DoxyChange – Visualization of Software Changes

Frédéric Plouffe

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Computer Science at

Concordia University

Montréal, Québec, Canada

September 2005

Your file Votre référence
ISBN: 0-494-10294-2
Our file Notre référence
ISBN: 0-494-10294-2

ABSTRACT

Visualization of Software Changes

Frédéric Plouffe

Visualization techniques are often used when a legacy software system becomes nearly unmaintainable, in which case software visualization can be applied to regain an understanding of the software system, such that maintenance activities can take place. One of such visualization techniques consists of identifying how a given module ended up in its current state. Therefore by visualizing the history of the source code repository, it may be possible to visualize software changes and understand how the software system has evolved over time. The presented work, which is part of the *CONCEPT framework (Comprehension Of Net-CEntered Programs and Techniques)* [RIL03], applies visualization techniques to any of the diagrams generated by *Doxygen* [HEE05] from the retrieved source code repository, in order to display software changes between two specified revisions. The graphics are obtained by reusing the powerful *Graphviz Dot* [GAN00] layout algorithm, which enables software engineers to easily grasp the complexity of a given module and facilitate the task of program comprehension by visualizing the system at a higher level of abstraction. Moreover, this project addresses some of the shortcomings, found in some reverse engineering tools, by providing an intuitive navigation and interaction within the generated UML diagrams. Finally, this research project is aimed to be use by any projects, which are currently using *Doxygen* as an online documentation tool [HEE05p], with the hope to bring visualization of software changes to these developers.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Juergen Rilling, for his encouragement, for his precious feedback and for his guidance and support, which have made the completion of my thesis possible.

I would also like to thank my family and friends for their encouragement and support. I would also like to dedicate this work to them.

I would also like to thank my colleagues within the CONCEPT framework for their insight and contribution to the overall CONCEPT framework [RIL03].

I would also like to thank Dimitri van Heesch for his *Doxygen* lexical scanner [HEE05], *Trolltech* for their *Qt Toolkit* [TRO05], *AT&T* for their *Graphviz Dot* tool [GAN00], Yuan Wang for his X-Diff tool [WAN05] and the *Apache* team for their *Xerces-C* library [ASF05x], which are all crucial components of this research project.

Finally, I would like to thank: *Le Fonds québécois de la recherche sur la nature et les technologies (FCAR) #86204* [QUE05], the *Natural Sciences and Engineering Research Council of Canada (NSERC)* [CAN05] and the *Hydro-Québec Graduate Awards* [HYD05] for funding and sponsoring this research topic and supporting me financially during my bachelor and master degree.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

The purpose of reverse engineering is to identify software components and to regain an understanding of the software architecture at different levels of abstraction. This research project describes various methods for recovering a given software system architecture, design and API to help maintainers understand the evolution of a given software system. As observed by Müller, the maintenance of legacy systems often leads to an acute software crisis inside a given company [MUL00] and as everyone in the industry might have observed, "well-specified, documented, disseminated and controlled architectures are the exception rather than the rule" [WOO99]. Moreover, legacy systems often provide incoherent, deteriorated and poor documentation, which is often due to time, economic or manpower constraints [GOV00][HOL99][OCA01][ORI98]. This research project mainly focuses on documenting a software API by producing a wide variety of UML diagrams and visualizing software changes in general.

Software comprehension is currently one of the main challenges in the field of software engineering. Software comprehension consists of analyzing a software system, in order to regain a better understanding of the system by using various reverse engineering methods [MUL00]. In 1978, Lientz and Swanson have revealed that 50% to 85% of a system's total costs was devoted to software maintenance purposes, with an in-depth survey of 69 organizations [LIE78][LIE80][PIG97]. Even though such studies proved for years the

usefulness of producing proper documentation during software development for maintenance purposes, many software systems are still currently developed without producing any proper documentation for maintainers. This means that many software systems are still being updated without any proper knowledge about the original system design [DAI02][REA03]. Even nowadays, documentation is still considered as a necessary evil [GUE01]. Many managers still do not see the importance of creating detailed documentation during development, since time spent on documentation largely reduces the amount of time spent on coding new features; consequently, increasing the cost and time to market period [GOV00][HOL99][OCA01][ORI98] without immediately providing any visible added value to the product [BER03]. No matter the reasons, the problem remains that "well-specified, documented, disseminated and controlled architectures are the exception rather than the rule" [WOO99]. As a result, most companies start using reverse engineering tools only when the gap between the desirable and available information becomes critical [MUL00]. This often occurs when a change is applied to the source code that leads to disastrous results. "At that point, reverse engineering techniques are inserted in a 'big bang' attempt to regain a useful understanding" of the software system at hand [MUL00]. In general, most software reverse engineering tools consist of:

- A parser, a syntax analyzer or a lexical scanner, which are often written using *Flex* [NEE00], *Bison* [PRO05] or *Yacc* [JOH75][DIC04]. Otherwise, a simple rule-based search engine or some automated scripting tool using regular expressions, such as: *grep* [LEV05], *sed* and *awk* [BER04][KOL05], *Perl* [HIE05] or similar are used.

- An analysis tool to derive meaningful information such as: an Abstract Syntax Tree, UML diagrams [OMG05w], design patterns, software metrics, source code documentation or any other meaningful information.

- A visualization tool to display navigable 2D, 3D or even textual representations of the retrieved information [SUN04d][HEE05].

1.1 Software Maintenance

Software development is an activity, which consists of designing and creating source code. During this process, many opportunities can arise for errors to be introduced, for instance, when the requirements or specifications are imperfectly or incorrectly specified [DEU82]. Consequently, software maintenance is an important, costly and critical phase of the software engineering life cycle. This phase starts after the delivery of the system where many tasks, such as: error corrections, fixing bugs, adding new features or refactoring, need to be performed on the legacy system. Some recent studies have even shown that over 90% of the total software cost will be spent during the software maintenance phase [KOS04][ERL00][MOA90]. Meanwhile, roughly 60% to 70% of the total effort will be directed towards software comprehension and analysis [RIL03c]. In this context, the usage of Computer Aided Software Engineering tools (CASE) can help software engineer to better understand critical software systems by visualizing how these systems currently work and provide sufficient insight, in order to save time and money during the software maintenance phase. Software visualization is important, since

it provides a higher level of abstraction view of the software system, while only displaying information relevant to the particular task. This is widely illustrated by the old proverb by the mathematician, Frederick R. Barnard, which once said: "One picture is worth ten thousand words" [BAR04]. While software visualization can be represented using textual or graphical representation, this research project's emphasis is on the graphical representation of source code by following the standard UML notation. UML is widely used since its standardization in 1997 [LEM05] to gain insight and to understand complex software system at various levels of abstraction by providing a key set of simple graphical elements to express various ideas about how a software system works or how a software system was designed. The main objective is to reduce the time, energy and effort required by software engineers to grasp critical information about a software system and to fully visualize the mental model needed to understand the various components and relationships among these components. As a result, a very large number of visualization tools, techniques and methods have flourished in the past decade to deal with the various visualization tasks to be performed.

1.2 Thesis Contribution

This research project proposes a new software visualization tool called *DoxyChange* that is part of the *CONCEPT framework (Comprehension Of Net-CEntered Programs and Techniques)* [RIL03] and also part of the *Doxygen* framework [HEE05]. This *DoxyChange* visualization tool focuses on visualizing software changes among various revisions of a given software system, in order to evaluate the evolution behind each revision.

This research project will focus on displaying various class diagrams using the Unified Modeling Language (UML) [OMG05] via a web interface using DHTML (Dynamic HyperText Markup Language) [REF05h], JavaScript [NET96] and SVG (Scalable Vector Graphics) [AND03] technologies. The diagrams being displayed are following the UML standard, since it is widely used for representing various software design concepts in both academic and enterprise environments [LEM05]. This research project will address some of the UML shortcomings by adding a graphical representation of software changes to the classical definition of UML class diagrams, which will be explained in detail in Chapter 5. The objective is to show in an intuitive manner important software changes within a single UML diagram such that a software engineer can easily evaluate what was changed between two revisions of the same software system. *DoxyChange* utilizes filtering and clustering techniques to hide or display meaningful contextual information, in order to provide a simple and usable human-like drawing of these systems, which is both scalable and navigable. The objective is to provide a program comprehension tool for the various levels of abstraction by pre-calculating some general guidelines as to which contextual information might be meaningful or not in the current visualization context. Finally, the textual source code can also be generated and therefore accessed from the diagram itself. The end result is automatically generated from two source code revisions with minimal input from the end user. The only requirement is to write a proper configuration file using the default configuration template.

1.3 Thesis Problem Statement

As part of this research project, a software visualization tool, called *DoxyChange*, was created to visualize changes between two distinct revisions of a given source code repository. The emphasis of this research project is to focus on visualizing and understanding the evolution of a software system, as it was recorded by the repository system, by using a graphical SVG/XHTML representation of these changes. Furthermore, a survey of the various technologies and techniques currently used to address the problem of parsing, synchronizing, merging, differentiating and visualizing source code will be presented. Finally, some sample use cases will be provided to illustrate possible real life scenarios where this technique may be applied right now in the industry. As a result, the research question of this thesis is the following: *"Which techniques and technologies can be used to visualize software changes from a UML perspective, in a distributed environment?"*

1.4 Problem Domain Assumptions

The following assumptions were made in order to restrict the problem domain:

- Source files must compile properly without any compilation errors or warnings. Otherwise, the *Doxyfile* configuration parameters must be used to work around any outstanding issues proper to any toolkit or proprietary compiler extensions or missing definitions.

- Any source files must have a current parsing context code fragment smaller than 256 KB. If this limit is ever reached, *Doxygen* will exit with a fatal compilation error saying: *"input buffer overflow, can't enlarge buffer because scanner uses REJECT"*. Otherwise, every generated *Flex* compilation unit must be increased within the mentioned limitations, by using the *increasebuffer.pl* script.

- The total size of all input source files for one revision must be smaller than 16 MB uncompressed for a machine with only 512 MB of RAM.

- The input documents for *X-Diff* must be valid *DOT-XML* files.

- *X-Diff* comparisons where one of the non-existing file pair is fail-safe matched against a script generated *DOT-XML* empty file, which will always produce false positive changes without any warning. However, such false positive changes can easily be detected, since the diagram will show every item as being inserted or deleted.

- Call graph files are associated generically by name without using the random number suffix and therefore, might produce incorrect call graph changes when more than one random number suffix exists.

- The *Doxygen* output must be inside a single flat directory with sufficient free *inodes*.

- *DoxyChange* requires disk space for roughly 12 to 20 times the size of both input source code pair.

- *Doxygen* must have enough heap memory, which must roughly be equal to 12 times the size of the input source code.

- Ideally, for revisions larger than 5 MB, all the input, temporary and output files should be loaded on a RAMDISK. Notice that if the RAMDISK is too small, *Doxygen* will abort and the entire batch job will have to be restarted from scratch.

- No virus scanner or intensive I/O tasks should be loaded in memory, while running *Doxygen*. Otherwise, the I/O overhead speed penalty will be roughly 4 times slower than without it.

- Items to be differentiated must have at least the name in common, since renames cannot be associated safely without creating improper false positive using *X-Diff*. Therefore, items with different names will not be match together and instead will be represented as two distinct insert/delete operations instead of one insert/delete pair.

- Item signature modifications are associated by name. Therefore, only a return type, parameter, attribute or protection modification on such item can be matched successfully and be represented as only one insert/delete pair.

- The documents once differentiated must produce a valid *Change-DOT* file to be converted into an equivalent *SVG* file by using the *Graphviz Dot* utility.

- The only user input required to visualize the software change is to create a proper configuration file.

- Large *Doxygen* tasks are heavily time consuming and should be performed offline.

- Differentiation of individual file pairs can be done online on pre-parsed *DOT-XML* files.

- Qt slots and signals are not parsed properly by *Doxygen* and will only show its function name without any parameters on the UML class diagrams.

1.5 Thesis Organization

This thesis is organized as followed:

- Chapter 2 provides an overview of software maintenance, program comprehension and software visualization.

- Chapter 3 introduces a survey of state-of-the-art version control system to keep track of the history of source code and their inherent advantages and limitations.

- Chapter 4 introduces a survey of state-of-the-art differentiating and merging tools with their limitation toward differentiating XML.

- Chapter 5 presents the motivations behind the presented research work, as well as the overall design and implementation details.

- Chapter 6 presents some case study and an evaluation of this software visualization tool by analyzing the collected test bench results.

- Chapter 7 presents the conclusions and includes the discussion about some future research work.

Chapter 2 Software Engineering Survey

Software Engineering is an active field of research and development since 1968 [RAN79], which consists of analyzing and proposing various methods, solutions and techniques to solve common day to day problem, related to large software system in general. More formally, Software Engineering is defined as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software" in general [RAD90].

Software Engineering consists of many sub-fields such as Reverse Engineering, which is a specialized domain of expertise that focuses uniquely on extracting vital information from software system, mainly to perform various maintenance tasks such as: adding features, fixing bugs or porting the system to a new platform. In 1990, Chikofsky and Cross defined Reverse Engineering as "the process of analyzing a subject system to (1) identify the system's components and their interrelationships and (2) create representations of the system in another form or at a higher level of abstractions" [CHI90]. Reverse engineering is crucially important for "systems with poor documentation", since "the code is the only reliable source of information about the system" [MUL00].

Software Comprehension is another specialized field within the Software Engineering domain, which focuses on the understanding and comprehension of software system through various methods, including various Software Visualization techniques. Mostly, "Software Comprehension is required when a programmer maintains, reuses, migrates, reengineers, or enhances software systems" [BRI03]. Software Comprehension can also be defined as "the process of taking computer source code and understanding it" [DEI90] or "the task of building mental models of the underlying software at various abstraction levels, ranging from models of the code itself, to models of the underlying application domain, for maintenance, evolution and reengineering purposes" [MUL94]. Finally, it can also be defined as "a process whereby a software practitioner understands a software artifact using both knowledge of the domain and/or semantic and syntactic knowledge, to build a mental model of its relation to the situation" [BRI03].

Software Visualization is "a discipline that makes use of various forms of imagery to provide insight and understanding and to reduce complexity of the existing system under consideration" [KNI98]. Software Visualization is an "exciting field of computational science spurred on in large measure by the rapid growth in computer technology, particular in graphics workstation hardware and computer graphics software" [ARE93]. In general, Software Visualization can be described as a method of computing, which "transforms the symbolic into the geometric, enabling researchers to observe their simulations and computations" [MCC87]. It is also "the process of transforming information into a visual form, enabling users to observe the information", where the "resulting visual display enables the scientist or engineer to perceive visually features

which are hidden in the data but nevertheless are needed for data exploration and analysis" [GER94]. In a certain way, "visualization offers a method for seeing the unseen" [MCC87], since the "goal of scientific visualization is to promote a deeper level of understanding of the data under investigation and to foster new insight into the underlying processes, relying on the humans' powerful ability to visualize. In a number of instances, the tools and techniques of visualization have been used to analyze and display large volumes of, often time-varying, multidimensional data in such a way as to allow the user to extract significant features and results quickly and easily" [BRO92, Chapter 1]. Furthermore, "choosing the appropriate representation can provide the key to critical and comprehensive appreciation of the data, thus benefiting subsequent analysis, processing, or decision making" [ROB91]. Consequently, "the primary objective in data visualization is to gain insight into an information space by mapping data onto graphical primitives" [SEN94] that can be useful to the software engineer.

2.1 Software Comprehension Techniques

Software comprehension techniques consist of analyzing software systems to better understand them. However, analyzing large software systems is a real challenge, since it becomes extremely difficult to filter out the large amount of information into something that can be understood and analyzed easily by a programmer [SIL00]. Reducing the amount of information displayed is crucially important, since psychologists have already demonstrated that displaying more than seven outstanding pieces of information is simply too much for the human brain [MIL56]. This is especially true when dealing with a large

number of model elements inside a single UML diagram [EGY00]. To be useful, such complex diagrams must be simplified by using some abstraction methods. One way to increase the level of abstraction is to use concept analysis [GIU99], concept assignment [BIG93] or rule-based techniques [KEL01][KRA96][PIN02]. These techniques mostly find or regroup similar concepts based on some given criterion for simplification purposes in order to obtain a high-level overview which clearly identifies implicit or explicit design decisions to be considered. As a result, *DoxyChange* can be configured to use or not this simple rule-based simplification technique, which mainly resizes the UML diagram by removing siblings which cannot be displayed within the allocated drawing space. Consequently, a class for which an immediate sibling was removed is drawn using a bright red border rectangular box using the truncated CSS stylesheet, as it will be explained in Figure 5.5.

2.2 Information Gathering Problem

Software systems can range from thousand to million lines of source code in certain complete case study [KDE05]. Obviously, the larger the system, the more complex and tedious it becomes to maintain it. In this context, it becomes critical to use reverse engineering tools and techniques to regain a sufficient understanding about the software system to perform common maintenance tasks at hand.

Currently, one of the major problems in software engineering is that the evolution of a given software system is extremely difficult to study, to analyze or to comprehend due to

their very large size and complexity. Consequently, when a bug is reported, it becomes extremely hard to understand: how this bug appeared, when it appeared, in what circumstance, how the system evolved from there, who is responsible for creating or not discover it, how it can be fixed and how similar bugs can be avoided in the future. Viewing such kind of information involves parsing several revisions of the source code repository and extracting crucial information on how the source code has changed over the years, while providing some insight about the internal design decisions, cohesion and coupling of the system as it evolved. In this context, the objective is to provide a meaningful way of displaying such information and to provide a proper tool for doing so. Furthermore, if such tool could provide a way of adding features, fixing bugs or refactoring the source code at a higher level of abstraction then that would be a nice feature to have, which would largely benefit the maintainability of large software systems.

2.3 Reverse Engineering and Architectural Design Recovery

Reverse engineering can be described as a process to "analyze a subject system to identify its current components and their dependencies, and to extract and create system abstractions and design information" [CHI90]. One of the major problems with large software system is that there exist at least two different types of architecture for any kind of software: the 'conceptual' or 'intended' architecture and the 'concrete' or 'implemented' architecture [RIV00][WOO99]. The term conceptual or intended architecture describe the architecture that exists conceptually within the software

documentation or within the minds of the software engineers, which might not reflect the current implementation due to some changes made during the maintenance phase. The current concrete or implemented architecture is the one that can be retrieved by analyzing the actual source code. Therefore, the goal of our reverse engineering tool is to regain some knowledge about the concrete architecture, in order to fix any misconceptions that may occur by providing a proper concrete documentation of the current software system, which might not possibly match the documented conceptual architecture. As a result, the main goal is to extract all suitable information from the source code, in order to reconcile both types of architecture. Consequently, this design reconciliation provides a means to figure out what the original design was, what the current design is, and how the future software system design should look like.

2.4 Reverse Engineering Tools

Currently, "many reverse engineering tools focus on extracting the structure of legacy systems with the goal of transferring this information into the minds of the software engineers trying to reengineer or reuse it. In corporate settings, reverse engineering tools still have a long way to go before becoming an effective and integral part of the standard tool set that a typical software engineer uses day-to-day" [MUL00]. The only widely known exceptions to this rule are *Javadoc* [SUN04d] and *Doxygen* [HEE05], which are used by a large pool of projects and companies spread all around the world [HEE05p]. Most others reverse engineering tools have a hard time realizing any market

breakthrough, apart from those which are already included in forward software engineering toolkit such as *IBM Rational Rose* [RAT05].

Existing reverse engineering tools mainly focus on the following software visualization aspects, which are: runtime relationships, data flows, code structures and various other kinds of diagrams. The problem is that some of the retrieved information might not coincide with the available documentation provided with the software system [KAZ97], due to false positive errors, incorrect implementation or simply documentation, which is not up to date. Consequently, it has long been recognized that no single set of tools will work for every case [KAZ97][WOO99]; therefore, various tools might be used to address different kinds of maintenance problems. To solve this specific issue, some authors [FER02][KAZ97][WOO99] have proposed the use of specialized reverse engineering frameworks, which integrates various reverse engineering tools within one specialized framework, rather than proposing a single reverse engineering tool that only work for a small set of maintenance tasks. The idea behind such a framework is to create an "open, lightweight environment that provides an infrastructure for opportunistic integration of a variety of tools and techniques" [WOO99] by specifying a plug-in format or by providing a formal schema for each data format being used [FER02]. A similar methodology is used within the *CONCEPT framework* [RIL03]; however, since *DoxyChange* used the *Doxygen* lexical analyzer [HEE05] instead of the *Sun™ Javac* [SUN04c] compiler framework used by other *CONCEPT* utilities, *DoxyChange* is more of a stand alone tool part of the *Doxygen framework* than a real *CONCEPT framework* component. The reason behind this important design decision will be explained in details in Chapter 5.

2.5 Information Gathering Techniques

Unsurprisingly, one of the major obstacles currently in reverse engineering is to find a proper parser for the C, C++ or Java programming language or simply a convenient way to easily extract information from such source code repositories into a higher level of abstraction. As everyone might know, writing a C++ parser is known to be an extremely painful task [PAR95] and finding a complete and fully working C++ parser that can be used for reverse engineering purposes is also a very painful task [SAM90][KNA99] [JAC99]. For instance, at the 1995 PCCTS workshop, Terrence Parr entitled his first slide as "C++ Parsing or how to induce a heart attack", since "C++ cannot be parsed without context-sensitive parsing, unbounded look-ahead and a great deal of pain" [PAR95]. In C++, most of the problems are related to parsing templates, namespace and ambiguous statements, or related to symbol, type and scope resolution problems. Consequently, one of the most time consuming parts of this research project was to analyze, compare and find a proper parser that could correctly parse Java or C++ source code and comments. As a matter of fact, more than 12 months were dedicated on this research project to find or create a 'proper compiler' for our specific reverse engineering needs until our *CONCEPT* group settled down on two possible choices: the *Sun™ Javac* compiler [SUN04c] or the *Doxygen* lexical analyzer [HEE05]. During this period, many parsers were evaluated including: a homegrown C++ parser, a *GCC-XML* extension [KIN04], a *JavaML* converter [BAD00], *IBM™ Jikes* [ABB05], the *Sun™ Javac* compiler [SUN04c], the *Doxygen* lexical analyzer [HEE05], among others. In the end, it was finally decided that *Doxygen* was a preferable choice for this research project, since it is widely used and it supports a wide variety of programming languages including: "C++, C, Java,

Objective-C, IDL (Corba and Microsoft flavors) and to some extent PHP, C#, and D" [HEE05]. It also generates UML diagrams [HEE05g], HTML and XML documentation by default [HEE05o]. Due to these outstanding parsing issues mentioned earlier, most other software comprehension research groups decided to resolve this problem by using a simple rule-based or partial syntax-based approach, instead of relying on a real compiler or lexical analyzer for various reasons [MUL00]. Some of these reasons are sometime barely frivolous to justify their approach while others are a bit more reasonable, such as: missing header files, implementation files or library files, usage of unsupported compiler extensions or source code which cannot be compiled for other reasons [WOO99]. In any case, the problem of using a partial rule-based approach, a partial lexer or a simple set of regular expressions [MAR03] to analyze the source code is that it is definitively not a complete parser. In other words, this means that the analysis process does not even try to understand the real parsing context or the overall source code design, since it is mostly relying on primitive keyword matching techniques, which does not work very well on an ambiguous language such as C++. For instance, the Figure 1 below shows a very short C/C++ code sample, which clearly "demonstrates exciting features of GNU C used in old versions of Linux" [BRI01]. Consequently, these handy partial query or lexer solutions have a bad tendency of producing false positive, useless or incorrect results in some of the corner cases of the C/C++ language. Furthermore, while such methodology might work for simple analysis and might scale well for extremely large system due to speed issues, in some other context this methodology will not work, since it cannot grasp the actual meaning of some complicated language constructs successfully. As a result, it is much more preferable to use a complete compiler or lexical analyzer that will cover every

corner cases of the programming language and therefore, provide trustworthy results for reverse engineering purposes.

```
1  int a, b;
2  typedef int t, u;
3  void f1() { a * b; }
4  void f2() { t * u; }
5  void f3() { t * b; }
6  void f4() { int t; t * b; }
7  void f5(t u, unsigned t) {
8    switch ( t ) {
9    case 0: if ( u )
10   default: return;
11   }
12 }
```

Figure 2.1: Demonstration of GNU C features used in old versions of Linux [BRI01]

2.6 Static versus Dynamic Source Code Analysis

In general, reverse engineering information may be retrieved using static or dynamic analysis. However, this research project focuses only on static analysis using a custom made *DOT-XML* extension to the open source *Doxygen* [HEE05] lexical analyzer. Most reverse engineering tools use static analysis [RIC99][WOO99], which involve parsing and analyzing the actual source code or software documentation [WOO99]. There also exist some reverse engineering tools that use information based on dynamic analysis, which utilize run-time debuggers or user-defined instrumentation for profiling purposes. Dynamic analysis typically consists of recording an execution trace at run-time for further analysis. Run-time debuggers are normally easier to use since they do not require any changes to the original source code, which is not the case for user-defined instrumentation, which must change the original source code explicitly by adding function calls between every statement to be profiled. In general, run-time debugging is

more accurate but also more costly, since it records everything including unnecessary information. Dynamic information mostly consists of the number of instances created, defined or used, concurrent thread information, and dynamic message or event information. Dynamic methods often provide search scope limitation, information that cannot be obtained statically, but also an incomplete coverage of the source code, which might be useful or not depending on the situation at hand. Finally, a dynamic execution trace is quite easy to obtain compared to the amount of work required by static methods to perform control flow analysis [RIC99].

2.7 Reverse Engineering and UML

Reverse engineering can produce all kinds of information from reports, metrics, statistics, diagrams and textual documentation, among others. Software engineering currently focuses for instance on creating formal requirement and specification documents [TRI98] and formal design documents [BAR98]. Such software engineering documents normally include various forms of descriptions and diagrams, such as UML diagrams. Consequently, one of the goals of reverse engineering is to recover such UML diagrams from the source code itself, in the case that these documents are not up-to-date or are simply inexistent.

2.7.1 The Unified Modeling Language

Since its introduction, the Unified Modeling Language (UML) has become one of the most commonly used graphical representation to visualize, specify, construct

and document software system artifacts [BOO99][MAT02][OMG05w]. Consequently, UML provides a *de facto* standard to document software systems, database systems and business processes. In general, UML can also document concrete things like classes, methods and reusable software components [BOO99].

2.7.2 UML Diagrams

In October 2004, the Object Management Group [OMG03a] standardized twelve different types of UML diagrams in their UML 2.0 specifications [SEL04], which are divided into three different categories to represent model management, dynamic behavior and static application structure [OMG05w]. The model management diagram subgroup includes package, subsystem and model diagrams. The behavior diagram subgroup consists of use case, sequence, activity, collaboration and statechart diagrams [OMG05w]. Finally, the structure diagram subgroup includes class, object, component and deployment diagrams [OMG05w].

2.7.3 Reverse Engineering and UML

Despite all the benefits of using UML, many software systems are still being developed without it. These systems could benefit from the abstraction and visualization properties provided by UML to facilitate their comprehension and, as a result, their maintenance activity [MAT02][OMG05w]. For the same reasons, a large number of tools have been developed to reverse engineer UML from source code [KOL00][MIT03][RAT02] [SEG02][TOG01]. In the following sub-sections, a survey of use cases, sequence and statechart diagrams and class diagrams often used in software engineering are presented.

In general, some reverse engineering tools, like this research project, provide a means by which a user can recreate UML diagrams from source code in order to provide an abstract view of the concrete implementation.

2.7.3.1 Use Case Diagrams

Use case diagrams normally facilitate the overall comprehension by describing the system behavior from the user's perspective. One of the UML reverse engineering approach involves the generation of use case diagram from the source code to visualize functional user requirements. For instance, Di Lucca et al. [LUC00] presented a reverse engineering approach for recovering use case diagrams from object-oriented source code. Their approach consists of exploiting the implicit link between use cases and object-oriented source code by using the notion of a thread, where a thread is defined as "an interleaved sequence of system inputs (stimuli) and outputs (responses)" [JOR94]. Since each use case describes a transaction consisting of different actions to be performed, a use case diagram can be derived by analyzing class method activation sequences triggered by input events and terminated by output events [LUC00].

The approach presented by Di Lucca et al. produced one or more use case diagrams at various levels of abstractions related to each other by including, extending or generalizing use case relationships [LUC00]. This approach helps software comprehension, since each use case can be traced back to the source code implementing it. On the other hand, this approach was tested on a small C++ software system where the produced use case model was successfully validated against the original software

requirement and specification documents. However, the validity of this case study is very limited due to the small size of the software system being analyzed. Unfortunately, no further analysis was performed to verify that this prototype actually works successfully for large software system.

2.5.3.2 Class Diagrams

While use case diagrams address the functional user requirement view of a system, a class diagram addresses only its static design view. In [MAT02], Matzko et al. presented *Reveal*, a tool which can reverse engineer class diagrams from C++ source code. To generate its class diagrams, *Reveal* uses *Graphviz Dot* [GAN00], a program that constructs directed graphs displayed in a hierarchical fashion. *Dot* is part of *Graphviz* [GAN00], a set of open source drawing tools developed by *AT&T Research Labs*. The class diagrams were generated in *PostScript* format [ADO00] and were displayed using *Ghostview* [KUH00]. To parse the input program under study, *Reveal* uses *Keystone* [MAL01], a parser front-end for the ISO C++ programming language [KOE96]. *Keystone* uses a bottom-up backtracking parsing algorithm enhanced by techniques such as token decoration to facilitate recognition of ambiguous language constructs [MAL01][MAT02]. While *Reveal* follows the UML standards as much as possible [OMG03b], it offers extensions to the standard notation to support namespaces, stand alone functions and friend functions found in the C++ language [KOE96].

In addition, Matzko et al. [MAT02] also compare *Reveal* with three other tools which reverse engineer class diagrams: *SuperWomble* [MIT03], *Rational Rose* [RAT05] and

Together [TOG01]. Their conclusion is that *Reveal* seems to be more precise than the three other tools [MAT02]. The suspected reason is that the other tools do not use a complete reverse engineering parsing process, but rather scan the source code for specific keywords without any context knowledge [ASV95][KOP97], which is only appropriate when dealing with missing header files or library files [WOO99]. In theory, a complete parser like *Keystone* [MAL01] should capture more meaningful and accurate information than a partial parser. This might partially explain why *Reveal* captures class relationships that are omitted by the other tools [MAT02].

As presented in [KOL02], Kollmann et al. also compare the capabilities of various class diagram reverse engineering tools. The examined tools consist of two popular commercial CASE tools: *Together* [TOG01] and *Rational Rose* [RAT05], and two research prototypes targeted on UML-based reverse engineering and round-trip engineering: *IDEA* [KOL00] and *Fujaba* [SEG02]. Their comparison shows that all the examined tools recognize the UML core elements such as basic class or interface and their inherent relationship since no fundamental element seems to have been omitted. The only exception to this rule is *Together* [TOG01], which failed to represent some parts of a plain association. Their survey demonstrates that abstract representation and advanced features recognition are clearly the domain of the research tools, since research tools mostly recognized all of the features, at least to a certain degree, while industrial tools did not address several of them at all. For instance, multiplicity, inverse associations and container resolutions were not addressed at all by *Rational Rose* or *Together* [KOL02]. As suggested by Matzko et al. [MAT02], this could be due to an incomplete

parsing and analysis process. The significant differences among the tools suggest that UML has not been used to its full potential in reverse engineering so far. This is somewhat surprising, since UML is *de facto* industrial standard for the representation of various design artifacts in object-oriented software development [KOL02][OMG05w]. In general, the abstractions provided by class diagrams have proven to be of great value when dealing with complex software systems [ICS99]. However, as software systems become more and more complex, UML class diagrams might simply contain too many model elements and might therefore need further abstractions in order to be fully comprehended [EGY00].

In [EGY99], Egyed and Kruchten proposed *Rose/Architect*, a plug-in tool for *Rational Rose* [RAT02], to visualize architecturally significant elements in a system design. Even though *Rose/Architect* can be used in forward engineering, it is much more valuable in reverse engineering, since it allows the extraction of meaningful architectural information from complex reverse engineered models. It mainly uses patterns and heuristics to define transitive relationships between classes, which are not directly connected. For instance, a transitive relationship may exist between two classes through a helper class forming a bridge between them, and therefore, hiding the inherent relationship [EGY99]. Transitive relationships from existing class diagrams can be derived by excluding classes and packages that are less meaningful for simplification purposes. This method can simplify complicated class diagram and therefore allow maintainers to better comprehend the software system. As a bonus, class can be assigned

to different abstraction planes, in order to create customized abstract view, where only the important classes and their relationships are displayed.

2.5.3.3 Sequence and Statechart Diagrams

Unlike use case and class diagrams, which focus on the static view of a software system, sequence and statechart diagrams model its dynamic aspects. Only a small number of tools currently support reverse engineering of dynamic information and *Shimba* is one of them. *Shimba* was developed by Systä [SYS00a], as a prototype environment to better understand software systems written in Java, by reverse engineering scenario and state diagrams. Scenario and state diagrams are similar to their UML counterparts, which are sequence and statechart diagrams. The main advantage of scenario diagrams is that they can be nested and they can easily express repetition of sequentially disconnected messages; however, they cannot express active and passive objects [SYS00a]. A state diagram can be characterized as a simplified statechart diagram, since it cannot be used to represent concurrency, history states, submachine references or timing constraints [SYS00b]. In order to reverse engineer scenario or state diagrams, *Shimba* runs the target Java system to be analyzed under a customized SDK debugger. Using the collected execution trace information about executed events, it generates diagrams using *SCHED* [KOS96][KOS98b], a dynamic modeling tool [KOS98a]. To validate their prototype, Systä models the internal functionality of the *Fujaba* class diagram tool [SEG02]. Using this case study, Systä found that the execution trace generated tends to grow rapidly, especially if it generates information at the object level [SYS00a]. This is due to the fact that a given object may take part in dozens of different scenario diagrams.

As a result, browsing the scenario diagrams can be very difficult and complicated [SYS00a]. On the other hand, the state diagram synthesis technique provides a quick and efficient way to focus on the overall run-time behavior of a single object [SYS00b].

2.7.4 UML Reverse Engineering Tools Summary

To fully understand the underlying architecture of an object-oriented software system, both static and dynamic analysis are needed [RIC99][SYS00a][WOO99]. However, most currently available reverse engineering tools only support UML diagrams using static information [SYS00a]. Therefore, UML diagrams are limited to the extraction of simple class diagrams without any additional information or abstraction levels. Even though a class diagram can aid with the understanding of a software system, it only provides a partial view of the underlying design decisions, since often there is no perfect mapping between the generated UML diagrams and the actual source code being analyzed [KOL02].

The dynamic view of a software system, provided by the Systä prototype [SYS00a], seems to be a good start towards the goal of reverse engineering dynamic information, which is very important for software systems that depend heavily on polymorphism. However, the diagrams generated by *Shimba*, do not follow the UML standard notation [OMG03b]. Even though they provide a variation of the sequence and statechart diagrams, *Shimba* should really follow more closely the standard UML notation, since it facilitates understanding and round-trip engineering due to the large amount of UML tools already available.

2.8 XML Definition

The eXtensible Markup Language (XML) [BRA00] is a W3C standard developed by the World Wide Web Consortium (W3C) [FUZ05w], since November 1996 [BRA96], to express any given set of data or document in a hierarchical fashion denoted with tags among a various range of computer systems. XML was mainly based on ideas developed from the Standardized Generalized Markup Language (SGML) [CON04] and the growing success and simplicity of the HyperText Markup Language (HTML) [RAG99] for exchanging information over the web. The main objectives were to be able to express any kind of hierarchical information and have a documented format to verify that the data followed a given grammar expressed by a Document Type Definition (DTD) [REF05d] or by an XML Schema [THO05]. Finally, XML was designed to support a wide range of character sets such as Unicode [UNI05], which provides a "consistent way of encoding multilingual text that enables the exchange of text data internationally" [UNI05i].

