Fine-grained Management of Software Artefacts: The ADAMS System

Andrea De Lucia¹, Fausto Fasano²*, Rocco Oliveto¹, and Genoveffa Tortora¹

¹ DMI, University of Salerno, Fisciano (SA), ITALY E-mail:{adelucia, roliveto, tortora}@unisa.it
² STAT Dept., University of Molise, Pesche (IS), ITALY E-mail: fausto.fasano@unimol.it

SUMMARY

We present ADAMS (ADvanced Artefact Management System), a web-based system that integrates project management features, such as work-breakdown structure definition, resource allocation, and schedule management as well as artefact management features, such as artefact versioning, traceability management, and artefact quality management. In this paper we focus on the fine-grained artefact management approach adopted in ADAMS, which is a valuable support to high-level documentation and traceability management. The traceability layer in ADAMS is used to propagate events concerning changes to an artefact to the dependent artefacts, thus also increasing the context awareness in the project. We also present the results of experimenting with the system in software projects developed at the University of Salerno.

KEY WORDS: software artefact management systems, fine-grained artefact management, traceability management, usability study

1. Introduction

In the last decade a lot of research effort has been devoted to the development of methodologies and technologies supporting coordination and collaboration of distributed software engineering teams. Examples are Computer Supported Cooperative Work (CSCW) and groupware, workflow management, and Software Configuration Management (SCM). The last technology is widely used in software engineering projects to face with coordination problems. In particular, SCM tools (see e.g., [3] [4] [5] [6] [7]) help to coordinate the activities of developers, by providing

---

†Part of the material contained in this article was published in a preliminary version in [1] and [2].
*Correspondence to: Fausto Fasano, STAT Dept. - University of Molise, C.da Fonte Lappone, 86090 Pesche (IS), ITALY. e-mail: fausto.fasano@unimol.it
capabilities that either avoid parallel development altogether, e.g., through locking, or assist in resolving conflicts, e.g., through merging.

While SCM is a well-established and common practice in the late phases of software development, notably during programming and integration, it is less commonly practiced in the early phases, i.e. analysis and design. One of the reasons might be the fact that in traditional software development methods there is usually a limited number of versions—and hardly any configuration—for analysis and design documents. However, modern iterative and incremental object-oriented development methods, e.g., the Unified Process, lead to a significant increase in the complexity and the number of versions of documents in the early phases [8]. In addition, analysis, design, and implementation are considered as overlapping activities. As a result, even simple modifications can affect several elements produced during different activities.

Due to the need of mainly supporting the late phases of software development, most of the current tools providing versioning functionalities, e.g., CVS [3] or Subversion [5], are specifically designed to manage files containing lines of text in a pretty-printed format, without considering the logical structure of the document contained in a file. Conversely, high-level software documents are filled with elements, e.g., images, diagrams, and tables, that cause the automatic merge functionality to fail. Most of these documents have a well-defined hierarchical structure. This structure is usually defined in the quality manual of an organisation and/or in the quality plan of a project and is essentially composed of chapters, sections, subsections, and paragraphs, each of them providing a part of the whole document content. Even diagrams, usually included in such documents must comply with a well specified meta-model, most of them have a graph-based structure, e.g., UML use case, class, state-chart, activity, component and deployment diagrams. Source code also has a logical structure; for example, java code can be organised in packages, each of them containing classes composed of attributes and methods.

Although high-level artefacts can be managed as single files, a finer-grained management is more desirable, where each single element of the document is managed as a different entity and the tool functionalities address both the atomic entity and the entities it is contained in. This gives developers the possibility to control the evolution of the single elements and more precisely identify the impact of changes through the definition of detailed traceability links. Indeed, adequate support for traceability management is another lack of SCM tools. Traceability can provide important insights into system development and evolution assisting in both top-down and bottom-up program comprehension, impact analysis, and reuse of existing software, thus giving essential support in understanding the relationships existing within and across software requirements, design, and implementation [9]. To this aim, specific traceability tools are used to manage traceability links [10] [11] [12] [13]. However, since their integration with the SCM tools is lacking, the traceability infrastructure fails during system evolution. Such an inadequate traceability is one of the main factors that contributes to project overruns and failures [14].

Another advantage of a fine-grained management of software artefacts is that it reduces the likelihood of conflicts arising when different software engineers apply changes to the same document. In fact, in case the document is decomposed in sub-artefacts, a conflict occurs only if changes concern the same sub-artefact. It is worth noting that, even if such kind of conflict (i.e., changes occurring in different parts of a document) can be usually addressed by automatic merge algorithms for textual documents, the scenario is much more complicated...
when dealing with non-textual documents, e.g., UML diagrams, where an automatic conflict resolution is difficult even when changes do not affect the same model element. Furthermore, reuse of components is enabled, as the software engineer can include the same low-level artefact in several composite artefacts, thus improving the consistency within the software project.

Finally, inadequate support to the management of the software project is another limitation of existing SCM tools. Managing the development team, the schedules and deadlines for the project artefacts, and the work breakdown structure of the project are essential features to monitor the development process. Generally, this has to be done using project management tools that are scarcely integrated with the tools managing the artefacts produced during the project [15]. With a fine-grained artefact management this gap can be partially overcome, as this approach naturally results in the definition of a product oriented work-breakdown structure: since an artefact is decomposed into a hierarchy of simpler objects, the definition of more precise access policies to specific parts of the artefact is also enabled, thus allowing a finer task allocation and responsibility identification.

In this article we present ADAMS (ADvanced Artefact Management System), a web-based system that integrates project management features, such as work-breakdown structure definition, resource allocation, and schedule management as well as artefact management features, such as artefact versioning, traceability management, and artefact quality management. The goal of the ADAMS project is to provide a full feature environment supporting the software engineer during all the phases of the software lifecycle paying special attention to the coordination problems occurring in cooperative development environments. However, ADAMS is different than traditional Process centered Software Engineering Environments (PSEE) [16] [17] [18] [19] [20] [21] [22] [23], as it is not designed around the definition and enactment of software processes. Rather, it is conceived for the definition and cooperative production of software artefacts. Besides the support provided by typical features of a configuration management system, collaboration is also supported by synchronous and asynchronous tools that improve the communication among the project team members. Moreover, ADAMS provides an event-based notification mechanism that allows the software engineer working on an artefact to be notified when something relevant occurs on the artefact or on related artefacts. ADAMS also integrates functionalities to manage artefact quality, through checklist based inspection processes.

ADAMS usability and usefulness was preliminary evaluated in [24]: the system was used as project and artefact management system in the Software Engineering courses of the Computer Science program at the University of Salerno (Italy). This preliminary study was conducted in 2005 and involved about 140 students, allocated on 17 projects. Students evaluated ADAMS through a questionnaire. The results achieved in the preliminary evaluation of ADAMS suggested the implementation of new functionalities to improve the fine-grained artefact management, such as automatic re-composition and versioning of composed artefacts. In this article, we focus on the new fine-grained management of software artefacts and on its impact on traceability management and event notification. We also replicated the experimentation to evaluate the usability of the new version of the system with respect to the previous version. The replicated experimentation spanned over two academic years, 2006 and 2007, and involved 90 students allocated on 12 projects. The results achieved in the second experimentation demonstrate that the added functionalities improve the fine-grained management of artefacts.
in ADAMS. Other than a qualitative analysis, we also present a quantitative analysis to analyse how artefacts are decomposed during software development and how the fine-grained artefact structure influences the management of the traceability information.

The main contributions of this paper can be outlined as follows:

- a full feature environment supporting the software engineer during all the phases of the software lifecycle, paying special attention to the coordination problems occurring in cooperative development environments. We focus the attention on the fine-grained management of software artefacts presenting the design and the implementation of advanced functionalities, i.e., automatic versioning and re-composition, to better support the fine-grained software management of software artefacts. The design of these functionalities were driven by the results of the preliminary evaluation of the system [24];
- a discussion on how the fine-grained artefact structure influences the management of the traceability information as well as the event notification mechanism that allows the software engineers working on an artefact to be notified when something relevant occurs on the artefact;
- a replication – spread across two years – of the preliminary ADAMS evaluation carried out in 2005. The goal was to analyse the usability and the usefulness of an integrated environment like ADAMS during the development of a software system. Moreover, the replication allows to analyse whether the enriched support for fine-grained artefact management improves the usability of the system.

The rest of the article is organised as follows. Section 2 presents an overview of ADAMS, Section 3 presents the fine-grained artefact management in ADAMS, and Section 4 describes the traceability management functionality of ADAMS. Section 5 presents and discusses the evaluation and experience of ADAMS within an educational environment. Section 6 discusses related work, while Section 7 concludes and gives some indications for future work.

2. ADAMS Overview

ADAMS enables the definition of a project in terms of the artefacts to be produced and the relations among them [1]. The system provides functionalities to manage resources, projects, and artefacts and is decomposed in several subsystems with a layered architecture, namely the Resource Management Subsystem, the Project Management Subsystem, the Artefact Management Subsystem, the Quality Management Subsystem, the Cooperative Development Subsystem, and the Event and Notification Management Subsystem. The user interface is implemented using JSP and HTML pages. In addition, a graphical front-end has been also implemented by developing an Eclipse [25] plug-in that allows the software engineer to synchronise artefacts in the ADAMS repository with his/her local workspace in the IDE.

The Resource Management Subsystem provides administrative functionalities for human resource and account management. Resources can be allocated on projects defined in ADAMS. When the user is allocated on a project he/she becomes member of the project team. ADAMS adopts a role based access control policy. The system administrator defines roles by enabling or denying the access to specific functionalities of the system. A user can be associated to many
different projects and can play different roles for each of the assigned project. An access control subsystem is used by each subsystem anytime it needs to verify whether a human resource has the permission to perform an operation as well as by the user interface to enable or disable functionalities according to the user roles.

