintenance tasks into a process aimed at a continuing stream of planned enhancements. In such an environment there may be no practical difference between the tools and procedures used for ‘development’ and those used for ‘maintenance’. Furthermore, the personnel themselves do not make any significant distinction between development and maintenance, which reduces motivation problems.

At the other extreme, there are maintenance organisations that are completely separate from development departments, and indeed may not work for the same company that developed the code they maintain. In such an environment, maintenance programmers may need specially designed tools to support their maintenance tasks.

Another issue is whether staff are responsible for the maintenance of a single product or group of products (i.e., a product portfolio). It is usual for an evolutionary style of development to be organised around a single product or product family, whereas a separate maintenance group usually looks after a portfolio of different products.

These are issues that should concern maintenance managers when service level agreements are defined, or when they are initially bidding for a maintenance contract.

3.5.2.3. Staff Skills. In general, the more skilled the maintenance staff, the better the productivity and quality of maintenance activities. Different activities require different skills, so these factors need to be controlled or specified during empirical studies of maintenance activities.

3.5.3. Customer and user staff

Customer and user issues that affect maintenance are:

- The size of the user population, which affects the amount of work required to support a particular application.
- The variability of the user population, which affects the scope of maintenance tasks. The more varied the user population, the more varied the problems they will encounter and refer to the maintenance staff.
- Whether or not the client and maintenance organisation are part of the same company. Relationships between client and maintenance group may be less co-operative if the groups are from different companies.
- The extent to which the customer/client and users have common goals. Customers/clients fund maintenance activities. If they do not understand the requirements of the real users, they may impose inappropriate service level agreements, to the detriment of the product users who will in turn become less satisfied with the maintenance organisation.

4. TWO MAINTENANCE SCENARIOS

4.1. Organisation distinction

Figure 7 shows the full maintenance ontology. In this section, we use this ontology to specify two different maintenance scenarios. Staff responsibility seems to be one of the most important factors in the above ontology. Our discussion at the WESS workshop continually returned to the issue of whether or not the maintainers and software developers were the same people.

Therefore, in this section, we define two maintenance scenarios based on this distinction:

- Evolutionary development, and
- Independent maintenance organisation.

We show how the factors identified in the ontology differ in the two scenarios. In addition, we consider for each the related industrial concerns.