2.9 Various XML Formats

Some XML formats [BRA00] were developed to express some well-known problems such as: mathematical equations using the Mathematical Markup Language (MathML) [AUS03], some chemistry molecules using the Chemical Markup Language (CML) [MUR01] and even chess games using the Chess Game Markup Language (ChessGML) [SIC00]. In 1999, computer graphics using vectors and XML started to appear in technical publications and research committees [EBE99][FER99][AND04]. One of these approaches was called GraX, GXL or the Graph eXchange Language depending on the publication

date [HOLT00][HOLT02]. GXL had a very simple semantic structure and therefore, lacked many advanced features found in usual vector formats [HOLT02]. Another XML graphical format called SVG or Scalable Vector Graphic (SVG) [AND03] appeared in February 1999 [FER99], as a World Wide Web Consortium [FUZ05w] standard working draft proposal. Since its introduction, various tools were developed around SVG, which became popular two years later in November 2001, when the *Adobe SVG Viewer 3.0* made its appearance as a free commercial tool [ADO01].

Some research groups such as Rigi [WON98] have used *GXL* [HOLT00][COV00] and *SVG* [HOLG02] as a back-end for their reverse engineer visualization tools, since it provides a friendly and heavily documented plug-in format for external manipulations. For similar reasons, *DoxyChange*, uses *DOT-XML*, *DIFF-XML* and *SVG* as back-ends for external manipulation purposes. *DOT-XML* is similar to *GXL* syntax wise. However, it is based on a homegrown *XML* version of the *DOT* language [NOR05]. Additionally, *DOT-XML* wraps each node with an encoded node name *XML* tag identified by its "L" prefix. This special encoding is required, since the *X-Diff* differentiation is based on pure XML tag grouping instead of tag and attributes. The *DIFF-XML* format is only provided for third-party usage and is fundamentally similar to the original *X-Diff* output format.

XML was chosen since it provides an intuitive mechanism to express internal hierarchical data structures into a portable format, which can be viewed, manipulated and used by a wide variety of commercial tools. Furthermore, SVG is a very sophisticated format that can be parsed easily [AND03] compared to its corresponding bitmap formats such as:

GIF [COM90], JPEG [CLA05], PNG [MER05][ROE05], EPS [ADO92], which cannot be easily decomposed into a smaller sets of drawing operations. The other major reason is that SVG is a W3C standard [AND03] and visualization plug-ins, such as the *Adobe SVG Viewer 3.03*, are currently available [ADO05]. Additionally, the *Adobe SVG Viewer* supports more than 95% of the current standard [ADO01][AND02] and covers all of the important features required by *DoxyChange* for its visualization purposes. Finally, SVG is also used in a wide variety of applications, such as translating *ChessGML* into SVG dynamically using XSLT as shown below in Figure 2.2 [SIC00] or visualizing chemistry molecules using *Chem2D/3D* [MUR01] or *Jumbo 3* [MUR01j] as shown below in Figure 2.3 and Figure 2.4.

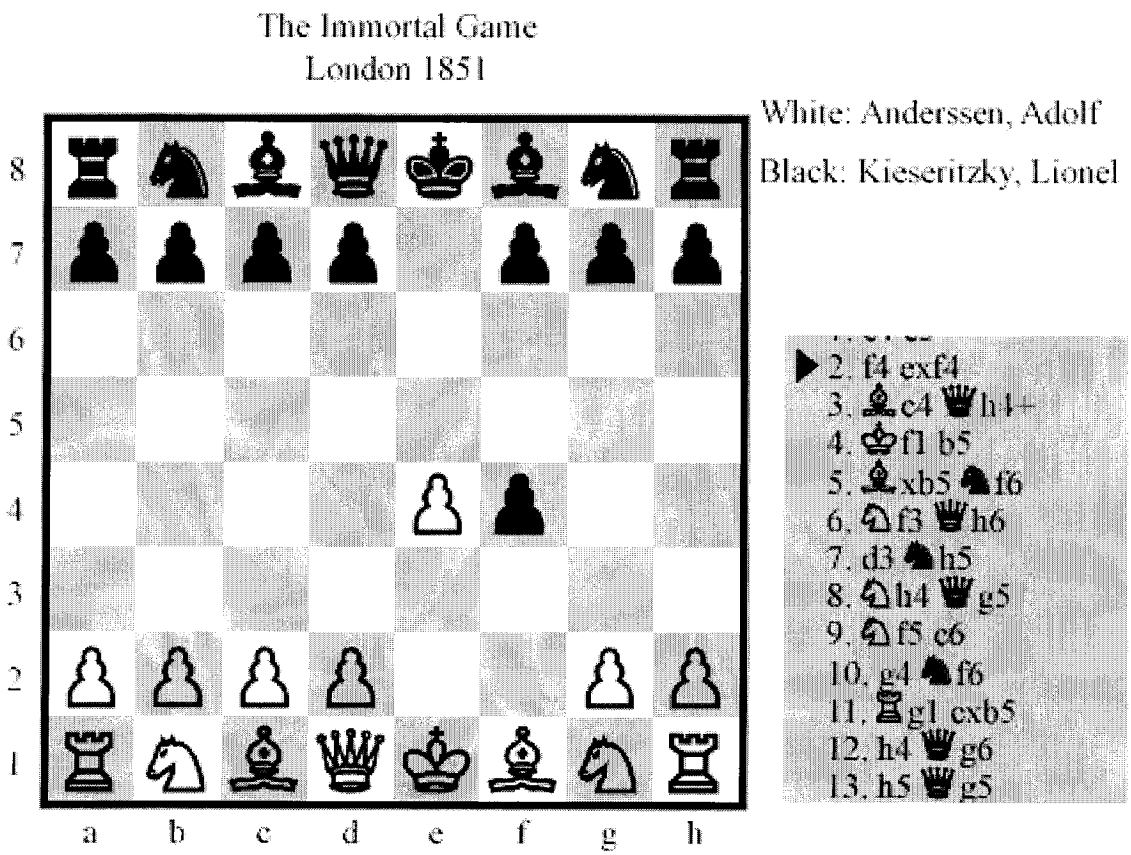

Figure 2.2: Screenshot: ChessGML to SVG – The Immortal Game [FRO02]

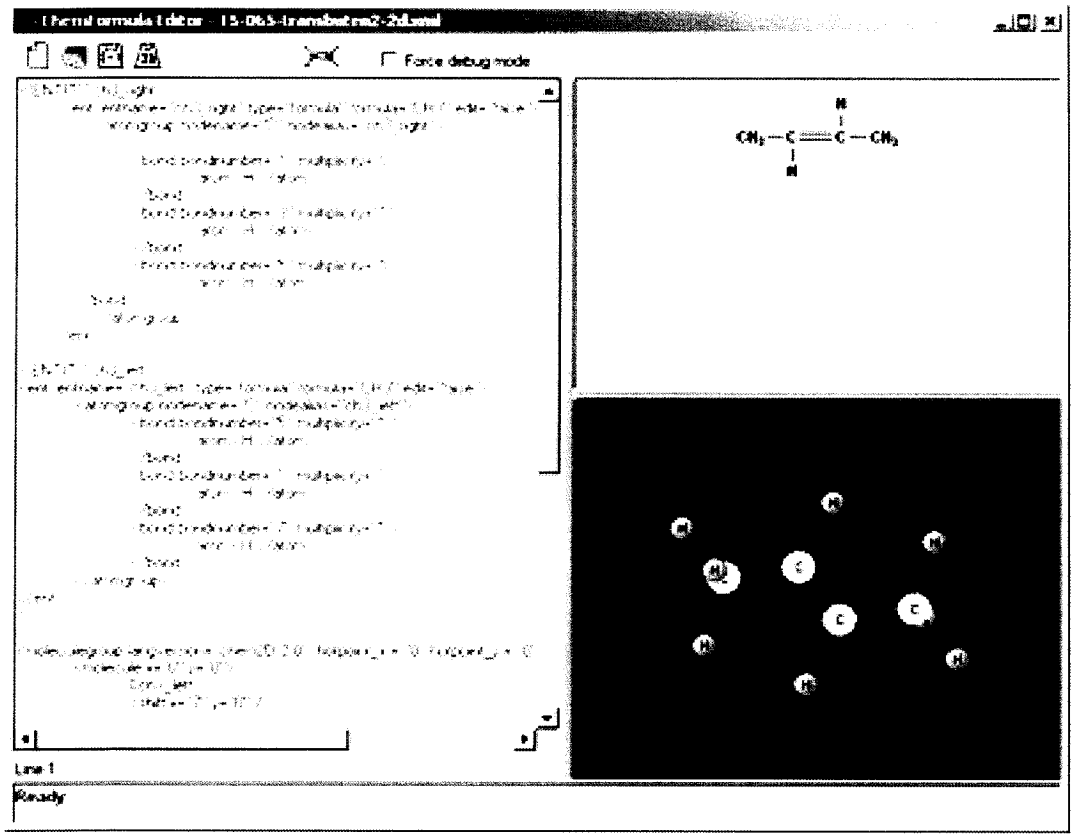

Figure 2.3: Screenshot: Chem2D/3D – Molecule Visualizer for CML [MUR01]

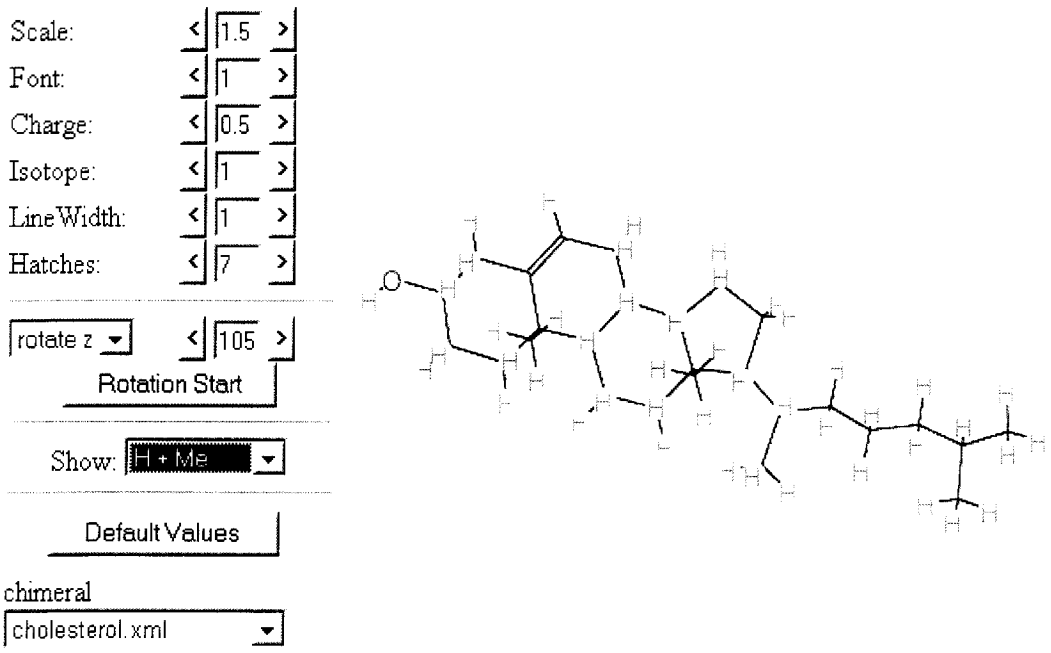

Figure 2.4: Screenshot: JUMBO 3.0-JS Molecule Browser for CML [MUR01j]

2.10 Software Evolution

In today's world, technology advancement is growing at a rapid pace and as a result new techniques, tools, technologies and ways of doing things are evolving constantly. In the end, it is very difficult for most companies to cope with the rapid technology advancement and to keep companies legacy system up to date with the new flavor *du jour* and the new ways of doing things. This problem is referred in the literature as software maintenance of legacy systems and software aging [PAR94]. Consequently, obsolescence is a fact that occurs every now and then, when old technologies are replaced with new ones, which are said to be more advanced, richer in features or just simply better. The same kind of obsolescence occurs with software and therefore, source code. Even though the algorithms or the business logic behind a given application might stay the same, the programming language might become obsolete, new ways of programming might have evolved or new programming concepts might have emerged. Even though software aging is inevitable, steps can be taken to limit its effects or temporarily fix it [GAL97][MUL00].

Software life cycle models are said to be the theory behind the life cycle of software systems, since they describes, using various theoretical models, how software system should be built in general. Most of these models, like the well-known waterfall model that was introduced by Winston Royce in 1970 [ROY70], are based on classic manufacturing process, which was successfully used for many centuries. By comparison, the field of software engineering is quite new compared to the few millennium of human history and the centuries of manufacturing process [FAN00]. Consequently, the field of software engineering is still considered to be in its young infancy and has a long way to

go before stabilizing into a proper manufacturing process. This is true even though a lot of research is currently performed, in order to find a real solution to this problem [FAN00]. In the mean time, major maintenance problems will continue to occur and the cyclic waterfall model is currently one of the common ways of doing proper software engineering, in general. Surprisingly, software engineering is not applied everywhere in the industry as statistics have shown that the number of companies which are at least CMM 3 [CMM02] are quite marginal [KUM01].

The classic reasons for not applying proper software engineering techniques are mostly tight budget, tight deadlines and an urgent need to rush a product to the market to maximize the immediate company profit before anyone else [FOR02]. As a result, software maintenance is currently estimated to cost as much as 65% to 90% of the total cost of a given legacy software system [KOS04][ERL00][EAS93][MOA90][HUF90] [POR88][LIE81]. Most of these legacy systems have an unknown architecture, no requirement, specification or design documents or even any proper documentation within the source code itself. Furthermore, to reduce the developing cost even further, some of these systems are developed by inexperienced programmers that have no software engineering background or knowledge about proper programming techniques in general. Consequently, some of these systems are highly coupled with low cohesion, built without any regards for the overall design, meaning that some of them will run into serious difficulties during the integration phase [RAY01], while leaving the maintenance phase into a well-known disastrous state [KOS04]. Another problem is that sometime, production system works with just enough luck and that special case or worst case scenario was not exercised during the simple "compile, run and ship" workflow.

This normally happens when serious unit or component testing [MYE79] is omitted and no proper code review or quality assurance process took place. This means that serious bugs or security flaws finally emerge in production when those corner cases are finally exercised [WHI02]. This usually ends up with a new "Service Pack" or "Patch" being released [TSP05] to gradually fix these major issues as they are finally discovered by hackers, researchers or end users [USC05]. As it might sound surprising, some large companies will completely ignore such issues, since the program works fine in the best case scenario and will not even bother why such worst case scenario might be important at all [BRO05]. If the issue is raised that test cases or complete documentation should be written, some companies will try to reduce the amount of time spent on doing those activities to a minimum. Simply to ensure that the team spent the maximum amount of time programming the system instead of validating, verifying, documenting or testing it; practically relying on pure luck for the production or maintenance phases.

Most companies start bothering with maintenance or documentation once a serious acute crisis arise that pose a real challenge during the maintenance phase [MUL00][EST00]. In such cases, some companies will simply restart the project from scratch over and over when maintenance becomes unmanageable. Some other companies will simply go out and try to regain an understanding of the software system by using a wide variety of reverse engineering tools [RIC00]. In the end, software design and architecture is still considered to be a craftsman skill, which is acquired by experience [ROU97]. As a result, it is important not only to create a proper software design and architecture but also to document it and to ensure that future maintenance activities respect this overall design and architecture documentation.

Chapter 3 Version Control System Survey

A crucial component to study the evolution of software system is the retrieval of source code history. Such source code history can be stored in various ways, through backups, release archives or more commonly from source code repositories. As a result, "source code repositories hold a wealth of information that is not only useful for managing and building source code, but also as a detailed log of how the source code has evolved during development" [WIL05]. This rich mine of information is stored in various format and special tools must be used to deal with the various source code release format available. The following sections will discuss the various places and formats where legacy source code can be freely and easily retrieved.

3.1 Open Source versus Closed Source

One way to obtain various source code revisions of a large software system is through the analysis of a well-known open source project [PER05], while another approach is to sign a non-disclosure agreement (NDA) [WIK05n] with some companies to access their source code repository. However, it is much easier to download source code, which is freely available, directly from an open source project web site than to find a company willing to share their source code for research purposes. Therefore, the former method is obviously preferred.

Many free web sites exist that are specialized in the hosting of free open source projects, such as: *SourceForge.net* [VAS05], *Savannah* [SAV05], *BerliOS* [FOK05], *Tigris* [COL05], *KDE* [KDE05], among others. Once an open source project is being chosen for further analysis, the source code must be retrieved in a way or another.

3.2 Release Archives versus Version Control Systems

One way to obtain one revision of the source code is by downloading a compressed tar ball [JOH94] or zip archive [PKW05], which normally can be found on the project *FTP* [POS85] or *HTTP* [FUZ05h] web site or from a mirror web site. For instance, *SourceForge.net* [VAS05] offers up to fifteen different mirrors spread all around the world for every release archives stored on its server in order to limit its own bandwidth usage.

Another method for retrieving source code from an open source project is to fetch the source code from various version control systems, such as: *CVS*, *SubVersion*, *Arch*, *Monotone*, *BitKeeper*, *Aegis*, etc. For instance, *SourceForge.net* [VAS05] and *Savannah* [SAV05] provides a free *CVS* repository to every project, while other sites such as *BerliOS* [FOK05] and *Tigris* [COL05] offer a free *SubVersion* repository to every project instead. Either way, the retrieval and analysis of the source code in question can be made based on a specific release number or for a specific time period, if nightly built archives are available or if the repository system accept queries for a specific time period.

Furthermore, such retrieval can be fully automated by scripting the various command line interfaces depending on the method used, using *Perl* for instance [HIE05]. The following sections analyze the different kinds of version control system used by the most popular open source projects in general and their inherent limitations and capabilities. Seven free and commonly used version control systems will be analyzed in details, while a comparison chart of fifteen different version control systems conclude this chapter.

3.3 RCS

RCS is a version control system developed by Tichy in the early 1980's [TIC85]. *RCS* is based on an implementation similar to *SCCS* with an improved command line interface. *RCS* uses *",v"* suffixes instead of the *"s."* prefix used in *SCCS* for each file archives. This small storage modification resolved some of the undesirable issues related to *SCCS* as explained in detail in [WEI98] and [JAM05]. The major issue in question is related to implicit *Makefile* rules, which are mainly based on suffix patterns. Therefore, some implicit *Makefile* rules would improperly handle *SCCS* files as real implementation files. Consequently, implicit *Makefile* rules and a *SCCS* based project could not be mixed easily. As an end result, the suffix approach became more popular and many concurrent version control systems are using this approach instead. Due to its popularity, some modern version control systems are using *RCS* internally or were derived from the *RCS* implementation [TIC85].

3.4 CVS

One of such version control system, which is using *RCS* internally, is called *CVS*. *CVS* is a widespread and well-known concurrent version control system [PES98][PRI05], which is also used by every open source projects hosted on *SourceForge.net* [VAS05] and *Savannah* [SAV05]. *CVS* supports file changes, merges, tagging and version easily. However, *CVS* lacks many wanted features like directory and file renaming, atomic commits, efficient change set merging, metadata and binary file handling. Due to these limitations, many other version control tools were developed to fix these issues. Even though such limitation still exists, *CVS* is still widely used and therefore, many friendly user interfaces were developed for it, such as: *TortoiseCVS* [CRA05], *WinCVS* [PAR05], *SmartCVS* [SIN05], among other plug-ins for various editors like *Eclipse* [YAM04], *Microsoft Visual Studio* or *Borland* [PUS05]. Finally, *CVS* is a simple, portable, efficient tool, which is sufficient for most projects, even with its current limitations.

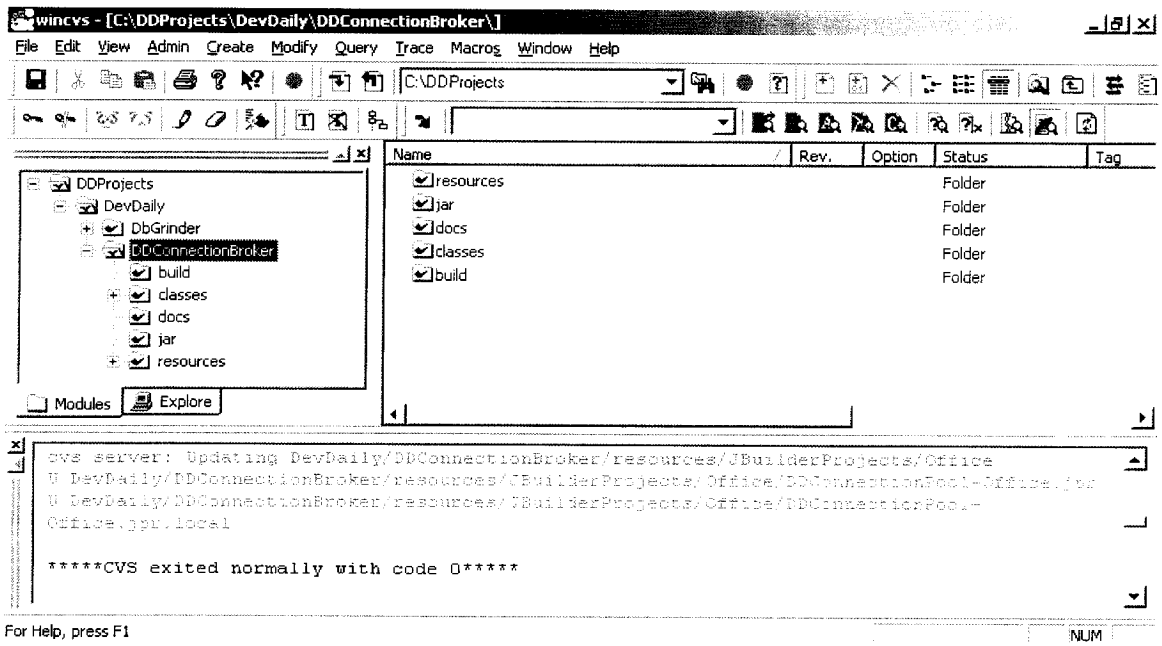

Figure 3.1: Screenshot: WinCVS [ALE05]

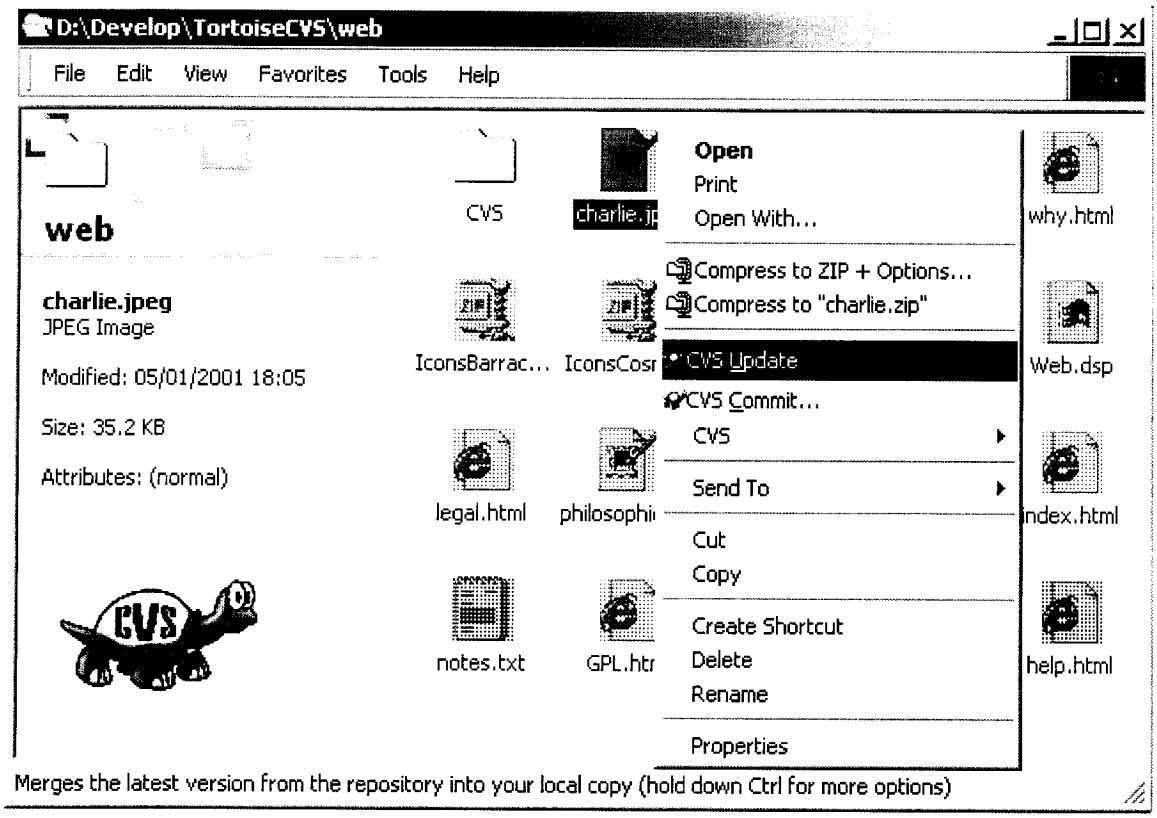

Figure 3.2: Screenshot: TortoiseCVS [TAY03]

3.5 SubVersion

SubVersion [RES05] is a concurrent version control system that is aiming at replacing *CVS* [PRI05]. *SubVersion* is mainly used by open source projects hosted at *BerliOS* [FOK05] and *Tigris* [COL05], and by many other large projects such as KDE [KDE05], among others. Moreover, *SubVersion* provides the following additional features [RES05]: directory, file renaming, symbolic links and file metadata is versioned. Commits are fully atomic. Branching and tagging are constant time operation and *SubVersion* supports better handling of binary files. Merging is proportional to the change size instead of the file size; therefore, merging a small change in a huge document takes less time than *CVS* [CUN05]. The storage backend can be either a Berkeley database [SLE05] or flat files.

The output messages were designed to be localized, humanly readable and easily scriptable. The *SubVersion* server is based on the *Apache 2.x WebDAV/DeltaV* protocol [ASF05][GOL99] or through a standalone server that can be tunneled over *SSH* [GER05]. *SubVersion* does not use custom version number instead all revision numbers are incremented upon every commit [RES05]. Consequently, an end user may need separate repositories for each project, which could be a tedious task to maintain properly instead of having one central repository for every project [RUS02]. On the positive side, there exists a friendly user interface called *TortoiseSVN* [KUN05], which was derived from *TortoiseCVS* [CRA05], to visually manipulate a *SubVersion* repository using a simple right mouse click context menu as shown below in Figure 3.3.

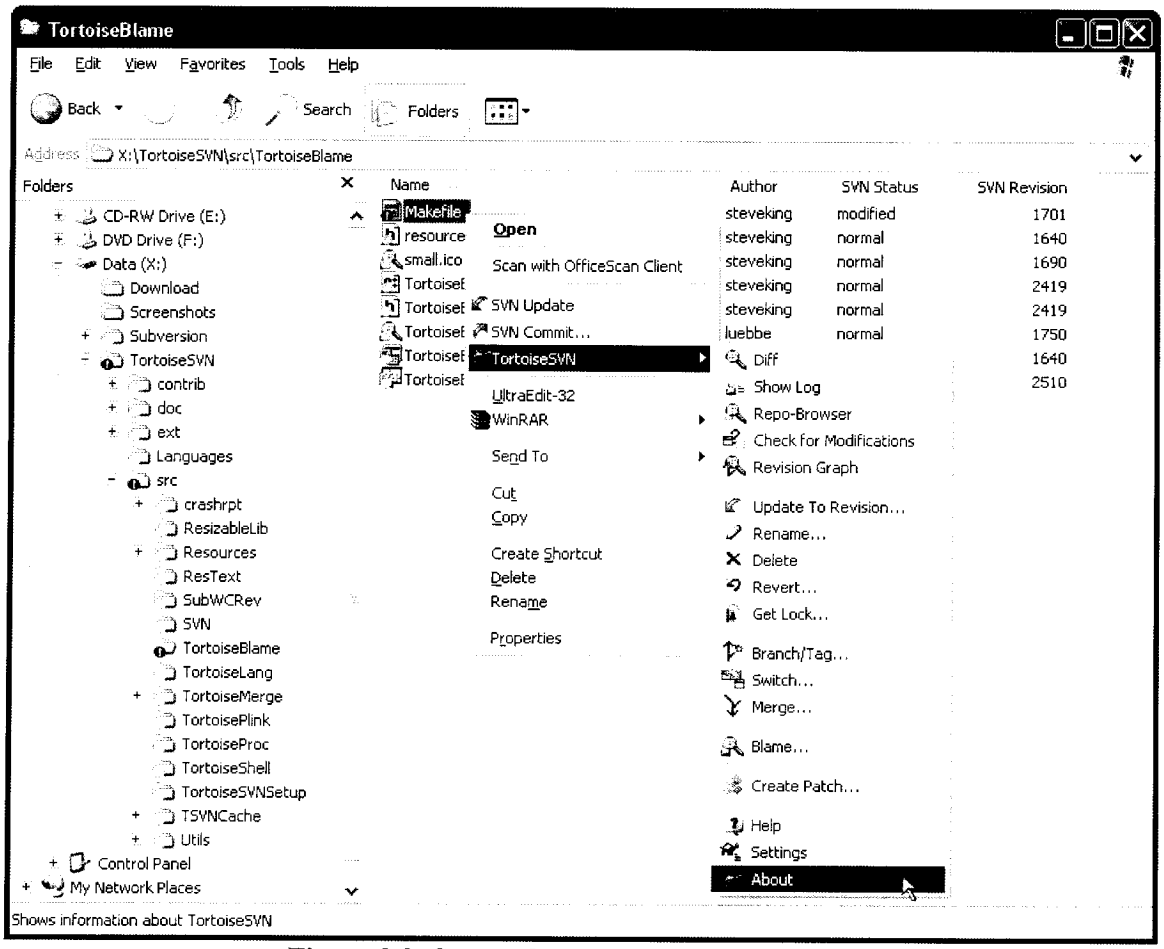

Figure 3.3: Screenshot: TortoiseSVN [KUN05]

3.6 Arch

Arch [SAU04] is a version control system, similar to *CVS* [PRI05] and *SubVersion* [RES05]. *Arch* was developed by Tom Lord and became a *GNU* [HAT05] sponsored project in July 2003 [LOR03]. *Arch* is licensed under the *GNU GPL v2 license* [TUR05] and was originally developed to replace *BitKeeper* [BIT05], as it will be explained later on. *Arch* is quite special in the sense that it does not require a repository server, instead every developer must have its own repository and *Arch* provides facilities to merge entire repository together in an *ad hoc* pull fashion. Therefore, this approach eliminates the need to provide write permissions to update a central repository. In other words, this means that every developer must have their own revision tree, where they can retrieve change sets from various trees and finally merge them using the *Arch* branching and merging support. While this is advantageous on one-hand, it means that every developer is forking the main branch and without a proper well established set of repository hierarchy, merging and synchronization becomes a real challenge. Moreover, in this *ad hoc* context, it becomes extremely difficult to know for a foreign developer, which tree is the main project. This happens, since hundreds of forks may exist at any given time, with each of them having their own inconsistencies and their own sets of features or bugs. Furthermore, each merge collision problems must be solved over and over by each individual, if no proper hierarchy actually exists. One way to work around this issue, which is the methodology used by the *Linux Kernel* [MOR05], is to adopt a proper subsystem maintainer hierarchy and to have one well-known web server mirroring each revision tree and one well-known central repository where the final merge actually occurs. On the positive side, *Arch* has many advantages, such as [LOR05]: distributed

repositories, advanced merging capabilities for manipulating branches and change sets, file renaming support, unobtrusive operations, since it allows trivial repository reorganization out of the box and revision libraries to build space efficient libraries of previous revisions. *Arch* also possesses a low barrier of entry, since it does not require any third party to execute any commands into some remote repository, until it is desired [YAS05]. Consequently, anyone can host on any machine their actual *Arch* tree that can be read from and can be remotely accessible using any existing Internet protocol such as: *HTTP* [FUZ05h], *HTTPS* [RES99], *FTP* [POS85], *SFTP* [GAL04], *WebDAV* [GOL99], etc. Finally, there exists a *Gnome* user interface called *ArchWay* [GOI04] and a *Qt 3.x* user interface called *Octopy* [SAI03] to visually manipulate an *Arch* repository as shown in the screenshots below in Figure 3.4 and Figure 3.5.

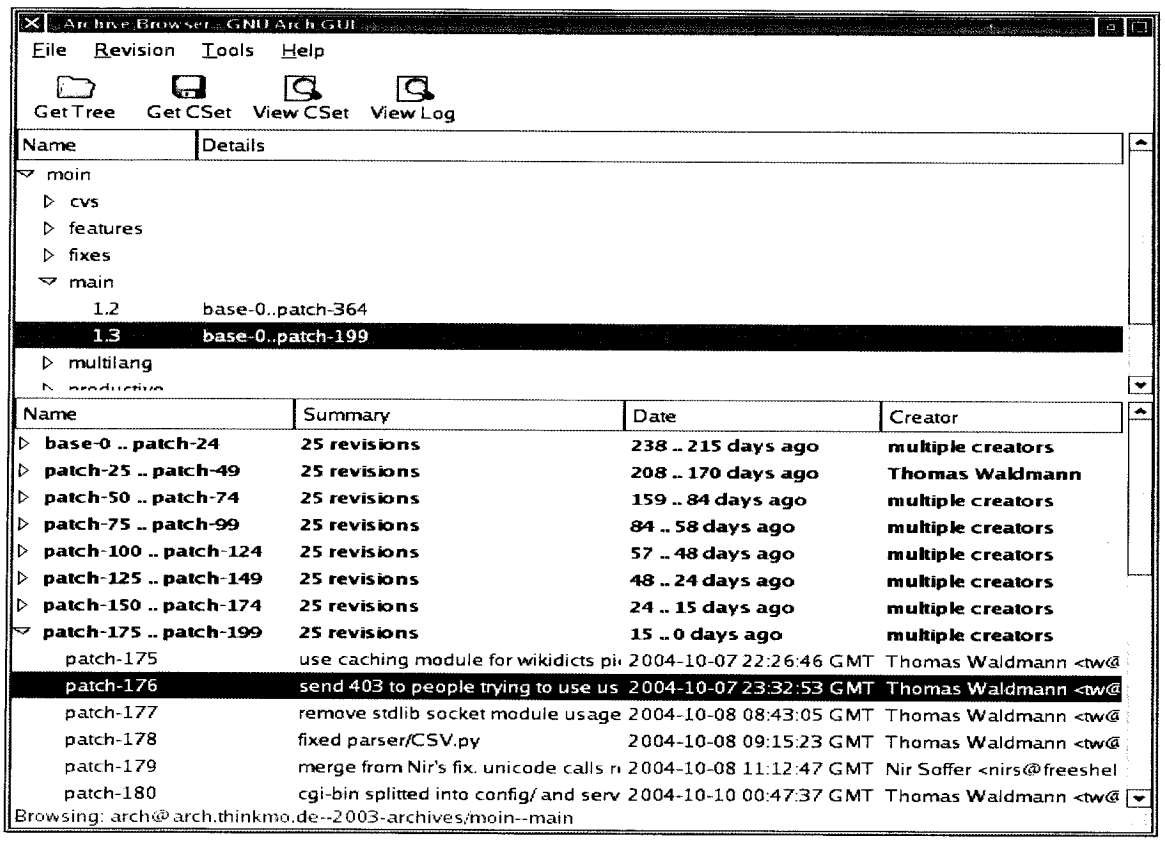

Figure 3.4: Screenshot: ArchWay – Archive Browser [GOI04]

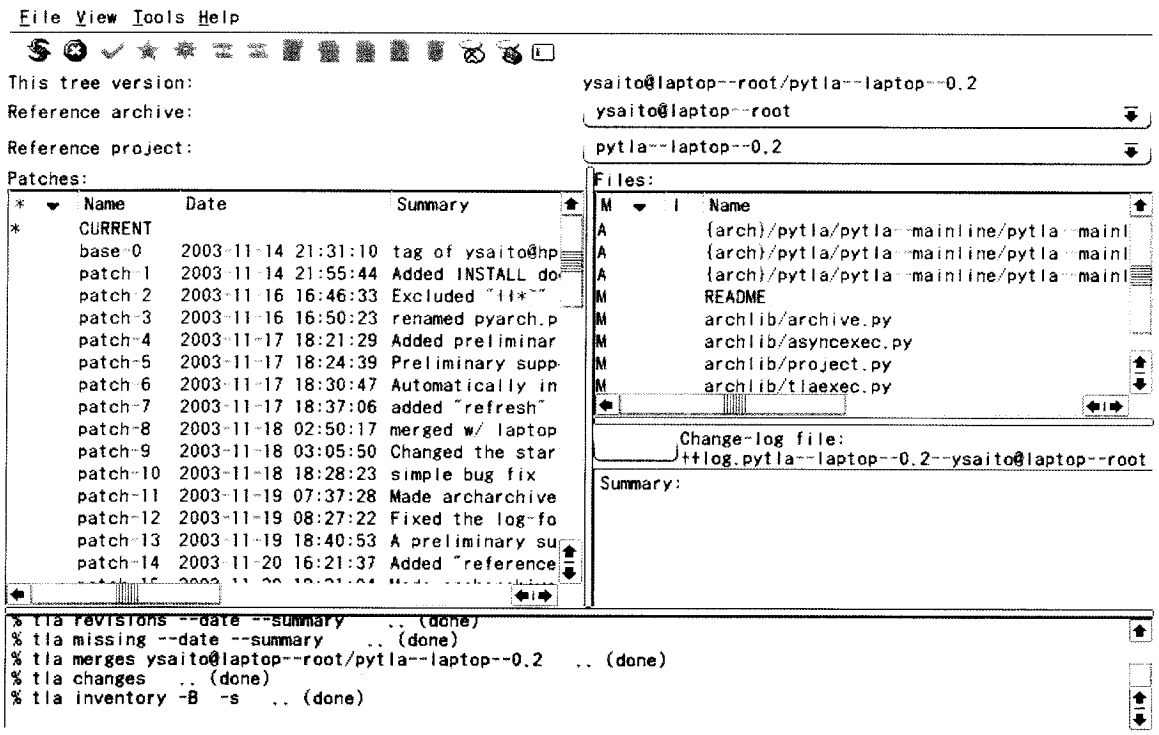

Figure 3.5: Screenshot: Octopy [SAI03]

3.7 Monotone

Monotone is a free and open source distributed version control system licensed under the *GNU GPL v2 license* [TUR05]. It was also developed as a possible successor to the *BitKeeper* repository [BIT05]. *Monotone* uses cryptographic *SHA1* [DES97][RSA04w] [SCH04h][SCH04r] version naming [HOA05v], client-side *RSA* certificates [RSA04c] [HOA05c], a transactional version database stored in a regular flat file and uses a custom network protocol for efficient database synchronization [HOA05f][HOA05s]. Furthermore, *Monotone* can help automate many tedious and error-prone tasks by providing many features [FIS05][HOA05d][HOA05i] [HOA05s], such as: *CVS* emulation, decentralized system, lightweight branches, file annotation and the ability to track file changes, copies,

renames and moves. Finally, change sets can be posted anywhere on the Internet, via *CGI* script [NCS98], *NNTP* newsgroups [KAN86], *FTP* [POS85] or *HTTP* [FUZ05h] web sites or mailing lists. On the positive side, there exists a *Gnome* user interface called *Monotone-viz* [AND05], shown below in Figure 3.6, to visualize and manipulate a *Monotone* repository.