The Project Management Subsystem provides project management functionalities, such as definition of the schedule and allocation of human resources to the artefacts. User tasks in ADAMS are represented by the output produced by the associated activities (an artefact). The definition of the schedule is performed during the creation of the artefacts by specifying the starting and due date for the artefact. When a software engineer accesses the list containing the set of artefacts he/she has to work on (to-do list), the system highlights late and overdue artefacts. Resources can be assigned to a specific artefact with specific roles, thus defining artefact teams. Each resource can be assigned a different set of roles for each artefact according to the role he/she has in the development of the artefact.

The Artefact Management Subsystem is responsible of the fine-grained management of software artefacts. This includes the hierarchical organisation of composite artefacts that provides a product-oriented work breakdown structure to the project manager. The Artefact Management Subsystem also provides concurrent development functionality. It uses a revision control subsystem to create, modify, and delete artefacts, as well as to manage the artefact state and versions. Traceability links between related artefacts can also be inserted and visualised through a traceability management subsystem and used for impact analysis during software evolution. The Event and Notification Management Subsystem is used to enhance context-awareness within the software project. Through this subsystem software engineers are notified when concurrent developers make changes to artefacts they are allocated to. Software engineers can subscribe events occurring to artefacts they would like to be notified about. Finally, the traceability layer can be used to propagate events involving an artefact to the artefacts depending on it. The functionalities provided by the artefact management subsystem and the subsystems it uses are the core of this article and will be discussed in detail in the following sections. The Artefact Management Subsystem is also responsible for the management of the artefact lifecycle. This includes the possibility to track the evolution of the artefact from the moment when it is conceived, through the different versions produced during the development, up to its release.

The Quality Management Subsystem provides functionalities to manage the quality within a project. This entails the definition of the artefact types managed by the system and the standard templates and review checklists associated to them. It also supports a distributed inspection process [24]. Software inspection is a software engineering practice to identify defects, reducing rework, and producing high quality software systems [26] [27]. ADAMS also supports a synchronous distributed inspection meeting, where all the inspectors can remotely meet, access the checklists of each other, and discuss about the artefact to produce the final defect log.

The Cooperative Development Subsystem includes tools used to enhance synchronous and asynchronous collaboration within ADAMS. Examples of synchronous collaboration tools are the internal chat and an UML collaborative editor that enables developers to access and modify the same diagram concurrently (STEVE [28]). STEVE allows a distributed team to discuss and model the system directly from within ADAMS, maintaining the versioning for each model.
element as well as for the information shared during the meeting. Examples of asynchronous collaboration tools are the internal email, comments and feedbacks that can be attached to a specific artefact (or to a specific artefact version), and the rationale management tool, that enables the software engineers to address open issues that need to be investigated by the team, using an argumentation based approach.

3. Fine-grained artefact management

ADAMS adopts a fine-grained management of software artefacts. In particular, an artefact in ADAMS can either be an atomic entity or it can be composed of an arbitrary number of atomic or further composite artefacts. The fine-grained management of artefacts proposed in ADAMS allows the software engineer to choose the granularity level of this decomposition according to the artefact type, the concurrency level needed for the human resources allocated on them, the number of software engineers that are likely accessing them at the same time, the necessity to specify responsibilities for specific unit of work, and the need for definition of traceability links between them.

3.1. Artefact structuring and versioning

In ADAMS, the composition of artefacts is realised by defining composition links between the composite artefact and each of the contained sub-artefacts. The system prevents cycles in the document structures. This is important to avoid the artefact re-composition process to be involved in an infinite loop (see Section 3.2).

Artefact hierarchies in ADAMS can be defined bottom-up (putting simple artefacts together to form a composite artefacts), top-down (starting from high level documents and incrementally adding sub-artefacts or decomposing parts of the composite artefact), or in any combination of the previous approaches, linking together already defined hierarchies. Such a composition process is dynamic, as the software engineer can decide to change the artefact structure at any time during software development. This gives the flexibility necessary in a highly changeable and evolving environment, like a software development project, and allows changing the grain of the decomposition as soon as the artefact structure becomes inadequate for the selected decomposition level.

Composition of artefacts can be performed through the hierarchical view of the project artefacts or using the compositional link management interface. The former supports the “drag & drop” among the artefacts to compose the hierarchy (see Figure 1) while the latter requires the software engineer to define each composition link specifying the parent and child artefacts. Drag and drop is also used to establish an ordering between children of the same parent (see ”‘Drag Before’” and ”‘Drag After’” options in Figure 1). This ordering is important to define the structure of the final document as the re-composition procedure (see Section 3.2) uses the ordering of the nodes as they appear in the hierarchical visualisation. The hierarchical view allows the software engineer to browse the artefact structure similarly to browsing a file system directory.
A graph-based view is also available, that shows the structure of composite artefacts by using a direct acyclic graph (see Figure 2). The graph-based view has been realised by using Grappa [29], an open source graph drawing package developed by AT&T Research Lab as a port of a subset of GraphViz to Java. Grappa has been customised with specific artefact type icons, different types of links for compositions and dependences, and new commands in the contextual menu (see Figure 2).

One of the benefits of the fine-grained approach is the possibility to share the same sub-artefact among several artefacts. The software engineer can decide to use the copy option to create a new compositional link without modifying the structure of the artefacts containing the sub-artefact to be shared (see Figure 1). Note that in this case any change applied to the shared artefact is reflected to all the composite artefacts it is contained in. It is worth noting that shared artefacts are best visualised in the graph-based view. As an example, in Figures 1 and 2 the artefact “Glossary” is shared between the RAD (Requirements Analysis Document) and the SDD (System Design Document). Note that both the views provide information about the structure of the composite artefact: the hierarchical view gives an overview of the composite artefact layout by respecting the sequential ordering between the artefacts and also visualises...
the dependences between the artefacts, while the graph-based view makes artefact sharing explicit.

ADAMS provides versioning functionality for each of the elements of the hierarchy of composite artefacts, thus including the composite artefacts and each leaf in the hierarchy. ADAMS versioning management is based on the check-out/check-in paradigm adopted in most of the current versioning and configuration management tools. Different versioning management systems adopt different strategies to store the version content and implement concurrency policies. Each of them has pros and cons in terms of performance, disk space usage, reliability, or concurrency criteria. ADAMS relies on a relational database to store the metadata related to the artefacts and their versions. The data content for each artefact, instead, is stored locally on the web server file system. This choice has been adopted to improve the performances of storing and retrieving functionalities. However, it is obviously possible to interface the revision control subsystem with a wrapper for other versioning systems, e.g., CVS [3] or Subversion [5], as well as to implement additional features such as caching and delta versioning.

Among the metadata stored with each artefact version, ADAMS records the timestamp of each check-in operation. Such a version timestamp is used to reconstruct the state of the project in a precise instant of the system evolution. The current state of the project is simply recovered by accessing the state of the project at the request time. However, it is possible to browse the artefact repository setting up a past timestamp and showing the snapshot of the project in that precise instant of the development process. This can be useful for configuration management,
but also provides support whenever it is necessary to recover any other intermediate version of the system.

Versioning allows the history management of the file associated to an artefact, providing a mechanism to revert to previous versions of an artefact and manage concurrent accesses to the same artefact by different software engineers.

3.2. Automatic re-composition and versioning of composite artefacts

The composition of software artefacts in ADAMS enables the definition of a hierarchical structure for a complex document, thus taking advantage of the benefits of a fine-grained management approach.

The distribution of the content over an arbitrarily deep and wide hierarchy of smaller sub-artefacts can cause the software engineers to spend a considerable amount of time simply to put the pieces together before the delivery of the documents, as well as anytime a complete version of the composite artefact is required. This time wasting activity can be avoided in case the structure of the document and the format of sub-artefacts can be automatically managed by the system.

The current version of ADAMS provides a specific support for two of the most common types of artefacts used during a software development process, namely UML diagrams and textual documentation, e.g., project documentation such as project plans, requirements specifications, design document, test documents, user manuals. Concerning documentation, currently ADAMS supports the main formats used, including Microsoft Word (doc), Open Document Text (odt), Rich Text Format (rtf), Portable Document Format (pdf), HTML, and text documents (txt and source code files). In this case, the re-composition functionality provided by ADAMS recomposes the document starting from its leaf nodes, by using the structure of the composite artefact to organise the document in sections and subsections, preserving the ordering defined among the nodes. ADAMS uses the Open Document format to normalise the content of the document components, then merges all the pieces in a single open document, and finally converts the merged document in the output format as required by the user. It is worth noting that software engineers can use their preferred editing tools to produce artefacts. Indeed, each piece of a composite artefact can be produced in any of the supported format with any authoring tool. The re-composition procedure can also use a template file to produce the cover of the document using dynamic information, including the project name, the artefact name, the artefact version, the developers of the document, and the project members. The revision history containing the list of all the versions of the recomposed documents together with the comments added to each version is also automatically included in the document. Finally, a summary of the document structure reporting the versions used for each of the leaf node used in the re-composition is attached to the document. The table of content of the document is automatically generated during the re-composition procedure using the name of the artefacts for sections and subsections.

The quality manager can also define the layout of the auto-generated parts of the document by uploading a special template file containing formatting information, e.g., the font and size to be used for each section level of the document. During the re-composition procedure, the software engineer is asked to choose how the leaf nodes have to be managed. By default, the
system creates a new section (or subsection) for each artefact (including leaves of the hierarchy). Note that, in some cases, it can be useful to have pieces of the document not included in a specific subsection. As an example, consider the section containing the functional requirements within a Requirements Analysis Document. It is a good practice to identify each requirement with an artefact, e.g., to trace requirements onto the implementing classes. However, the software engineers could decide to avoid producing the final document having one section for each requirement. To this aim, ADAMS allows the software engineer to specify whether a leaf node content must be considered as a subsection or as a simple piece of the artefact, e.g., a functional requirement. On the other hand, the software engineer can specify that a particular leaf node does not imply the creation of a new subsection. Finally, it is also possible to specify that an artefact is not intended to be used during the re-composition. This will cause the procedure to skip the artefact as well as all its descendants (in case of a composite artefact).