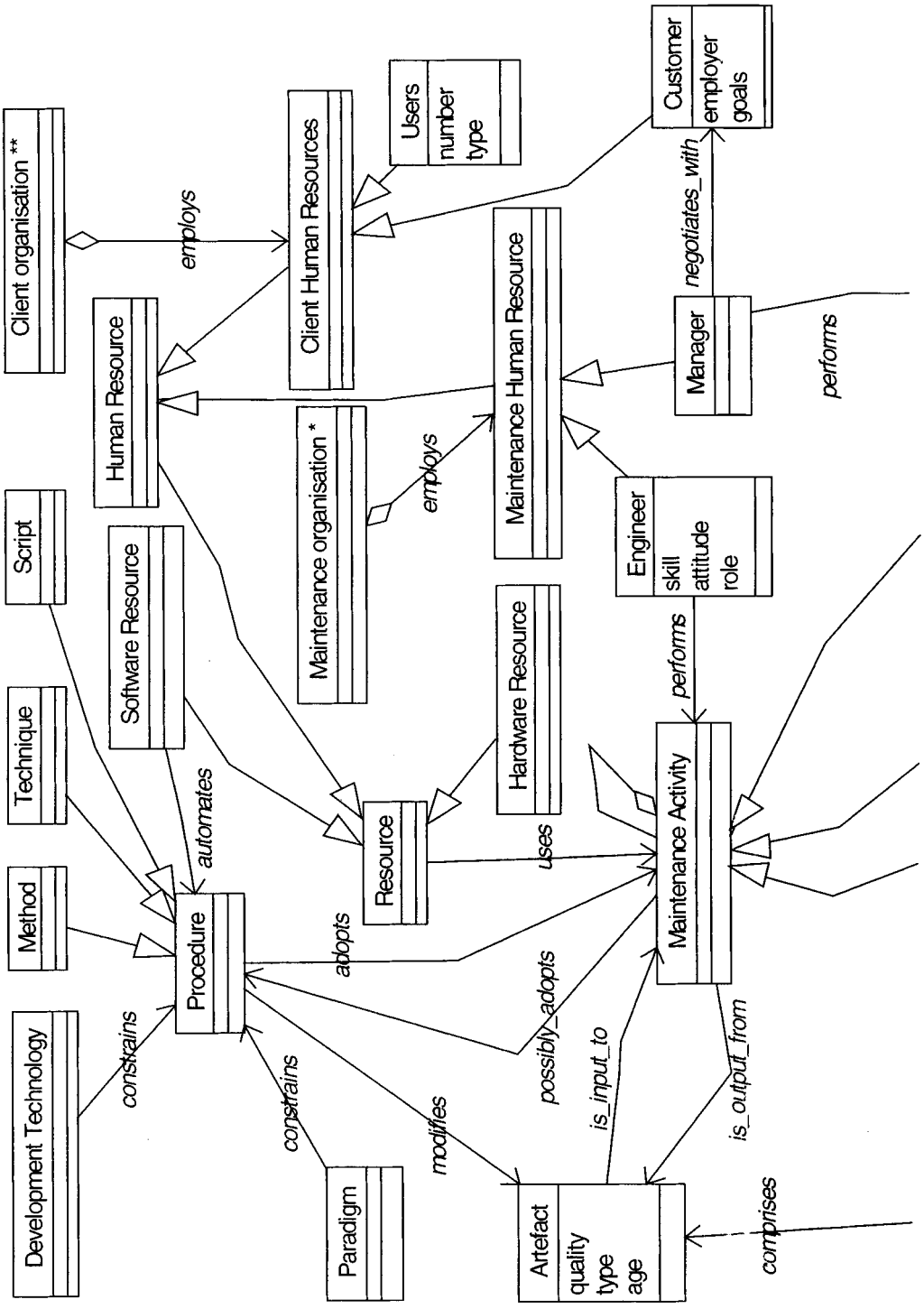
4.2. Evolutionary development

Table 8 specifies the evolutionary development scenario. In this maintenance scenario, practitioners are often concerned with optimising the evolutionary process. Particular concerns include:

- optimisation (and/or minimisation) of inter-release intervals,
- prediction of release quality/reliability,

Table 8. Evolutionary development scenario

Staff responsibilities	Maintenance engineers are responsible both for producing new product upgrades and for correcting problems in past releases. Staff are responsible for the evolution of a single product or product family.
Product size	Usually large. Examples: Space Shuttle, Microsoft Word, ICL VME Operating System. Note, however, large products often encourage small companies to produce small add-on products. These small products track the evolution of larger products. For example PKZIP tools have evolved in line with Microsoft products from DOS to Windows 3.1 to Windows 98.
Development technology	The maintenance and development technologies are identical. Maintenance activities do not require additional staff skills or tools.
Application domain	Application domain knowledge is required both for maintenance and development.
Product age	As the product ages, the original software developers will move to other jobs so some expertise is lost. However, there is also some continuity resulting from the overlap between older staff leaving and new staff joining the group.
Product maturity	The impact of maturity on an evolving product depends on the client and user population. For shrink-wrapped products, there is a danger that maintenance requests arising from a large user population will interfere with enhancement activities. For example, defect reports arising from release n will be received during the development of release $n + 1$. This can be even more complicated if different clients do not upgrade in the same time scale, so some client will be reporting defects with release $n - 2$ while others are reporting problems with release $n - 1$. If one product release is of particularly poor quality, it may generate enough defect reports to prevent software developers working on the next planned release. For custom products, such as the Space Shuttle, releases are co-ordinated with the specific client activities so there is less of a problem.
Maintenance management process	The management will need to provide a means to administer the stream of defect reports from users. Release schedules are based on prioritising customer requirements. Enhancements are funded either by clients (analogous to development projects), or licensing agreements or product sales. Licensing agreements or product sales usually covers maintenance costs.
Maintenance group organisation	Support levels are often used to separate software developers from support staff who interface with users.
Staff attitudes	Staff regard themselves as software engineers rather than developers and maintainers so there are less likely to be problems motivating staff.
Types of maintenance	All enhancement activities are referred to as evolutionary development.
Customer and user types	See product maturity.
Document quality	In principle, the original software documentation would continue to be updated as part of the evolutionary release cycle. However, in practice this would depend on the organisational culture and management practices.