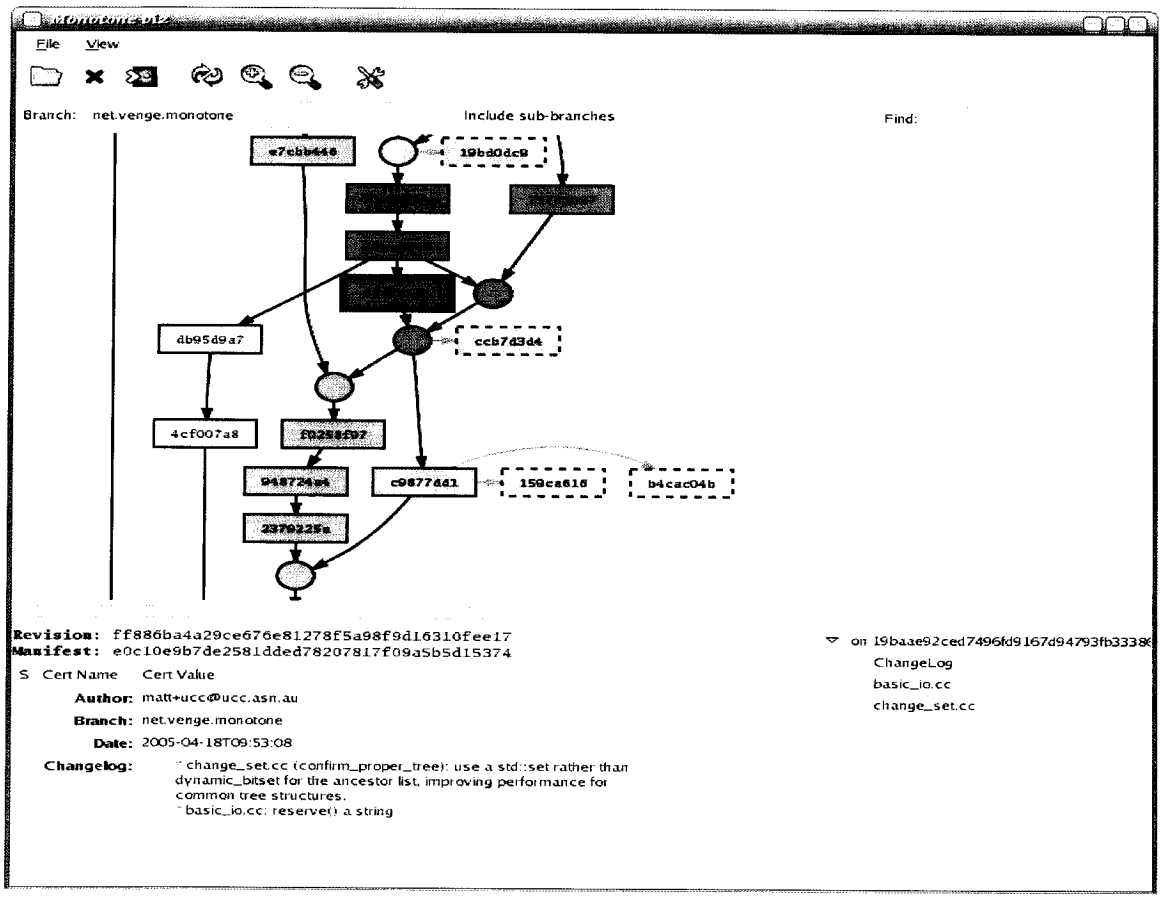

Figure 3.6: Screenshot: Monotone-viz [AND05]

3.8 BitKeeper

BitKeeper [BIT05][BIT04s] is a commercial configuration management system that was used by the *Linux Kernel* from February 2002 [BIT04h][SHA03] to April 2005. In the end,

BitKeeper was replaced by *git* [TOR05], due to the development of *SourcePuller* by Andrew Tridgell [TRI05][ORL05], which somehow violated the free *BitKeeper* license [MOF01] originally provided by Larry McVoy [MCV02] by not respecting the newly added non-compete clause [EKL02][BAR05][PEE05].

In the case of the *Linux Kernel* development methodology, *BitKeeper* provided a means of having multiple central repositories that could easily be merged within the central *BitMover* repository server [BIT05d]. Having a central repository was solving a lot of the issues that are common with solutions like *Arch* and *Monotone*, since it makes the inter-repository synchronization much easier than otherwise. This *BitKeeper* methodology seemed to be very appropriate and very useful for such a large and globally distributed open source project, since the patch and change set management system allowed for better filtering of changes and dependencies than any other available solutions back in 2002. Under this schema, change sets were gradually promoted to a higher revision tree, once enough peer reviews were performed until they were included into the main branch [MOR05]. This methodology was created to perfectly match the corresponding *Linux Kernel* methodology, in the first place. As a result, *BitKeeper* was introduced to the widely known *Linux Kernel Mailing List* [SPA05] to eliminate the "*Linus does not scale*" problem that occurred occasionally during the 2.4 series [BAR02]. As a result, *BitKeeper* provided "a set of techniques and tools to control changes to information over time, made by multiple, sometimes simultaneous, often geographically distributed modifiers" [BEA05]. Furthermore, "*BitKeeper* was designed from the beginning to work with multiple source repositories and to facilitate moving patches from one repository to another" [EKL99]. Finally, *BitKeeper* provides a large set of features as

described in [BEA05][BIT04f][BIT04a]. For instance, *BitKeeper* provides a decentralized system, compressed hierarchical repository, automated merging, rollback on any operation, checksum integrity checks, change set utilities, support for tracking file renames and symlink, event notification triggers and a highly scalable architecture to support huge projects such as the *Linux Kernel*. It also provides a portable *Tcl/Tk* graphical user interface as shown below, in Figure 3.7 [OUT04].

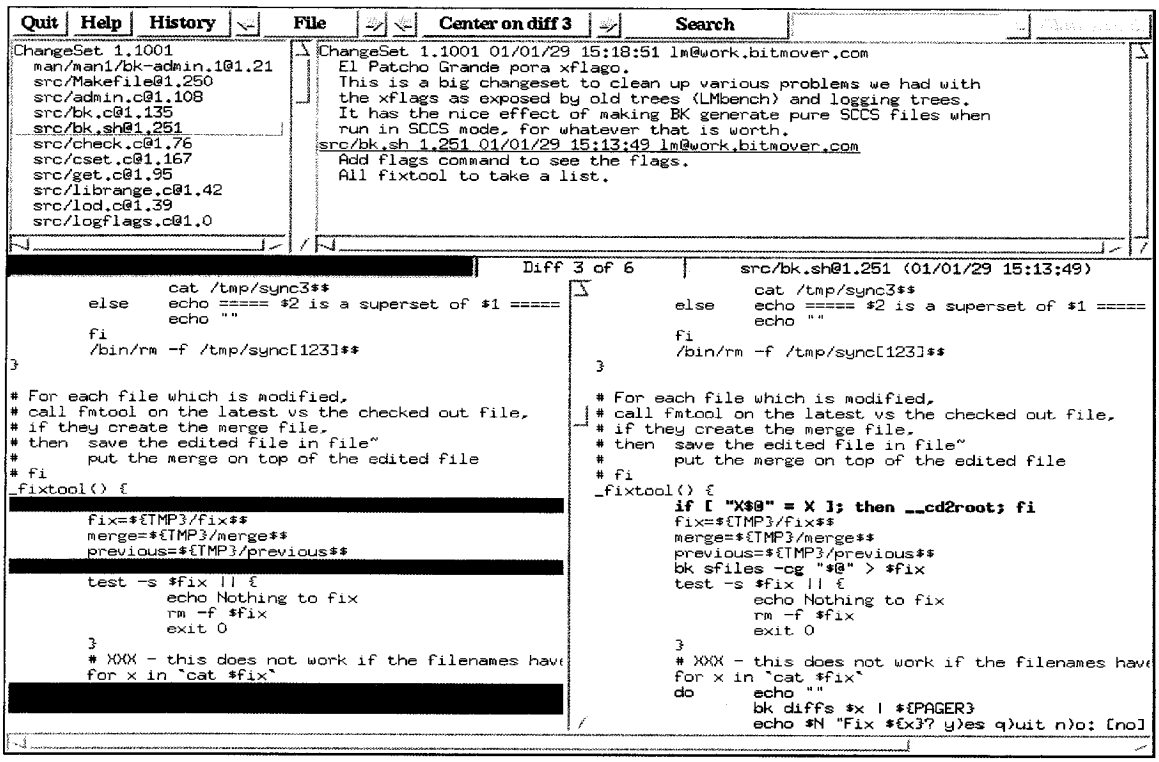

Figure 3.7: Screenshot: BitKeeper – Change set Tools [OUT04]

3.9 AEGIS

Aegis is another free and open source distributed version control system, which is actively maintained since 1991 [MIL05] and is also licensed under the GNU GPL license v2 [TUR05]. The name stands for protection or defense and it comes

from the Greek word *aigis*, which was the "weapon of Zeus and Athena" in the ancient Greek mythology [ANS05].

Aegis provides many features as described in [MIL05][MIL05c][MIL05p][MIL05q] [CHA04][WAT04][BRU00][APP98]. *Aegis* was designed around incremental development and was therefore designed for repository security, availability, integrity and confidentiality. On the positive side, change sets and branching can be of any desired depth and are independently treated. Change sets are atomic and must pass well-defined test cases before being integrated. However, while *Aegis* supports large directory trees, it cannot handle as many files as *CVS* and it has a wide range of restrictions that are not found in other SCM. Consequently, *Aegis* is still marginally used by other open source project. As a final note, *Aegis* support projects hosted on *SourceForge.net* and has a web interface to track every activities being performed within an *Aegis* repository [MIL05w] as shown below in Figure 3.8.

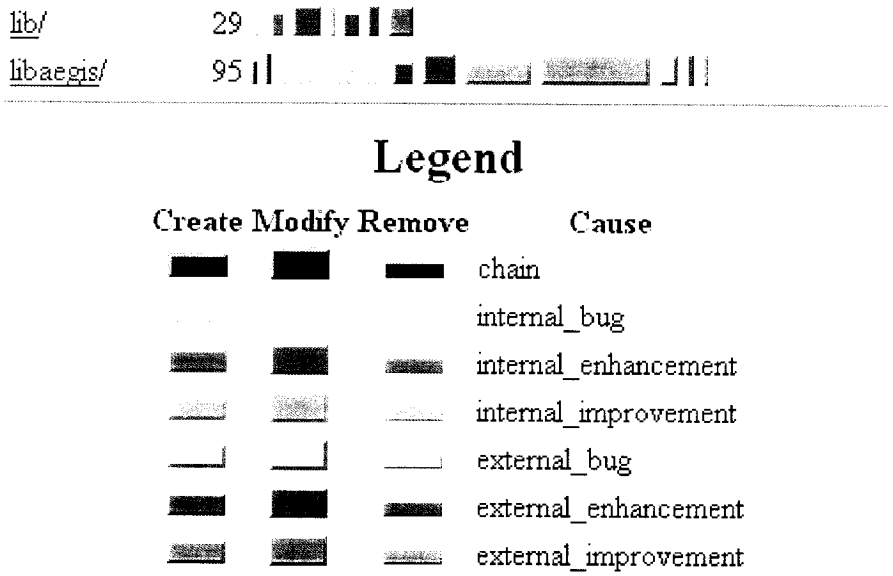

Figure 3.8: Screenshot: Aegis – Change Density by File [MIL05w]

3.10 Problematic of Distributed Version Control Systems

The major problem with distributed version control systems like *Arch*, *Monotone*, *Aegis* and *BitKeeper* is that many conflicting forks might exist at any given time. While this may seem appealing at first sight, over time it makes merging and patching a real challenge, since various change sets or trees might become incompatible to the point were various development trees cannot be safely merged back into the main repository. For instance, some small [THO04] and large [TOR05] projects, which used the distributed version control systems methodology, suffered problems related to instability, synchronization, stabilization, feature integration, bug reporting and bug resolving. It also created additional confusion, since at any point of time, many releases existed concurrently which do not have the same sets of bugs or features. This methodology proved itself to require discipline and willingness to ensure that no explicit fork exists and that everyone in the end synchronizes itself with one and only one main and widely known repository [MOR05]. As a result, no perfect version control system currently exists to cover every possible use cases. Each of them has their own advantages and shortcomings, which must be examined carefully before choosing one over another. To help developers into their quest of choosing a proper version control system the following table provides a list of every common feature available in fifteen different version control systems.

3.11 Version Control System Tools Comparison Table

Table 3.1: Comparison of various version control software systems [FIS05c]

	CVS	Aegis	Arch	BitKeeper	CM Synergy	Co-op	Darcs	Monotone
Atomic commits	N	Y	Y	Y	Y	Y	Y	Y
File renaming	N	Y	Y	Y	Y	Y	Y	Y
File move	N	Y	Y	Y	Y	Y	Y	Y
Directory renaming	N	Y	Y	Y	Y	Y	Y	Y
Directory move	N	Y	Y	Y	Y	Y	Y	Y
File copy	N	N	N	Y	Y	N	N	Y
Directory copy	N	N	N	Y	Y	N	N	Y
Remote repository replication	N	Y	Y	Y	Y	N	Y	Y
Propagating changes to parent repositories	N	Y	Y	Y	Y	Y	Y	Y
Repository permissions	Limited	Y	Y	N	N	N	N	Partial
Support changesets	N	Y	Y	Y	Y	Y	Y	Y
Tracking line-wise file history	Y	Y	Partial	Y	Partial	N	Y	N
Ability to work on only one directory of the repository	Y	N	Partial	N	Partial	N	Y	N
Tracking uncommitted changes	Y	Y	Y	Y	Y	Y	Y	Y
Per-file commit message	N	N	Y	Y	Y	N	N	Y
Documentation	Excellent	Medium	Medium	Very good	Medium	Very good	Good	Good
Ease of deployment	Good	Medium	Excellent	Good	Medium	Very easy	Very good	Excellent
Command set	CVS	Complex	Complex/ CVS	Complex/ CVS	Complex/ CVS	CVS	Compact	Complex/ CVS
Network support	Good	Poor	Excellent	Good	Good	Simple LAN	Good	Good
Portability	Good	Medium	Good	Very good	Very good	Win95	Very good	Excellent
Web Interface	Y	Y	Y	Y	Possibly	N	Y	N
Graphical User Interface	Many	1	3	some	some	integrated	N	N
Price	Free	Free	Free	Free/ Costly	20K$/Server, 4K$/Client	159$/client	Free	Free
License	GPL	GPL	GPL	Proprietary	Proprietary	Proprietary	GPL	GPL

	OpenCM	Perforce	SourceSafe	Subversion	Super version	SVK	Vesta
Atomic commits	Y	Y	N	Y	Y	Y	Y
File renaming	Y	N	N	Y	N	Y	Y
File move	Y	N	N	Y	N	Y	Y
Directory renaming	Y	N	N	Y	N	Y	Y
Directory move	Y	N	N	Y	N	Y	Y
File copy	N	Y	Y	Y	N	Y	Y
Directory copy	N	Y	Y	Y	N	Y	Y
Remote repository replication	N	Y	N	Y	Y	Y	Y
Propagating changes to parent repositories	N	N	N	Y	N	Y	Y
Repository permissions	per branch	Y	per user	Y	N	Y	per package
Support change sets	Y	Y	N	Partial	Partial	Partial	N
Tracking line-wise file history	N	Y	N	Y	N	Y	N
Ability to work on only one directory of the repository	N	Y	Y	Y	N	Y	Partial
Tracking uncommited changes	Y	Y	Y	Y	Y	Y	Y
Per-file commit message	N	N	Y	N	Y	N	N
Documentation	Good	Very good	Medium	Very good	Poor	Very poor	Good
Ease of deployment	Very good	Very good	Very good	Good	Very good + JDK1.4	Good + CPAN	Medium
Command set	CVS	Very extensive	GUI	CVS	GUI	CVS	5-20 cmds
Network support	Good	Good	network share	Very good	Good	Very good	NFS/ RPC
Portability	Good (Unix)	Excellent	Win95 + MainSoft (Unix)	Excellent	Excellent	Good	Good
Web Interface	N	Y	N	Y	N	Y	Y
Graphical User Interface	N	Y	integrated	many	integrated	N	N
Price	Free	750$/client + 150$/client per year	MSDN	Free	Free	Free	Free
License	GPL/BSD/ CPL	Proprietary	Proprietary	Apache/ BSD	GPL	Perl	LGPL

Chapter 4 Merging and Differentiating Survey

Merging and differentiating is a challenging research area. Since 1970, various tools were developed to perform such task, such as: *diff* in 1974 by Douglas McIlroy [WIK05d], *patch* in 1985 by Larry Wall [WIK05p], *GNU diff3* in 1988 by Randy Smith [SMI88][SMI94] and *GNU sdiff* by Thomas Lord [WAL02]. It is worth mentioning that most of the modern *GNU* versions of these Unix utilities [WAL02] are mostly based on research work published by Myers in 1986 [MYE86].

For the last decades, most open source projects were heavily relying on *GNU diff/patch* technologies for merging and differentiating source files. One example is the *Linux Kernel* project, which always refused to use *CVS* at all costs. After lots of discussions on the *Linux Kernel Mailing List*, the conclusion ended up being that *CVS* was simply inappropriate, as stated by Linus Torvalds and other *Linux Kernel* developers [TOR95]. As a result, many *Linux Kernel* developers were relying on *GNU diff*, *patch*, *grep* and *find* utilities for manipulating the *Linux Kernel* source code [LOV02][SHA03]. However, in 2002, when some new merging technology became a major selling argument for *BitKeeper*, when it introduced transparent, integrated and automated merging of change sets within its proprietary repository solutions [BIT04s][BIT04a], Linus Torvalds was consequently approached by Larry McVoy [MCV02], who offered free *BitKeeper* licenses to all *Linux Kernel* developers to help them manage their *Linux Kernel* project [ANDR02].

However, many critics such as Richard Stallman criticized this offer, by saying that "the decision to use *BitKeeper*, reflects the attitude of the original developer of Linux, a person who thinks that "technically better" is more important than freedom" [STA02]. As a result, many similar open source solutions were developed after its introduction in the *Linux Kernel* in February 2002 and various approaches with similar merging solutions were researched to solve this difficult problem. Consequently and as discussed in the previous chapter, many open source solutions flourished out of this initiative such as: *SubVersion*, *Arch*, *Monotone* and *git*, which all evolved in parallel as possible *BitKeeper* replacements for the *Linux Kernel* repository requirements and scalability problems. As a side effect, this research field also flourished during this same period, with many solutions that appeared to solve this kind of problems and similarly, other kinds of problems such as the differentiation of XML documents.

As mentioned by Estublier in 2000: "Mergers found in today's tools simply compare (on a line by line basis) the two files to merge, and a file that is historically common to both (the common ancestor). If a line is present in a file but not in the common ancestor, it was added, and must be kept; if a line is present in the ancestor and not in the file, it was removed and must be removed. If changes occurred at different places, the merger is able to decide automatically what should be the merged file. This algorithm is simply a heuristic that provides in output a set of lines with absolutely no guaranties about correctness. Nevertheless mergers proved to work fine, to be very useful and became almost unavoidable" [EST00]. A more in-depth survey of merging technologies that were previously available in 1995, is provided by Buffenbarger in [BUF95]. The problem with

line-by-line differentiation technique is that it is completely inappropriate for XML purposes or non line-by-line based hierarchical data structures to be compared, which is the case of the various *DoxyChange* intermediate file formats. The following sections discuss some of the different kind of merging and differentiating technologies available and their inherent limitations and capabilities. Five merging and differentiating tools will be analyzed in details, while a comparison chart of six different differentiating tools conclude this chapter.

4.1 GNU diff/diff3

The *GNU diff/diff3* command line tools is a free open source tool made by *GNU* [SMI88] [SMI94] to differentiate two given files based on the algorithm developed by Myers in [MYE86]. *GNU diff/diff3* is quite efficient, portable, heavily documented and easy-to-work with. The operation is decoupled with synchronization and optimistic locking. It only requires input files and it also works with XML, but only on a line-by-line basis. However, it does not provide any valid XML output or neither support any XML syntactic context. *GNU diff/diff3* does not support copy or move. *GNU diff/diff3* also does not support tree differentiating, tree patching or tree encoding. It also lacks structural merge, merge tracking, subtree move, subtree copy or move/update [ZS97][LIN01p]. It is mainly aimed at line-by-line differentiation between two lines, since it cannot detect lines that are moved around. Originally, one of the primary goals was to produces a minimal edit script to transform one file into another by using a set of insertion and deletion operation. On the positive side, an easy-to-use

graphical back-end for the K *Desktop Environment* [KDE05] named *KDiff3* was developed by Joachim Eibl [EIB05] and is freely available under the *GNU GPL v2 license* [TUR05]. Some *KDiff3* screenshots are shown below in Figure 4.1 [EIB05s].

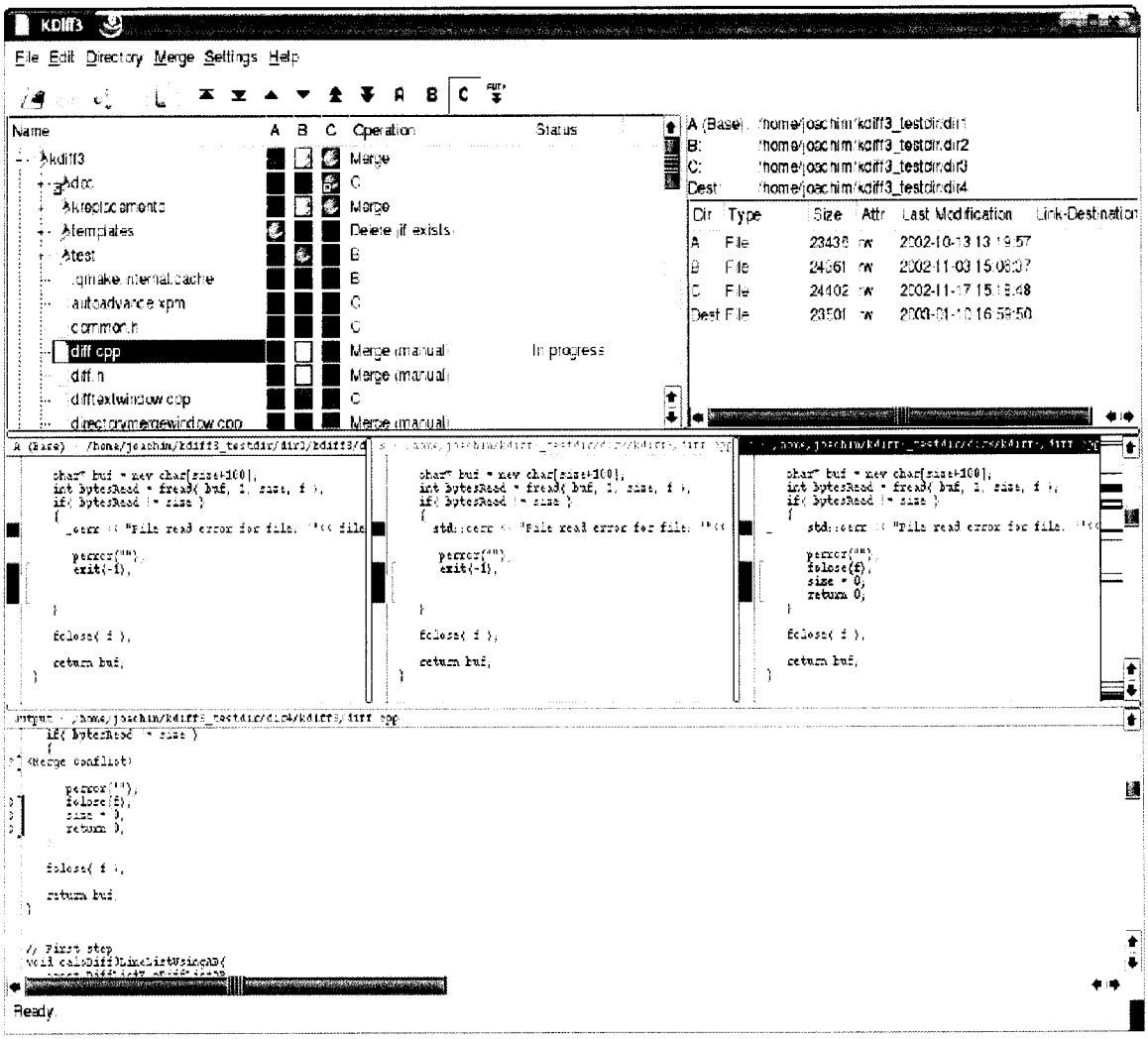

Figure 4.1: Screenshot: KDiff3 [EIB05s]

4.2 XMLTreeDiff

The *XMLTreeDiff* is a proprietary tool developed by *IBM AlphaWorks* to differentiate XML files. *XMLTreeDiff* was developed in November 1998 by Francisco Curbera

[CUR98], it seems to be the first tool that was publicly available that attempted to differentiate XML files, since it was released only nine months after the first W3C recommendation of the eXtensible Markup Language (XML), version 1.0 [BRA98]. It does show tree differences and encodes them. *XMLTreeDiff* can be considered to be a bit inefficient and it does not support tree patching, move or copy. However, since *XMLTreeDiff* was retired, any further analysis cannot be performed. As a side note, many research papers and projects, which were published in 2001, 2002 or later, mention this research tool, such as [MCZ01][JEU02] [HAR02][LIN01p], among others.

4.3 3DM

The *3DM* XML differentiating and merging tool is another free open source tool written in Java, which handles move and copy of entire subtree [LIN01p]. It also supports node renaming, copying, insertion and deletion. However, it assumes that both input XML files have the same DTD; therefore, a change in the DTD between two given revisions will not work properly [LIN05p][LIN05w]. The algorithm complexity is **O(n log n)** as mentioned in [LIN04]. Originally, the *3DM* tool was supposed to be used to differentiate two *DOT-XML* documents. However, extensive testing of *3DM* proved to be quite useless for differentiating minor or major changes within two XML documents, since it was not able to detect small changes properly, only additions or deletions. As a result, another differentiating tool had to be found to fulfill this task. A screenshot of the *3DM TreeDiff* graphical output is shown below in Figure 4.2 [LIN01s].

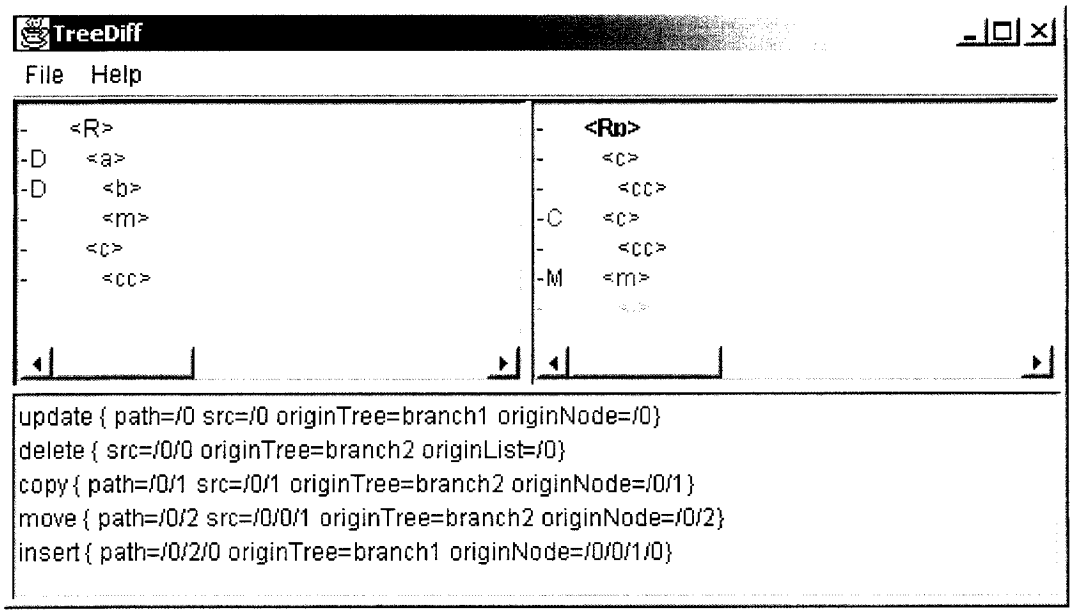

Figure 4.2: Screenshot: 3DM TreeDiff View [LIN01s]

4.4 XMLDiff

The *XMLDiff* differentiating tool is a proprietary tool made by *IBM Alphaworks* [IAW01], which works very well with input XML files. Based on the acquired experience after using and testing the *XMLDiff* tool, it became apparent that *XMLDiff* was quite reliable for differentiating minor or major changes within two XML documents. Moreover, the algorithm complexity is quite acceptable by being $\mathbf{O(|T1| \cdot |T2| \cdot depth(T1) \cdot depth(T2))}$ as mentioned in [SHA02]. Some effort was made to make *XMLDiff*, a core component of *DoxyChange* until the obvious limitations were finally discovered.

First of all, *XMLDiff* is a closed source product that cannot be extended to the specific needs of this research project. Furthermore, *XMLDiff* is implemented in Java, which creates a huge overhead penalty when used intensively in batch mode, since the Java

Virtual Machine initialization must be performed over and over for every XML files to be generated, which can easily be estimated to be more than ten thousand times. The most disastrous limitation is the very restrictive license, which is unfortunately limited to 90 days for non-commercial usage and only for evaluation purposes [IAW01]. As an end result, this unfortunate license limitation is simply too restrictive and completely inappropriate for this research project. In other words, a more license friendly XML differentiation tool had to be found to replace this one, even though it produced interesting and valid results.

4.5 X-Diff

X-Diff differentiating tool is a free open source tool written in C++. Additionally, a Java version also exists [WAN05]. *X-Diff* supports node changes, insertion and deletion. In the end, *X-Diff* was chosen since it was as successful as *XMLDiff* for identifying minor or major changes within two XML documents and a C++ version existed. Later on, a work-around had to be inserted, since *X-Diff* completely ignores node ordering and attributes by design [WAN05]. As a result any named group had to be wrapped using specially crafted naming node tag, to ensure that *X-Diff* would group them properly. Since the tag had to be associated somehow, it was chosen that items would be associated only by names and not by signatures. This small change in the XML structure allows small changes in the signature for the same node name to be clearly identified as a signature rename, instead of being silently ignored as a normal delete/insert pair. As mentioned earlier wrapping tag use a "L" prefix with a special encoding for names

that contains incompatible XML characters for tag names, such as mathematical symbols often used in C++ member functions.

4.6 Differentiating Tools Comparison Table

The following Table 4.1 provides an overview of the various tools that were analyzed during this XML differentiating survey. It is worth mentioning that *XY-Diff* was also briefly evaluated. However, since major development had already took place within *X-Diff* and that *X-Diff* seemed to be offering better output results as mentioned by its corresponding documentation and technical papers, which claimed to be superior to *XY-Diff* and other similar tools such as *XMLDiff*. As a result, *XY-Diff* was left behind without further testing, mostly due to time constraints.

Table 4.1: Comparison of various XML differentiating tools

	GNU Diff	XMLTreeDiff	3DM	XMLDiff	X-Diff	XY-Diff
Handle Text	Yes	No	No	No	No	No
Handle XML	No	Yes	Yes	Yes	Yes	Yes
Insertion	Yes	Yes	Yes	Yes	Yes	Yes
Deletion	Yes	Yes	Ys	Yes	Yes	Yes
Change	No	Unknown	No	Yes	Yes	Unknown
Rename	No	Unknown	No	Yes	Yes	Unknown
Reliable XML changes	No	Unknown	No	Unknown	Work-around	Unknown
Portability	Yes	Yes	Yes	Yes	Yes	Yes
Language	C	Java	Java	Java	C++ or Java	C++ or Java
Tested	Yes	No	Yes	Yes	Yes	Unsuccessful
Source code	Yes	No	Yes	No	Yes	Yes
Price	Free	Retired	Free	Costly	Free	Free
License	GPL	Proprietary	LGPL	Proprietary	SleepyCat / BSD-like	LGPL

Chapter 5 Contribution

The main purpose of this research project is to make a small contribution within the field of software engineering. More specifically this research focuses on developing a new tool called *DoxyChange*, which enables programmers or software engineers to visualize the evolution of a given software system. *DoxyChange* is an addition to the existing *Doxygen* open source project [HEE05], which is already widely used to document and visualize more than 400 projects worldwide [HEE05p]. Another benefit of *Doxygen* is that it provides a friendly parser framework that can fully handle complicated and ambiguous C/C++ constructs, such as templates. Another factor is that *Doxygen* generates a large variety of *Graphviz Dot* UML diagrams [HEE05g]. Therefore, making it easier to extend this architecture to support software change visualization rather than either to create a new one from scratch or to build one on top of the *Sun*^{TM} *Javac* framework [SUN04c] currently used by the *CONCEPT* framework. The main contributions of this research project are the integration and the extension of existing tools and algorithms, such as: writing the glue code for the intermediate file parsers and converters, optimizing each component and in some case, writing a portable layer such that some Linux components would work and compile properly on the Windows platform. The following sections will discuss the contributions in more details.