The re-composition procedure can be executed starting from any composite artefact within the project. This allows the software engineer to recompose an entire document, as well as a single chapter, section or subsection of the document. Note that in ADAMS even a composite artefact may have a file associated with each of its versions. The usefulness of having a file associated with composite artefacts (whose content is actually provided by accessing its children content) is twofold. In case the software engineer uses file formats not supported by the open document format, the file associated with the composite artefact can be used to maintain the manually recomposed artefact versions. Even in case the file format is supported by the system, the file associated to the internal node of the hierarchy can be useful. Indeed, it can be used to store the content of specific artefact versions, e.g., the specific version built and delivered to the client, or an intermediate version manually recomposed by the software engineer, e.g., to solve some formatting problems due to unsupported document types.

Note that the storage of intermediate versions of a composite artefact can cause the content of the composite artefact be not synchronised with respect to the actual content of the artefacts it is composed of. In fact, when a software engineer modifies the content of a leaf node, the stored versions of the artefacts containing it become out-of-date with respect to the new version of the leaf artefact. Even if the automatic document re-composition or a manual merge can be performed to build the artefact again using the newest version of each leaf artefact, the content of the composite artefact is not synchronised until the software engineer freezes it as a new version. To inform the software engineer that composite artefact content is not synchronised with the versions of the artefacts it is composed of, the hierarchical and the graph-based views of ADAMS show these artefacts with an out-of-synch decorator. In particular, an exclamation mark over the artefact icon is used in the graph-based view and close to the name of the artefact in the hierarchical view (see for example the RAD artefact in Figures 1 and 2).

A special consideration has to be done for another type of document used during the software development process, namely UML diagrams. The system supports UML diagrams by maintaining XML-based files describing each diagram component and the diagram itself, e.g., containing colours, positions, and dimension for the element within the diagram. The latter file is used by the system to automatically recompose the artefact starting from its children node content. In this situation, the XML representation of the artefact is maintained at the composite artefact level. It is worth noting that, being such a file versioned by the system, it is possible to modify the internal structure of the composite artefact, i.e., how to
put its components together, whenever needed, by merely updating the file associated to the composite artefact. This also guarantees the versioning of the diagram as well as the versioning of each of its component to be easily provided.

3.3. Support to concurrent development

The support for collaborative development in ADAMS is provided through typical configuration management features. ADAMS enables groups of people to work on the same artefact, depending on the required roles. Different software engineers can access the same artefact according to a lock-based policy or concurrently, if branch versions of the artefact are allowed.

In case a lock-based concurrency policy is adopted, to modify an artefact the software engineer has to declare his/her intentions by performing a check-out operation, thus locking the artefact. Once all the changes are applied, the file is checked-in again, causing the production of a new revision. This approach prevents conflicts due to modifications applied by different software engineers to the same artefact. In case a composite artefact is checked-out, locks should be recursively propagated across the hierarchy, to avoid concurrent changes to the composite artefact and some of its components. However, to limit the effects of indirect locks (lock to ancestors or descendants) when the software engineer accesses an artefact to modify it, ADAMS distinguishes among four different types of locks:

artefact lock lock affecting the artefact to modify, due to a previous check-out of the same artefact

container lock lock to one or more ancestors of the artefact

safe lock lock involving non-leaf descendants node

The different types of lock are visually identified by using different colours for the lock icon: red for artefact locks; yellow for content or container locks; and green for safe locks. An artefact lock does not allow concurrency, except in the case the branch and merge policy has been specified for that artefact. However, in case of a content lock, the system allows the software engineers to work on composite artefacts containing the locked content. This behaviour is motivated by the assumption that changes to a composite artefact are generally due to the necessity to recompose the artefact starting from the content stored in the components. This operation is usually quicker (especially if the automatic re-composition functionality is used) than the production of a new version of a leaf artefact, where the change has to be analysed, designed, and performed. As a consequence, the new version of the composite artefact is generally produced before the locked leaf artefact is unlocked. However, there is the possibility that a new version of the leaf node is created before the composite artefact is recomposed and checked-in. In this case, during the check-in of the composite artefact, the system detaches this situation (comparing the timestamps of the check-in operations) and highlights it to the software engineer by tagging the new version as out-of-date. Container locks are managed similarly to content locks. They differ in the sequence of the check-out operations. In particular, a container lock occurs when a change has to be applied to a leaf artefact whose content is
being used to create a new version of a composed artefact. In this situation, the system allows
the software engineer to work on the leaf artefact, while the composite artefact is tagged as
\textit{out-of-date} only in case it is checked-in after the new version of the leaf artefact has been
created. Finally, safe locks are shown only for context awareness. Indeed, since they do not
involve any leaf artefact, i.e., any content change, they never cause problems with respect
to the synchronisation between the content of the composite artefact and the content of its
components.

In case a software engineer has to modify concurrently an already check-out artefact
(especially in case of a leaf artefact), an artefact lock occurs, requiring him/her to wait
until the lock is released. In some situations, this serialisation of changes can excessively
slowdown the development process. In order to allow the concurrent development within
a cooperative development environment, ADAMS supports the branching and merging of
artefacts. Whenever a locked artefact has to be modified concurrently by another software
engineer, he/she can decide to create a new branch. When he/she has finished working on
the artefact, his/her changes have to be merged with the changes applied by the concurrent
workers. This can cause conflicts in case different changes are applied to the same piece of
document, thus requiring a manual conflict resolution. Note that for textual documents, such
as source code files, automatic merging algorithms based on file differencing can be used.

The concurrent development functionality of ADAMS can be applied to any type of artefact
managed by the system. As an example, let us consider the software requirements specification,
a high level project document. The software engineers involved in the requirements elicitation
and analysis phases can work concurrently on the document, each of them accessing the
sections related to the functionalities they have been assigned to. Managing the requirements
analysis document as a composition of finer-grained sub-artefacts allows the project manager
to precisely assign the responsibilities for each document section and the software engineers
focus on the work product they have been assigned to. This means that two different software
engineers allocated on different functional requirements are allowed to work independently
from each other without the necessity to take care of locking and unlocking mechanism as
well as conflict resolution issues. This is particularly important in a distributed software
development environment, where the work breakdown structure is defined, tasks are assigned
to the software engineers, and the intermediate work products, e.g., requirements and actors
description, scenarios, use cases, sequence diagrams, test cases, are successively merged in a
deliverable document.

Besides complex documentation artefacts, another type of artefact that benefits from the
fine-grained versioning management provided by the revision control subsystem is represented
by diagrams. Diagrams and components are managed as different configuration items. Each
element of a diagram can be related to a description, e.g., the description of a use case can
contain its preconditions and post-conditions, the flow of events, and exceptional conditions.
As a consequence, the diagram and each of its elements can have files associated. The file of
the composite artefact is used to store the corresponding diagram representation, while the
descriptions of each diagram element are stored in different files. Such a separation is also
valuable for the reuse of components across different diagrams.

When an UML diagram is checked-out, both the diagram and the contained elements are
locked. As a consequence, in case UML diagrams sharing elements are checked-out at the same
time, the shared elements can only be edited in the first checked-out diagram, while they are locked in the other diagrams. Thus, developers working on other diagrams can modify all diagram elements except the shared elements previously locked in other diagrams.

4. Traceability management

Software artefact traceability has been defined as “the ability to follow the life of a requirement in a forward and backward direction” [30]. Maintaining traceability across software artefacts can be helpful in a number of tasks. Traceability links between pieces of code and documentation, such as requirements, design documents, or manual pages, aid both top-down and bottom-up comprehension. In other words, traceability allows developers to map high-level documents, and thus abstract concepts, to low-level artefacts. This also improves the software maintainability: once a maintainer has identified the high-level document, e.g., requirement or use case, related to the feature to be changed, traceability helps to locate the pieces of design, code, and whatever else needs to be maintained.

In ADAMS, a traceability link represents a relationship between a source artefact and a target artefact. Traceability links can be inserted manually: the software engineer responsible for maintaining the consistency of the traceability layer (the traceability manager), selects the source and the target artefacts among the already defined project artefacts. Filters on the artefact type and/or on the name of the artefact can be specified to guide the traceability manager during the identification of both the source and the target for the traceability link (see Figure 3). ADAMS shows a list of candidate links calculating all the combinations of source and target artefacts. Thus, the software engineer selects the links to trace specifying for each link a stereotype. The stereotype can be used to identify the nature of the link (see Figure 4).

The fine-grained management of software artefacts in ADAMS enables the definition of traceability links between entities within the same document, as well as between parts of two different documents or diagrams, thus supporting a more precise impact analysis. Clearly, there is a trade-off between the quality of traceability information and the effort to maintain it. In particular, the quality of the traceability layer depends on the granularity of software artefacts. The finer the granularity of software artefacts, the higher the quality of the traceability links between artefacts, as well as the higher the effort to generate and maintain them [31]. For this reason, in ADAMS the software engineer can trace links at different levels of granularity. Once the artefact structure is defined, the software engineer can decide to trace the component artefacts (e.g., requirements, use cases, methods) or the composite artefacts (e.g., document sections, whole document, class). If for economical reasons the software engineer decides to trace at a higher level of granularity, he/she can decide to trace the composite artefacts. In this way he/she is able to trace a set of artefacts, i.e., composite artefacts, on another set of artefacts. On the other hand, if the software engineer is interested in having very high quality traceability information he/she can trace the component artefacts.