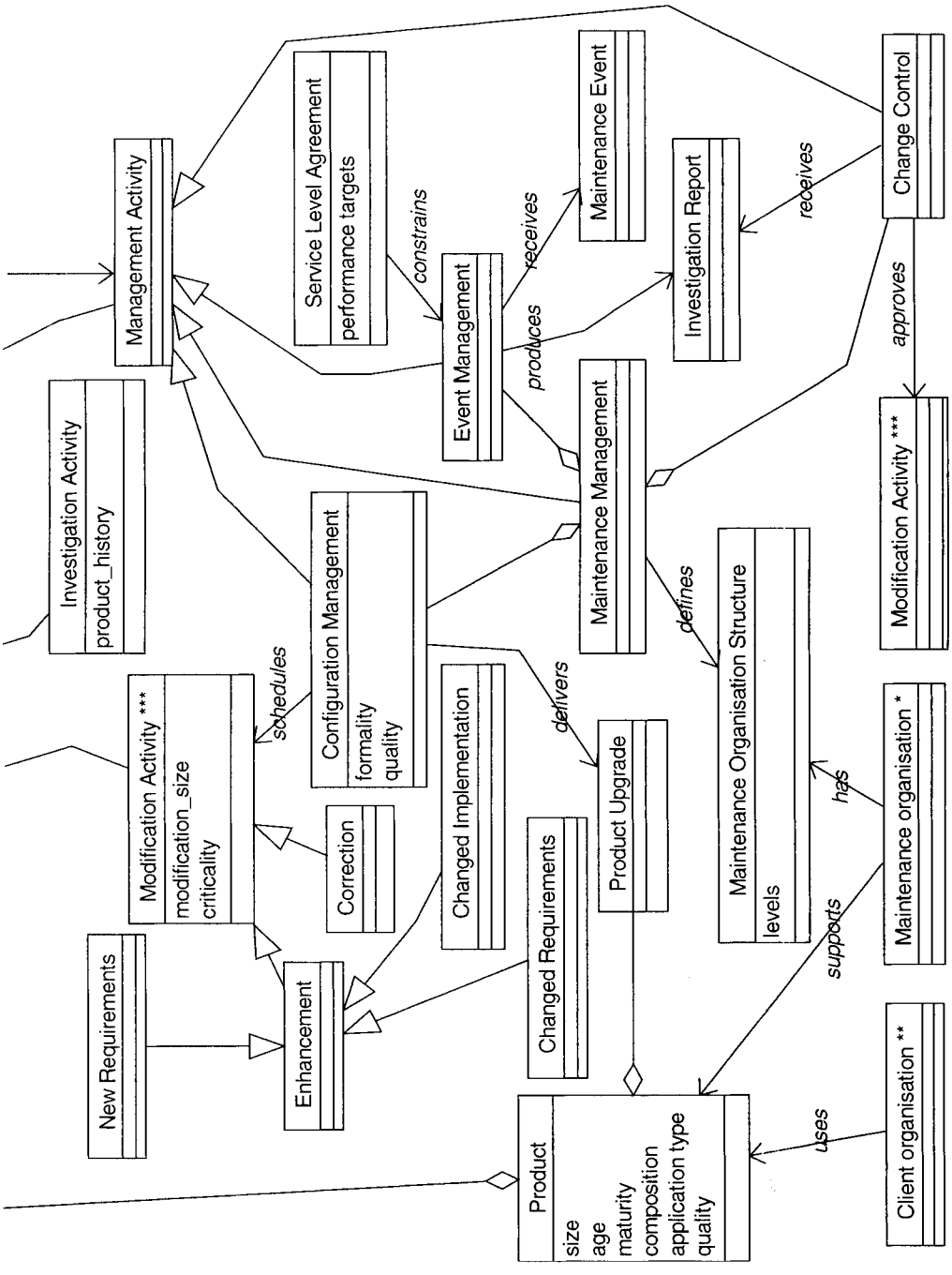


Figure 7. The software maintenance ontology. (The asterisks indicate single classes that are repeated in this Figure to enhance readability)

Table 9. Independent Maintenance Group Scenario

Staff responsibilities	Maintenance engineers are responsible for producing product upgrades that may include changes due to enhancements and corrections of maintenance tasks. They will usually not have been involved in original product development. Staff is usually responsible for a portfolio of products.
Product size	Individual elements in a portfolio will be of different sizes.
Development technology	Usually different products in different portfolios will have been produced using different technologies. The maintenance organisation will often need to support many different technologies although the technologies, required by an individual maintainer will usually be restricted.
Application domain	If the portfolio of products is very diverse, it will be difficult to ensure that all maintenance staff have appropriate domain knowledge.
Product age	Different products will have different ages. This makes the maintenance of portfolios complex and planning and costing maintenance activities difficult.
Product maturity	Different products will have different levels of maturity.
Maintenance management process	The management will need to provide a means to administer the stream of defect reports from users. They need fairly complex estimating and risk management procedures to cope with the complexity inherent in administering portfolios. This will be less formal if the client and maintenance group work for the same company. Relationships with customers are usually mandated by a service agreement, although adaptive maintenance may be managed like a development project.
Maintenance Group Organisation	Support levels are often used to separate software developers from support staff who interface with users.
Staff attitudes	Motivation is likely to be particularly important in maintenance groups.
Types of maintenance	All the standard types of maintenance are performed.
Customer and user types	There seem to be two different scenarios: One client—many users. e.g. in-house support groups. Many Clients—many users, e.g. a third party maintenance shop. Note that in some cases the number of items in the portfolio is important. Some maintenance shops support one large custom product in each client portfolio, e.g. Department of Defense in the U.S.A.
Document quality	This is a critical issue for third party maintenance shops since they seldom have any access to software developers. For in-house support groups it may be less of a problem because they may have access to the original developers.