In summary, *Doxygen* was extended to provide a *DOT-XML* output within the HTML output generator, some configuration variables and optimizations were added to decrease the amount of output files and consequently, the time it takes to generate them. *X-Diff* and its dependencies were modified to compile on Windows, *X-Diff* was also refactored and a new *Change-DOT* output generation was added. The change filter was created and it provides among others, the following core functionalities: reformatting, restructuring, grouping, performing some CSS styling and modifying encoded strings into SVG polygons, circles and arrows. It is worth mentioning that this chapter only describes the successful steps towards the completion of the *DoxyChange* tool and excludes any other research work which was actually performed, but later on abandoned, modified or reworked towards obtaining a fully working solution. This includes work performed on many other compiler frameworks, such as: a homegrown C++ parser, a *GCC-XML* extension [KIN04], a *JavaML* converter [BAD00], *IBM™ Jikes* [ABB05], the *Sun™ Javac* compiler [SUN04c] and other *Doxygen* extensions that were first investigated and then abandoned. These *Doxygen* extensions include a direct SVG generation and a complete *XML-SVG* generation and some work towards modifying *Graphviz Dot* such that it could be statically inlined within *Doxygen* for speed reasons. Furthermore, attempts were made to refactor *Graphviz Dot* such that XML data would be kept intact from the input *dot* file to its corresponding output SVG format. For instance, at one point, the goal was to have all phases integrated into a single static application instead of four separate entities, but due to outstanding technical difficulties, it was simply preferable to keep them separate to distribute the load and therefore use them as is, even with the well-known I/O overhead due to temporary files, since it can be worked around by using a RAMDISK.

As a result, this research project focuses mainly on the visualization of large scale software system with approximately the same speed and size limitations than *Doxygen* itself. It has to be pointed out that *DoxyChange* can already scale up to software system, in the order of 16 MB of C++ source code without any major problems, as illustrated in Chapter 6. Furthermore, techniques are also available and are already used by large projects to work around these speed and size limitations, such as distributing the load on many computers and using a partial recompilation techniques that reuse pre-computed parts and recomputing other parts on an as needed basis. Another component being used in this project is called *Graphviz Dot* [GAN00], which is widely known for its graph layout capabilities and its ability to create clear and concise UML diagrams. The final third-party component being used is an extended version of *X-Diff* [WAN05] to differentiate two generated UML diagrams in *DOT-XML* format. Third-party libraries are also implicitly used such as *TrollTech Qt 2.2* library [TRO05], *Apache Xerces-C* library [ASF05x] and the *Standard Template Library* (STL) [JOH05][STL04a][STL04j][SGI00].

5.1 Software Evolution Use Case

Currently, most companies have difficulties to visually track changes to their original context or keeping track of software changes in general. Normally, the documentation, if it is still available, is mostly outdated or does not represent the current software status, mostly due to progressive modification of the source code during its maintenance life cycle. The idea is to automatically update the documentation and to produce a process framework, which can easily keep track of changes and to correlate modifications that

was performed between two releases. Ideally the documentation, like other test cases, should be generated for each revision of the software system (basically for every night built), to ensure that it is always up to date and to allow the developer to easily track changes using *DoxyChange*.

5.2 Doxygen External Architecture

Currently, the *Doxygen* lexical analyzer can be seen as a black box, which takes input source files and a configuration file describing the task to be performed and it outputs a wide variety of output files depending on the task to be performed. For the test cases described in Chapter 6, *Doxygen* is configured to parse input C++ source code and to output *HTML*, *XML*, *DOT*, *DOT-XML*, *MAP* and *PNG* files. Internally, *Doxygen* calls *Graphviz Dot* during the output generation process to generate *DOT*, *MAP* and *PNG* files. The Figure 5.1 shown below explains the entire process which takes place. Normally, *Doxygen* creates temporary *DOT* files generated by *Graphviz Dot* for graph resizing and truncation purposes, in order to know if the end result diagram fits within the specified *MAX_DOT_GRAPH_WIDTH* and *MAX_DOT_GRAPH_HEIGHT* drawing space. In order to reduce the *Doxygen* I/O overhead, our custom version of *Doxygen* provides a new *NO_DOT_RESIZE* configuration variable to disable this feature. It also provides a new *NO_DOT_RUNJOB* configuration variable to disable any calls to *Graphviz Dot* except for resize. A new *NO_DOT_MD5* configuration variable was also added to disable the *md5sum* calculations [RIV92] used to know if the diagrams should be regenerated or not.

Finally, a new *NO_HTML_OUTPUT* configuration variable was added to disable the HTML output altogether if it is not immediately required.

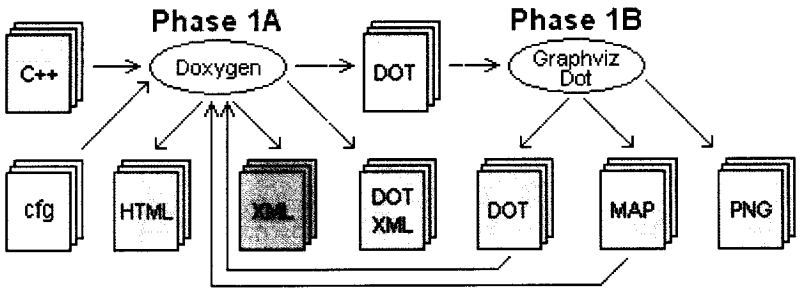

Figure 5.1: Doxygen External Architecture Overview [HEE05a]

To compensate for the lack of HTML generation, a new *HTML_URL_PREFIX* configuration variable was added, which replace the internal HTML links to external HTML links where the actual HTML documentation was previously generated. These special parameters allows *Doxygen* to reduce its extremely time consuming I/O throughput significantly as illustrated in Chapter 6 and allows the user to generate the HTML documentation on one machine and the *DOT-XML* generation on another machine to decrease the overall run-time. The *DOT-XML* generation is enabled using the *GENERATE_DOT_XML* configuration variable. It can also be tuned using the existing *UML_LOOK* configuration variable, which normally should be set to *YES*.

The display can be extended or simplified by tuning the appropriate custom extensions called: *SHOW_SIMPLIFY_PARAMS*, *SHOW_FUNCTION_PARAMS* and *SHOW_TYPE*. These configuration variables will show the full signature simplified or not, show function parameters or not, and show the return types or variable types or not.

Currently, types can be shown in C++ style or in the common ADA style used by *IBM Rational Rose* [RAT05]. By default, types are shown in ADA style, while parameters are shown in simplified C++ style. Any *const*, *static*, *virtual*, *inline*, *final*, *register*, *class*, *struct*, *union* or similar properties, including any array size are removed from the function signature for simplification purposes and also to reduce the size of the function signature. If any of these simplifications are changed, they will not be shown directly on the UML diagram, the corresponding class variable or member functions will instead be shown in orange, since the change is obviously not visible after simplification.

5.3 Doxygen Internal Architecture

Internally, *Doxygen* consists of a highly coupled, high cohesion and modular architecture. The highest degree of coupling is toward the *Qt 2.2 Toolkit*, the remaining coupling is due to the configuration singleton [GAM93], the parser and the output generator infrastructure, which is heavily inter-coupled. However, the output generator children classes are interfaced using the Visitor design pattern [GAM93], which at least eliminates the coupling between specific output generators. In others words, the HTML output generator is completely independent to the XML, RTF or $L^{A}T_{E}X$ output generator and vice-versa. In general, as it can be seen in the Figure 5.2 data flow overview diagram, *Doxygen* mostly consists of a unique configuration parser singleton [GAM93], a C/C++ preprocessor, a language parser, a tag file parser, a set of data containers, a source parser, a document parser and a set of output generators.

As shown below in Figure 5.2, *Doxygen* works in roughly four major stages, with some minor sub-stages, where the data is being retrieved, refined and gradually enhanced, in order to be finally converted into every of the specified output formats.

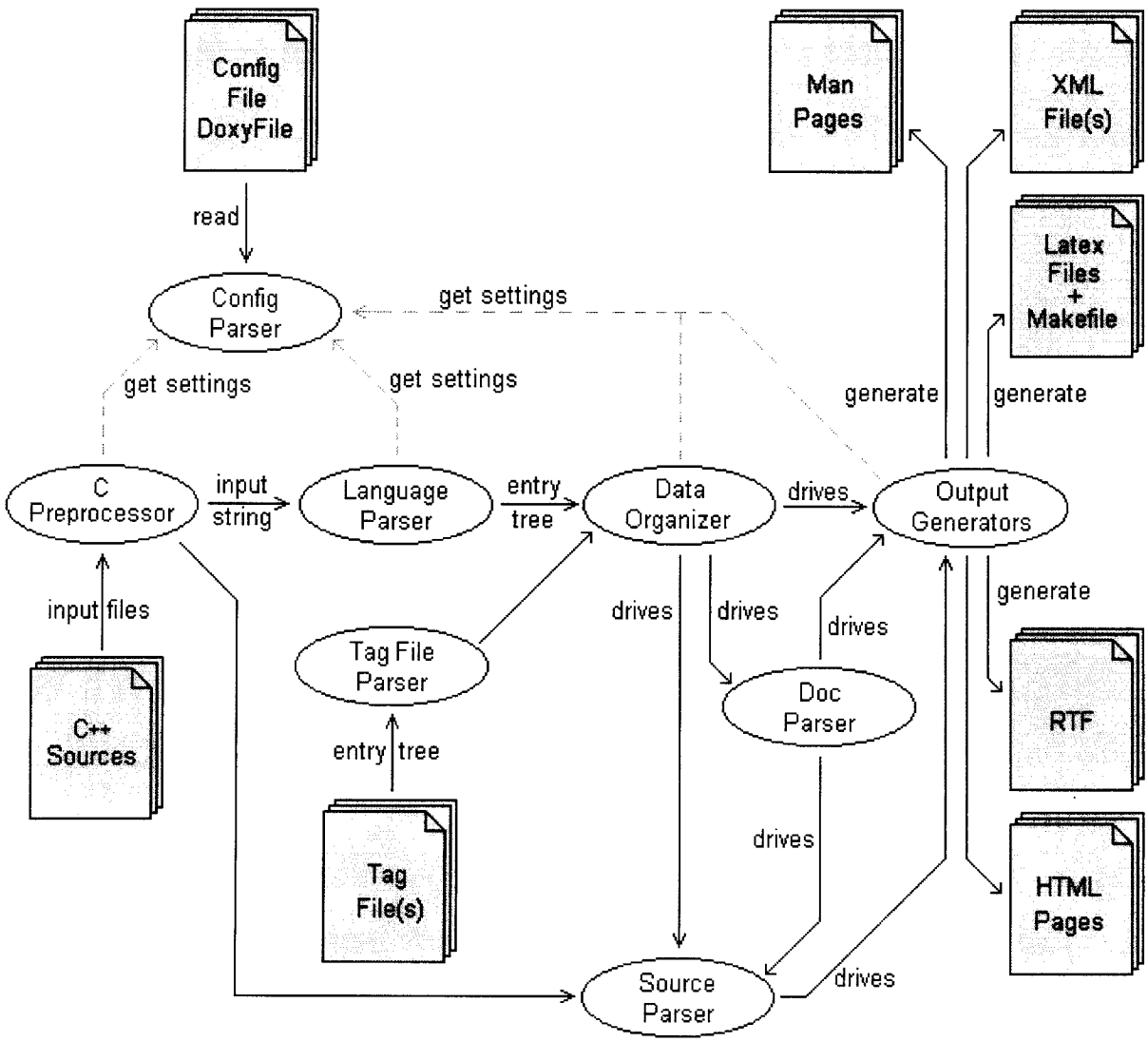

Figure 5.2: Doxygen Data Flow Overview [HEE05a]

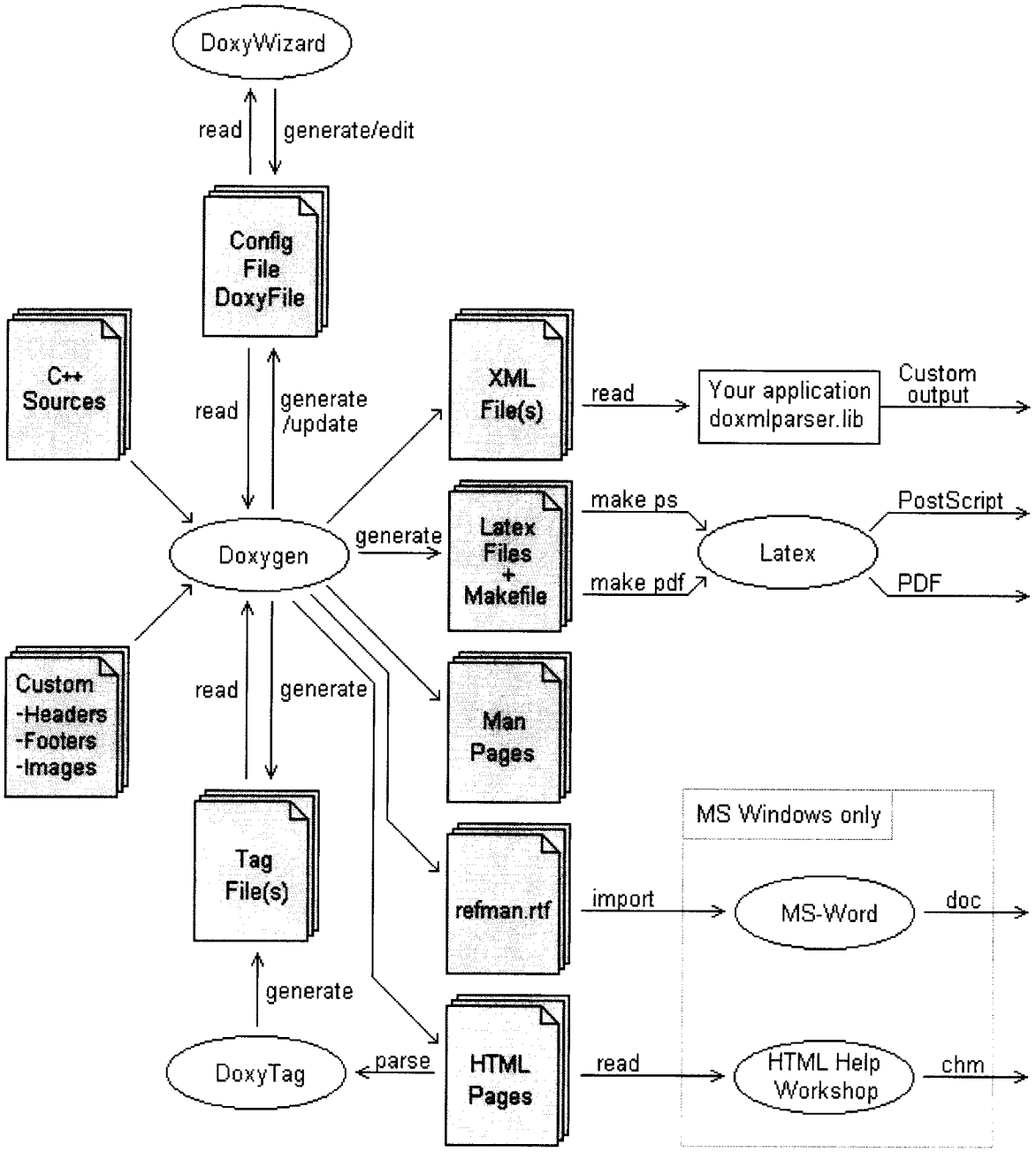

Figure 5.3: Doxygen Information Flow Overview [HEE05s]

The information flow overview diagram, shown above in Figure 5.3, shows some of the various *Doxygen* external components and how they relate to some of the *Doxygen* inputs and outputs. For instance, *DoxyWizard* is used to create or tune a configuration file, *DoxyTag* is used to create tag files from generated HTML pages and LaTeX is used to

create *PostScript* or *PDF* output. Moreover, *Microsoft Word* can be used to convert RTF files into *Microsoft Word* documents and the *HTML Help Workshop* can be used to transform the HTML pages into a *CHM* help file, also known as *Microsoft Windows 98 Help* file format. It is to be noticed that the XML files can be used as an input to other XML based system, which can used the *doxmlparser.lib* library, if needed instead of writing a new dedicated parser. Obviously, the *Graphviz Dot* interactions are missing from this information flow diagram.

5.4 DoxyChange Architecture

The overall *DoxyChange* Architecture Overview is shown below in Figure 5.4.

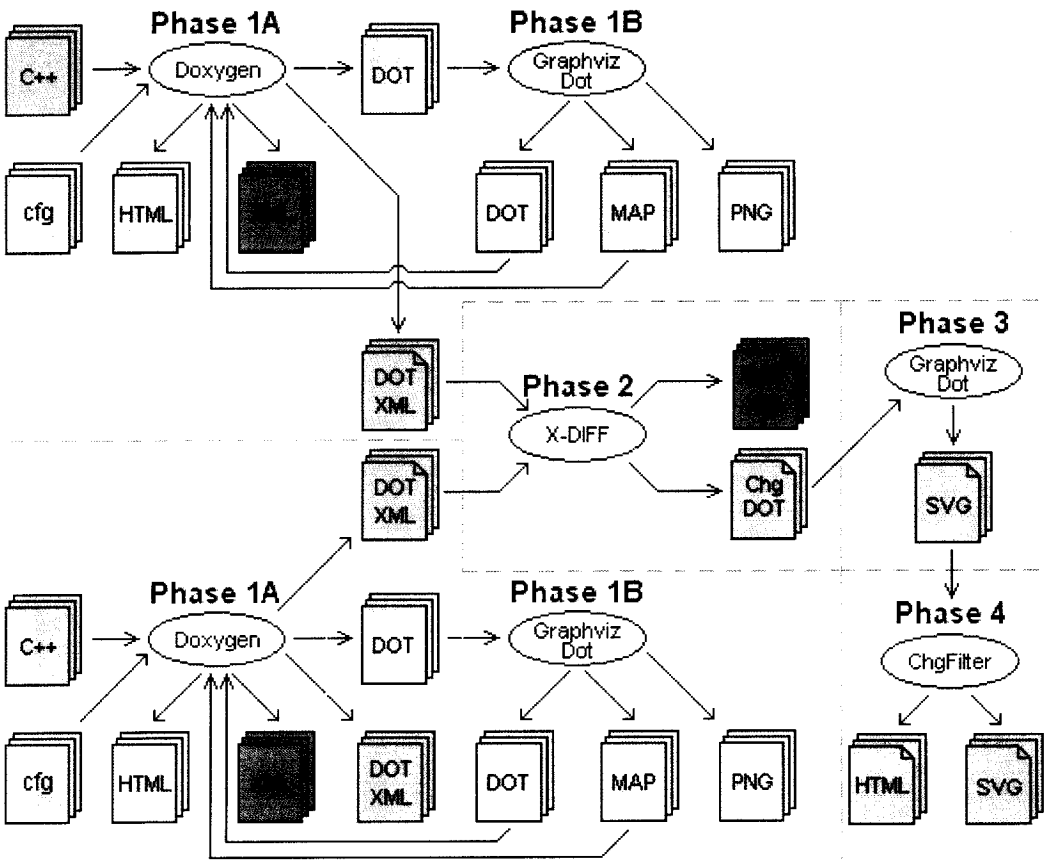

Figure 5.4: DoxyChange Architecture Overview

5.4.1 DoxyChange: Phase 1 – Parsing

Phase 1 of the *DoxyChange* process consists of parsing the input source code such as header files or implementation files. As mentioned earlier, *Doxygen* supports a wide variety of programming languages such as "*C++*, *C*, *Java*, *Objective-C*, *Python*, *IDL (Corba and Microsoft flavors)* and to some extent *PHP*, *C#*, and *D*" [HEE05]. *DoxyChange* supports all of those languages by default; however, this research will concentrate on C++, since the original *Doxygen* lexical scanner was designed in the first place for parsing this particular kind of source code and generating various kinds of diagrams from it. Moreover, *Doxygen* recently included a *UML_LOOK* configuration variable to its custom diagram, such that for instance, UML class diagram could be generated instead.

All the other programming languages are parsed by dynamically adapting the lexical scanner depending on the language context and therefore, by adding more fuzzy rules into each of the *Flex* parsers [NEE00]. Moreover, another method can also be used for other more extreme programming languages, by simply changing the input source code into some sort of fuzzy C++ input, in order to be parsed by *Doxygen* [HEE05q]. As a result, other languages such as: *JavaScript*, *Visual Basic*, *Object Pascal*, *MatLab* and *Pro*C* are indirectly supported using this methodology by using a third-party preprocessor to convert the source code into some C++ flavor compatible with *Doxygen* [HEE05h]. Consequently, *Doxygen* is a very powerful multi-language open source parser that is freely available [HEE05d] under the *GNU GPL v2 license* [TUR05]. *Doxygen* is also multi-platform and is available for Windows, Mac and wide variety of *nix flavors,

due to its use of the *Qt 2.2 Toolkit* [TRO05]. Furthermore, *Doxygen* is very flexible and easy to maintain due to its modular C++ object-oriented approach. Additionally, *Doxygen* uses a wide variety of design patterns, such as the Visitor and Iterator design patterns, while also incorporating some Interface, Delegation, Factory Method, Composite, Façade, Proxy and Singleton design patterns. Furthermore, *Doxygen* also have a configurable interface, which makes heavy usage of *QDict*, *QList* and *QStack* for storing data [TRO05t], *QTextStream* classes [TRO05i] and polymorphic generator subclasses to generate the content through a Visitor/Iterator [GAM93] feedback loop.

The parser is modular and is divided into eleven independent *Flex* parser modules [NEE00] that perform various tasks such as:

1. The ***config.l*** parser for retrieving the configuration file options;
2. The ***pre.l*** parser that acts as a preprocessor;
3. The ***scanner.l*** parser that acts as a scanner;
4. The ***code.l*** parser that acts as a code handler;
5. The ***defargs.l*** parser for functions or templates parameters;
6. The ***declinfo.l*** parser for declarative statements;
7. The ***commentcnv.l*** parser to retrieve and convert comments;
8. The ***commentscan.l*** parser that handles verbose *Doxygen* comments.
9. The ***constexp.l*** parsers to handle numerical or Boolean expressions;
10. The ***doctokenizer.l*** parser that acts as a document tokenizer;
11. The ***doxytag.l*** parser for the *Doxytag* generated HTML tag files;

The *DOT*, *HTML*, *MAP* and *PNG* output files shown in pink are generated by default by *Doxygen* for every input source files and every parsing code context that it judges meaningful. As a result the following diagrams are generated: include file inheritance

diagram, include file dependency diagrams, class inheritance diagrams, class collaboration diagrams, call graphs, context specific dependency diagrams and class hierarchy diagram.

To browse these diagrams, *Doxygen* also generates a wide variety of HTML files to document further the source code. One of them is the HTML documentation with enhanced *Doxygen* or *Javadoc* style comment of every C++ input files with the source code *inlined* with every keyword colored and linked to their specific HTML documentation. This output is crucial for *DoxyChange*, since it is in this specific part of ***Phase 1*** where the diagrams are generated and therefore, where the *DOT-XML* add-on takes place. Most of the processing is done in *dot.cpp*, which is the module that handles the *dot* language related generation.

Doxygen also generates a list of class and files with abbreviated comments, a textual and graphical class hierarchy, a textual directory hierarchy, an overall member lists, variable list, function list, typedef list, enumeration list, enumerator list and property list with the class name or file name where they are used. It also generates a *PHP* search engine, if the *Doxygen* configuration file variable *SEARCHENGINE* is set to YES. The same documentation is offered in a variety of flavor such as: *HTML*, *L^AT_EX*, *XML*, *RTF*, man pages and as a bunch of *Perl* modules for scripting purposes. The *DOT-XML*, *Change-DOT*, *HTML* and *SVG* output shown in purple are generated by the various components of *DoxyChange*. The *XML* output and *DIFF-XML* output shown in green are facultative output which might be omitted, since *DoxyChange* does not currently use any of these files and these files are only provided to be used by any other third-party tools.

The ***Phase 2***, ***Phase 3*** and ***Phase 4*** is performed by calling a *Perl* script, which creates a list of all input *DOT-XML* files and then create a corresponding empty *XML* file to complete any file pair that cannot be matched successfully.

5.4.2 DoxyChange: Phase 2 – DOT-XML Differentiating

The ***Phase 2*** uses a custom made *Qt 2.2* version of *X-Diff* with a new *XPortability.hpp* header file added for portability reasons, in order to make *X-Diff* compile on Windows. Even though *X-Diff* looks quite innocent with its ten source files, it relies on not really portable and hard to compile external libraries and functionalities, which are barely documented. The only hint provided is the README file which contains the following clue: "Required: *Xerces C++ v1.4.0*, *g++ v2.9.** ". *Apache Xerces C++ version 1.4.0* is no longer maintained and the last version of the v1.x series is 1.7.0, which is incompatible with *X-Diff*. Furthermore, the *g++ v2.9.x* is four years old and the latest version of this series was released in February 2001 [ANG04] and is no longer maintained. The newest GCC version is *v4.0.1*, which is incompatible with the Windows platform without using *Cygwin*. As a side effect, *g++* relies on the *GNU libstdc++* implementation; more specifically, *X-Diff* relies on the *GNU libstdc++ hash_map* implementation, which is not available on Windows. To solve these two issues some research had to be done to first identify what a *hash_map* is and if a compatible implementation exists on Windows. Even though *hash_map* is part of the *std* STL namespace in *libstdc++*, it is only a C++ recommendation [AUS03] and most STL implementations do not support it.

Consequently, *Qt 2.2* was used to replace the non-portable *hash_map* STL class to work around the *GNU GCC* [ANG04] non-portable extension such as: specific 64-bit integer type and the *nix time structure *timeval* and *timezone*, which was replaced by some proper #define macros, typedef or some custom stubs implementation using *clock_t* and *CLOCK_PER_SEC*. Other work around were also used such as: some *#pragma*, to ignore some extremely noisy template name length warnings and some *#define*, to mask the small function signature difference between each platform. *QMap* was adapted to behave like a *hash_map* using some *#define* and a new default constructor where the size parameter is ignored. The reasons for using *QMap* instead of some of the STL *hash_map* implementation is that it was nearly impossible to find a *hash_map* implementation that would work under *Microsoft Visual C++ 5.0 or 6.0 (MSVC)*. Some widely known implementation such as *libstdc++* [JOH05], *SGI* [SGI00] and *STLport* [STL04a][STL04j] would not work properly or would simply collide with the existing partial STL implementation already provided by the Microsoft compiler. The major problem was that all of these STL implementations provided their own heavily inter-coupled implementations of every STL class that cannot be worked around to use the Microsoft STL implementation. However, even when the third party STL classes were tweaked, in order to be used with the Microsoft compiler, compilation errors were still generated. As an alternative solution, the *Qt 2.2 Toolkit* was considered since it already provided a set of portable container classes that are fully compatible with the Microsoft compiler. For instance, *QMap* provided by *Qt 2.2 Toolkit* was very close to the *hash_map* interface and provided similar functionality. To overcome the minor naming interface problems, some #define macros were introduced, since the *GNU libstdc++* uses

hash_map<T1, T2>::const_iterator, while *Qt 2.2* uses *QMap< T1, T2 >::ConstIterator*. Notice the upper case naming convention used in *Qt 2.2* and the lower case and underscore naming convention used by the STL. This macro approach is required, since the *hash_map* template cannot be typedef and a third argument is used to compare strings in *hash_map*, which do not exist in *QMap*. Finally, the *hash_map* constructor takes an integer parameter to indicate the default size, which does not exist in *QMap*. This can easily be worked around by replacing the *QMap()* constructor definition by *QMap(int sz=0)* and simply ignoring the size parameter. As a result, a portability layer called *XPortability.hpp* was written and the X-*Diff* source code was modified to use this portability layer instead of calling the functions directly.

The only major limitation related to the usage of *Qt 2.2* is the requirement that the entire source code must be licensed under the *GNU GPL v2 license*. This restriction is not really problematic, since the *X-Diff* source code is available on some flavor of the BSD license, which is compatible with the *Qt 2.2* GPL license. Moreover, this GPL restriction upon *X-Diff* can easily be removed by simply eliminating the *QMap* dependency.

On the other end, the *Apache Xerces C++ version 1.4.0* dependency had to be resolved. The problem was to find a *MSVC* project or *Makefile* which would work with the *Apache Xerces C++* library. Since the version 1.4.0 refused to compile, some insight suggested that maybe the latest version would provide the proper bug fix. However, the latest version 2.6.0 is incompatible with the version 1.4.0, after the *Apache* source base underwent some major refactoring. Some attempts were made to fix these issues;

however, the overall complexity involved was overwhelming and another solution had to be found. Since this approaches was not conclusive, the latest version of the 1.x series was selected, since it did not have any of the version 2.x refactorings in it, while it would still addressed some of the problems inherent with the *Apache Xerces C++ v1.4.0*. However, *Apache Xerces C++ version 1.7.0* does not provide a compatible *MSVC* project or a compatible *Makefile*. Consequently, the *Makefile* from version 1.4.0 was adapted for version 1.7.0, which eventually worked just fine.

Another major implementation issue was that *Xerces* is designed to be dynamically linked (DLL) instead of being statically linked. The problem with DLL is that they cannot be debugged live by stepping through the source code, when *X-Diff* calls any *Xerces* functions internally. This was important since the resulting modified version 1.7.0 was not working properly and would cause a *Segmentation Fault* crash for no apparent reason. The library was modified significantly to compile it statically against *X-Diff*. Once, *Xerces* was successfully statically linked the guilty unsupported C++ construct *"try {...} catch(...) { throw; }"* was found and *"throw;"* was commented out, since it is not supported by the *Microsoft* compiler.

Attempts were made to trim down the size of *Xerces* from what looked like redundant components. However, this is not really possible, since *Xerces* is extremely and heavily inter-coupled within itself. In other words, every file in *Xerces* barely needs every other file in *Xerces*, in one way or another. As a small example, the SAX compiler is tightly coupled against the DOM compiler and vice-versa [WOO00], even if does not make any

sense at all at first sight. Furthermore, these two XML compilers API are forced to use a *Validator*, which must use a *Model*, which in turn must use every possible *Model* and/or *Validator* and that is just the beginning. Now to parse or display some text, a *Transcoder* is needed, but since one of them is needed, all of them need to be loaded, initialized and are therefore, needed just in case. Since nearly 75% of the files are tightly coupled, it is unsurprising that these specific files also require, in one way or another, the remaining source files. This might explain why *Xerces* is provided as a DLL or dynamically loaded library, since it probably makes no sense at all to compile it statically except for debugging purposes.

As a result, the *Xerces* dependency is roughly 10 MB of source code and roughly 70 MB of temporary files once compiled in debug mode, which must be added to the size of *QMap* and all its *Qt 2.2* dependencies. Size reduction was however possible for the *Qt 2.2* dependencies by using the configuration header file named *<qconfig.h>*. Consequently, after some compiler optimization, the *X-Diff* executable was reduced to 2.3 MB without any external dependency, which finally sounds reasonable. Esepcially, for a small utility that originally contained only 10 C++ source file within a single tar ball, since the *hash_map* and *Apache Xerces-C* dependencies were not included, neither clearly identified at first sight.

Once *X-Diff* was optimized, the *XDiff.cpp* was split into three C++ files: *XDiff.cpp*, *XDiffMain.cpp* and *XDiffWriteXML.cpp*. A new file called *XDiffWriteSVG.cpp* was then added, which is based on the *XDiffWriteXML.cpp* implementation. This file is used by

DoxyChange to generate the *Change-DOT* output as explained in the *DoxyChange* Architecture Overview diagram shown in Figure 5.4.

5.4.3 DoxyChange: Phase 3 – SVG Generation

During ***Phase 3***, the *AT&T Graphviz Dot* utility is executed from the command line by requiring an *SVG* output instead of a normal *DOT*, *MAP* or *PNG* output which was previously generated by the *Doxygen* program.

5.4.4 DoxyChange: Phase 4 – SVG Beautifier

The ***Phase 4*** transform the encoded XML strings into a more visually pleasing SVG format by replacing the encoded text into a new set of decoded strings and finally by adding colors, circles, arrows and boxes decoration to represent the various software changes that occurred between two revisions. For instance, *"///"* delimitates the signature before and after the change actually occurred. Therefore, *"f()///"* means that the function *"f()"* was removed, while *"///g()"* means that the function *"g()"* was added and *"h()///h(bool b)"* means that the function *"h()"* signature was changed into its corresponding *"h(bool b)"* signature. Moreover, *"h()@@@"* signifies that *"h()"* was modified internally or that its full signature was modified, but not its abbreviated signature. For instance, if the private class variable *"QString a[2]"* is changed to *"QString a[4]"*, then it will be shown as modified, since the abbreviated signature in both cases is *"- a : QString[]"*.

Phase 4 is performed using a custom made C++ utility, which heavily depends on the *Qt 2.2 Toolkit*, more precisely it depends on *QCString*, *QString* and *QRegExp* [TRO05t]. The first version was written in *Perl* using some simple regular expressions and a nearly compatible C++ syntax knowing that it would be converted later on into C++ using the *Qt 2.2 Toolkit*. The advantage of using *Perl* as a fast prototype tool is that it provides some very handy verbose structure debugging information via the *Data::Dumper* package [MAR03] and that the compile time is nearly inexistent compared to the few minutes required to rebuild any revision of a *Qt 2.2* project statically. As a result, the complete *Perl* script and its C++ version were written and fully debugged in less than a day. More specifically, once the *Perl* script was fully working, it took less than a few hours to translate it into proper C++ source code. The main overhead was that the *Qt 2.2 Toolkit* [TRO05t] compared to the *Qt 3.3.4 Toolkit* [TRO05n] do not support grouping within regular expressions or any advanced regular expression features. Therefore, the regular expression logic had to be reduced into a set of fine-grained statements to simulate its corresponding multi-matching regular expression statements.

After the translation was complete, additional functionalities were added and some of the underlying logic was refined to support additional features, such as partial signature renaming. Furthermore, some context and CSS styles were added to reduce the file size, to clarify the resulting SVG code and to make dynamic JavaScript view easier to program by simply searching and changing the CSS style. Finally, circle decoration was added for arrows and edges. The *Polygon*, *Edges* and *Text* nodes were grouped and handled recursively. The grouping was required to ensure that the context could be properly

optimized and that the logical flow and order was ensured. For larger graphs, edges had a tendency of being declared before their target nodes. To overcome this issue, edges are buffered within a *QVector* to be decoded once every node is properly handled.

5.7 Doxygen Legend

The following section explains the nomenclature used by *Doxygen* when drawing its UML diagrams. As it can be seen in Figure 5.5, *Doxygen* use a combination of color and strike to represent different kinds of protections and associations between classes.

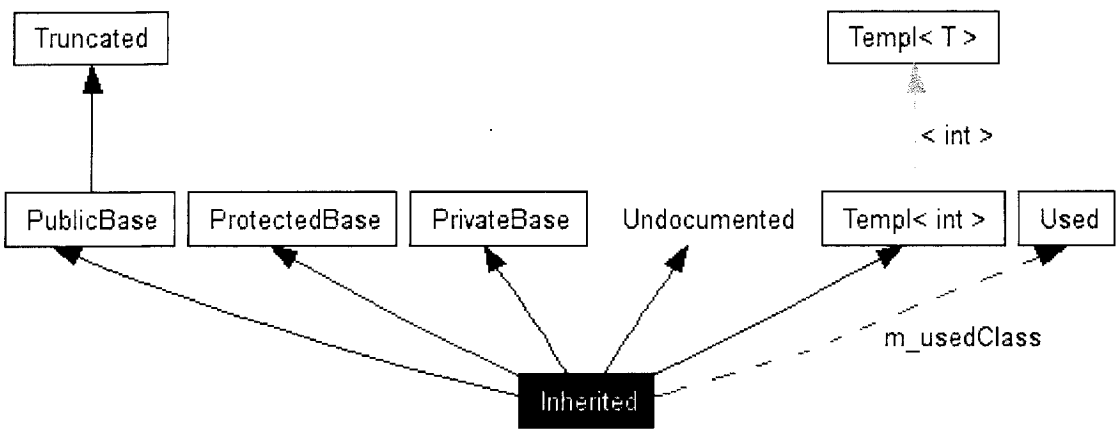

Figure 5.5: Doxygen Arrow and Box Legend [HEE05g]

The arrow colors and shapes have the following meaning:

- A dark blue arrow represents a public inheritance relationship between two classes.

- A dark green arrow represents a protected inheritance relationship between two classes.

- A dark red arrow represents a private inheritance relationship between two classes.

- A purple dashed arrow represents a variable aggregation dependency between two classes where the labeled arrow is the variable name, which points to the variable class type.

- An orange dashed arrow represents a relationship between a template instance and the derived instantiated template class where the labeled arrow represents the template parameters being used.