Traceing links at a fine-grained level of granularity is an error prone and time consuming task because of the high number of links that have to be identified [31]. To reduce the effort for tracing new links, ADAMS has been enriched with a traceability recovery tool,
Figure 3. Filtering candidate traceability links

Figure 4. Tracing a new link manually
namely ADAMS Re-Trace [32] that supports the software engineer during the traceability link definition. ADAMS Re-Trace uses Latent Semantic Indexing (LSI) [33], an Information Retrieval method, to calculate the similarity measure among the artefacts within a project. The tool is based on the conjecture that artefacts having a high similarity probably contain similar concepts, so they are likely good candidates to be traced onto each other. See [32] for a detailed description of the traceability recovery functionality of ADAMS.

Finally, traceability links evolve together with the evolution of the system, i.e. they can be added or removed, when new artefacts are added to the project, existing artefacts are removed or newer versions are checked-in. To this aim, traceability links in ADAMS are versioned, as well as composition links. Thus, it is possible to modify the nature of a link without losing its history.

4.1. Artefact browsing

Traceability links can be organised in a traceability graph where nodes identify the artefacts and the traceability links are represented by edges of the graph. Within a traceability graph we identify traceability paths, i.e. sets of artefacts connected by traceability links. Obviously, the identification of such paths is very useful for impact analysis. An artefact along a traceability path could be impacted by each artefact appearing in the part of the path preceding it as well as it could impact on each artefact appearing in the part of the path following it.

To identify the impact set for a specific artefact, the software engineer can use the traceability graph provided by ADAMS. To this aim, he/she selects the source artefact from the artefact list and defines the dependence types to visualise in the graph. It is worth noting that the traceability graph is built starting from a source artefact and finding all the dependences of a specific type that involve the source artefact either as source or target artefact. Moreover, the software engineer has also the possibility to show the dependences that involve a specific version of the source artefact.

The traceability graph visualisation has been realised by using Grappa, as well as the graph-based view of the work breakdown structure. Grappa has been also customised to provide enhanced visual information about the artefacts. In particular, tooltips have been used to show immediate artefact information as soon as the user positions the mouse over an icon. The artefact used as query has been shown in red to point out its position within the traceability graph. Finally, each different traceability link type has been associated with a specific notation. In particular:

- a traceability link is represented by a solid blue arrow labelled with the stereotype associated to the link;
- a composition is represented by a solid green arrow.

Furthermore, the contextual menu shown on a right click of the mouse over an artefact of the graph has also been customised to directly access to some artefact information (see Figure 5).

Artefact comments allow the software engineer to share information concerning a specific software artefact, such as additional information needed to better understand the artefact they refer to. They can be attached to the artefact as well as to a specific version of the artefact, and can be tagged as public (visible to the project team) or private (visible only
to the resources allocated on the artefact). Feedbacks are a special type of comments, used to highlight potential problems in the artefact. Thus, feedbacks are intended for the artefact manager and usually require the artefact to be validated. For this reason they are considered as priority comments by the system and managed differently.

ADAMS has also been enriched with a rationale management tool, that enables the software engineers to address open issues that need to be investigated by the team, using an argumentation based approach [34]. This tool enables the management of relations among the rationale elements and the other artefact types, e.g., the functional requirements raising an issue or the non-functional requirements addressed by a project criterion. Obviously, also this information can be obtained browsing the traceability graph. The software engineer can easily identify issues related to a specific artefact.

4.2. Event subscription and notification

ADAMS provides an event notification mechanism that allows the software engineers working on an artefact to be notified when something relevant, e.g., another branch is created by another worker, occurs on the artefact. Unfortunately, event based notification has revealed
some scalability problems, as the number of messages generated can slowdown the system and cause the developer to ignore notifications because they require a lot of time to be managed. Moreover, not all the users need to be notified for specific events and not all the events are perceived as equally important by different users.

For these reasons, ADAMS enables software engineers to subscribe events they would like to be notified about (see Figure 6), and specify the artefacts they are interested in, allowing each user to define the best compromise between an adequate context-awareness level and an endurable quantity of messages received. The software engineer can use the traceability graph to browse the artefact repository and select the artefacts to subscribe.

A number of events are automatically notified without any need for subscription. Examples include notifying a software engineer he/she has been allocated on a project or an artefact. By default, all the artefacts allocated to a developer are intended as fully subscribed (all events concerning the artefact are notified), whereas all the other artefacts are intended as unsubscribed. However, each developer can override such behaviour specifying for each artefact the most appropriate notification level.

To support the developers during the customisation of the types of events they would like to be notified about, the ADAMS Event and Notification Management Subsystem assigns a relevance level (an indication of how much important is the notification of the event for the software engineers) to each event and uses this classification during the subscription phase. Developers are only requested to choose the minimum relevance level for which they want to be notified (see Figure 6) and the system will notify them all the events with a relevance level higher than or equal to the specified level. As an example, developers could choose to be notified each time an artefact they are allocated on is checked-out, to prevent the isolation problem, or they could only be interested on meaningful changes to that artefact. Such a distinction is possible due to the improvement of the granularity of change management introduced in ADAMS. In particular, each version of an artefact checked-in into ADAMS is tagged with an impact value determined comparing the new version with the previously stored version. For this reason, developers are required to specify the impact of the changes implemented when they check-in a new version of an artefact, so that it is possible to distinguish between unimportant changes, e.g. text corrections, code comments, language improvements, and meaningful changes, e.g. requisite modifications, new functionalities, error corrections. This information, together with subscriptions, is used by the Event Management Subsystem to decide whether to send a notification for the newer version or not.

Note that the event notification level is automatically tuned by the system that monitors the behaviour of each single software engineer whenever he/she receives a new notification. In particular, a software engineer that repetitively ignored notifications concerning an artefact, will realistically ignore later similar notifications and for this reason the subscription level is decreased accordingly. Obviously, the level can be re-established by the software engineer.

4.3. Event propagation

The traceability graph is also used to propagate event notifications, so that the developers can be aware of changes applied to artefacts their work depends on. This can contribute to anticipate potential changes thus reducing expensive later modifications. Event propagation
also reduces the complexity of subscribing several events for notification, avoids worthless redundancies, and prevents from forgetting to subscribe indirectly linked artefacts.

It is worth noting that the propagation across the traceability graph and the composition hierarchy can cause the proliferation of the number of generated messages. To avoid this proliferation, developers are allowed to specify whether they want to be notified about events concerning artefacts indirectly impacting their work, i.e., artefact not subscribed being source of an indirect traceability link with respect to artefact they subscribed. In particular, they can specify the maximum number of traceability links (distance) an event can cross (see Figure 6). If the subscribed artefact is linked to the artefact the event is related to by a chain of traceability links shorter than or equal to the number specified during the subscription, the software engineer will receive the notification message. With respect to the layered structure of the event classification, only events involving changes with high impact (meaningful changes) are propagated across the traceability links.

Composition links can also be used to propagate events. Even in this case, the software engineer can decide whether to be notified when the content of a composite artefact, i.e., one of the leaf nodes in the subtree, changes (see Figure 6). The propagation of events across a hierarchical composition follows a bottom-up direction. This means that events are propagated only from children to parents. Moreover, only events involving a leaf node are propagated to the composite artefact containing it. Differently from propagation across traceability links, events are always propagated across the hierarchy independently on the change impact, since even minor changes to a component artefact affect the composite artefact. Note that the exclamation mark used by the system to highlight that the content of a composite artefact is out-of-sync, is another (visual) notification mechanism. As a consequence, even in case the artefact manager does not subscribe notifications of changes to sub-artefacts, e.g., to reduce
the number of notifications, he/she can use this mechanism to be aware of the synchronisation among composite and component artefacts.

5. ADAMS evaluation

In this section we report the results of a usability study performed on ADAMS. In particular, the system was used as a process support system and document management system in the Software Engineering courses of the Computer Science program at the University of Salerno (Italy).

5.1. Experimentation definition and context

The goal of our study was to analyse the support given by ADAMS during the development of a software system. We performed a usability study to evaluate management functionalities and the support for the management of software artefacts, e.g., versioning and automatic recomposition, provided by ADAMS. Several motivations lead us to this experimentation. First, our aim was to test the support given by ADAMS in the definition of the work breakdown structure of the software system to develop. Second, we wanted to test the usability of ADAMS and to evaluate the benefits provided by an integrated environment for managing different types of software artefacts, e.g., requirements, use cases, UML diagrams, and source code, produced during the development process. Finally, we were also interested in analysing at which level of granularity artefacts are managed and how traceability information are maintained during software development. Thus, the quality focus was ensuring high support to concurrent software development, while the perspective was both of (i) Researchers, evaluating how effective is an advanced artefact management like ADAMS during the development of a software project; and (ii) of Project managers, evaluating the possibility of adopting such a tool within their own organisation.

The study was executed at the University of Salerno and involved Master and Bachelor students. In particular, ADAMS has been evaluated within the projects conducted by students of the Software Engineering courses of the Computer Science program at the University of Salerno (Italy). The experience involved 249 students allocated in 27 software projects to develop software systems with a distributed architecture (typically three tier) using mainly Java and web technologies and a relational DBMS (typically MySQL). During the development of these software systems ADAMS was used to schedule the work among developers and to manage the software artefacts produced.

5.2. Experimentation design and execution

To create a real usage scenario of the system we simulated —in an academic environment—a small industrial environment. In particular, we organised students in several teams where each project team included between six and eight undergraduate students with development roles, i.e., team members, and two master students with roles of project and quality management, respectively. The team members were Bachelor students (2nd year B.Sc.) attending the
course of Software Engineering, while Master students (2nd year M.Sc.) attending a course of Software Project Management participated to the experience with management roles. Both undergraduate and graduate students were from the same class with a comparable level of background. In particular, Bachelor students had previously attended Programming and Database Management courses, while Master students had also previously attended advanced courses of Programming and Software Engineering.