- effort estimation for individual enhancement projects, and
- planning functional contents of releases to minimise the risk of destabilising the product while achieving customer/client required functionality.

Another important concern is the impact of new development paradigms on system evolution, e.g., RAD products, COTS-based products and object-oriented products.

4.3. Independent maintenance group

Table 9 specifies the independent maintenance group scenario. In this scenario, industry concerns differ according to whether or not the maintenance 'shop' is in-house or a third-party organisation. In particular, third-party organisations have concerns about bidding for maintenance contracts (in terms of estimation processes and accuracy and risks), that are less important for in-house maintenance groups (unless they are candidates for outsourcing). Furthermore, outsourcing organisations—particularly those that takeover in-house organisations—have major management concerns about the issues of achieving a common organisational culture and changing the working methods of organisations they absorb (Tittle, 1998; Ketler and Willems, 1999).

All types of maintenance group have concerns about maintenance task estimating and planning and improving efficiency of maintenance activities. An important issue for such organisations is the need for re-engineering methods and tools to address the problem of lack of adequate specification/design documentation in older products.

5. CONCLUSIONS

This paper has presented an ontology of software maintenance aimed at assisting researchers to report sufficient contextual detail for other researchers and practitioners to understand the results of empirical studies. We developed the ontology from our personal experiences of the maintenance process and have discussed two different maintenance scenarios in terms of the ontology. Figure 7 summarises the ontology, modelled in UML.

One of the problems with the model is that competency questions provide a criterion for inclusion of a factor in the model, but they do not provide completion criteria, nor do they provide any concept of relative importance. Thus, the elements identified in the model are things that a researcher needs to report when describing empirical studies, but there may be other factors we have not included. We must emphasise that, even using this ontology as a guide, it is still the responsibility of the individual researcher to attempt to identify any special conditions that apply to his/her results.

Formally, the ontology presented in this paper is not complete. We have not attempted to formalise the ontology using predicate logic, nor have we fully evaluated it. Furthermore, since we are not attempting to integrate our ontology into a knowledge-based system, we do not believe such a formalisation is necessary. In its current form, we believe the ontology provides useful insights into the type of information researchers should report if we are to understand fully the results of empirical studies of maintenance. Only if the software maintenance community were considering a large-scale database to register empirical research results, would a formalised, fully-evaluated ontology be necessary.

References

de Almeida FR, de Menezes SC, da Rocha ARC. 1998. Using ontologies to improve knowledge integration in software engineering environments. In *Proceedings of 2nd World Multiconference on Systemics, Cybernetics and Informatics*, Volume I / *Proceedings of 4th International Conference on Information Analysis and Synthesis*, Volume I; International Institute of Informatics and Systemics: Caracus, Venezuela; pp. 296–304. Also available at URL: http://www.inf.ufes.br/˜falbo/download/pub/sci98.zip [28 October 1999].

Fowler M, Scott K. 1997. *UML Distilled: Applying the Standard Object Modeling Language*; Addison-Wesley Publishing Co.: Reading MA.

Gruber TR. 1995. Towards principles for the design of ontologies used for knowledge sharing. *International Journal of Human–Computer Studies* **43**(5/6):907–928.

Haworth DA, Sharpe S, Hale DP. 1992. A framework for software maintenance: a foundation for scientific inquiry. *Journal of Software Maintenance: Research and Practice* **4**(2):105–117.

Hasselbring W. 1999. Technical opinion: on defining computer science terminology. *Communications of the ACM* **42**(2):88–91.

Karam GM, Casselman RS. 1993. A cataloguing framework for software development methods. *IEEE Computer* **26**(2):34–45.

Ketler K, Willems JR. 1999. A study of the outsourcing decision: preliminary results. In *Proceedings of the 1999 ACM SIG CPR Conference on Computer Personnel Research*; ACM Press: New York NY; pp. 182–189.

Kung H-J, Hsu C. 1998. Software maintenance lifecycle model. In *Proceeding International Conference on Software Maintenance*; IEEE Computer Society Press: Los Alamitos CA; pp. 113–117.