The box colors have the following meaning:

- A filled black box represents the current class focus for which the graph was generated.

- A box with a black border represents a fully documented class.

- A box with a gray border represents an undocumented class.

- A box with a red border represents a documented class that was truncated for clarity reasons, where all the class specific inheritance or aggregation relationships were voluntarily omitted. A graph is voluntarily truncated if the graph size does not fit within a drawing page as specified by the *Doxygen* configuration file under the variables *MAX_DOT_GRAPH_WIDTH* and *MAX_DOT_GRAPH_HEIGHT*.

The text colors and shapes have the following meaning:

- Black text represents a class name, an attribute or a function, which was not changed.

- Light green text represents a class name, an attribute or a function, which was added.

- Stricken red text represents a class name, an attribute or a function, which was deleted.

- Orange text represents a class name, an attribute or a function, which was modified without changing its signature.

- Stricken red text followed by light green text represents a class name, an attribute or a function for which its signature changed. For instance, it was renamed, its protection changed or its type or parameter type was changed.

The box shapes have the following meaning:

- A non-dashed regular box represents a class, which was not added, nor deleted.

- A small dashed box represents a class, which was added.

- A wide dashed box represents a class, which was removed.

5.8 DoxyChange Legend

The following section explains the nomenclature used by *DoxyChange* when drawing its UML diagrams. It is obviously an extension to the *Doxygen* legend shown in Section 5.7. As it can be seen in Figure 5.6, *DoxyChange* also use a combination of color, strike and circle to represent different kinds of changes between classes and also within a single class.

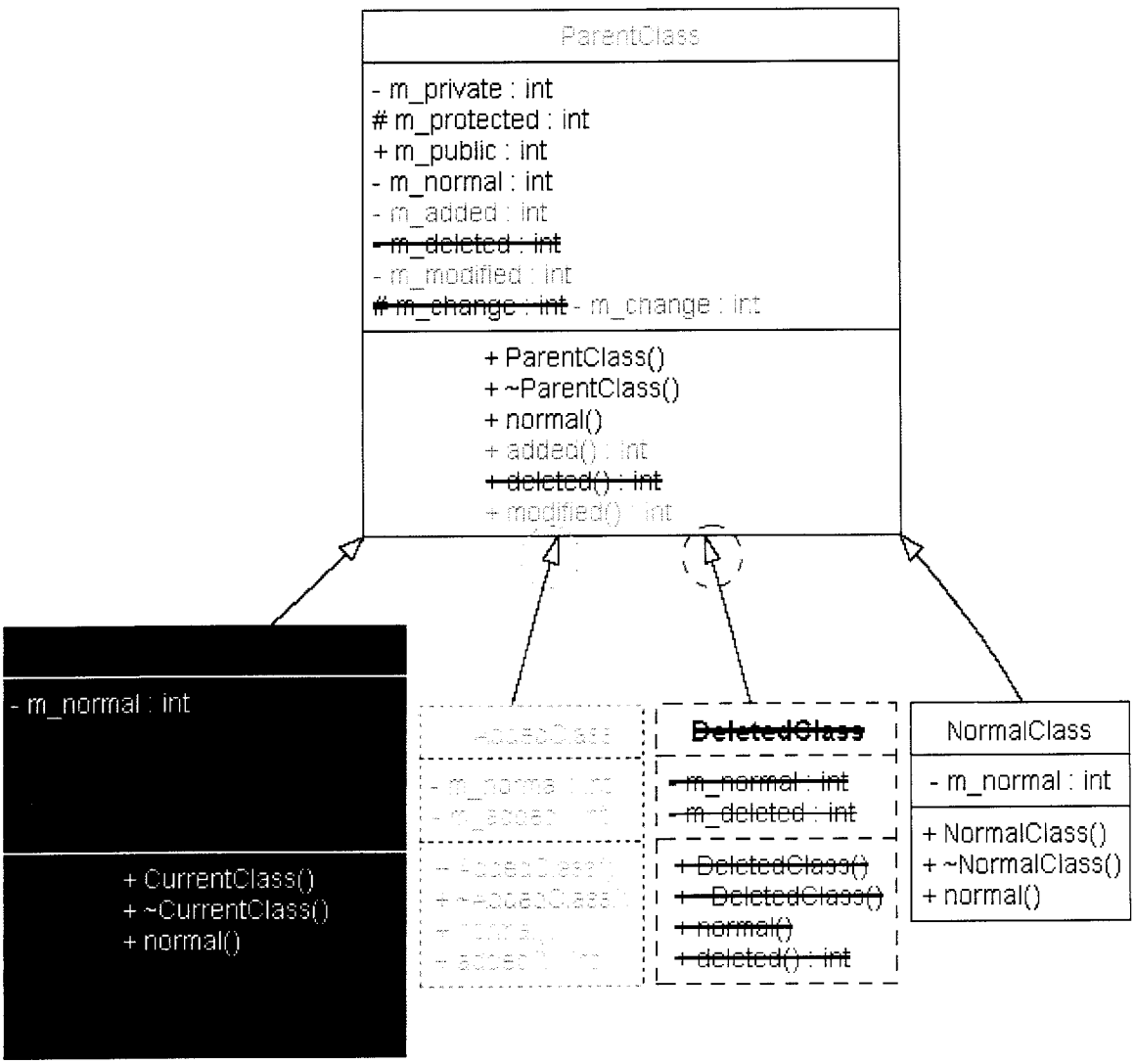

Figure 5.6: DoxyChange Arrow, Box and Text Legend

The circle colors and shapes around arrows have the following meaning:

- An arrow without any circle around it represents an association, which was not changed.

- An arrow with a small dashed green circle represents an association, which was added.

- An arrow with a wide dashed red circle represents an association, which was deleted.

The symbols prefixing attributes or functions have the following meaning:

- A plus symbol '+' represents a public attribute or function.

- A sharp symbol ' # ' represents a protected attribute or function.

- A minus symbol '-' represents a private attribute or function.

Chapter 6 Case Study

All test cases were performed on a Dell Inspiron 5100 Laptop equipped with a Pentium 4 2.66GHz processor, 512MB of 266MHz DDR SDRAM, 30GB 5400RPM hard drive, running Microsoft Windows XP Service Pack 2 Home Edition with the usual drivers and default settings. The test case using Mandriva Linux 2005 Limited Edition using 16x16MB RAMDISK was also performed on this Dell Laptop setup. The RAMDISK setup was found to be 2 to 4 times faster than without any RAMDISK depending on the actual target. However, since the output could not be generated on a flat directory structure, due to *tmpfs* and *inodes* limitations, this generated output could not directly be used for the following phases without some major changes. It is also worth mentioning that the *tmpfs* implementation provided with the *Linux Kernel* with our current Mandriva setup did not support RAMDISK size to be larger than 16MB, as a result 16 small RAMDISK had to be used instead.

6.1 Trivial Homegrown Case Study

The first case study is inherently trivial. It is small, concise and easy to analyze. It was designed and crafted to test the various features of *DoxyChange*, while reducing the amount of time spent on each test. This test case is crucial, since running *DoxyChange*

hundreds of time on a real use case for tweaking the SVG output generation and tuning the configuration parameters would not be very time efficient. In comparison, this trivial test case takes less than a few minutes to be fully executed, while the *KDE Utils* or *KDE Libs* use cases take few hours or days, depending on the parameter settings. The input file for this trivial test case consists of a single C++ header file containing all the desired class information. The generated output of this use case was previously shown in Figure 5.6.

Another test case was also made that consisted of a class having the following public attributes *"int previous, next;"* in one revision versus *"void *m_next, *m_prev;"* in the other revision. However, *X-Diff* could not successively match them correctly by default. This result was performed, prior to the node name tag wrap addition. In this context, the rename was associated in appearance order, basically relying on pure luck. As a result, the diagram showed that *"previous"* was renamed into *"m_next"* and *"next"* was renamed into *"m_prev"*. After applying the node name patch, these operations are now distinct and unassociated with each other. In other words, four operations are now indicated, more precisely, *"previous"* and *"next"* were deleted and *"m_next"* and *"m_prev"* were added, ensuring consistent and proper results for such corner cases. Obviously as shown in Figure 5.6 simple type change or protection change or other signature changes, which are not influencing the item name, should always be match correctly.

Another important corner case, which might not always be handled properly, is the following: *"const string& operator=(const string&)"* and *"const T&*

operator=(const T&)" versus *"const D& operator=(const D&)"* and *"const QString& operator=(const QString&)"*. In this case the function name *"operator="* is the same; therefore, the association could be matched inconsistently in appearance order. A trickier use case is *"Array(int); Array(const Array&);"* versus *"Array(); Array(int); Array(const Array&);"*, again the constructor name is the same. Similar corner cases happens, for any two or more function overloading, where at least one of them changed its appearance or signature between two revisions. In practice, those parameter corner cases seem to be handled properly, especially if the protection is the same. However, such results cannot be guaranteed and improper resolution might eventually occur. For instance, in one of the use cases, *"+ RegExp();"* was changed into *"+ RegExp(bool); - RegExp();"*, *DoxyChange* associated such change in appearance order; therefore, first by protection, then by type. The reason behind this is that the diagrams are generated internally by iterating over every item from each protection sets, therefore, visually ordering them by protection first, then by type. The resulting change context was that *"+ RegExp();"* was renamed into *"+ RegExp(bool);"* and *"- RegExp();"* was therefore added, instead of having *"+ RegExp();"* to be renamed into *"- RegExp();"* and *"+ RegExp(bool);"* to be added. However, in this specific case, this modification is correct, since *"- RegExp();"* was added to ensure that no default constructor is created by the C++ compiler and to prohibit it's usage. In any case, if such improper mismatch actually happens, it is far from catastrophic and it can easily be detected and understood by simply looking at the generated diagram. Obviously, adding harder restrictions on the signature would create more harms than goods; therefore, these problematic corner cases are already established as a known *DoxyChange* limitation as stated in Chapter 1. Notice that if the order of

appearance does not change, the logical associations are more likely to be matched successfully.

6.2 KDE Utils Case Study

The second case study is a real application bundle, one that changed significantly during the last few years. The bundle in question is part of the standard KDE 3.x distribution [KDE05]. It mainly consists of various handy KDE utilities as shown in the Figure 6.1 below. *KDE Utils 3.x* consists of several applications that were removed or added that will not be discussed in detail, since the change diagrams are either completely red or green respectively. Therefore, not really interesting from a software evolution point of view. As shown in Figure 6.1, *KAB* was removed along with *KCardTools*, *KDEPasswd*, *KLJetTool*, *klpq* and *klprfax* from the *KDE Utils* 3.4.1 bundle. On the other hand, *KDElirc*, *KGPG*, *KMilo*, *KSim* and *KWallet* were added to the *KDE Utils 3.4.1* bundle. Out of this bundle, five specific applications have significantly evolved, which are: *KRegExpEditor*, *KHexEdit*, *KDiskFree*, *KTimer* and *Ark*. These specific KDE applications will be analyzed in further details, including their UML class diagrams and file dependency diagrams when judged meaningful. The end result is to evaluate how useful the change diagrams really are and to see if this technique can be useful for a large scale project such as KDE [KDE05]. In general, the amount of coupling between each utility should be as low as possible, since they are essentially independent stand-alone executables.

It is worth mentioning that *DoxyChange* is not always able to distinguish clearly between a new feature and a feature that was moved to a new file. In some cases, surrounding diagrams will clearly indicate it and therefore clarify, that the feature was present in another class or not, but this is not always the case. As a result, the *KDiskFree* and *Ark* utility will be partially discussed, since the various diagrams could not clearly identify if these classes are inexistent in the previous release or were simply moved from one file to another file by simply looking at the various surrounding diagrams. From the file list, shown in Figure 6.2, some of them might have been moved around, refactored or removed, while certain new features were obviously added such as the support for the *7-zip* file format [PAV05] as shown by the file names *sevenzip.h* and *sevenzip.cpp*. Nevertheless, some interesting changes in the *Ark* utility will be presented in section 6.3.5.

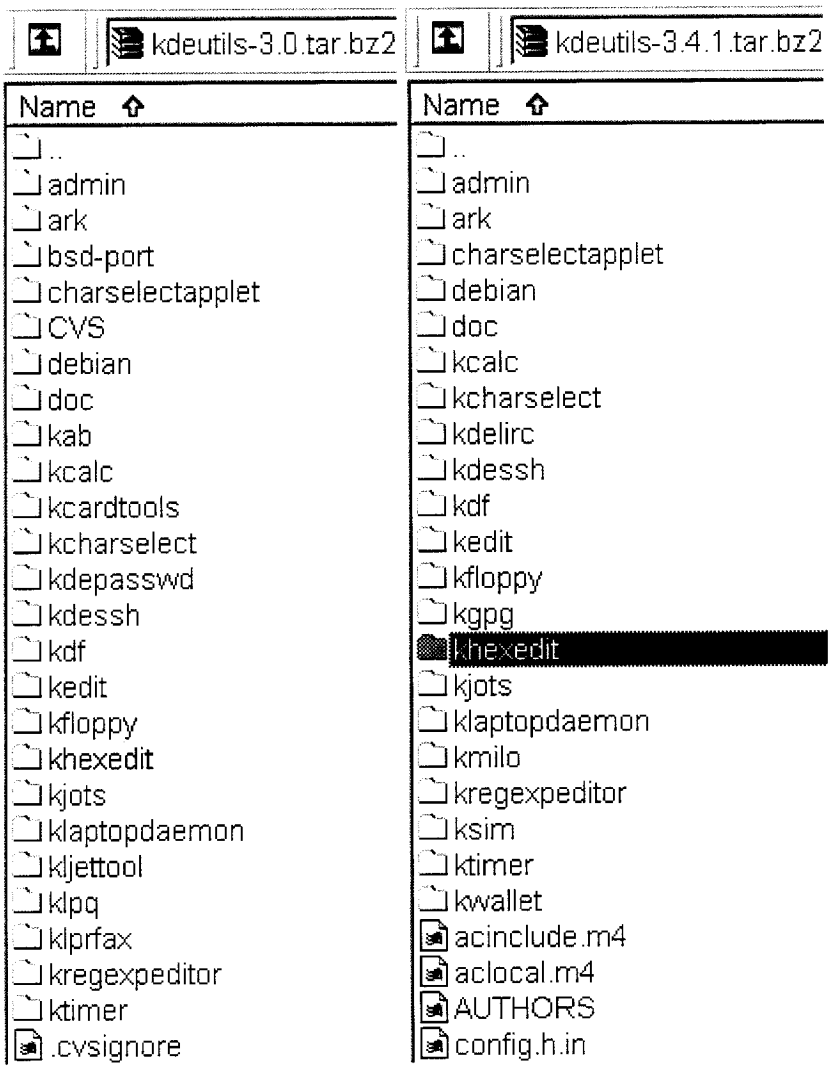

Figure 6.1: KDE Utils 3.0 vs 3.4.1 – Application Listing

Figure 6.2: Ark 3.0 vs 3.4.1 – File Listing

The other *KDE Utils 3.x* applications that will not be evaluated in further details, are either extremely stable such that no changes occurred between each revision or outstanding changes occurred such that no useful comparison could be made. For instance, *kjots*, shown in Figure 6.3, had all its files renamed such that no comparison could take place. *KFloppy*, shown in Figure 6.4, was stable apart for the *format.h* and *format.cpp* file add-on, which was not obvious from the UML diagrams, since every item is either added or deleted.

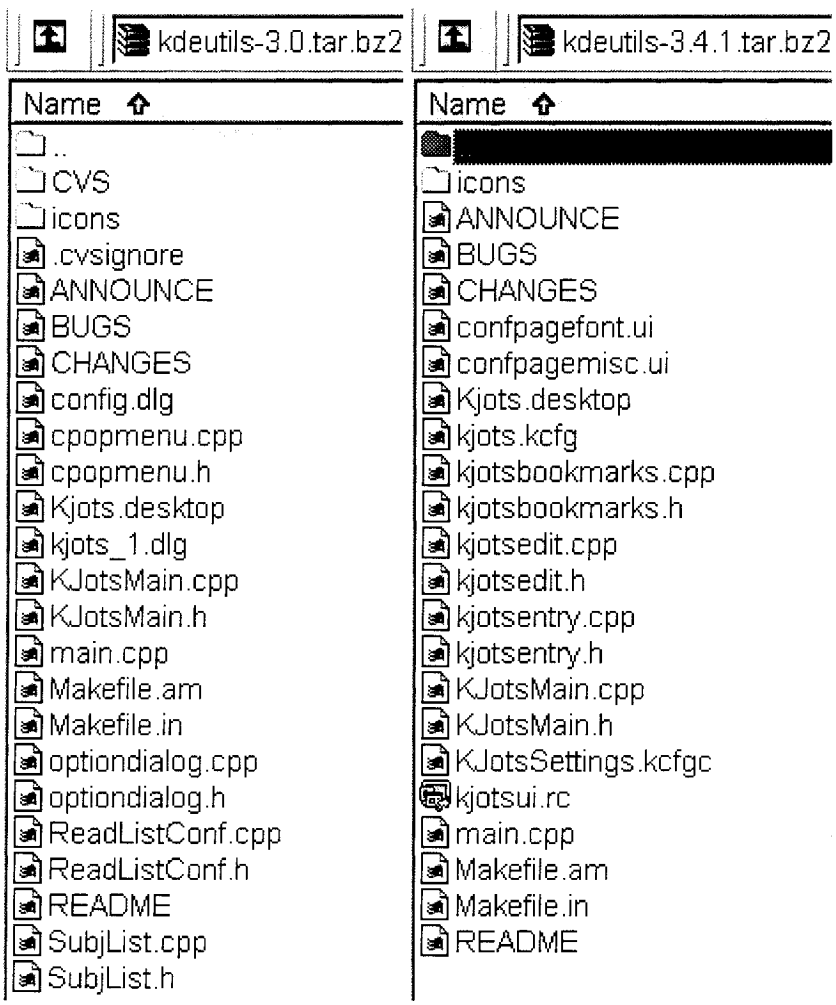

Figure 6.3: kjots 3.0 vs 3.4.1 – File Listing

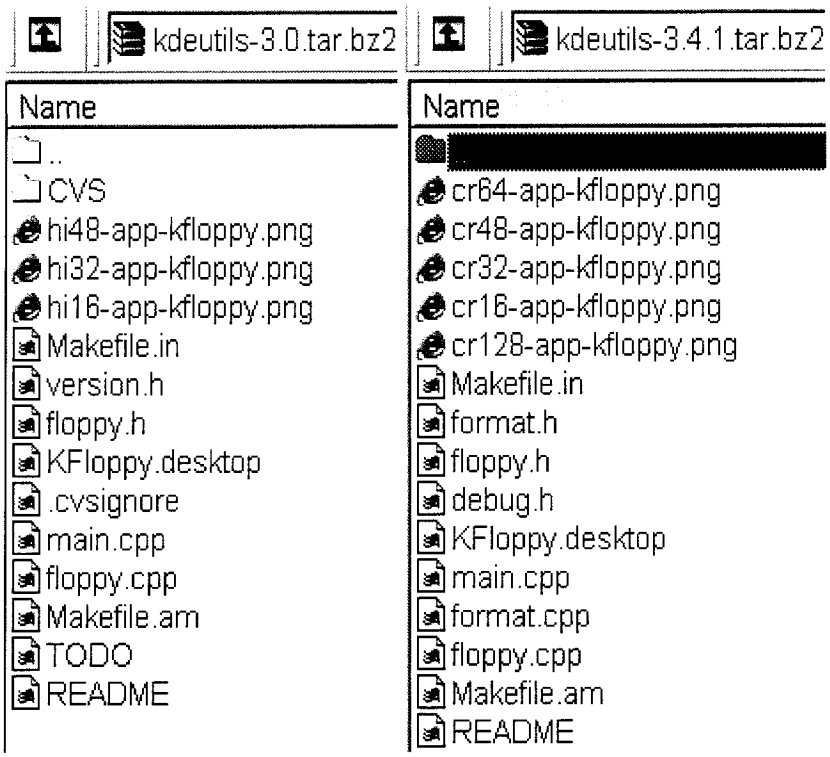

Figure 6.4: KFloppy 3.0 vs 3.4.1 – File Listing

KEdit, shown in Figure 6.4, was stable apart from the *optiondialog.h*, *optiondialog.cpp*, *mail.h* and *mail.cpp* source files being removed. *KDESSH* and *CharSelectApplet*, shown in Figure 6.6 and Figure 6.7 respectively, did not change significantly. Finally, *KCalc*, shown in Figure 6.8, underwent a major change that included adding or moving source code to too many new files to make any observable change using *DoxyChange*. Most of the diagrams, which did not show any changes, were also skipped.

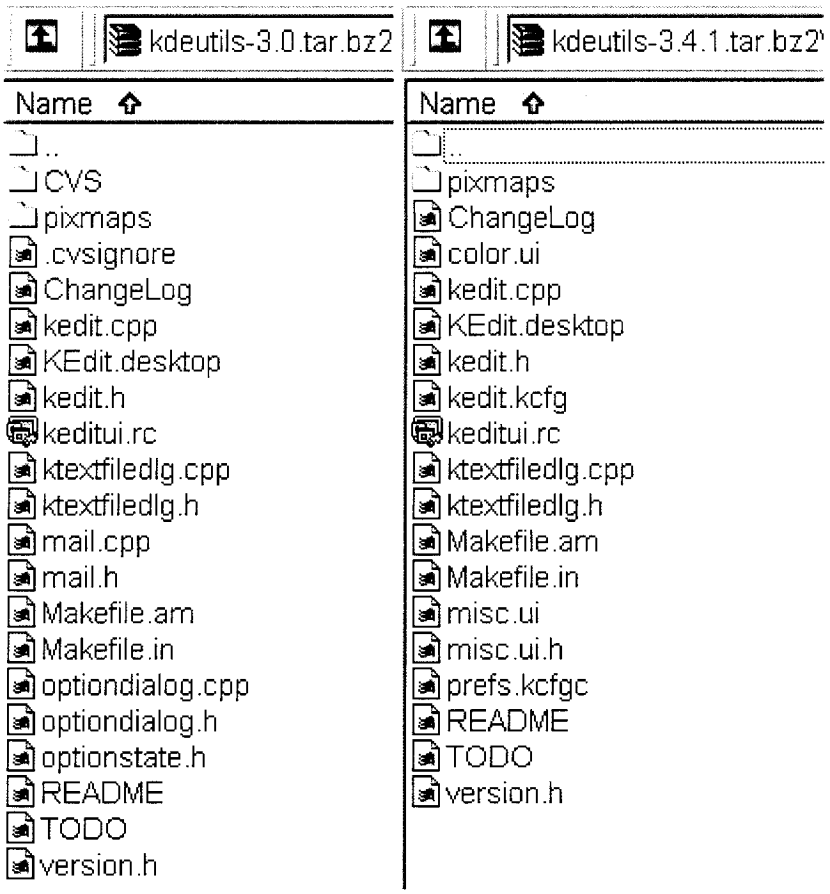

Figure 6.5: KEdit 3.0 vs 3.4.1 – File Listing

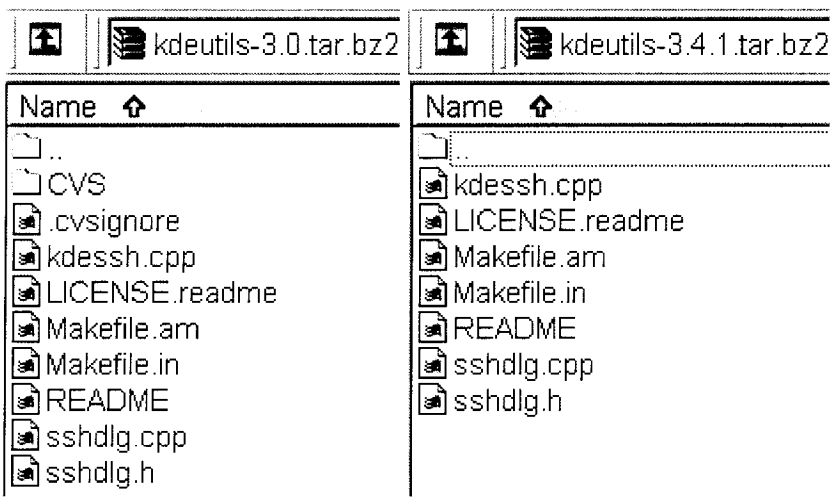

Figure 6.6: KDESSH 3.0 vs 3.4.1 – File Listing

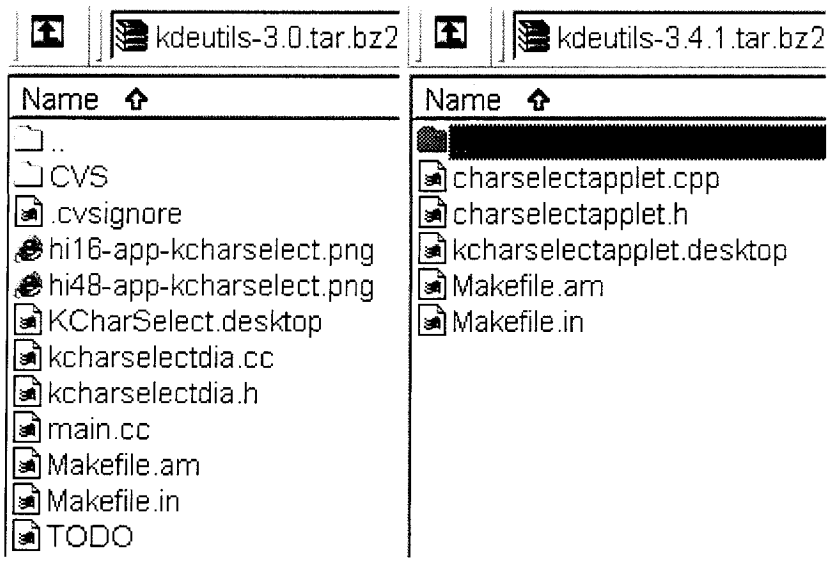

Figure 6.7: CharSelectApplet 3.0 vs 3.4.1 – File Listing

As an example, the *SingleFactory* inheritance diagram was not modified as shown in Figure 6.9 and any diagrams like such were also skipped. Furthermore, every diagram that showed every class as being added or removed were also skipped. This happens when *DoxyChange* compares a diagram with an empty file, since the other corresponding diagram does not exist for this mismatched file pair. In other words, either the file was renamed or moved or the corresponding file simply does not exist. Finally, another fact that can be seen from these file archive listings is that the *CVS* files were removed in *KDE Utils 3.4.1* archive. The reason behind this is that KDE moved to *SubVersion* in May 2005 and as a result, any *CVS* related files are no longer required [RID05].

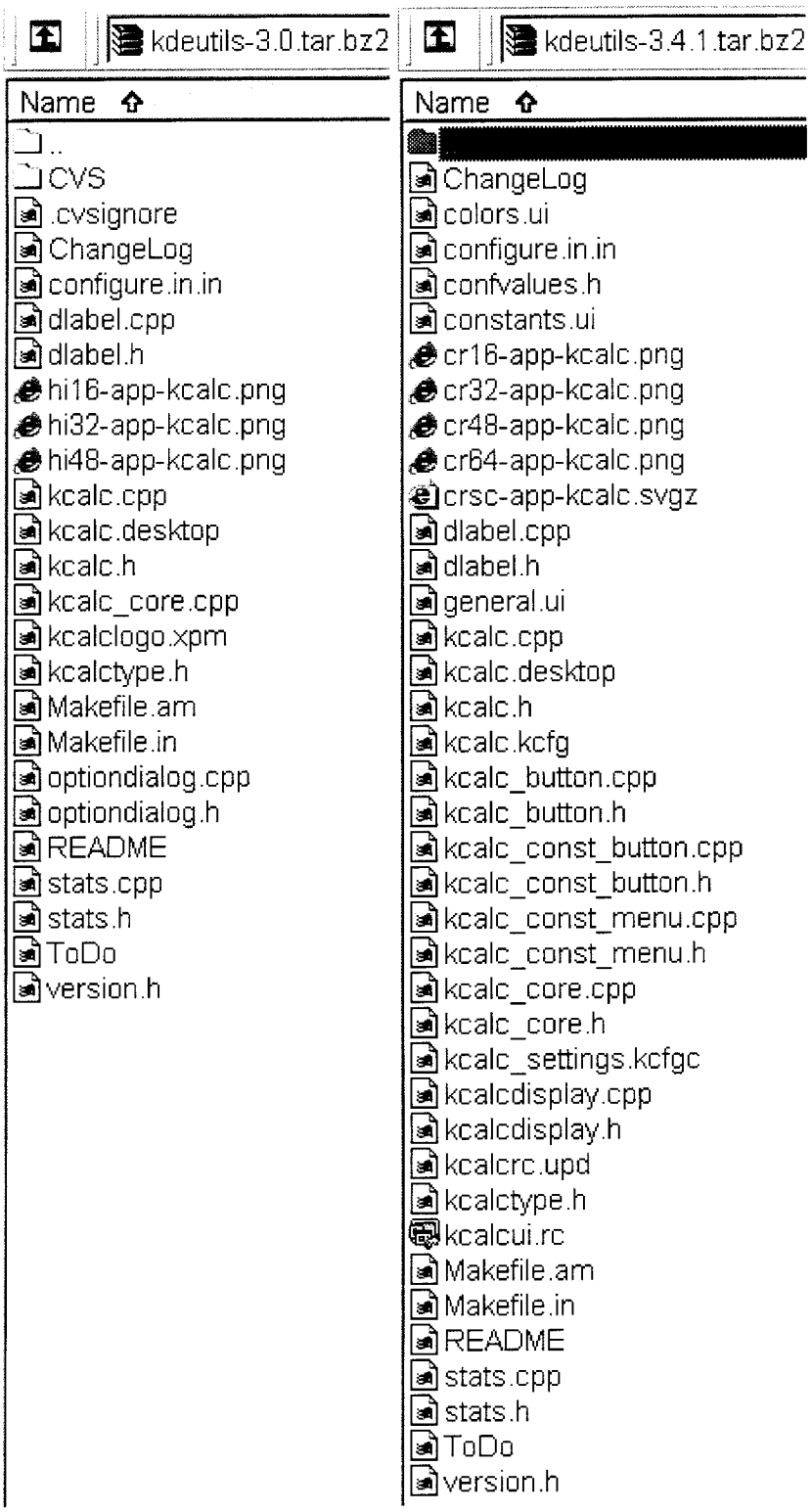

Figure 6.8: KCalc 3.0 vs 3.4.1 – File Listing

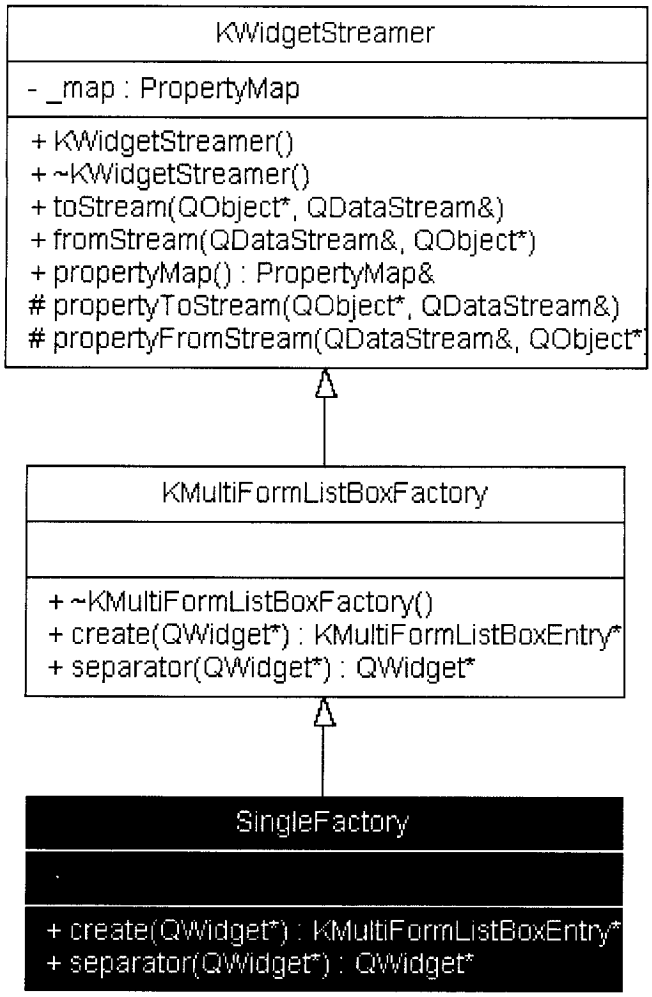

Figure 6.9: KRegExpEditor: classSingleFactory__inherit__graph

6.2.1 KRegExpEditor Case Study

KRegExpEditor is a small KDE utility, which provides a convivial user interface for editing regular expressions as shown in Figure 6.10. The *KRegExpEditor* use case was one of the first real use cases to be tested by *DoxyChange*. It hopefully offers a real test bench to study software evolution without taking an outstanding amount of time and hard disk space to be generated. Basically, this use case is a simplification of the *KDE Utils 3.x* case study, in an attempt to take one of the most interesting utility,

which changed the most compared to any of the other utilities included in the *KDE Utils 3.x* tar ball without being too noisy. The version being analyzed is 3.0 and 3.4.1, which are currently the first version and the latest version of the KDE 3.x series.

Some partial analysis was also performed using version 3.1 against version 3.0; however, the amount of changes was not really interesting, since the amount of changes were far from being significant. Some other partial analysis was also performed using version 3.1 against version 3.4.1; however, the changes showed were nearly the same than the 3.0 versus 3.4.1 analysis; therefore, highly redundant. It is worth mentioning that the internal and external API must normally be stable during a major KDE release. The reason behind this requirement is that applications must be binary compatible within each other inside a major KDE series. This is also known as the BIC or *binary incompatible changes* avoidance requirement as stated in [HAU02][ETT05].

Another advantage seen by this case study is that *DoxyChange* proved that visualizing changes at a UML level of abstractions summarizes a large amount of information into a tiny set of visual changes in most cases. In other words, it takes an enormous amount of effort to "corrupt" a UML diagrams between two revisions. The only exception to this rule is in the presence of well-known corner cases as mentioned previously in Chapter 1 and 6. Another way, would be to perform a complete deep change in the naming convention, let say from a lower case with underscore BSD style naming convention to a mixed case Java style naming convention or vice-versa.

In any way, this case study was performed against *KDE Utils 3.0* versus *KDE Utils 3.4.1*. The following Table 6.1 illustrates the amount of source code and source code files found in *KDE Utils 3.x*.