The process model adopted for software development was incremental: the students performed a complete requirements analysis and high level design of the software system to be developed and then proceeded with an incremental development of the subsystems. The goal of the software project was to release at least one increment by the deadline for closing the projects. The project manager was responsible for coordinating the project, defining the project schedule, organising project meetings, collecting process metrics, and allocating human resources to tasks. The quality manager was responsible for defining process and product standards of the project, collecting product metrics, and organising checklist-based artefact reviews for quality control. The project and quality managers were also in charge of building the traceability matrix for the software artefacts developed in their project. Finally, the choice of the granularity level for each of the artefacts to be produced was left to the managers of each project.

A preliminary evaluation of ADAMS was conducted in 2005 and involved about 156 students allocated on 17 projects. However, the evaluated version of ADAMS provides different functionalities as compared to the version presented in this paper, in particular concerning the fine-grained management of software artefacts. For example, the artefact composition was only realised by specifying a stereotyped traceability link and features related to automatically merging the composing artefacts into the composite artefact were not implemented. The results achieved in 2005 suggested to add and/or enrich some feature of ADAMS, in particular concerned with fine-grained artefact management. For example, we decided to separate and appropriately manage compositional and traceability links and add a functionality for the automatic composition of artefacts, thus enabling specific support to the management of composed artefacts.

After the enhancement of ADAMS, we conducted a replicated experimentation of the system that spanned over two academic years, i.e., 2006 and 2007. In 2006 the experimentation involved 32 students allocated on 4 software projects, while in 2007 the experimentation involved 61 students allocated on 6 software projects. In the next sections we report and compare the results of the two evaluation periods, i.e., 2005 and 2006-2007, to verify whether the added functionalities resulted in an improvement of the fine-grained artefact management of ADAMS. We distinguish between the two versions of the system and group data concerning 2006 and 2007, referring to these two experimentation as to the second evaluation period, while the results achieved in 2005 are related to the first evaluation period.

At the end of the projects, students evaluated ADAMS through a questionnaire. Each question refers to an attribute of four quality characteristics of interest, namely functionality, usability, reliability, and performance. Besides such characteristics, some quantitative evaluation questions have been provided to the subjects. A quantitative analysis was also performed to analyse how artefacts are decomposed during software development and how the fine-grained artefact structure influences the traceability management.
Table I. Projects of the first evaluation (2005)

<table>
<thead>
<tr>
<th>Project</th>
<th>Resources</th>
<th>Artefacts</th>
<th>Versions</th>
<th>Traceability Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>10</td>
<td>172</td>
<td>854</td>
<td>49</td>
</tr>
<tr>
<td>1-2</td>
<td>10</td>
<td>189</td>
<td>912</td>
<td>158</td>
</tr>
<tr>
<td>1-3</td>
<td>10</td>
<td>151</td>
<td>636</td>
<td>52</td>
</tr>
<tr>
<td>1-4</td>
<td>9</td>
<td>192</td>
<td>713</td>
<td>120</td>
</tr>
<tr>
<td>1-5</td>
<td>10</td>
<td>152</td>
<td>784</td>
<td>299</td>
</tr>
<tr>
<td>1-6</td>
<td>9</td>
<td>167</td>
<td>414</td>
<td>117</td>
</tr>
<tr>
<td>1-7</td>
<td>9</td>
<td>198</td>
<td>556</td>
<td>189</td>
</tr>
<tr>
<td>1-8</td>
<td>9</td>
<td>150</td>
<td>767</td>
<td>90</td>
</tr>
<tr>
<td>1-9</td>
<td>8</td>
<td>201</td>
<td>864</td>
<td>205</td>
</tr>
<tr>
<td>1-10</td>
<td>8</td>
<td>325</td>
<td>854</td>
<td>107</td>
</tr>
<tr>
<td>1-11</td>
<td>10</td>
<td>140</td>
<td>584</td>
<td>45</td>
</tr>
<tr>
<td>1-12</td>
<td>10</td>
<td>244</td>
<td>719</td>
<td>131</td>
</tr>
<tr>
<td>1-13</td>
<td>7</td>
<td>213</td>
<td>628</td>
<td>39</td>
</tr>
<tr>
<td>1-14</td>
<td>8</td>
<td>163</td>
<td>490</td>
<td>121</td>
</tr>
<tr>
<td>1-15</td>
<td>10</td>
<td>190</td>
<td>565</td>
<td>126</td>
</tr>
<tr>
<td>1-16</td>
<td>9</td>
<td>215</td>
<td>1002</td>
<td>96</td>
</tr>
<tr>
<td>1-17</td>
<td>10</td>
<td>483</td>
<td>635</td>
<td>131</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>9.18</strong></td>
<td><strong>191.24</strong></td>
<td><strong>704.53</strong></td>
<td><strong>122.06</strong></td>
</tr>
</tbody>
</table>

Table II. Projects of the second evaluation (2006-2007)

<table>
<thead>
<tr>
<th>Project</th>
<th>Resources</th>
<th>Artefacts</th>
<th>Versions</th>
<th>Traceability Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>10</td>
<td>386</td>
<td>1020</td>
<td>130</td>
</tr>
<tr>
<td>2-2</td>
<td>7</td>
<td>600</td>
<td>1669</td>
<td>202</td>
</tr>
<tr>
<td>2-3</td>
<td>7</td>
<td>581</td>
<td>1365</td>
<td>195</td>
</tr>
<tr>
<td>2-4</td>
<td>8</td>
<td>421</td>
<td>869</td>
<td>142</td>
</tr>
<tr>
<td>2-5</td>
<td>16</td>
<td>1100</td>
<td>2954</td>
<td>248</td>
</tr>
<tr>
<td>2-6</td>
<td>9</td>
<td>167</td>
<td>361</td>
<td>81</td>
</tr>
<tr>
<td>2-7</td>
<td>9</td>
<td>838</td>
<td>1901</td>
<td>117</td>
</tr>
<tr>
<td>2-8</td>
<td>9</td>
<td>678</td>
<td>1591</td>
<td>300</td>
</tr>
<tr>
<td>2-9</td>
<td>9</td>
<td>260</td>
<td>701</td>
<td>178</td>
</tr>
<tr>
<td>2-10</td>
<td>9</td>
<td>448</td>
<td>652</td>
<td>181</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>9.3</strong></td>
<td><strong>547.9</strong></td>
<td><strong>1308.3</strong></td>
<td><strong>177.32</strong></td>
</tr>
</tbody>
</table>
5.3. Experimentation results

During the first experimentation period, the average number of artefacts produced within each project was 191 (140 artefacts were produced for the smallest project and 325 artefacts for the largest project). On average, for each artefact about 3.78 different versions were produced (in some cases, up to 22 different versions of the same artefact were produced). During the second evaluation period, the average number of artefacts produced within each project was 548 (167 artefacts were produced for the smallest project and 1100 artefacts for the largest project). On average, for each artefact about 2.35 different versions were produced. See Tables I and II for details.

The distribution of answers for each question of the survey questionnaire can be found in Tables III, IV, and V. The tables report the questions and the distribution of answers for each question and for each period of experimentation, i.e., 2005 and 2006-07. This is useful to highlight the improvements achieved by implementing some of the suggestions collected at the end of the first evaluation period. Aggregate data are not reported since the system has changed between the two evaluation periods.

In general, we noticed that the user’s perception of the tool usefulness has globally incremented during the second experimentation period. In particular, the overall perceived usefulness of ADAMS rose from 75.00% to 86.90% (see question Q1.1.1 in Table III). Regarding the evaluation of the usability for the main functionalities of ADAMS, the project and resource management functionalities are considered intuitive by over 80% and 90% of the users, respectively (Q.2.1.1 and Q.2.1.2 in in Table V). Moreover, versioning management functionalities as well as the software artefact lifecycle management have been considered intuitive (questions Q.2.1.3 to Q.2.1.5 in Table V).

The user interface is considered friendly (in the questionnaire we used the term friendly as synonym of “easy to use and understand”, as this term was very popular among undergraduate students) by about 80% of the users in both the evaluation periods (Q.2.2.1 in Table V). Similar results have been achieved in 2005 for easiness of Input / Output (Q.2.2.5 and Q.2.2.6 in Table V) and data retrieval (Q.2.2.7 in Table V). These results have also been confirmed in the second evaluation (see Table V).

Reliability and performance evaluation show good results in terms of adequacy of the performances with respect to the user needs and considering the limitation of a web application in terms of bandwidth restrictions. However, question Q.2.4.1. does not show a high satisfaction for the response time, in particular in the first evaluation period. This was mainly related to the time needed to access the work breakdown structure of the project. In fact, in case of projects with a very high number of artefacts, the system suffered some scalability problems.

In particular, the version of ADAMS used during the first experimentation period represented the project work breakdown structure by means of an XML document visualised by means of a XSL parser tool. However, this approach required the system to load the complete structure of the selected project into memory, resulting in a waste of system resources. Moreover, the

†The ADAMS project itself has been put under configuration management, resulting in about 2,000 source code artefacts and more than 1,0000 documentation artefacts with a hierarchy depth of 10 levels.
Table III. Evaluation: Generic Functionalities