Lehman MM, Perry D, Ramil JF. 1998. Implications of evolution metrics on software maintenance. In *Proceedings International Conference on Software Maintenance*; IEEE Computer Society Press: Los Alamitos CA; pp. 208–217.

Maxwell K, Wassenhove LV, Dutta S. 1996. Software development productivity of European space, military and industrial applications. *IEEE Transactions on Software Engineering* **22**(10):706–718.

Mellor P. 1983. Modelling software support. *ICL Technical Journal* **3**(4):407–438.

Niessink F, Vliet Hv. 1998. Towards mature IT services. *Software Process—Improvement and Practice* **4**(2):55–71.

Pfleeger SL. 1998. *Software Engineering: Theory and Practice*; Prentice-Hall, Inc.: Saddle River NJ; pp. 44–75.

Pressman RS. 1997. *Software Engineering: A Practitioner's Approach*, 4th edition; McGraw-Hill Companies, Inc.: New York; pp. 805–825.

Singer J. 1998. Practises in software maintenance. In *Proceedings International Conference on Software Maintenance*; IEEE Computer Society Press: Los Alamitos CA; pp. 139–145.

Swanson EB, Chapin N. 1995. Interview with E. Burton Swanson. *Journal of Software Maintenance: Research and Practice* **7**(5):303–315.

Tittle J. 1998. Software Maintenance. A Perspective on Some Issues from the Trenches, Keynote Address, *Third Annual Workshop on Empirical Studies of Software Maintenance, WESS '98*. (Slides available at URL: http://www.cs.umd.edu/users/travasso/tittle.ppt [28 October 1999].)

Authors' biographies:

Barbara A. Kitchenham is Managing Director of Butley Software Services Ltd. and Principal Researcher in Software Engineering at the University of Keele. Her main research interests are software metrics and empirical software engineering. She is a visiting professor at the Universities of Bournemouth and Ulster. Her email address is: barbara@cs.keele.ac.uk

Guilherme H. Travassos is an Associate Professor at COPPE-Federal University of Rio de Janeiro. His research interests include empirical software engineering, software architecture, software testing, quality and software engineering environments. He is a Visiting Associate Professor at the Department of Computer Science of the University of Maryland at College Park. His email address is: travassos@cs.umd.edu

Anneliese von Mayrhauser is a Professor at Colorado State University and Director of the Colorado Advanced Software Institute, a consortium of businesses and Colorado universities supporting Technology Transfer research. Her research interests include software testing and maintenance. She holds an M.S. and Ph.D. from Duke University and a Dipl.Inf. from the Karlsruhe Technical University in Germany. Her email address is: avm@cs.colorado.edu

Frank Niessink is a Ph.D. candidate at the Vrije Universiteit, Amsterdam, in the Faculty of Sciences. His research interests are software measurement, software maintenance and process improvement. Frank received an M.Sc. in Computer Science and an M.Sc. in Economics from the Vrije Universiteit. His email address is: F.Niessink@cs.vu.nl

Norman F. Schneidewind is Professor of Information Sciences at the Naval Postgraduate School in Monterey CA. He developed the Schneidewind software reliability model that is used by NASA to predict the software reliability of the Space Shuttle. Also, he is a Fellow of the IEEE, elected for his contributions to software measurement. His email address is: nschneid@nps.navy.mil

Janice Singer is a research officer with the National Research Council in Canada. She works in the Software Engineering Group of the Institute for Information Technology in Ottawa. Her research interests include maintenance, human computer interaction, and empirical studies. She holds a Ph.D. in cognitive psychology from the University of Pittsburgh in Pennsylvania. Her email address is: singer@iit.nrc.ca

Shingo Takada is an Assistant Professor at the Department of Information and Computer Science, Keio University in Japan. His research interests include software engineering, especially software reuse and software maintenance, as well as information exploration. He received his Ph.D. in Computer Science from Keio University in 1995. His email address is: michigan@doi.cs.keio.ac.jp

Risto Vehvilainen is a Managing Director of an IT consulting company. He holds an M.S. in Mathematics from Helsinki University and also he is a doctoral student at the Swedish School of Economics and Business Administration in Helsinki. His main research area is software maintenance as a service. His email address is: risto.vehvilainen@kolumbus.fi

Hongji Yang is a Principal Lecturer in Computer Science Department at De Montfort University, UK and leads the Software Evolution and Re-engineering Group. His research interests include software maintenance, reverse engineering, re-engineering and reuse. He served as a Program Co-Chair at International Conference on Software Maintenance in 1999. His email address is: hjy@dmu.ac.uk