Table 6.1: KDE Utils – SLOC count

	SLOC	Comments	Blank Lines	Total	Files
KDE Utils 3.0	65,798	11,621	15,722	93,141	387
KDE Utils 3.4.1	100,826	23,381	23,881	148,088	699
Difference	+35,028	+11,760	+8,159	+54,947	+312

Table 6.2: KDE Utils – Run-Time Cost Analysis

KDE Utils	3.0	3.4.1	Difference
No HTML	417.22 s	844.14 s	+426.92 s
No HTML, DOT, MAP, PNG	228.32 s	481.38 s	+253.06 s
No HTML, DOT, MAP, PNG, MD5	223.31 s	500.07 s	+276.76 s
No HTML, DOT, MAP, PNG, MD5 & No dot resize	75.69 s	154.43 s	+78.74 s

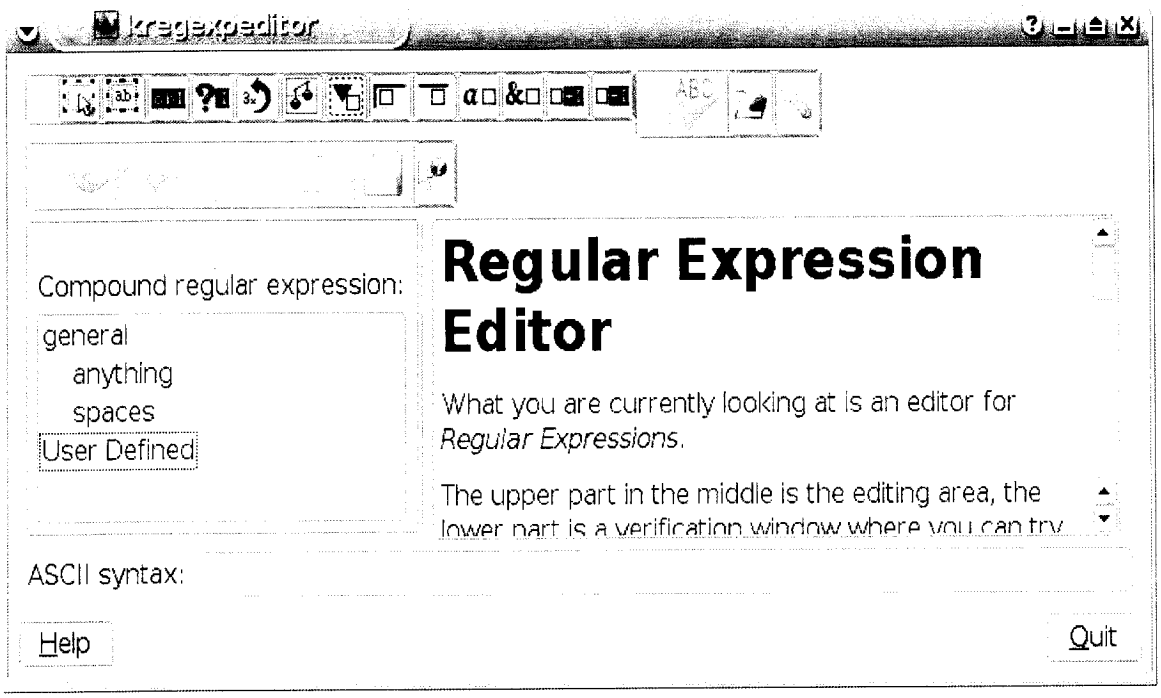

Figure 6.10: Screenshot: KRegExpEditor [MEY05r]

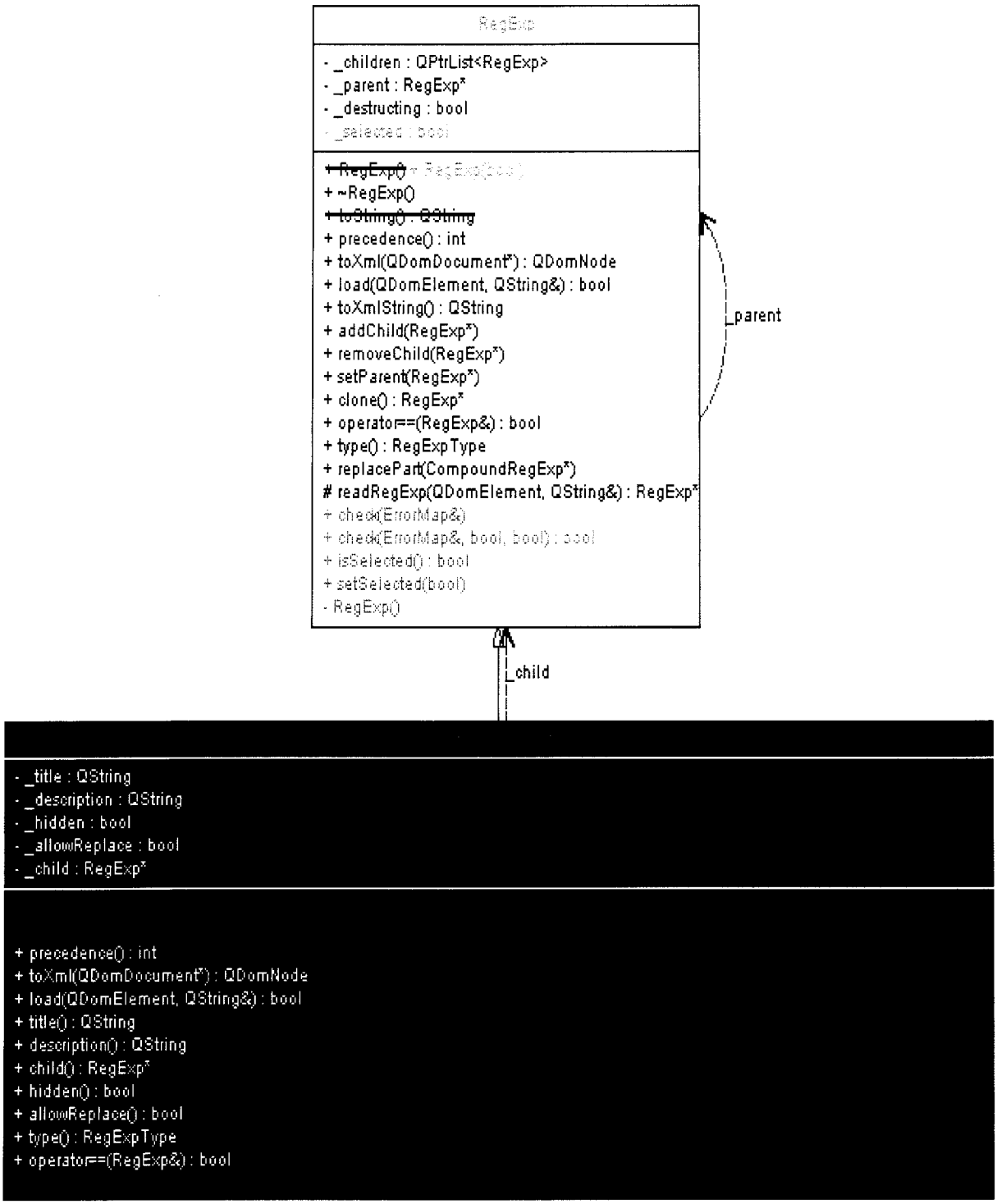

Figure 6.11: KRegExpEditor: classCompoundRegExp__coll__graph

In the above *classCompoundRegExp* collaboration diagram shown in Figure 6.11, in the *classTextRegExp* collaboration diagram below shown in Figure 6.12 and in the *classPositionRegExp* inheritance diagram below shown in Figure 6.13, for each of these diagrams, the class *RegExp* have now a new private Boolean attribute called *_selected.* Its public default constructor is now private and finally, *RegExp* has a new public constructor, which takes a Boolean parameter.

It is obvious from the diagram that *X-Diff* made the matching emphasis on keeping the same protection and appearance order, instead of enforcing the same parameters; therefore, *"+ RegExp()"* was matched with *"+ RegExp(bool)"* instead of being matched with *"- RegExp()"*, which is absolutely correct in this specific case as mentioned earlier. It has two newly added access functions: *isSelected()*, which returns a Boolean and *setSelected(bool)*, which take a Boolean parameter. The *"QString toString()"* function was removed from the parent and its children. The *"void check(ErrorMap&)"* overloaded public function was added to the parent and the *"bool check(ErrorMap&,bool,bool)"* public function was added to the parent and its corresponding children. Each children public constructor got a new Boolean prefix parameter, from the diagram it is probably the parameter sent to the new parent public constructor. Finally, *TextRegExp* no longer requires a *"QString escape(QString, QPtrList<QChar>, QChar)"* protected function, since it was removed.

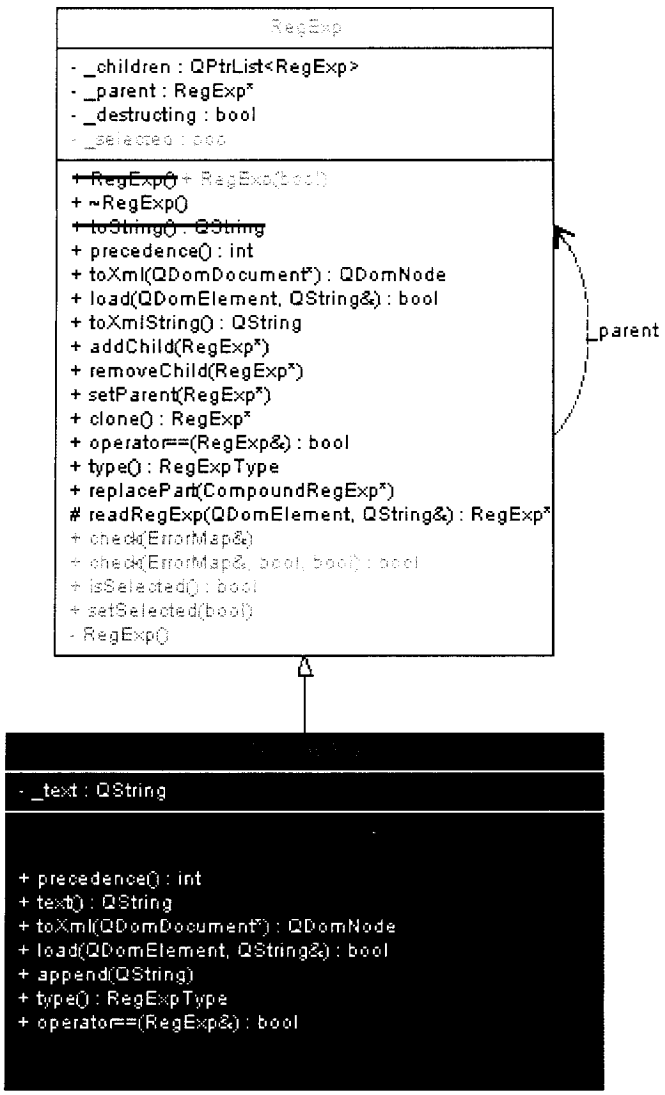

Figure 6.12: KRegExpEditor: classTextRegExp__coll__graph

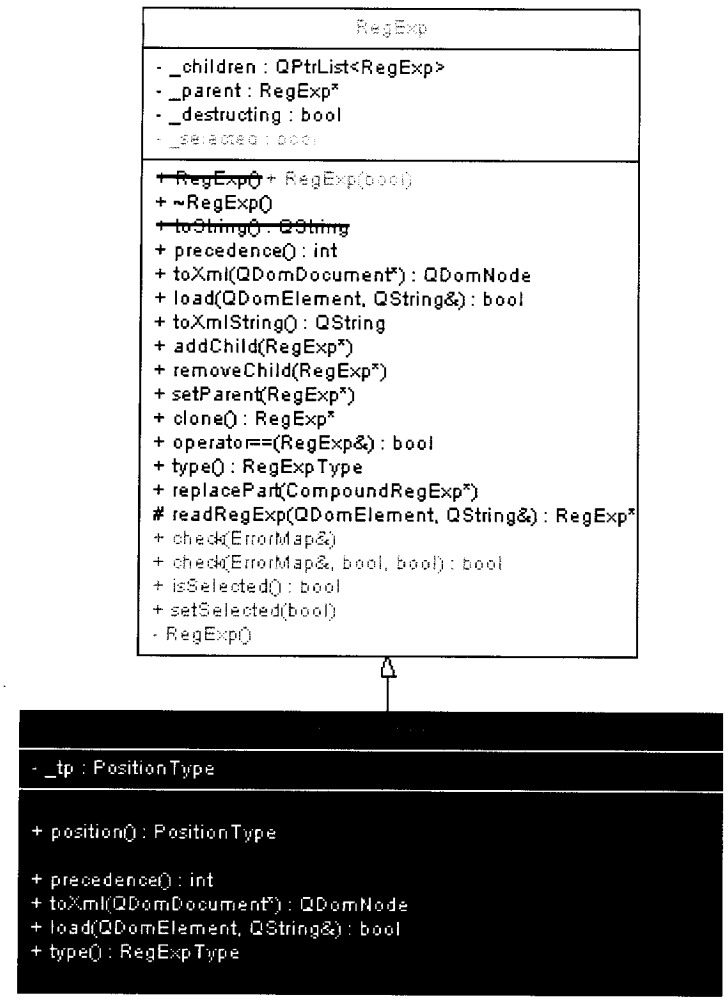

Figure 6.13: KRegExpEditor: classPositionRegExp__inherit__graph

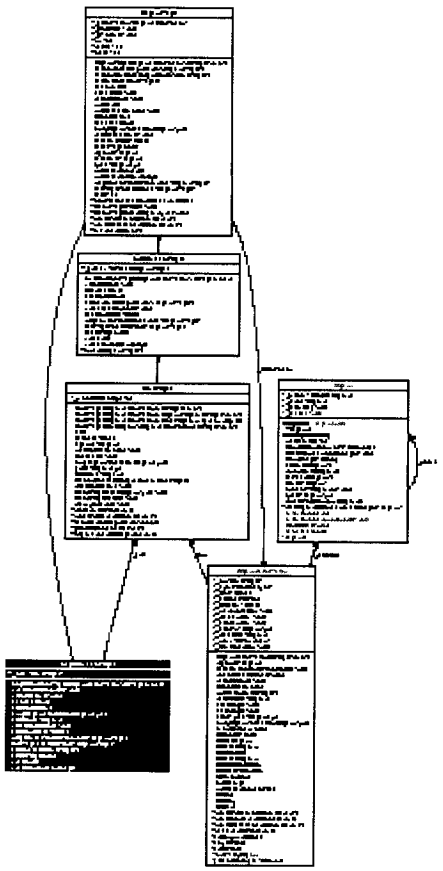

Figure 6.14: KRegExpEditor: classSingleContainerWidget__coll__graph - Part 1/3

The above *classSingleContainerWidget* collaboration diagram, shown in Figure 6.14, provides an overview of the changes being made, while the following diagrams, shown in Figure 6.15 and Figure 6.16, provide a closer view of the actual changes. The middle right class is *RegExp*, which was explained in detail earlier, while the bottom right class is *RegExpEditorWindow*. The class *RegExpEditorWindow* had two new feature being added: a protected slot *"virtual void emitVerifyRegExp()"* was added and a private function *"QIconSet getIcon(const QString&)"* was added as shown in the source code [PET05]. Notice that the currently focused class *SingleContainerWidget* did not change during the last few years.

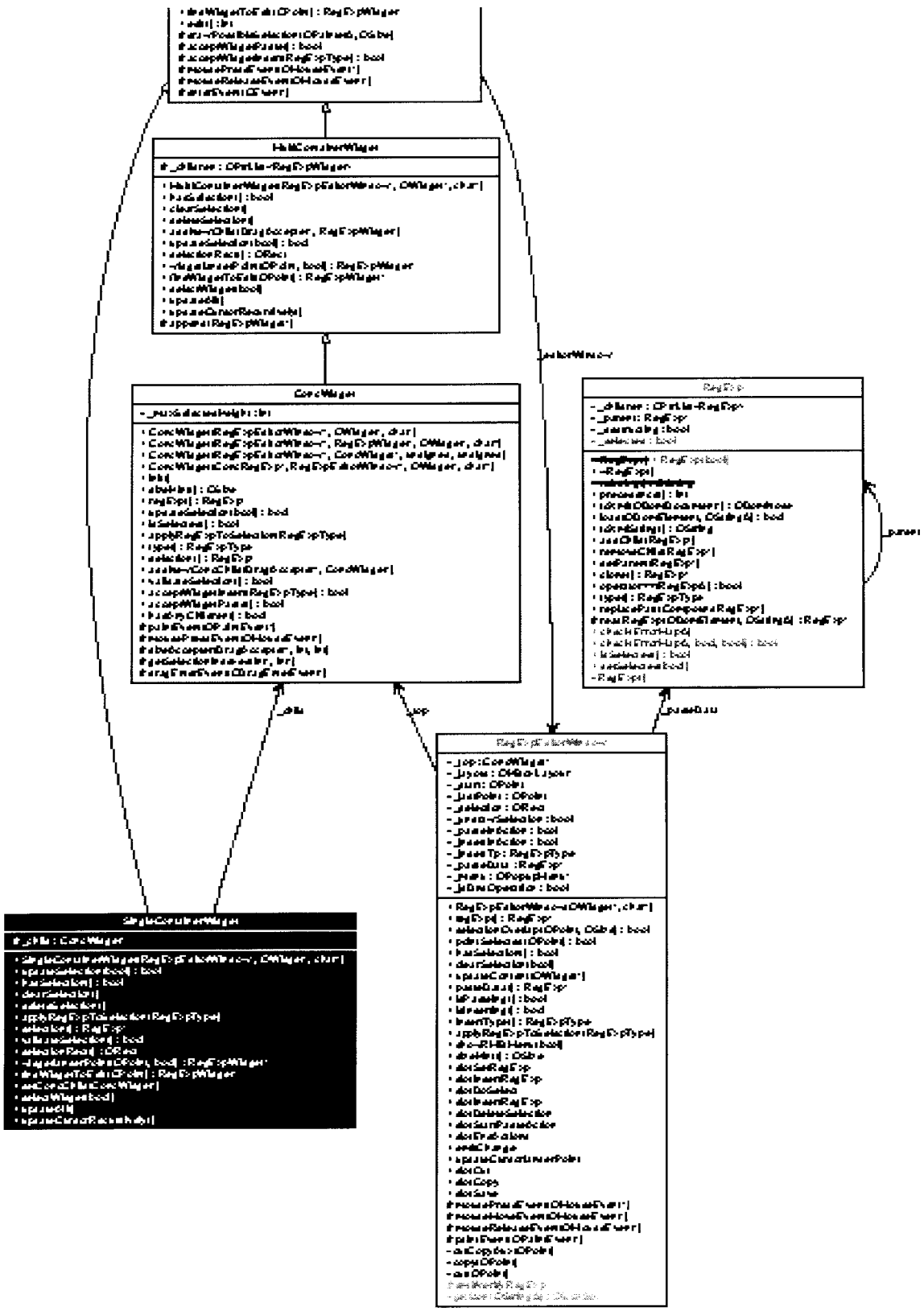

Figure 6.15: KRegExpEditor: classSingleContainerWidget__coll__graph - Part 2/3

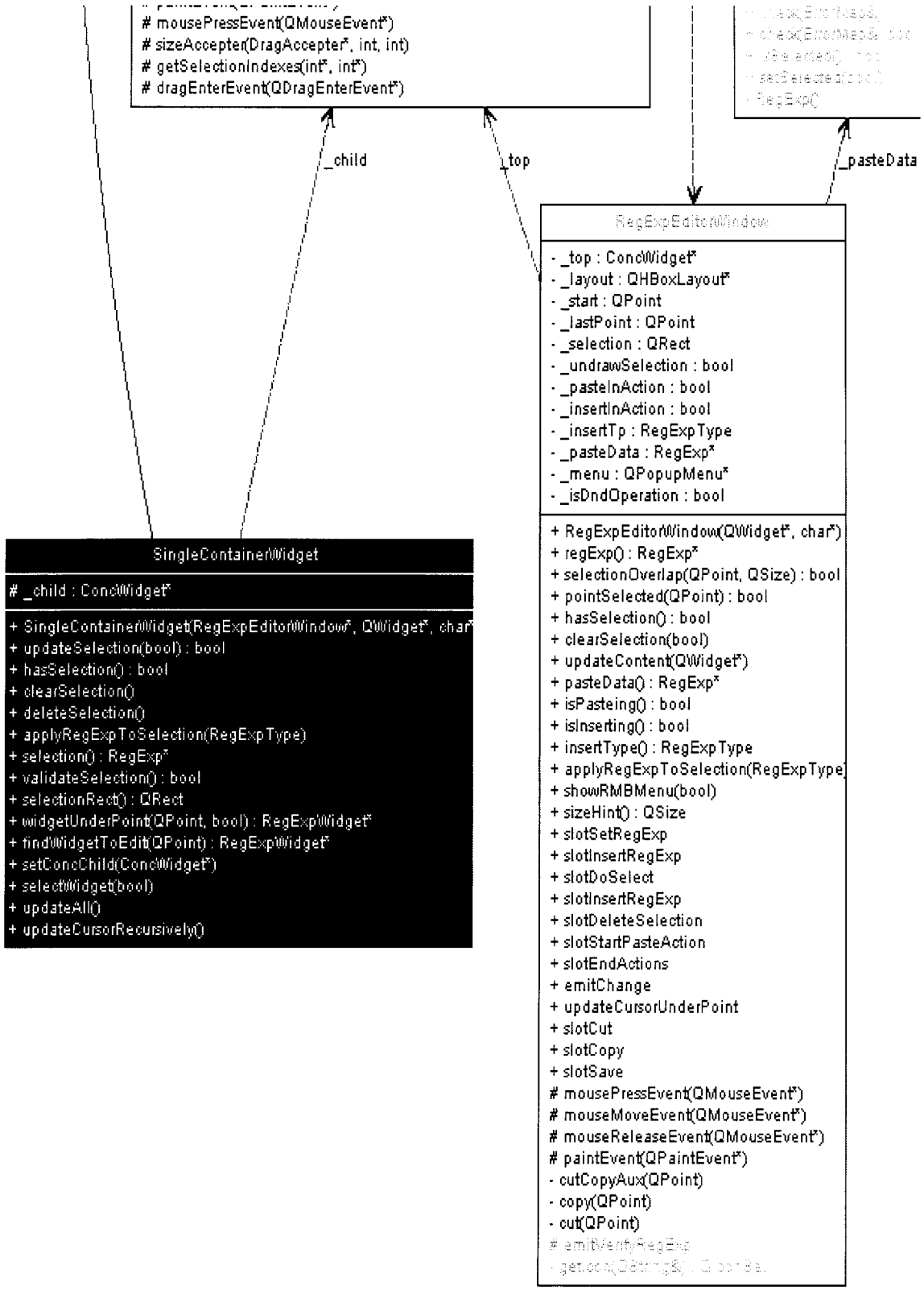

Figure 6.16: KRegExpEditor: classSingleContainerWidget__coll__graph - Part 3/3

Figure 6.17: KRegExpEditor: classSingleContainerWidget__inherit__graph

The above *classSingleContainerWidget* inheritance diagram, shown in Figure 6.17 and Figure 6.18, shows two classes being modified, which are *CompoundWidget* with a new protected function *"QPixmap getIcon(const QString&)"* and *LookAheadWidget* with a new public function *"RegExpWidget* findWidgetToEdit(QPoint)"*. Notice that *getIcon* returns a *QPixMap* instead of a *QIconSet* return type.

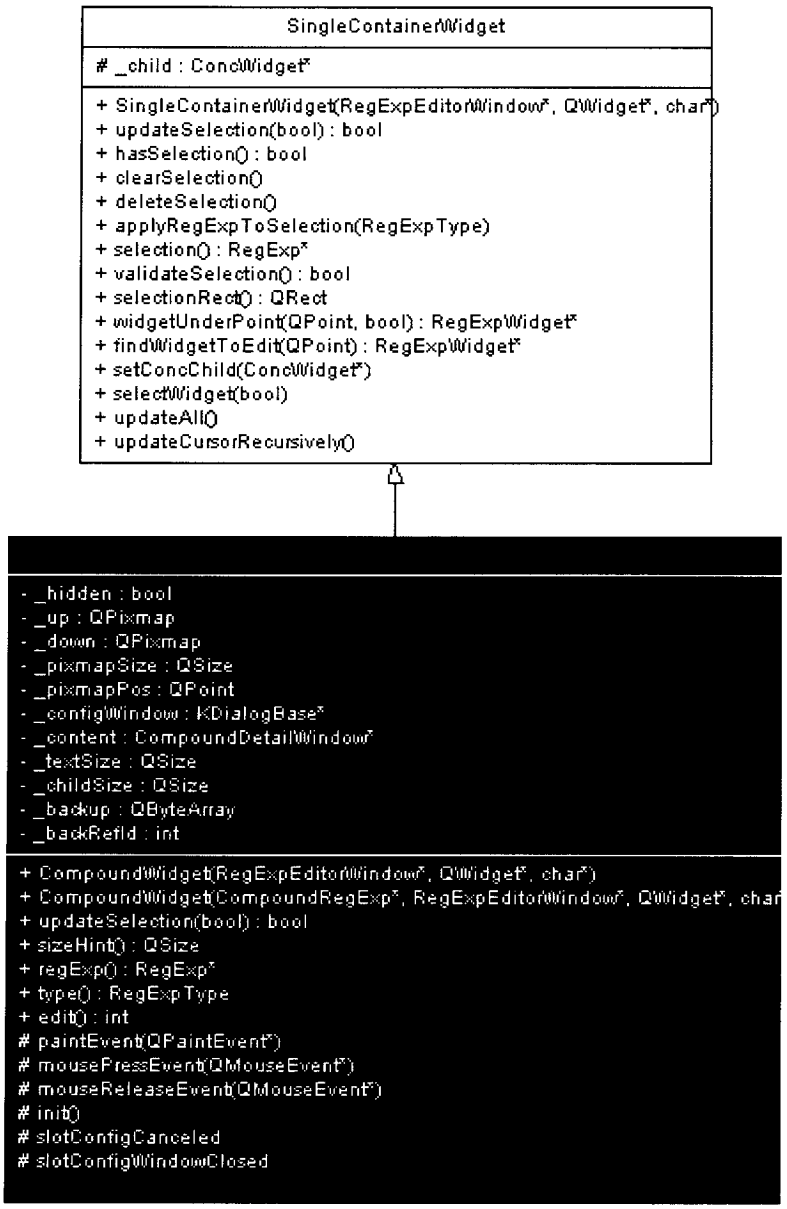

Figure 6.18: KRegExpEditor: classCompoundWidget__inherit__graph

6.2.2 KHexEdit Case Study

KHexEdit is a small KDE utility, which offers "a versatile and customizable hex editor. It displays data in hexadecimal, octal, binary and text mode", as shown below in Figure 6.19. "New input can be inserted into the file or it can replace current data just like in a common text editor. Undo/Redo functionality is included as well as formatted printing and HTML export. *KHexEdit* is designed to integrate into the KDE environment but can be used as a standalone program" [SAN99h].

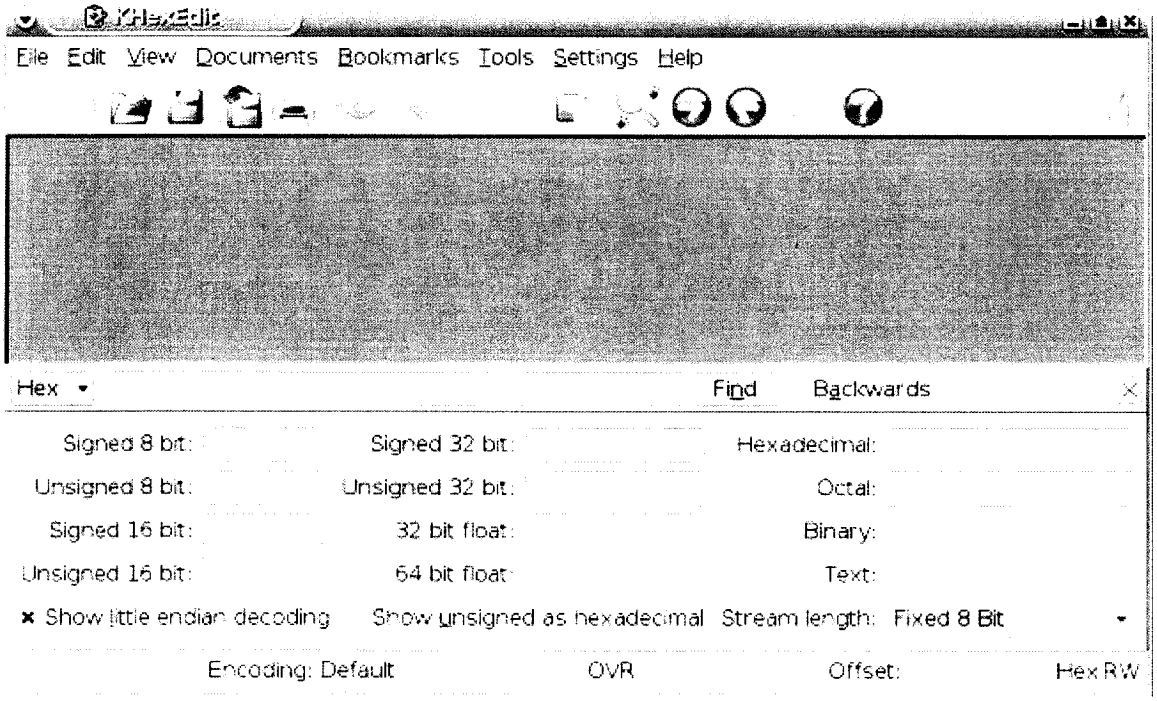

Figure 6.19: Screenshot: KHexEdit [MEY05h]

In the *classCStringDialog* collaboration diagram shown below in Figure 6.20, the class *KDialogBase* was added as a public inherited parent to *CStringDialog*. From this diagram, it is impossible to tell if *KDialogBase* is a new class in version 3.4.1 or if the class already existed in version 3.0. The only thing that this diagram says is that this class

is now a new parent for *CStringDialog* and that this class definition was not part of the diagram in version 3.0. A quick search in the source code confirms that the *KDialogBase* class was part of the KDE 3.0 distribution and was also used within the *KRegExpEditor* utility.

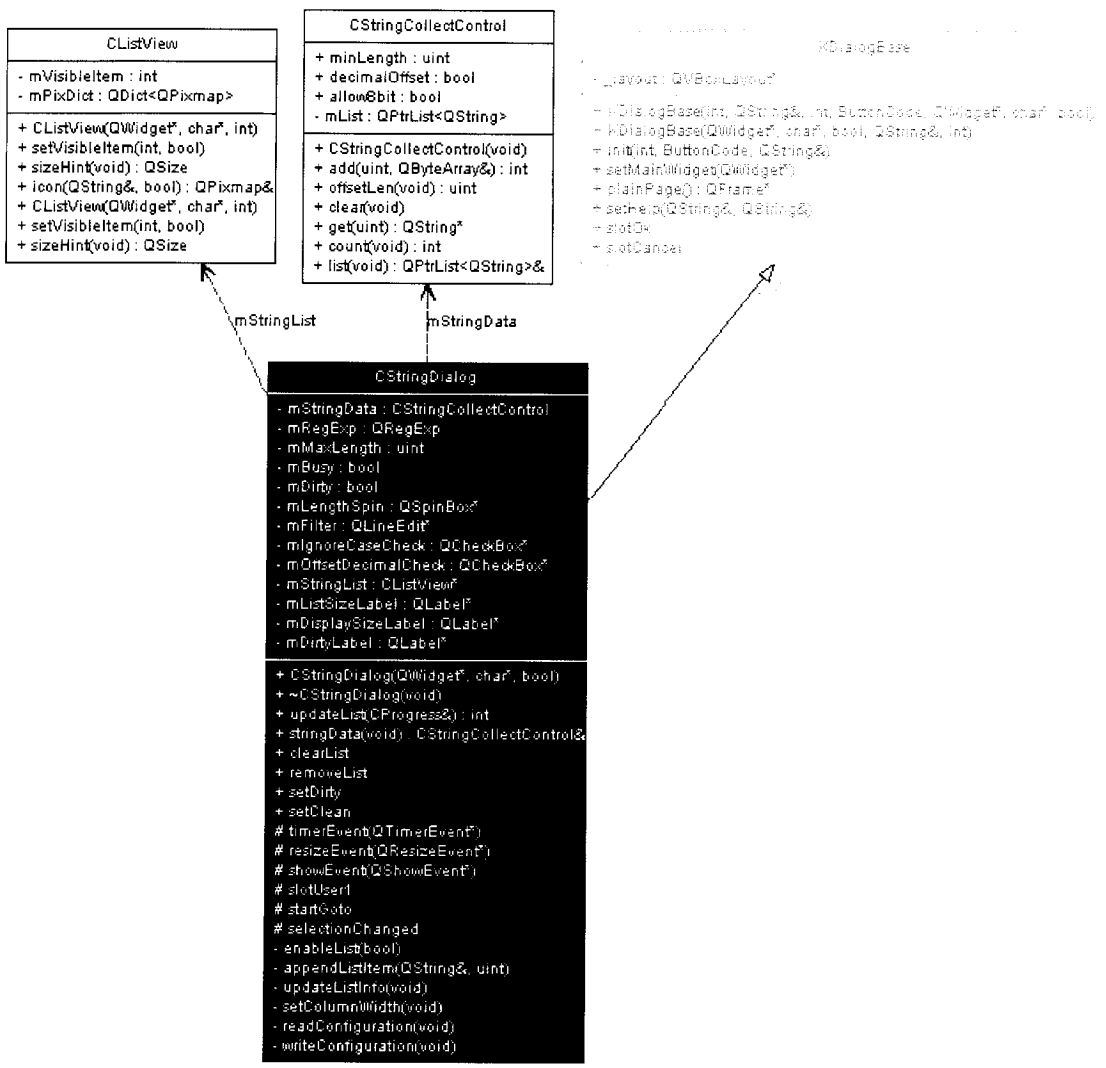

Figure 6.20: KHexEdit: classCStringDialog__coll__graph

In the *classWidgetWindow* collaboration diagram shown below in Figure 6.21, the class *KDialogBase* was added as a public inherited parent to *WidgetWindow*.

As stated above, from this diagram, it is impossible to tell if *KDialogBase* is a new class in version 3.4.1 or not.

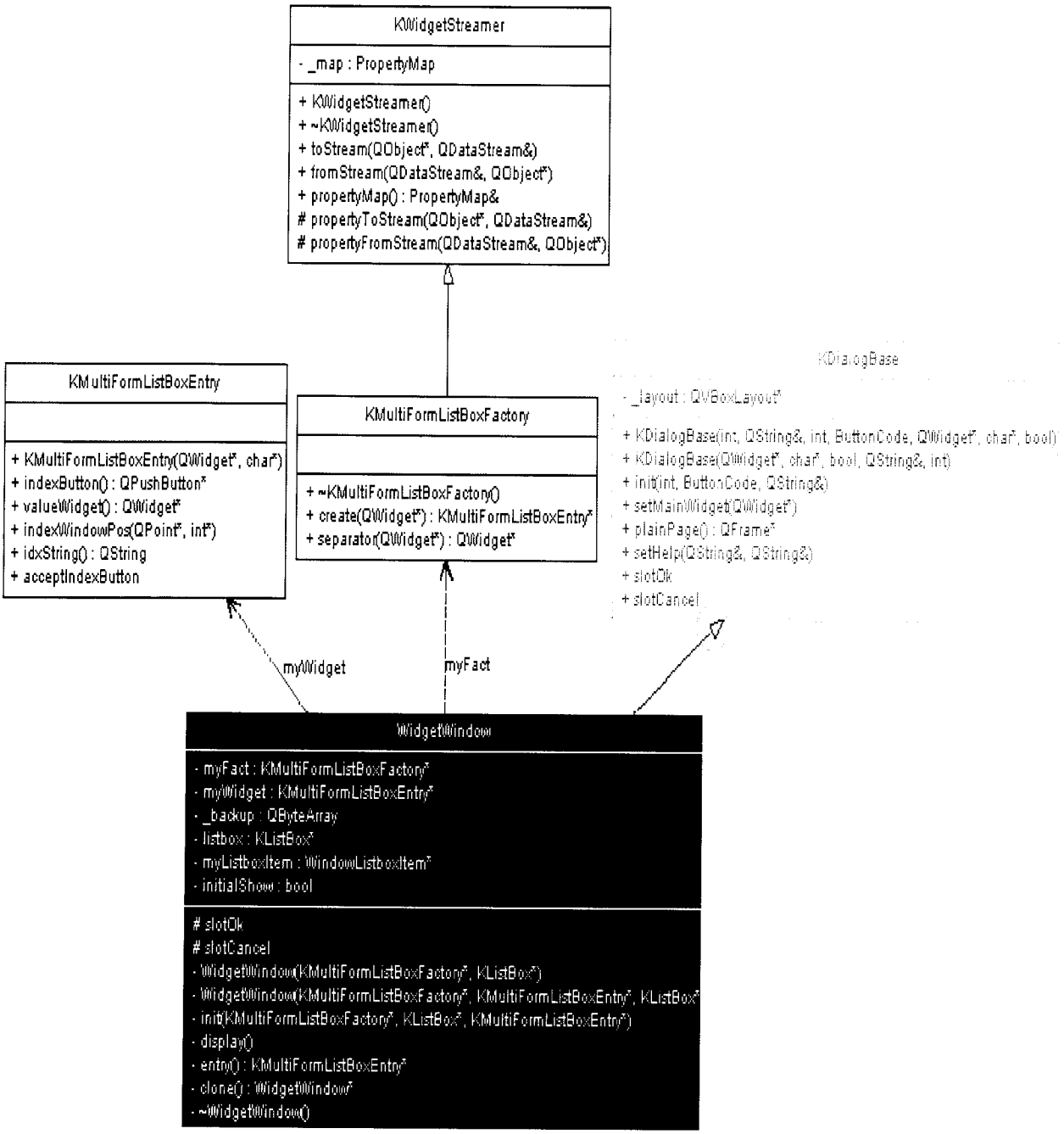

Figure 6.21: KHexEdit: classWidgetWindow__coll__graph

The following *classCHexEditorWidget* collaboration diagram shown below in Figure 6.22 is quite large; however, it is incomplete. This is shown by the red border box, which states that the relationships between this class and other classes are not shown, as explained in the *Doxygen Arrow and Box Legend* shown in Figure 5.5.