<table>
<thead>
<tr>
<th>Question</th>
<th>2005</th>
<th>2006-07</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1.1.1. Was ADAMS useful during the software development process?</td>
<td>75.00</td>
<td>25.00</td>
</tr>
<tr>
<td>Q1.1.2. Was the possibility to add comments to artefacts useful?</td>
<td>42.14</td>
<td>57.86</td>
</tr>
<tr>
<td>Q1.1.3. Was the possibility to send a feedback to the artefact manager useful?</td>
<td>42.45</td>
<td>57.55</td>
</tr>
<tr>
<td>Q1.1.4. Was the functionality to visualise the users allocated to a specific artefact useful?</td>
<td>80.58</td>
<td>19.42</td>
</tr>
<tr>
<td>Q1.1.5. Was the functionality to visualise the graph composed by the artefacts of the project and the traceability links useful?</td>
<td>55.40</td>
<td>44.60</td>
</tr>
<tr>
<td>Q1.1.6. Was the possibility to subscribe specific artefacts useful?</td>
<td>51.80</td>
<td>48.20</td>
</tr>
<tr>
<td>Q1.1.7. Was the event hierarchy useful?</td>
<td>66.19</td>
<td>33.81</td>
</tr>
<tr>
<td>Q1.1.8. Did you preferred working at a fine-grained artefact management level (instead of a coarse-grained artefact management)?</td>
<td>41.43</td>
<td>58.57</td>
</tr>
<tr>
<td>Q1.1.9a. Did you use branching and merging functionalities?</td>
<td>29.29</td>
<td>70.71</td>
</tr>
<tr>
<td>Q1.1.9b. Did you use branching and merging functionalities?</td>
<td>18.97</td>
<td>81.03</td>
</tr>
<tr>
<td>Q1.1.10. Do you agree with the proposed event hierarchy?</td>
<td>36.59</td>
<td>63.41</td>
</tr>
<tr>
<td>Q1.1.10a. Did you use branching and merging functionalities?</td>
<td>68.35</td>
<td>31.65</td>
</tr>
</tbody>
</table>

size of the XML file transferred from the web server to the browser caused high response times due to the bandwidth limits especially for students connected at home with a dial-up connection. To address this issue, in the new version of ADAMS (used in the second experimentation period) the user interface used to visualise the project artefacts has been implemented with the AJAX (Asynchronous JavaScript and XML) technology. When the software engineer accesses the project artefacts, the system provides only the root artefacts. Anytime the software engineer selects one of the artefacts visualised, the web browser recovers the children of the selected artefact from the database, thus expanding the visualisation of the artefact hierarchy. In this way, only the information about the artefacts whose visualisation is actually required is extracted from the database. This greatly improved the response time and the scalability of the system.

5.4. Fine-grained versioning and traceability management

The third part of the functionality evaluation questionnaire addresses user consideration about the tool usage and design decisions such as the role-based access control policy, the fine-grained approach, and the usefulness of branching and merging when a fine-grained approach is preferred. In particular, it is worth focusing the attention on questions Q.1.1.8 and Q.1.1.9 as they give an indication of the effect of fine-grained management of software artefacts on user behaviour. In fact, the percentage of software engineers that used the branching and merging functionalities is very low, about 30% in the first experimentation and less than 3% in the second experimentation.

It is important to note that the results of question Q.1.1.9 is influenced by the answers of the software engineers that decided to use a coarse-grained management that leads to...
### Table IV. Evaluation: Management Functionalities

<table>
<thead>
<tr>
<th>Question</th>
<th>2005</th>
<th>2006-07</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q.1.2.1. Were ADAMS team management functionalities adequate?</td>
<td>56.25</td>
<td>43.75</td>
</tr>
<tr>
<td>Q.1.2.2. Did Project Management functionalities fit your needs?</td>
<td>93.75</td>
<td>6.25</td>
</tr>
<tr>
<td>Q.1.2.3. Did Resource Management functionalities fit your needs?</td>
<td>87.50</td>
<td>12.50</td>
</tr>
<tr>
<td>Q.1.2.4. Did the revision process fit your needs?</td>
<td>77.14</td>
<td>22.86</td>
</tr>
<tr>
<td>Q.1.2.5. Did the Versioning Management functionality fit your need?</td>
<td>76.98</td>
<td>23.02</td>
</tr>
<tr>
<td>Q.1.2.6. Do you agree that a role-based policy to access the artefacts is preferable to a permission-based access control?</td>
<td>81.25</td>
<td>18.75</td>
</tr>
<tr>
<td>Q.1.2.7. Was the possibility to define different horizontal links useful?</td>
<td>81.25</td>
<td>18.75</td>
</tr>
<tr>
<td>Q.1.2.8. Were negative traceability links useful to discover content overlapping among artefacts?</td>
<td>43.75</td>
<td>56.25</td>
</tr>
</tbody>
</table>

### Table V. Evaluation: Usability, Reliability, and Performance

<table>
<thead>
<tr>
<th>Question</th>
<th>2005</th>
<th>2006-07</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q.2.1.1. Are Project management functionalities intuitive?</td>
<td>93.75</td>
<td>6.25</td>
</tr>
<tr>
<td>Q.2.1.2. Are Resource Management functionalities intuitive?</td>
<td>93.75</td>
<td>6.25</td>
</tr>
<tr>
<td>Q.2.1.3. Are creation and activation procedures intuitive?</td>
<td>80.00</td>
<td>20.00</td>
</tr>
<tr>
<td>Q.2.1.4. Are downloading and uploading procedures intuitive?</td>
<td>83.57</td>
<td>16.43</td>
</tr>
<tr>
<td>Q.2.1.5. Is the artefact revision procedure intuitive?</td>
<td>82.14</td>
<td>17.86</td>
</tr>
<tr>
<td>Q.2.1.6. Is the traceability link creation intuitive?</td>
<td>87.50</td>
<td>12.50</td>
</tr>
<tr>
<td>Q.2.1.7. Is the similarity measure between two artefacts intuitive?</td>
<td>71.88</td>
<td>28.13</td>
</tr>
<tr>
<td>Q.2.2.1. ADAMS User Interface is friendly</td>
<td>82.86</td>
<td>17.14</td>
</tr>
<tr>
<td>Q.2.2.2. User Interface components layout is satisfactory</td>
<td>91.43</td>
<td>8.57</td>
</tr>
<tr>
<td>Q.2.2.3. Understanding and memorising the User Interface components’ behaviour is easy.</td>
<td>90.71</td>
<td>9.29</td>
</tr>
<tr>
<td>Q.2.2.4. Understanding the main features is easy</td>
<td>91.43</td>
<td>8.57</td>
</tr>
<tr>
<td>Q.2.2.5. Providing input to the system is easy</td>
<td>83.57</td>
<td>16.43</td>
</tr>
<tr>
<td>Q.2.2.6. Obtaining output from the system is easy</td>
<td>83.57</td>
<td>16.43</td>
</tr>
<tr>
<td>Q.2.2.7. Searching for data is easy</td>
<td>83.57</td>
<td>16.43</td>
</tr>
<tr>
<td>Q.2.2.8. Terms used for commands are clear</td>
<td>90.00</td>
<td>10.00</td>
</tr>
<tr>
<td>Q.2.3.1. The User interface prevents errors</td>
<td>80.71</td>
<td>19.29</td>
</tr>
<tr>
<td>Q.2.3.2. The system allows the user to easily suspend or abort an operation at any time.</td>
<td>87.86</td>
<td>12.14</td>
</tr>
<tr>
<td>Q.2.3.3. In case an error occurs, the system allows an easy recovery.</td>
<td>76.43</td>
<td>23.57</td>
</tr>
<tr>
<td>Q.2.3.4. ADAMS support for error recovery is satisfactory</td>
<td>89.29</td>
<td>10.71</td>
</tr>
<tr>
<td>Q.2.4.1. Waiting time between request and response is tolerable.</td>
<td>60.00</td>
<td>40.00</td>
</tr>
<tr>
<td>Q.2.4.2. ADAMS Project and Resource Management Subsystems’ performances are satisfactory.</td>
<td>81.25</td>
<td>18.75</td>
</tr>
<tr>
<td>Q.2.4.3. ADAMS Traceability Management Subsystem’s performances are adequate.</td>
<td>84.38</td>
<td>15.63</td>
</tr>
<tr>
<td>Q.2.4.4. The ADAMS Artefact Management Subsystem’s performances are adequate.</td>
<td>84.17</td>
<td>15.83</td>
</tr>
<tr>
<td>Q.2.4.5. ADAMS Event &amp; notification Management Subsystem’s performances are adequate.</td>
<td>93.53</td>
<td>6.47</td>
</tr>
</tbody>
</table>
Table VI. Number of fine-grained artefacts for the main project deliverables (first evaluation)

<table>
<thead>
<tr>
<th>Project</th>
<th>RAD</th>
<th>SDD</th>
<th>ODD</th>
<th>Testing</th>
<th>Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>48</td>
<td>24</td>
<td>18</td>
<td>54</td>
<td>25</td>
</tr>
<tr>
<td>1-2</td>
<td>30</td>
<td>35</td>
<td>42</td>
<td>45</td>
<td>34</td>
</tr>
<tr>
<td>1-3</td>
<td>24</td>
<td>24</td>
<td>37</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>1-4</td>
<td>31</td>
<td>27</td>
<td>32</td>
<td>72</td>
<td>28</td>
</tr>
<tr>
<td>1-5</td>
<td>24</td>
<td>30</td>
<td>45</td>
<td>31</td>
<td>21</td>
</tr>
<tr>
<td>1-6</td>
<td>27</td>
<td>16</td>
<td>22</td>
<td>71</td>
<td>28</td>
</tr>
<tr>
<td>1-7</td>
<td>32</td>
<td>21</td>
<td>24</td>
<td>74</td>
<td>45</td>
</tr>
<tr>
<td>1-8</td>
<td>24</td>
<td>29</td>
<td>44</td>
<td>44</td>
<td>6</td>
</tr>
<tr>
<td>1-9</td>
<td>32</td>
<td>33</td>
<td>38</td>
<td>24</td>
<td>72</td>
</tr>
<tr>
<td>1-10</td>
<td>52</td>
<td>33</td>
<td>23</td>
<td>65</td>
<td>150</td>
</tr>
<tr>
<td>1-11</td>
<td>22</td>
<td>22</td>
<td>37</td>
<td>27</td>
<td>29</td>
</tr>
<tr>
<td>1-12</td>
<td>39</td>
<td>27</td>
<td>25</td>
<td>79</td>
<td>73</td>
</tr>
<tr>
<td>1-13</td>
<td>29</td>
<td>22</td>
<td>15</td>
<td>104</td>
<td>42</td>
</tr>
<tr>
<td>1-14</td>
<td>26</td>
<td>19</td>
<td>26</td>
<td>73</td>
<td>18</td>
</tr>
<tr>
<td>1-15</td>
<td>31</td>
<td>22</td>
<td>25</td>
<td>76</td>
<td>41</td>
</tr>
<tr>
<td>1-16</td>
<td>25</td>
<td>41</td>
<td>26</td>
<td>62</td>
<td>59</td>
</tr>
<tr>
<td>1-17</td>
<td>29</td>
<td>24</td>
<td>31</td>
<td>79</td>
<td>18</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>30.88</strong></td>
<td><strong>26.41</strong></td>
<td><strong>30.00</strong></td>
<td><strong>59.47</strong></td>
<td><strong>42.41</strong></td>
</tr>
</tbody>
</table>

Table VII. Number of fine-grained artefacts for the main project deliverables (second evaluation)

<table>
<thead>
<tr>
<th>Project</th>
<th>RAD</th>
<th>SDD</th>
<th>ODD</th>
<th>Testing</th>
<th>Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>201</td>
<td>12</td>
<td>44</td>
<td>31</td>
<td>55</td>
</tr>
<tr>
<td>2-2</td>
<td>403</td>
<td>18</td>
<td>16</td>
<td>37</td>
<td>90</td>
</tr>
<tr>
<td>2-3</td>
<td>384</td>
<td>20</td>
<td>27</td>
<td>37</td>
<td>74</td>
</tr>
<tr>
<td>2-4</td>
<td>157</td>
<td>12</td>
<td>1</td>
<td>52</td>
<td>50</td>
</tr>
<tr>
<td>2-5</td>
<td>297</td>
<td>16</td>
<td>48</td>
<td>268</td>
<td>257</td>
</tr>
<tr>
<td>2-6</td>
<td>65</td>
<td>15</td>
<td>10</td>
<td>26</td>
<td>50</td>
</tr>
<tr>
<td>2-7</td>
<td>213</td>
<td>32</td>
<td>69</td>
<td>105</td>
<td>357</td>
</tr>
<tr>
<td>2-8</td>
<td>158</td>
<td>27</td>
<td>36</td>
<td>116</td>
<td>245</td>
</tr>
<tr>
<td>2-9</td>
<td>137</td>
<td>12</td>
<td>7</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2-10</td>
<td>164</td>
<td>72</td>
<td>85</td>
<td>57</td>
<td>66</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>217.90</strong></td>
<td><strong>23.60</strong></td>
<td><strong>34.30</strong></td>
<td><strong>73.00</strong></td>
<td><strong>134.40</strong></td>
</tr>
</tbody>
</table>
the need for more branching and merging operations. To better analyse the effects of fine-grained management with respect to the necessity of branching and merging, we have further investigated the questionnaire answer taking into consideration the answers of the software engineers that preferred a fine-grained management (Q.1.1.9a) and a coarse-grained management (Q.1.1.9b). In this case, the results confirm the conjecture that the need for branching and merging decreases when a fine-grained management is used. In fact, concerning the users that preferred the fine-grained artefact management, less than 19% of users in the first experimentation period and none of the users in the second experimentation period used branching and merging functionalities (see question Q.1.1.9a in Table III).

To better understand the use of the fine-grained functionalities of ADAMS we analysed the composition of produced artefacts. We observed that the main software documents produced were the Requirements Analysis Document (RAD), the System Design Document (SDD), the Object Design Document (ODD), the Test Plan, and the test execution documents, besides project and quality management documents (all these documents were produced following the templates defined by Brügge et al. [35]). Using the composition links, such documents were decomposed into a large number of sub-artefacts. Tables VI and VII report the number of fine-grained artefacts the main documents were decomposed into in the first and second experimentation period, respectively. The table does not report data about source code artefacts, since students paid more attention to the production of high-level artefacts, i.e., analysis and design artefacts, than low-level artefacts, i.e., source code. In particular, students produced a low number of source code artefacts, as they implemented only a part of the designed system. Moreover, in many cases they used ADAMS to upload an archive containing the entire project source code in the first experimentation period or entire packages in the second experimentation period.

By analysing the data reported in Tables I, II, VI, and VII, we note that comparing the two evaluation periods, on the average, the number of fine-grained artefacts incremented from 191 to 548. In particular, the number of artefacts composing the RAD documents has incremented, on average, from 31 to 218. The increased number of artefacts for RAD documents in the second evaluation was due to their decomposition at requirements, use cases, and UML diagrams (sequence, activity, and state-chart diagrams) level. As a consequence, the managers were able to trace links at a finer granularity level. However, even if the number of artefacts increased, there was a disproportionately lower increase in the number of links, i.e., on average the number of traceability links only increased from 122 to 177 per project. By analysing the average number of artefacts for each type of project deliverable we observed that students produced almost the same number of artefacts for the SDD, ODD, testing, and management documents (see Tables VI and VII). In the second experimentation students produced a higher number of artefacts for the RAD document as compared to the RAD artefacts produced in the first experimentation (217.30 vs 30.80). From the analysis of the RAD decomposition we observed that in the first experimentation students extracted from the RAD document only the use cases, thus managing only a few number of RAD artefacts at a fine granularity level. Consequently, only use cases were traced onto design and testing artefacts. In the second experimentation students performed a finer decomposition of the RAD document managing at a fine granularity level different types of analysis artefacts, such as requirements, use cases, UML diagrams, and screen mock-ups. Nevertheless, students only traced use cases onto design
and testing artefacts following the same behaviour as the students in the first experimentation. As a consequence, the average number of traced links did not proportionately increase with the number of artefacts produced.

It is important to note that students generally preferred to pay more attention to activities such as document review, code inspection, and testing, rather than traceability management. Moreover, the traceability information was not continuously maintained during software development. Indeed, students traced links occasionally during the different phases of the project. In particular, since they were required to submit a traceability management report (as well as project management and quality management reports) monthly, generally students updated the traceability layer few days before the deadline for submitting the report. Thus, due to the strict deadlines students did not have so much time for traceability identification. Consequently, they decided to trace only “core” artefacts.

5.5. Threats to validity

This section describes the threats to validity that could affect our experience. During the experiments we tried to simulate a real working environment, although we used undergraduate and master students. In particular, managers have been selected among final-year master students, so they have a very good analysis, development and programming experience, and they are not far from junior industry analysts. In addition, team members are students enrolled in a software engineering course, so they have sufficient knowledge of the software development life cycle. Unfortunately, in a real working environment there are other pressures so the results cannot be completely generalised to the industrial context and thus the experiments should be replicated with practitioners.

The artefact repositories built by the students during this experience have a small/medium size but they represent a good benchmark as they cover many different projects with different goals. However, students mainly use the fine-grained approach during the RAD development, managing functional requirements, scenarios, use cases, participating objects, and sequence diagrams as different artefacts. On the other hand, the granularity of the ODD was quite coarse. This has been influenced by the fact that most of this document was composed of automatically generated documentation for the interfaces of the application objects, that were provided as html documents by the javadoc utility.

A separate potential threat to validity is related to the design of the post-study questionnaire. To this aim, we tried to follow the guidelines provided in [36]. A design point that may pose a validity threat is the choice of the answer. Most studies ask questions on a scale of measure, because the answer is rarely so absolute as “yes” or “no”. Indeed, more precise results could be achieved asking questions on a scale of measure (e.g., Likert scale [36]). In our questionnaire we opted for true/false questions. However, we asked subjects to clarify their answers. In this way we mitigated the threat represented by true/false questions and we also collected feedbacks from the subjects. This was particularly useful in the first evaluation period, as user feedbacks allowed us to improve the fine-grained artefact management functionalities of the system before the second evaluation period.

Finally, the results achieved are based only on the perception students had of the usability of the user interface. Stronger results should be achieved performing another type of usability
study based on a quantitative analysis. For instance, counting the error (or measuring the time) performed by students carrying out an assigned task. The lower the number of errors performed (or the time spent to perform the task) the higher the usability of the user interface. Unfortunately, we did not perform such a study and we evaluated the usability of the user interface only through a qualitative analysis of data collected through a survey questionnaire.

6. Related Work

ADAMS is an extensible system for software artefact and project management aiming to provide a full feature environment supporting software engineers during software development and evolution. In this article we have mainly focused on the fine-grained artefact management of ADAMS, in particular the hierarchical versioning and the traceability management. For this reason, related work will address these two topics.

6.1. Hierarchical Versioning Management Systems

Several commercial and research Configuration Management systems providing versioning have been proposed [3] [4] [5] [6] [7]. However, most of these tools do not provide an active support to hierarchical documentation versioning. Even if the user can organise composite artefacts using folders and subfolders, such an approach is limited to an enumeration of components, without providing any structural information relating components to the composite.

Hierarchical versioning has been approached by Asklund et al. [37] who propose a unified extensional versioning model. In this model, a document is a tree structure consisting of local data, a composite node (which is a collection of nodes), or a link node (which is used to represent arbitrary relations between documents). The model has been implemented in COOP/Orm [38] and CoEd [39]. COOP/Orm [38] is a research tool developed at the University of Lund, in Sweden, supporting the natural hierarchical structure of textual documents, and providing version control and merge facilities within distributed groups. CoEd [39] is a visual environment for textual hierarchical document versioning, developed at the University of Aalborg, in Denmark. CoEd uses the same document management approach adopted by COOP/Orm, focusing on support to \LaTeX{} documents. Both COOP/Orm and CoEd are limited to textual documentation, thus they do not provide a suitable support to the documentation produced within a software project. The support to the development team is limited to the versioning management functionality. In particular, context awareness and communication mechanisms are poor. Despite these tools allow a document to be organised in a tree-based structure, mechanisms to automatically recompose the overall document are not provided, resulting in scalability problems when the document hierarchy is complex. Moreover, traceability between different artefacts is not provided.