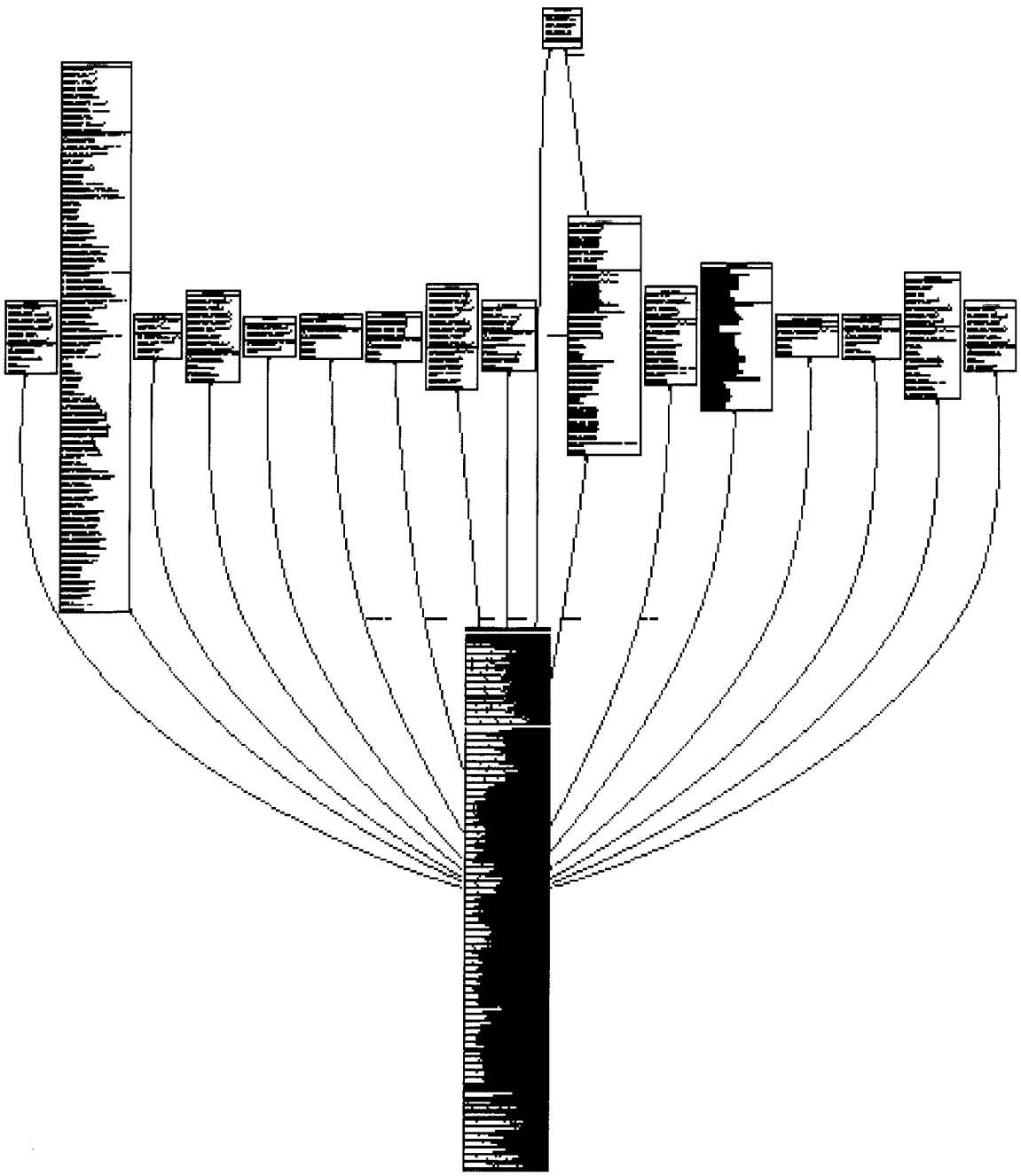

Figure 6.22: KHexEdit: classCHexEditorWidget__coll__graph - Part 1/5

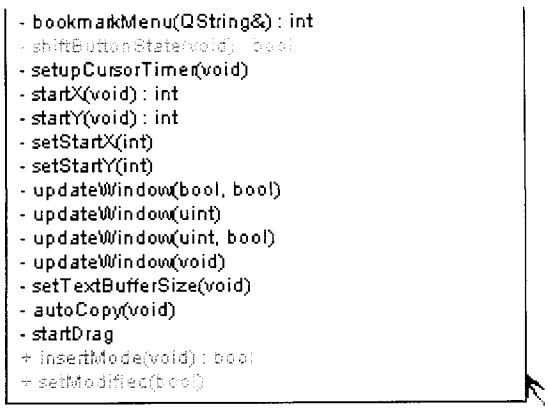

Figure 6.23: KHexEdit: classCHexEditorWidget__coll__graph - Part 2/5

The closer view shown above in Figure 6.23 presents changes which were made to the *CHexViewWidget* class, which is partially visible due to space constraints. Nevertheless, it clearly shows that the *"bool shiftButtonState(void)"* private function was changed and two new public functions *"bool insertMode(void)"* and *"void setModified(bool)"* were added. In this case, the change within the *"shiftButtonState"* function shows that the internal function implementation was changed between the two releases. The Figure 6.24 presents some of the changes that were performed within the *CFilterDialog* class. For instance, the private attribute *"QString mOperandString"* got changed from a *QString[4]* array type to a *QString[5]* array type. On the other hand, many changes occurred within the *COptionDialog* class. It no longer requires *SOptionState* class or *SSpellWidgets* class, since both private attribute *mState* and *mSpell* were removed. Two new private Boolean attributes were added: *dataChanged* and *configChanged.* Overloaded public functions based on state parameters were removed such as: *"void setFont(SFontState&)", "void setColor(SColorState&)", "void setColor(SColorState&)", "void setSpell(SSpellState&)", "void setMisc(SMiscState&)"* and *"void setState(SOptionState&)"*. Finally, a protected slot is added in the implementation file.

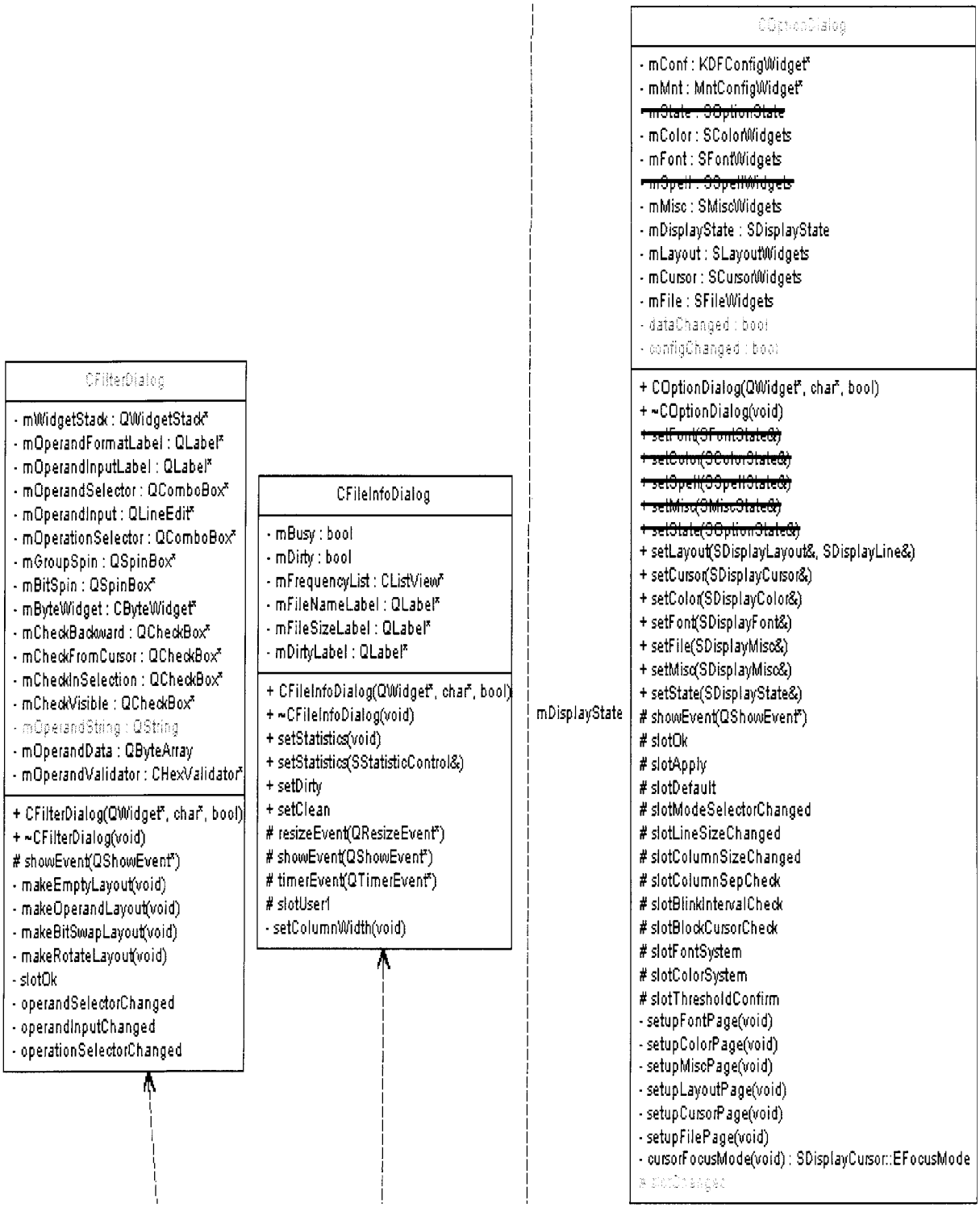

Figure 6.24: KHexEdit: classCHexEditorWidget__coll__graph - Part 3/5

Figure 6.25: KHexEdit: classCHexEditorWidget__coll__graph - Part 4/5

In Figure 6.25 and Figure 6.26, the *KIOslave* [PFE01] related private attributes were removed from the *CHexEditorWidget* class *"QPtrDict<QString> mKioRemote"*, *"QPtrDict<QString> mKioLocal"*, *"QPtrDict<QString> mKioFlag"* and its protected slots *kioReadFinished* and *kioWriteFinished.* Furthermore, the private attributes

"CPrinterDialog mPrintDialog"* and protected slot *printText* were also removed. The private function *readURL* now takes a *KURL* instead of a *QString*. The private function *writeFile* was changed and *createTemporaryFileName* was removed. Finally, a new public slot called *toggleInsertMode* was added.

Figure 6.26: KHexEdit: classCHexEditorWidget__coll__graph - Part 5/5

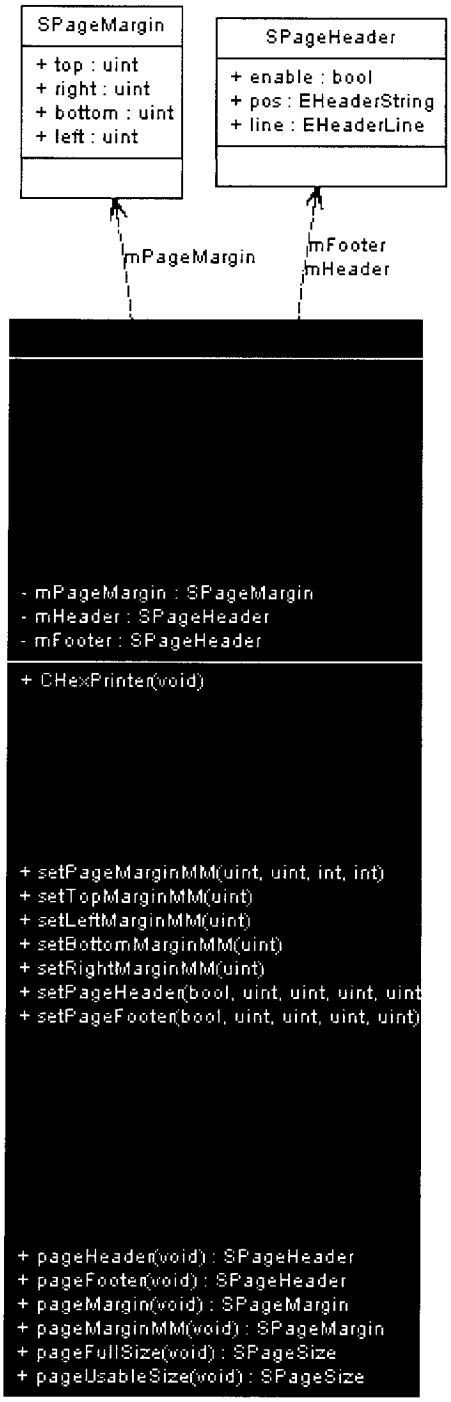

Figure 6.27: KHexEdit: classCHexPrinter__coll__graph

In Figure 6.27, the *classCHexPrinter* collaboration diagram had many private attributes removed such as: *"bool mAsTest", "bool mAll", "bool mInSelection", "bool mInRange",*

"bool mOutputToStdout", "bool mScaleToFit", "bool mPrintBlackWhite", "uint mStartOffset" and "uint mStopOffset". As a result the corresponding public access function were also removed: *"void setAsText(bool)", "void setAll(bool)", "void setSelection(bool)", "void setRange(bool,uint,uint)", "void setOutputToStdout(bool)", "void setScaleToFit(bool)", "void setPrintBlackWhite(bool)", "bool asText()", "bool all()", "bool selection()", "bool range()", "uint startRange()", "uint stopRange()", "bool outputToStdout()", "bool scaleToFit()"* and *"bool printBlackWhite()"*.

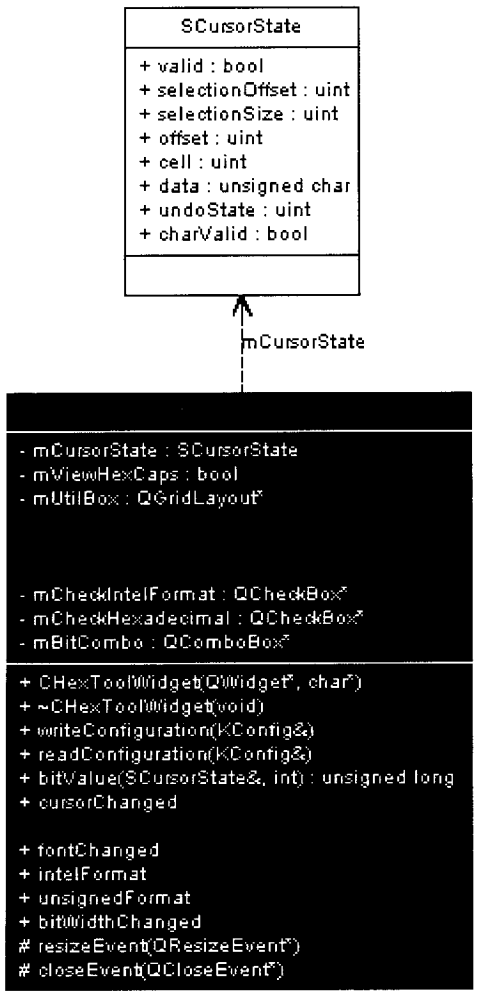

Figure 6.28: KHexEdit: classCHexToolWidget__coll__graph

In Figure 6.29, the *classCHexToolWidget* collaboration diagram, the class *CHexToolWidget* was modified as follow: its public slot *"void paletteChanged(void);"* was removed and its private attribute *mText1*, *mText2*, *mText3* types were changed from *QLabel** to *QLineEdit** as shown in the changes made to the CVS repository from 1999 [SAN99] to 2003 [SAN03].

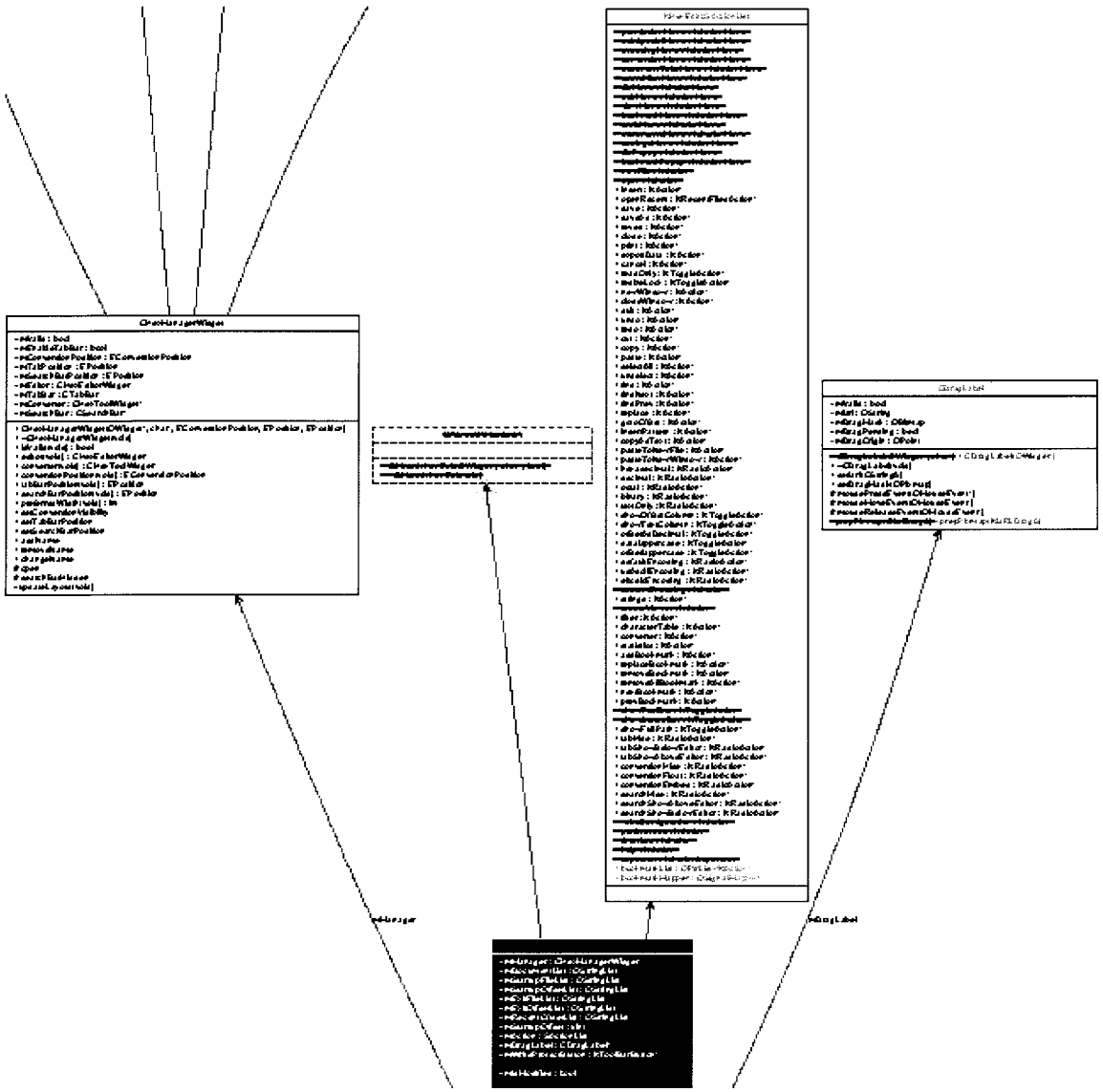

Figure 6.29: KHexEdit: classKHexEdit__coll__graph - Part 1/4

Futhermore, in Figure 6.29, the class *CAboutKHexEdit* and the associated private attribute *"CAboutKHexEdit mAboutKHexEdit"* of *KHexEdit* were removed from the

diagram. The private attributes *"bool mShowToolBar"* and *"bool mShowStatusBar"*, with their corresponding public function *showToolBar*, *showStatusBar*, *showAboutApplication* and *setStartupGeometry* were also removed from *KHexEdit*. Furthermore, *KHexEdit::SactionList* internal structure removed every *KActionMenu** public attribute, such as: *permissionMenu*, *editSpecialMenu*, *encodingMenu*, *conversionMenu*, *documentTabsMenu*, *searchBarMenu*, *fileMenu*, *editMenu*, *viewMenu*, *bookmarkMenu*, *toolsMenu*, *documentsMenu*, *settingsMenu*, *filePopup* and *bookmarkPopup*. Also, the *KAction** *newFile*, *open*, *customEncoding*, *recordViewer*, *writeConfiguration*, *preferences*, *favorites* and *help* were removed. Finally, *"KToggleAction showToolBar"*, *"KToggleAction showStatusBar"* and *"KActionSeparator separator"* were removed, while *"QPtrList<KAction> bookmarkList"* and *"QSignalMaper* bookmarkMapper"* public attributes were added. From this diagram, two things are obvious: the program got heavily refactored and where these features are now implemented. After some background research, the answer to this question is simple. *KDE Libs 3.x* in the middle of its life cycle enhanced its API to allow common functionalities to be shared under the common library umbrella to reduce the maintenance overhead and code duplication. As a result, all those standard actions and menus are now defined in the *KHexEdit* constructor using the new *"KStdAction::*"* library functions. This allows KDE to standardize widely used user interactions and therefore to easily obtain a uniform and standard behavior within every KDE application and also to simplify its code base. All this can easily be observed from these simple diagrams. Moreover, any KDE developers can rapidly know if a KDE application was ported to the new *KStdAction* API by simply comparing the original version with the latest version. On the other hand, the

class *CDragLabel* also got refactored, by removing the string parameter from its constructor and changing its *prepPixmap* private function parameter from *QUriDrag* to *KURLDrag*, in order to use the standardized KDE API instead of the default Qt API.

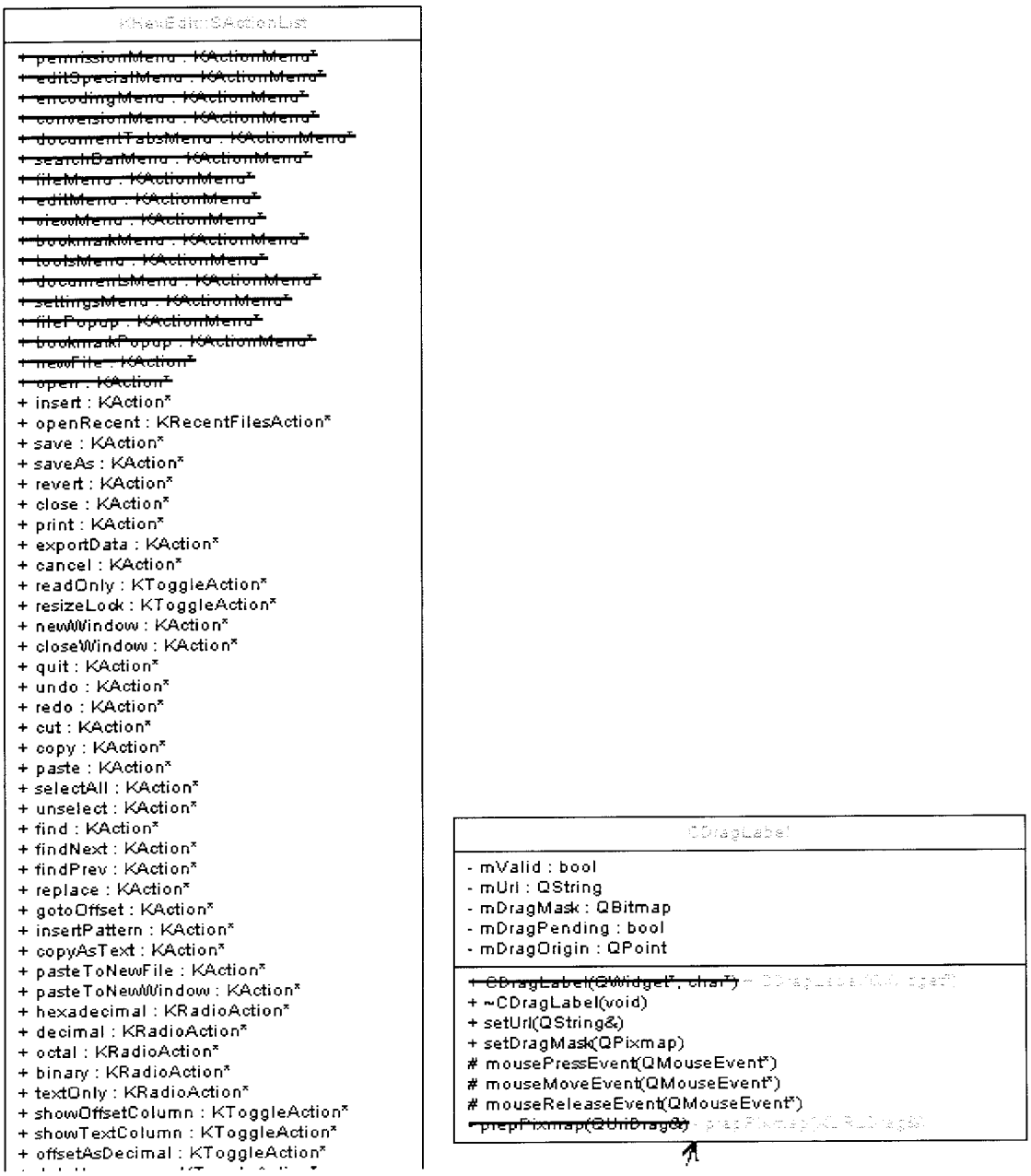

Figure 6.30: KHexEdit: classKHexEdit__coll__graph - Part 2/4

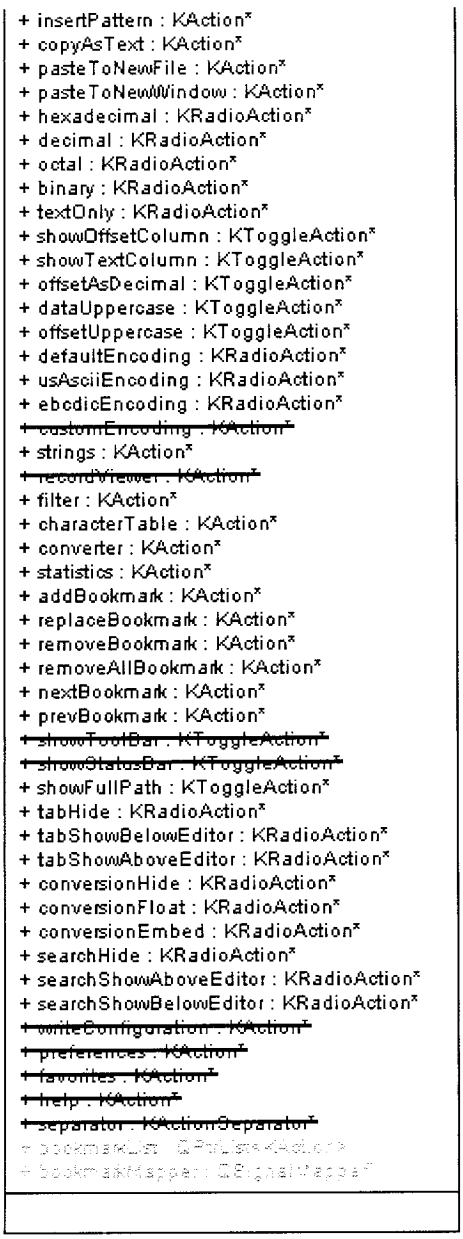

Figure 6.31: KHexEdit: classKHexEdit__coll__graph - Part 3/4

Figure 6.32: KHexEdit: classKHexEdit__coll__graph - Part 4/4

6.2.3 KDiskFree Case Study

KDiskFree is a small disk free utility for KDE, which shows mount points and its related disk usage using a convivial user interface as shown below in the Figure 6.33.

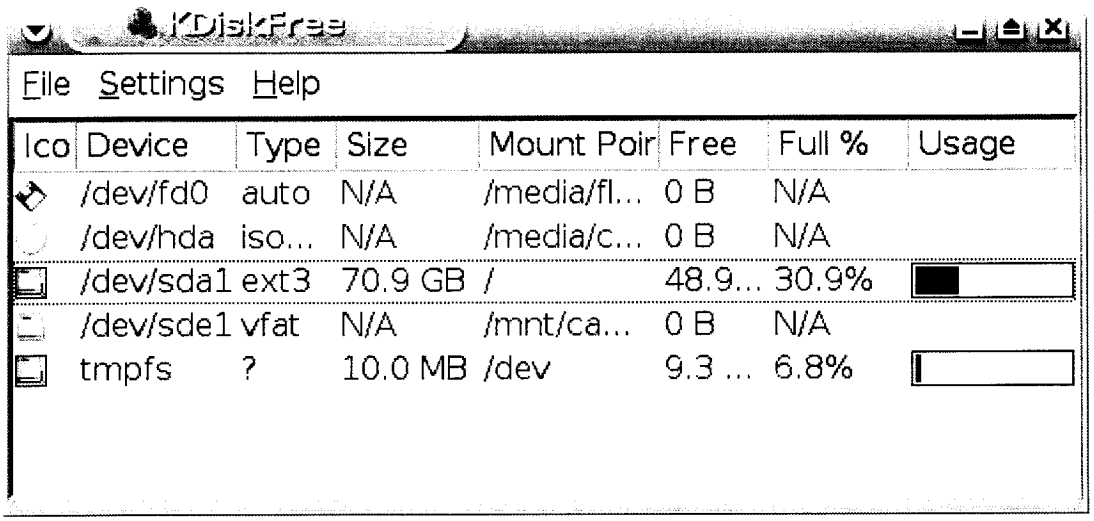

Figure 6.33: Screenshot: KDiskFree [MEY05d]

The only outstanding change within the *DiskList* class, as illustrated below in Figure 6.34, is a new private attribute called *"bool updatesDisabled"* was added with its corresponding public function called *"void setUpdatesDisabled(bool)"* and a new public function *"void deleteAllMountedAt(const QString&)"* were also added. The former feature is to disable continuous updates to reduce CPU and Battery usage, while the other feature is to delete any tracking of disk usage for disk mounted with the specified mounting point name. This feature might be useful in a networked environment for shared drives with nearly unlimited disk space. For instance, the primary disk storage area used on ECE or CS file servers.

Figure 6.34: KDiskFree: classDiskList__coll__graph

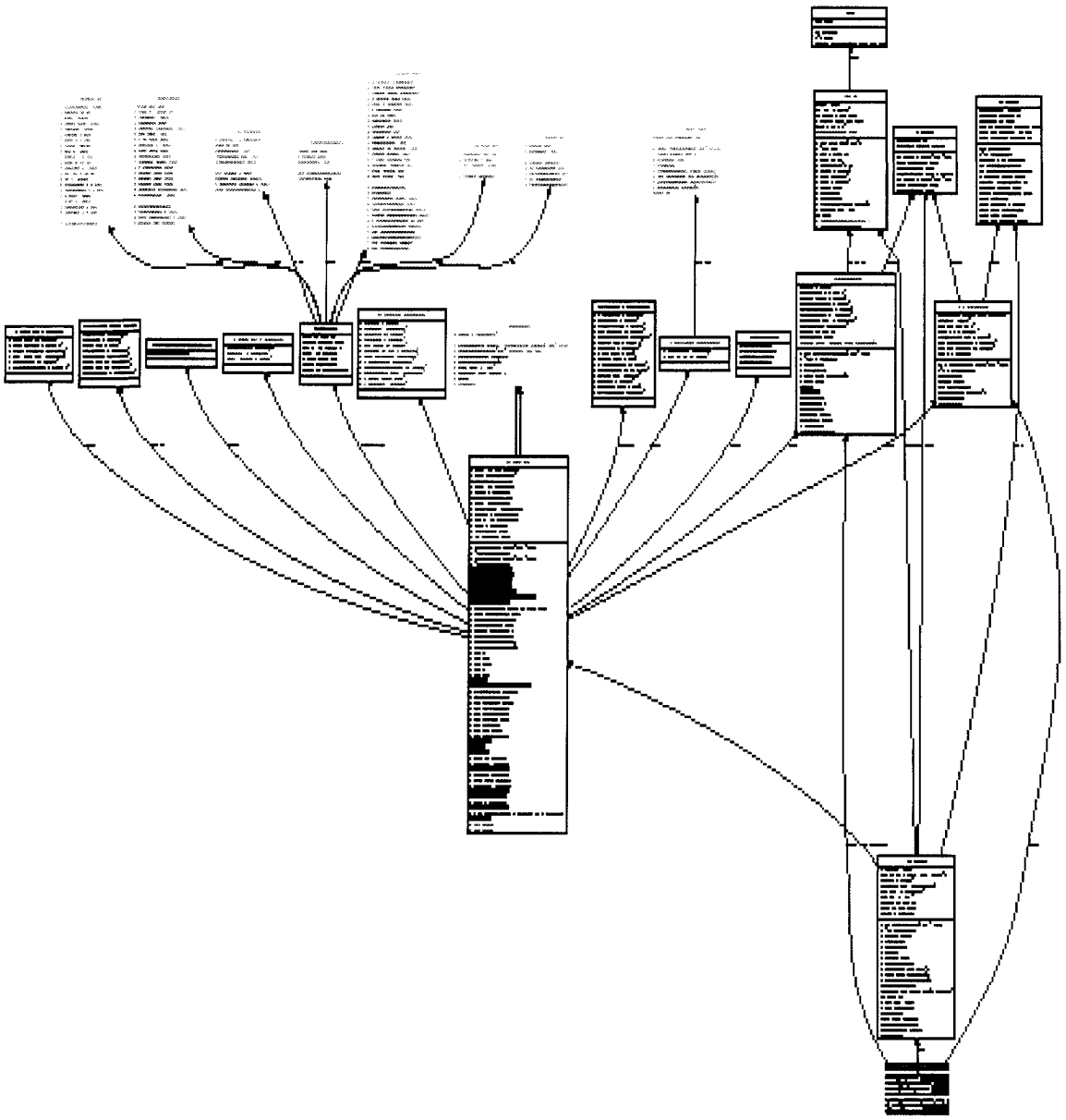

Figure 6.35: KDiskFree: classKDiskFreeWidget__coll__graph

The *classKDiskFreeWidget* collaboration diagram shown above in Figure 6.38 provides a nice overview of every outstanding change that occurred within the *KDiskFree* application. More than twelve new class associations were added, two classes associations were removed and at least six other classes were modified without going into further details.

6.2.4 KTimer Case Study

KTimer, shown in Figure 6.36, "allows you to execute commands after a certain amount of time. It allows for looping commands as well as delaying the execution of a command" [NGU04]. *KTimer* application is quite stable; however, in the *KTimerPref* collaboration diagram shown in Figure 6.37, the protected slot *"void browse();"* was removed from the class API has illustrated by the CVS changes from 2001 [SCH01] to 2003 [SCH03].

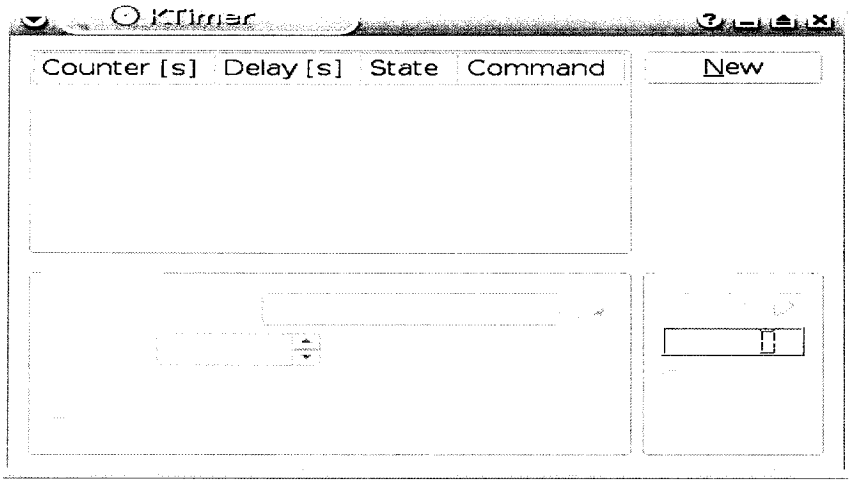

Figure 6.36: Screenshot: KTimer [MEY05t]

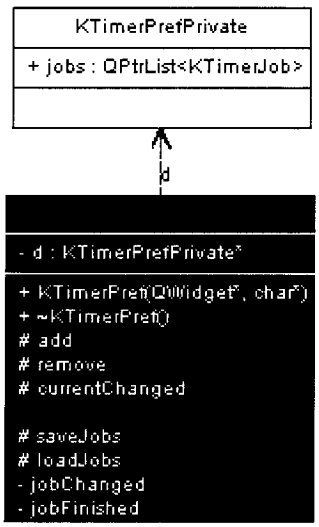

Figure 6.37: KTimer: classKTimerPref__coll__graph

6.2.5 Ark Case Study

Ark is a small KDE utility for quickly managing and extracting archives. It supports a wide range of archive formats such as: *zip*, *tar*, *tar.gz*, *tgz*, *tar.bz2*, *tbz*, *tzo*, *tar.Z*, *rar*, *zoo*, *lha*, *ar*, among others. It also provides a friendly user interface similar to *WinZip* [WZI05], as shown below in Figure 6.38 and Figure 6.39.