CoVer [40] is a hypermedia version server implemented on top of the HyperBase hypermedia engine [41]. Versioned objects are represented by multi-state objects, representing a composite of references to the states of versioned objects. Similarly to COOP/Orm and CoEd, CoVer is designed to manage a specific type of artefact (hypermedia documents). As a consequence, the support provided during the software development process is lacking.
Stellation [42] is another fine-grained CMS aiming to provide versioning functionalities for source code. Instead of having the code organization of the system dominated by its layout in source files for storage, programs and related development artefacts are stored as small, fine-grained units called fragments. In Stellation, all storage is in terms of fragments and aggregation is used for combining groups of fragments into larger structures. Stellation is limited to the fine-grained versioning of source code, so it does not support all the other types of documents produced during a software project.

Volzer et al. [43] use composition to define hierarchical views of product structure, independently of the underlying artefact-repository structure. The approach is implemented in SubCM, a lightweight tool designed to be used on top of a CMS. However, the hierarchical versioning management in SubCM is conceived to provide configuration management of hierarchies of subsystems, thus being not suitable for software artefact management.

Molhado [44] is an interesting fine-grained version control system supporting versioning of any fine-grained units at any structural levels in a software document (including source code and documentations), and providing version history and change information at the semantic level. Molhado manages the evolution of a system and its software components as a whole entity: the project entity. This entity contains a directed graph in which each node is associated with a component that contains a reference to an atomic or another composite component of the software system. Similarly to ADAMS, Molhado [44] provides fine-grained versioning functionalities for different types of artefacts, covering most of the artefacts produced during the software development process. However, besides providing a great flexibility to the software engineer that is allowed to tailor the granularity level of the artefact decomposition according to his/her needs, ADAMS also supports the main types of artefacts produced during the software development process, providing functionalities to automatically recompose complex artefact hierarchies when needed. Moreover, Molhado does not provide specific support to the development team. In particular, the scheduling for the artefacts to be produced is not managed, thus requiring the software engineers to use external tools to manage the work breakdown structure of the project.

Molhado also allows the software engineers to manage traceability links providing versioning for them, allowing users to return to a consistent previous state of a software document. However, no support for semi-automatic traceability link recovery is provided. Versioning of diagrams is provided by Molhado through the versioning of the XML representation of the diagram. However, in contrast to ADAMS, the elements contained in the diagram are not directly managed, thus limiting the advantages of the fine-grained approach adopted by the tool, as the sharing of diagram components among different diagrams is not enabled.

Odyssey-VCS [45] is an interesting version control system providing the fine-grained versioning of UML diagrams. However, even if it implements an algorithm to merge and detect conflicts between two different versions of the same UML diagram element, synchronous modelling functionality is not provided as well as support for different types of artefacts. Similarly to previous approaches, the tool is designed to manage a specific type of artefact, thus being not suitable to manage software project’s diverse artefacts.
6.2. Traceability Management Systems

Several research and commercial tools are available that support traceability between artefacts. Unfortunately, all the proposed tools are intended for managing the traceability links during their definition and (in some cases) evolution. As a consequence, they are usually apart from the artefact management or the versioning management system, thus requiring the software engineer to maintain the information between the repositories consistent. DOORS [13] and Rational RequisitePro [12] are commercial tools that provide effective support for recording, displaying, and checking the completeness of traced artefacts using operations such as drag and drop [13] or by clicking on a cell of a traceability link matrix [12]. RDD.100 [10] (Requirements Driven Development) uses an Entity Relationship Attribute (ERA) repository to capture and trace complicated sets of requirements, providing functionalities to graphically visualise how individual pieces of data relate to each other and to trace back to their source. TOOR [11] is a research tool in which traceability is not modelled in terms of simple links, but through user-definable relations that are meaningful for the kind of connection being made. This lets developers distinguish among different links between the same objects. Moreover, TOOR can relate objects that are not directly linked using mathematical properties of relations such as transitivity. Recently, artefact traceability has been tackled within the Ophelia project [46] which pursued the development of a platform supporting software engineering in a distributed environment. In Ophelia the artefacts of the software engineering process are represented by CORBA objects. A graph is created to maintain relationships between these elements and navigate among them.

An important issue of traceability management concerns the evolution of traceability links. Nistor et al. [47] developed ArchEvol an environment that manages the evolution of architecture-to-implementation traceability links throughout the entire software life cycle. The proposed solution maintains the mapping between architecture and code and ensures that the right versions of the architectural components map onto the right versions of the code (and vice versa), when changes are made either to the architecture or to the code. Maletic et al. [48] propose an approach to support the evolution of traceability links between source code and UML artefacts. The authors use an XML-based representation for both the source code and the UML artefacts and applies meta-differencing whenever an artefact is checked-in to identify specific changes and identify traceability links that might have been affected by the change.

Traceability in ADAMS is also used to propagate events that occurred on an artefact to the dependent artefacts. A similar functionality is provided in [49], [50], and [51]. OSCAR [49] is the artefact management subsystem of the GENESIS environment [15]. It has been designed to non-invasively interoperate with workflow management systems, development tools, and existing repository systems. Each artefact in OSCAR possesses a collection of standard meta-data and is represented by an XML document containing both meta-data and links to artefact content that may include linking relationships with other artefacts. OSCAR introduces the notion of “active” software artefacts that are aware of their own evolution. To support such an active behaviour, every operation on an artefact generates events that may be propagated by an event monitor to artefacts deemed to be interested in such events by their relationships with the artefact generating the event. Chen and Chou [50] have proposed a method for consistency management in the Aper process environment. The method is based on maintaining different
types of traceability relations between artefacts, including composition relations, and uses triggering mechanisms to identify artefacts affected by changes to a related artefact. Cleland-Huang et al. [51] have developed EBT (Event Based Traceability), an approach based on a publish-subscribe mechanism between artefacts. When a change occurs on a given artefact having the publish role, notifications are sent to all the subscriber (dependent) artefacts. Despite supporting context awareness by managing event notification and using the traceability layer to propagate events, these tools do not offer facilities to limit the proliferation of notifications. As an example, the main problem of EBT, is the higher number of messages that are generated within a process if too many links are maintained [51] and the fact that often traceability links are not correctly maintained during a software development process [52]. ADAMS gives developers the possibility of customising the set of events they would like to be notified about, thus avoiding undesired notifications on one hand and receiving notifications that are not planned by the traceability layer on the other hand. Moreover, it implements an automating tuning of the subscription level taking into consideration the software engineer behaviour to optimise the event notification mechanism.

Context awareness support in ADAMS is also provided through the possibility to visualise the resources involved in the development of an artefact and detailed information about the locks, such as which artefacts are locked (even indirectly), who has checked-out the artefact, and when he/she performed the operation.

Another distinctive feature of ADAMS with respect to other approaches is the support for the management of the entire life cycle of artefacts that includes a checklist-based inspection and review phase. An artefact lifecycle management is provided by OSCAR, by means of the "active" artefacts. However, OSCAR does not directly support artefact review as well as automatic re-composition of fine-grained artefacts.

Traceability has also been used in the PROSYT environment [19] to model software processes in terms of the artefacts to be produced and their interrelationships. This artefact-based approach results in process models composed of simpler and more general operations than the operations identified using an activity based approach. PROSYT is able to manage the inconsistencies between a process instance and the process model [53] by tracing the deviating actions and supporting the users in reconciling the enacted process and the process model, if necessary.

Finally, none of the tool presented provide functionality to support the software engineer during traceability recovery. Traceability recovery methods have been proposed in the literature [52] [54] [55] [56] [57] [58] [59] [60] but they have not been implemented in artefact management systems.

7. Conclusion

In this article we have presented ADAMS, an extensible system that integrates project management and artefact management features. Besides coordinating distributed software engineers working on the same artefact and providing versioning facilities like configuration management systems, ADAMS also supports context-awareness through event subscription and notification and artefact traceability. In particular, traceability links are used to propagate
events concerning changes to an artefact to the dependent artefacts. We focused on the fine-grained management of software artefacts provided by ADAMS, in which a complex artefact is decomposed in a hierarchy of sub-artefacts where each piece is managed as a different entity and versioning is provided for both the atomic entity and the composite entities it is contained in, giving the possibility to define more detailed traceability links. The software engineer is able to specify traceability links between entities within the same document, as well as between parts of different documents, resulting in a more precise impact analysis. Fine-grained artefact management reduces the likelihood of conflicts arising when different software engineers apply changes to the same document. In fact, in case the document is decomposed in sub-artefacts, a conflict occurs only if changes concern the same sub-artefact. Finally, a composition of finer-grained sub-artefacts allows the project manager to precisely assign the responsibilities for each document section and the software engineers to focus on the work product they have been assigned to. This is particularly important in a distributed software development environment, where the work breakdown structure is defined, tasks are assigned to the software engineers, and the intermediate work products, e.g., requirements and actors description, scenarios, use cases, sequence diagrams, test cases, are successively merged in a deliverable document.

ADAMS has been used as project and artefact management system in the Software Engineering courses of the Computer Science program at the University of Salerno (Italy). At the end of the projects, the students evaluated ADAMS through a questionnaire. Each question referred to an attribute of four quality characteristics of interest, namely functionality, usability, robustness, and performance. The results of the evaluation questionnaire have been presented and discussed. The results seem promising, although replication in different contexts, with different subjects and software projects, is needed to corroborate our findings. Replicating the experimentation in real context and with professional software engineers is part of the agenda of our future work. Another direction for future work will be the integration with widely used concurrent versioning systems (e.g., CVS or Subversion) to provide the functionalities of ADAMS for existing software repositories as well as to allow the access to the ADAMS repository information from external tools e.g., for mining software repository, metrics extraction, or repository analysis and visualisation tools.

Acknowledgments

We would like to thank the anonymous reviewers for their detailed, constructive, and thoughtful comments that helped us to improve the presentation of the results in this paper. Special thanks are also due to the students who were involved in the usability study of ADAMS as subjects. The work described in this article is partially supported by the project TOCAI (Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet), funded by MiUR within the FIRB 2005 program.

REFERENCES