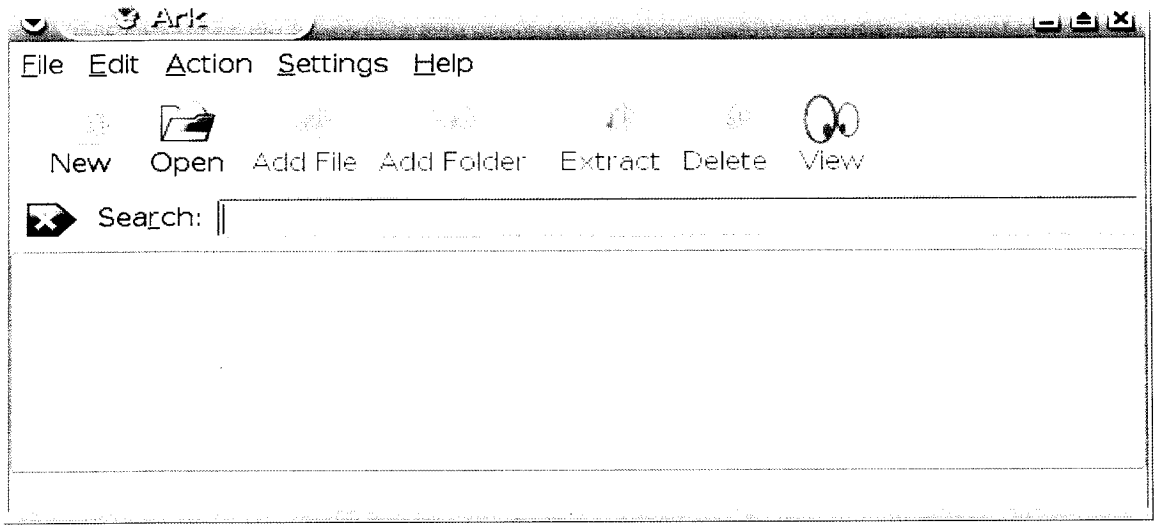

Figure 6.38: Screenshot: Ark [MEY05a]

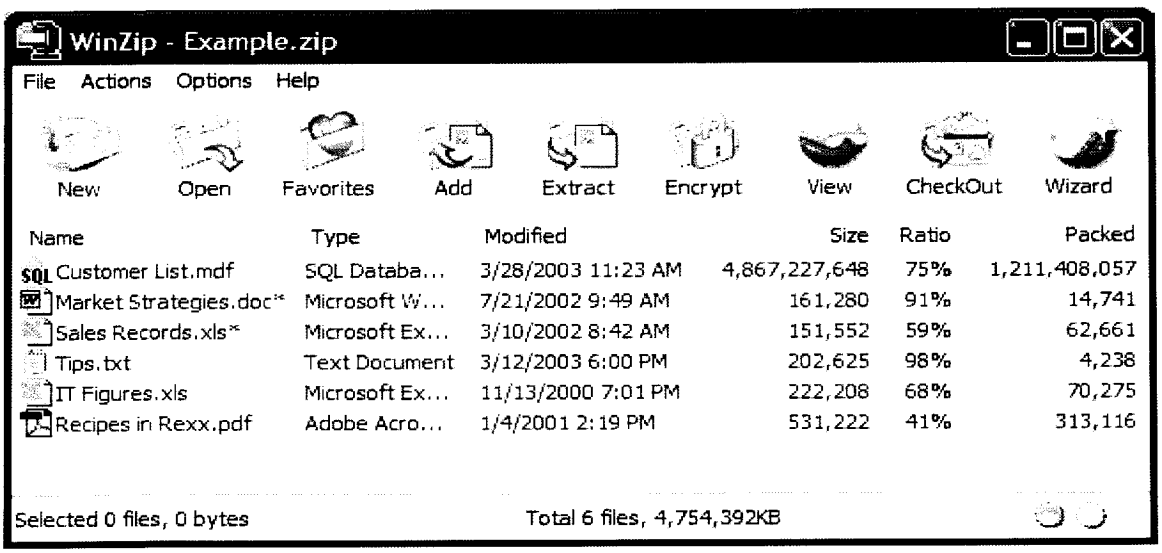

Figure 6.39: Screenshot: Winzip [WZI05]

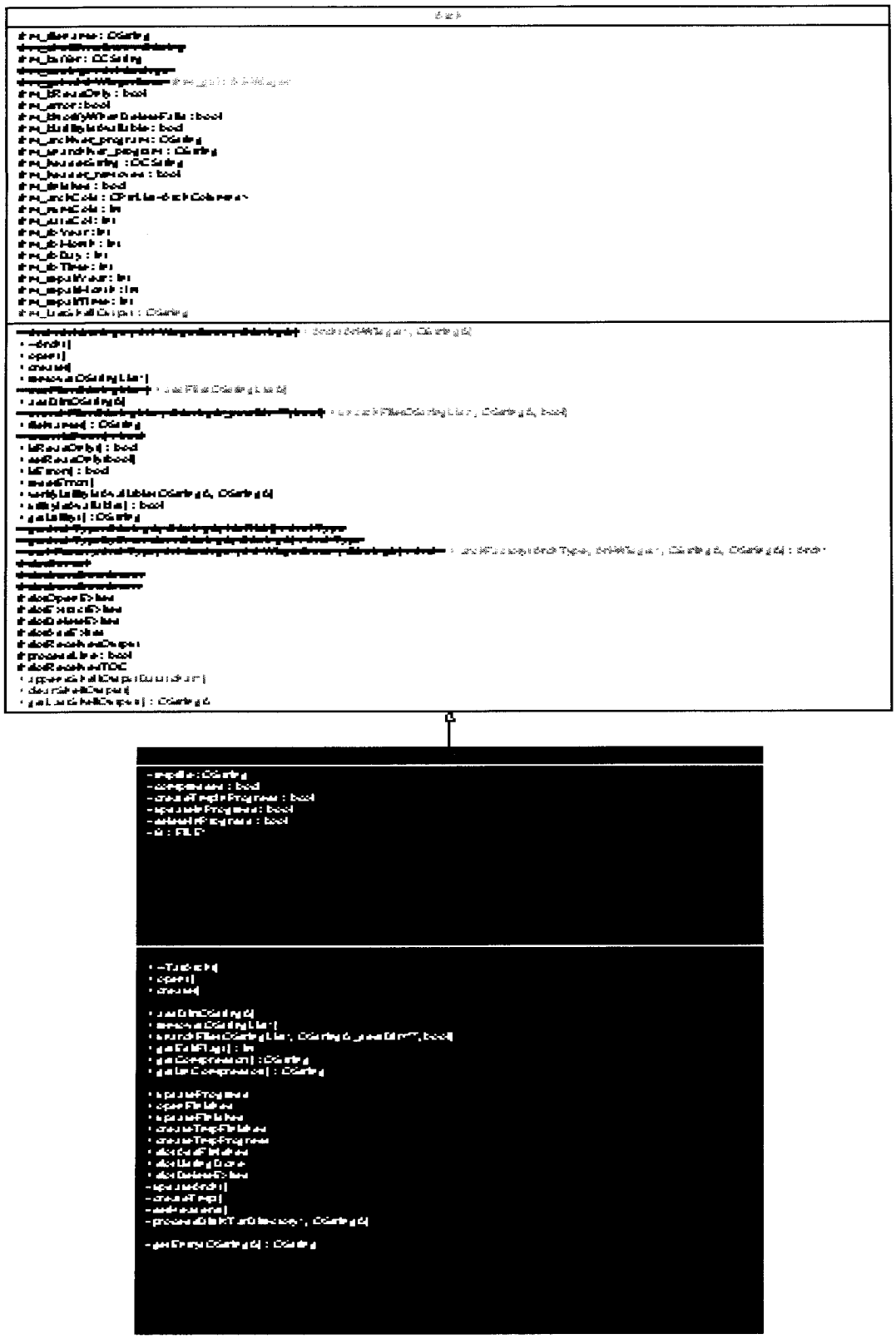

Figure 6.40: Ark: classTarArch__inherit__graph - Part 1/3

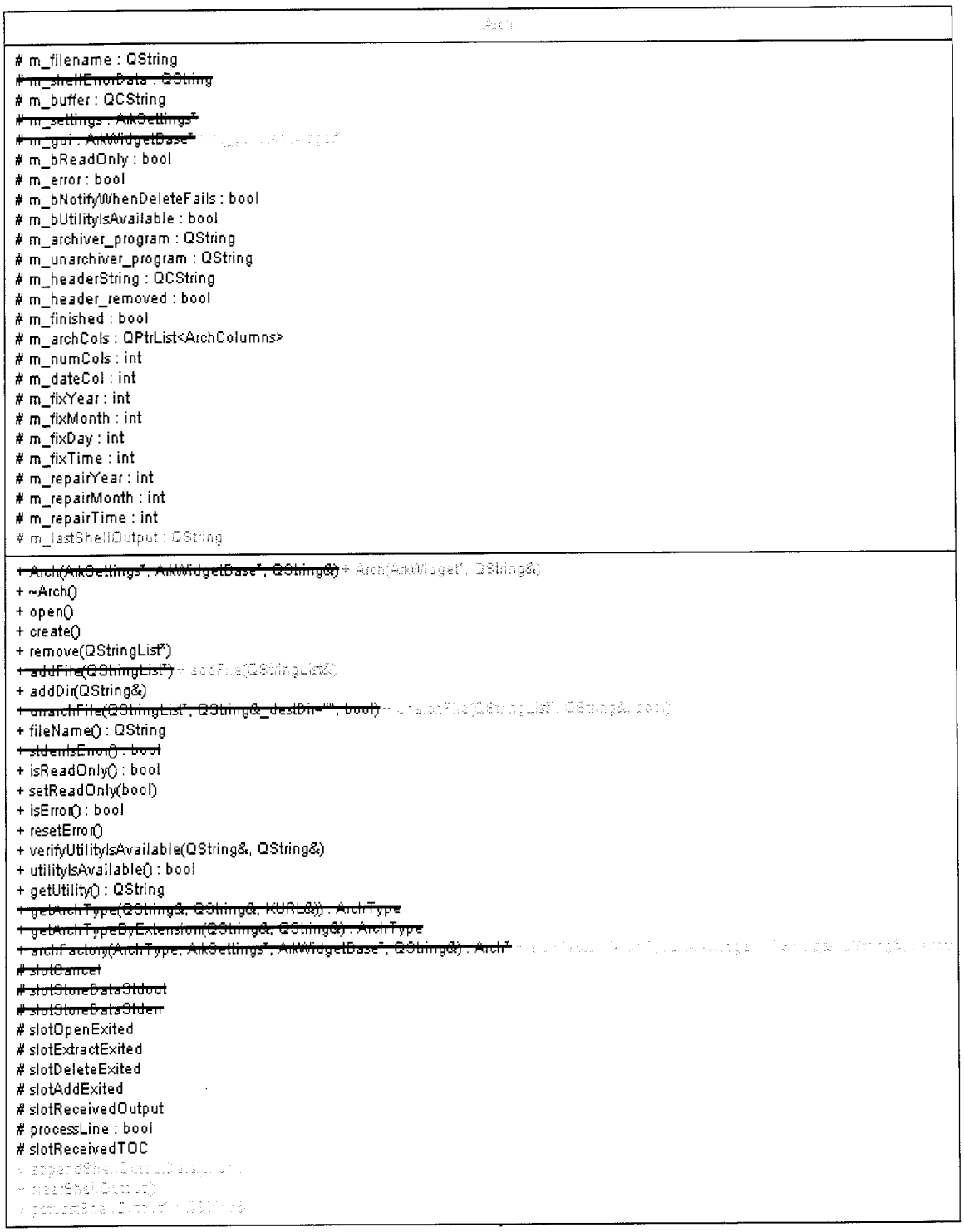

Figure 6.41: Ark: classTarArch__inherit__graph - Part 2/3

In the *classTarArch* inheritance diagram shown in Figure 6.40 and Figure 6.41, many changes were performed to the *Arch* class, which stands for archives. The *"QString m_shellErrorData"*, *"ArkSettings* m_settings"* protected attributes were deleted, *"QString m_lastShellOutput"* was added and the *"m_gui"* type was changed from *ArkWidgetBase** to *ArkWidget**. The public constructor lost its first *ArkSettings* argument, *addFile* now takes a reference instead of a pointer, *unarchFile* lost its unused default parameter. The public functions *"bool stderrIsError();"*, *"static ArchType getArchType(const QString&, QString&, const KURL&);"*, *"static ArchType getArchTypeByExtension(const QString&, QString&);"* and *"static Arch* archFactory(ArchType, ArkSettings*, ArkWidgetBase*, const QString&);"* were removed. The protected slots *"void slotCancel();"*, *"void slotStoreDataStdout(KProcess*, char*, int);"* and *"void slotStoreDataStderr(KProcess*, char*, int);"* were also removed. Finally, some public functions were added such as: *"void appendShellOutputData(const char*)"*, *"void clearShellOutput()"* and *"const QString& getLastShellOutput() const"*.

In the *classTarArch* inheritance diagram shown in Figure 6.40 and Figure 6.42, the *TarArch* class, which publicly inherits from the parent *Arch* class and which handles the tar file format and its derivative, also changed significantly. Like the *Arch* class, it lost its *ArkSettings** constructor parameter and *addFile* now takes a reference instead of a pointer; however, *unarchFile* still have an unused default parameter. The private function *deleteOldFiles* also takes a reference instead of a pointer for *QstringList* and the public function *"QString getUnCompressorByExtension()"* was removed. On the other side, many new features were added.

It now has the following added private attributes to handle temporary decompressed tar files:

*"KTempDir * m_tmpDir;",*

"QString m_fileMimeType;",

"QStringList m_filesToAdd;",

"QStringList m_filesToRemove;",

"KProcess m_pTmpProc;",*

"KProcess m_pTmpProc2;",*

"KTar tarptr;",*

"bool failed;",

"bool m_dotslash;".

Finally, new private slot functions were added such as:

"void openFirstCreateTempDone();",

"void openSecondCreateTempDone();",

"void deleteOldFilesDone();",

"void addFileCreateTempDone();",

"void addFinishedUpdateDone();",

"void removeCreateTempDone();"

and *"void removeUpdateDone();".*

Figure 6.42: Ark: classTarArch__inherit__graph - Part 3/3

In summary, these are the major changes that occurred during the *KDE Utils 3.x* series, including some of the refactoring and new features that were introduced during this stable BIC release cycle. Notice, that most features that were changed are either private, protected, slots or virtual functions to ensure that no outstanding BIC changes occur between the various releases within shared classes. Application specific classes do not have this restriction, if they are not shared or externally used.

6.3 KDE Libs Case Study

This fourth case study is inherently complex. It includes every core library component, which forms KDE. Every item in the core KDE library are inherently inter-coupled and also weakly coupled to any external KDE programs, obviously by design. The major problematic with the KDE library is its size. The KDE library is inherently large, very large. Due to the large amount of classes and files that are included in this test case only the important test generation data are provided for comparison purposes. The goal behind those test generations is to measure accurately extreme test cases, which *DoxyChange* might face in a production environment. It also provides a better idea, where *DoxyChange* might not be suitable and some possible solutions are therefore proposed to work around these well-known issues as described in Chapter 1. The following table provides an estimation of the *KDE Libs* size for comparison purposes. By comparing Table 6.1 with Table 6.3, *KDE Libs* is roughly six times larger than *KDE Utils* in terms of line of source code and nearly five to seven times larger in terms of the amount of source files. The amount of changes is also six times larger even though these changes were performed within a stable release cycle.

Table 6.3: KDE Libs – SLOC count

	SLOC	Comments	Blank Lines	Total	Files
KDE Libs 3.0	388,639	130,884	86,305	605,828	2,196
KDE Libs 3.4.2	590,032	206,714	129,022	925,768	2,988
Difference	+201,393	+75,830	+42,717	+319,940	+792

Table 6.4: KDE Libs 3.0 versus 3.4.2 input files

	All files	.h files	.c files	.cpp files	.cc files	Source files
KDE Libs 3.0	6,543	1,096	57	965	78	2,196 **files**
1385 **Directories**	33,237,900	5,201,724	812,394	10,175,368	1,018,229	17,207,715 **bytes**
KDE Libs 3.4.2	8,633	1,482	78	1,345	83	2,988 **files**
1115 **Directories**	60,790,558	8,132,022	940,522	16,825,067	1,229,074	27,126,685 **bytes**
Difference	2,090	386	21	380	5	792 **files**
-270 **Directories**	27,552,658	2,930,298	128128	6649699	210,845	9,918,970 **bytes**

This large number of changes was the result of a major refactoring that took place in the middle of the KDE 3.x life cycle. The refactoring introduced the usage of *KStdAction* and similar *KDE Libs* new core features that unified and standardized the KDE API for the great benefit of every KDE developers and maintainers. Overall, it took several hours, if not days to generate the data. As a result, an extremely large amount of computing cycles and of hard disk space is required to simply generate the *Doxygen* output for this specific case study. Some further analysis showed that the KDE team performs *Doxygenation* of their documentation in very small increments, folder by folder, by reparsing the dependency as needed. This might explain why *Doxygen* was used extensively for each revision, during the entire KDE 3.x life cycle. *Doxygen* is also included in the default *Makefile.am* part of the KDE build system, which is automatically generated by the *KDevelop* default project template.

Since KDE provides on its web site, the *Doxygen* HTML documentation and since running one third of *Doxygen* over *KDE Libs 3.0* took more than 36 hours on a standard Dell CS computers running anti-virus software, some optimization had to be performed to remove any unnecessary I/O overhead from the default *Doxygen* output. This optimization step was crucial to create a test generation in reasonable amount of

time, which is nearly 5 hours per *KDE Libs* release, averaging 11 hours total for every phase, if it is performed in a sequential manner.

The load charge, memory, CPU utilization and similar metrics are provided in the following Table 6.5, Table 6.6 and Table 6.7, including some averages about the amount of time spent in each phase and the amount of output I/O required using a minimalist configuration. These software metrics were taken using the *Doxygen* generation, which did not output any PNG, MAP, MD5 files, neither any computed DOT or resized DOT files. Every DOT diagrams generated are first attempts without any resizing or layout optimization for the specified drawing page size.

Table 6.5: System Usage for Phase 2 performed on KDE Libs 3.0 vs 3.4.2

Phase 2 for KDE Libs 3.0.0 vs 3.4.2		No Load	96% Load	Difference
Totals	Handles	6,489	7,995	+1,506
	Threads	353	423	+70
	Processes	37	45	+8
Commit Charge in KB	Total	162,912	310,092	+147,180
	Limit	1,280,160	1,280,160	0
	Peak	379,312	379,312	0
Physical Memory in KB	Total	523,604	523,604	0
	Available	289,960	153,524	-136,436
	System Cache	279,432	199,000	-80,432
Kernel Memory in KB	Total	58,704	54,876	-3,828
	Paged	40,816	37,912	-2,904
	Non-Paged	17,888	16,964	-924

Table 6.6: X-Diff Timing for Phase 2 performed on KDE Libs 3.0 vs 3.4.2

X-Diff Timing per file	Minimum	Maximum
Parsing file 1	0.01 s	0.05 s
Parsing file 2	0.01 s	0.05 s
Diff operation	0.01 s	0.05 s
Writing diff	0.02 s	0.18 s
X-Diff Total	**0.05 s**	**0.20 s**

Table 6.7: KDE Libs 3.0 versus 3.4.2 statistics

	XML Files	Total size (bytes)	Total Run-Time (sec)	% CPU Usage	Memory Usage (KB)
KDE Libs 3.0	7,731	55,001,120	15,157.75	99%	262,144+
KDE Libs 3.4.2	10,886	157,167,808	20,590.20	99%	262,144+
Difference	+3,155	+102,166,688	+5,432.45	-	-
Empty supplement	4,631	1,157,750	3,572.33	96%	147,180
X-Diff XML Delta	6,347	190,887,551			
X-Diff DOT Delta	6,347	29,218,560			
X-Diff SVG Delta	6,347	104,901,801	1,321.298	90%-96%	2,240-6,895
Clean SVG Delta	6,347	93,644,827	307.7275	42%-91%	5,512
HTML Embed	6,347	2,941,795			

[+] The test took more than 256 MB and run out of heap memory about half-way; however, it was successfully resumed.

During the *Doxygen* generation, the program crashed about halfway, since it run out of heap memory. A second attempt was made in debug mode, which could be successfully resumed. This crash did not occur during the HTML generation, which took more than 36 hours. However, the crash actually occurred after the instant where the 36 hours HTML generation job was interrupted, which was estimated to be roughly 30% of the total HTML generation.

6.4 KDE Base Case Study

Another possible case study that was evaluated is the *KDE Base* package, which consist of the main GUI code, menus, applet and desktop behind the *K Desktop Environment*. Similar to the *KDE Libs* size, this KDE core component suffered fewer changes than *KDE Libs* by a ratio of 10:1 in terms of line of source code, as shown in the following comparison table. However, the number of file changes is only twice as less than *KDE Libs* for the same amount of source code. Due to time constraints, this test case was not

actually performed, only evaluated. However, it is reasonable to estimate that the amount of time required to generate a minimalist output would be in the same order of magnitude, roughly 11 hours of computation time without any HTML documentation.

Table 6.8: KDE Base – SLOC count

	SLOC	Comments	Blank Lines	Total	Files
KDE Base 3.0	322,713	53,867	56,212	432,792	1,615
KDE Base 3.4.2	346,586	72,820	72,834	492,240	2,054
Difference	+23,873	18,953	16,622	+59,448	+439

6.5 Case Study Summary

These case studies demonstrate that in most cases, *DoxyChange* is capable of identifying successfully software changes properly. It was also observed that *DoxyChange,* as it was also demonstrated, suffer from the same limitations, inherent to the usage of *Doxygen*, more precisely issues related to the intensive I/O throughput and random disk seek penalty.

Overall, *DoxyChange* provided meaningful diagrams, except for these corner cases where the corresponding file pair does not exist. It can also be observed that the diagrams generated were specified to be at most 4096x4096 pixels. This configuration parameter was tweaked as such, since it was known in advanced that most of the *KDE Utils 3.x* classes would have too many attributes, slots and function members to be displayed correctly with their surrounding associations at a normal 640x480 pixels drawing size.

Nevertheless, even at such high resolution, diagrams can be scaled up and down and are scrollable to visually see the overall class diagram.

It would have been possible to create even larger diagram using *DoxyChange*, in order to fit more elements within a single diagram. However, while examining each case study, it was obvious that generating a larger diagram such as 10240x10240 pixels would have been relatively useless since most diagrams did not have a significant amount of truncated relationships. The only exceptions were some insignificant truncated class within some of the *KHexEdit* class diagrams.

Based on the experience gained from enhancing *Doxygen*, it would require a significant amount of effort to modify *Doxygen* to generate an overall project class diagram, where every class would be linked to each other. However, as it can be seen in some of the diagrams currently generated, this type of diagram would greatly suffer in readability, since arrow crossing would become a major issue. Nevertheless, having such large and general overview diagram would be very useful for online web manipulation and generation by allowing the end user to hide or expand some parts of the diagram as needed.

Chapter 7 Conclusions and Future Work

This research provided a major overview of program comprehension, reverse engineering, software maintenance, software visualization and software evolution. It also provided a review of various concurrent version control systems being used by many open source projects and a survey of a wide variety of XML differentiating tools currently available. Overall, the major goals of this research project was to generate meaningful change diagrams for large scale software system, in order to observe and better understand how a given software system has evolved over time between two major revisions. As provided by the case studies discussed in Chapter 6, the change diagrams being generated by *DoxyChange* can help a developer or software engineer to easily and clearly identify some of the larger modifications and refactorings that took place within any software systems. In cases where the high level views did not provide sufficient information, navigating through the surrounding classes can generally provides a better understanding of the change context. In some cases, *DoxyChange* can provide an idea of the type of change which took place, while in some of the extreme cases as described in Chapter 1, *DoxyChange* cannot currently provide a meaningful diagram. One example is when a file was renamed, moved, added or deleted. Furthermore, *DoxyChange* also offer the possibility to have a user friendly browsing platform to further refine and understand the critical changes by simply clicking through the specified class and see the specific changes that took place within the source code itself through the online *Doxygen*

documentation of each revision. It is also possible to configure *DoxyChange* to generate an *hdiff* [SUN02][BER96][DB96][DBC98] of both *Doxygen* documentation revisions by showing them side-by-side as needed. The end result can easily be link within any of the *DoxyChange* diagram output by properly configuring the *HTML_URL_PREFIX* configuration variable, such that it points to the generated *hdiff* output instead.

Some of the possible future work might consist of maintaining the never ending enhancement of the parser infrastructure, by synchronizing the grammar changes with the original *Doxygen* project. Creating a web interface to allow a dynamic interaction between the end user visualizing the source code, the source code repository itself and the offline pre-generated server-side *Doxygen DOT-XML* files. Furthermore, a code generator could be inserted within the web server to enhance the end user experience by directly allowing the end user to manipulate the source code visually from the SVG interface itself using an *XML-RPC* interface [WIN03]. Additionally, the diagrams could be generated on the fly by using the *webdot* server application [ELL05]. This kind of manipulation would allow, the addition, manipulation and removal of certain items, while showing or hiding types or parameters, or editing a diagram and generating its corresponding source code skeleton. On the other hands, other kinds of enhancement could take place, such as better handling of file rename, copy, delete or move. Better handling of complete file, class, function or variable renames by using a Leveistein and semantics matching or similar advanced algorithm. Better linkage between the formal IEEE documentation and the generated diagrams and documentation. Finally, a more optimized I/O throughput, memory footprint and enhanced SMP capabilities for faster generation of *DoxyChange* would be extremely desirable.

Reference

[ABB05] Abbey, C.; Charles, P.; Shields, D. et al.; "Jikes' Home", IBM developerWorks, 29 July 2005. <http://jikes.sourceforge.net/>

[ADO92] Adobe Systems Inc.; "Encapsulated PostScript (EPS) – File Format Specification – Version 3.0 #5002", 34 pages, 1 May 1992. <http://partners.adobe.com/public/developer/en/ps/5002.EPSF_Spec.pdf>

[ADO00] Adobe Systems Inc.; "PostScript Specifications", December 2000. <http://partners.adobe.com/asn/developer/technotes/postscript.html>

[ADO01] Adobe Systems Inc.; "Current Support for SVG – Adobe SVG Viewer – Version 3.0 (Build 76)", November 2001. <http://www.adobe.com/svg/indepth/pdfs/CurrentSupport.pdf>

[ADO05] Adobe Systems Inc.; "Web Center Features – SVG – Manual Download – Adobe SVG Viewer 3.03", April 2005. <http://www.adobe.com/svg/viewer/install/main.html>

[ALE05] Alexander, A. J.; "WinCVS - How to check out a CVS Project", DevDaily Interactive Inc., (current August 2005). <http://www.devdaily.com/wincvs/HowToUseWinCVS/CheckingOutAProject.shtml>

[AND02] Andersson et al.; "SVG 1.1 Conformance Suite Implementation Status – Release 1.0", *W3C SVG Working Group*, 15 November 2002. <http://www.w3.org/Graphics/SVG/Test/20021115/matrix.html>

[AND03] Andersson et al.; "Scalable Vector Graphics (SVG) 1.1 Specification", *SVG Working Group, W3C: World Wide Web Consortium*, 14 January 2003. <http://www.w3.org/TR/SVG/>

[AND04] Andersson et al.; "W3C Scalable Vector Graphics (SVG) – News History", *SVG Working Group, W3C: World Wide Web Consortium*, 21 June 2004. <http://www.w3.org/Graphics/SVG/History>

[AND05] Andrieu, Olivier; "Monotone-viz", 6 July 2005. <http://oandrieu.nerim.net/monotone-viz/>

[ANDR02] Andrews, J.; "Linux: Linus And Kernel Patches", *KernelTrap.org*, 15 May 2002. <http://kerneltrap.org/node/193>

[ANG04] Anglin, J. D. et al.; "GCC news and announcements – GNU Project – Free Software Foundation (FSF)", *The GCC Team, Free Software Foundation Inc.*, November 2004. <http://gcc.gnu.org/news.html>

[ANS05] Answer.com; "aegis: Definition, Synonyms and Much More From Answers.com", *GuruNet Corporation*, (current August 2005). <http://www.answers.com/topic/aegis>

[APP98] Appleton, B.; "Brad Appleton's Software Configuration Management Links", *CM Crossroads*, 26 February 1998. <http://www.cmcrossroads.com/bradapp/links/scm-links.html>

[ASF05] Apache Software Foundation; "Apache HTTP Server Version 2.0 Documentation – Apache HTTP Server", (current August 2005). <http://httpd.apache.org/docs-2.0/>

[ASF05x] Apache Software Foundation; "Xerces C++ Parser", (current August 2005). <http://xml.apache.org/xerces-c/>

[ARE93] Aref, H.; Charles, R. D.; Elvins, T. T.; "Scientific Visualization of Fluid Flow". In Pickover, C. A. and Tewksbury, S. K. (editors), "Frontiers of Scientific Visualization", *Wiley Interscience*, pp. 7-43, 1993. <http://citeseer.csail.mit.edu/context/369866/0>

[ASV95] Asveld, P. R. J.; "A Fuzzy Approach to Erroneous Inputs in Context-Free Language Recognition", *Proceedings of the 4th International Workshop on Parsing Technologies (IWPT'95),* Prague-Karlovy Vary, Czech Republic, pp. 14-25, September 1995. <http://wwwhome.cs.utwente.nl/~infprja/ABS/1995IWPT04.pdf>

[AUS03] Ausbrooks, R. et al.; "Mathematical Markup Language (MathML) Version 2.0 (Second Edition)", 21 October 2003. <http://www.w3.org/TR/MathML2/>

[BAD00] Badros, G. J.; "JavaML Home Page (Greg J. Badros)", *University of Washington, Computer Science and Engineering Department,* 16 November 2000. <http://www.badros.com/greg/JavaML/>

[BAR98] Barnard, H. J. et al., "IEEE Std 1016-1998 (Revision of IEEE Std 1016-1987)", *IEEE Recommended Practice for Software Requirements Specifications*, 23 September 1998. Software Engineering Standards Committee of the IEEE Computer Society. ISBN: 0-7381-1456-1. <http://web.nps.navy.mil/~nschneid/is3020/PDF/1016-1998.pdf>

[BAR02] Barr, J.; "Linus tries to make himself scale", *Linux World Magazine*, 11 February 2002. <http://www.linuxworld.com/story/32722_p.htm> or <http://linux.sys-con.com/read/32722.htm>

[BAR04] Barnard, F. R.; "Proverbs compiled by GIGA", *Giga Quotes*, 22 November 2004. <http://www.giga-usa.com/gigaweb1/quotes2/quautbarnardfrederickrx001.htm>

[BAR05] Barr, J.; "BitKeeper and Linux: The end of the road?", *NewsForge*, 11 April 2005. <http://os.newsforge.com/article.pl?sid=05/04/11/118211&tid=152&cid=110195>

[BAT99] Di Battista, G. et al.; "Graph Drawing: Algorithm for the Visualization of Graphs", *Prentice Hall*, New Jersey, 400 pages, 1999. <http://www.amazon.com/exec/obidos/tg/detail/-/0133016153/>

[BEA05] Beale, T.; "Using BitKeeper – OVERVIEW", *OpenEHR*, 27 July 2005. <http://www.openehr.org/developer/t_bk_um_overview.htm>

[BER96] Berk, E.; "HtmlDiff: A Differencing Tool for HTML Documents", COS 461, *Princeton University*, December 1996. <http://citeseer.ist.psu.edu/context/1204015/0>

[BER03] Berry, J.; "ROI Guide: Economic Value Added", *Computerworld inc.,* Febuary 2003. <http://www.computerworld.com/managementtopics/roi/story/0,10801,78514,00.html>

[BER04] Berry, K.; "Gawk – GNU Project", *Free Software Foundation Inc.*, 31 August 2004. <http://www.gnu.org/software/gawk/gawk.html>

[BIG93] Biggerstaff, T. J., Mitbander, B.G.; Webster, D.; "The Concept Assignment Problem in Program Understanding", *Proceedings of the 15th International Conference on Software Engineering*, Baltimore, Maryland, pp. 482-498, May 1993. <http://portal.acm.org/citation.cfm?id=257679>

[BIT04a] BitMover Inc.; "BitKeeper – The Scalable Distributed Software Configuration Management System – Advantages", 2004. <http://www.bitkeeper.com/Products.BK_Pro.Advantages.html>

[BIT04f] BitMover Inc.; "BitKeeper – The Scalable Distributed Software Configuration Management System – Feature Summary", 2004. <http://www.bitkeeper.com/Products.BK_Pro.Feature.html>

[BIT04h] BitMover Inc.; "BitKeeper – The Scalable Distributed Software Configuration Management System – BK/Bits server statistics", 2004. <http://www.bitkeeper.com/Hosted.BK_Bits.html>

[BIT04s] BitMover Inc.; "BitKeeper – The Scalable Distributed Software Configuration Management System – Do you need BitKeeper?", 2004. <http://bitkeeper.com/Sales.Do.html>

[BIT05] BitMover Inc.; "BitKeeper – The Scalable Distributed Software Configuration Management System – Welcome", 15 June 2005. <http://www.bitkeeper.com/>

[BIT05d] BitMover Inc.; "BitKeeper – Download Request Form", 1 July 2005. <http://www.bitmover.com/cgi-bin/download.cgi >

[BOO99] Booch, G.; Rumbaugh, J.; Jacobson, I.; "The Unified Modeling Language User Guide", First Edition, *Addison-Wesley*, 1999. <http://www.amazon.com/exec/obidos/tg/detail/-/0201571684/>

[BRA96] Bray, Tim; Sperberg-McQueen, C. M.; "Extensible Markup Language (XML) – W3C Working Draft 14-Nov-96", *W3C: World Wide Web Consortium,* 14 November 1996. <http://www.w3.org/TR/WD-xml-961114.html>

[BRA98] Bray, Tim; Paoli, Jean; Sperberg-McQueen, C. M.; "Extensible Markup Language (XML) 1.0 ", *W3C: World Wide Web Consortium,* 10 February 1998. <http://www.w3.org/TR/1998/REC-xml-19980210>

[BRA00] Bray, Tim et al.; "Extensible Markup Language (XML) 1.0 (Second Edition) ", *W3C: World Wide Web Consortium,* 6 October 2000. <http://www.w3.org/TR/2000/REC-xml-20001006>

[BRI01] Brisset, P.; "A 3D animation of Linux source code development", June 2001. <http://perso.wanadoo.fr/pascal.brisset/kernel3d/kernel3d.html>

[BRI03] O'Brien, M. P.; "Software Comprehension – A Review & Research Direction", Technical Report UL-CSIS-03-3, *University of Limerick, Department of Computer Science & Information Systems*, Ireland, 29 pages, November 2003. <http://www1.csis.ul.ie/plone_media/Techrpts/UL-CSIS-03-3.PDF>

[BRO92] Brodlie, K. W. et al. (editors); "Scientific Visualization: Techniques and Applications", *Springer-Verlag*, New York, USA, 284 pages, 1992. ISBN: 0-387-54565-4. <http://portal.acm.org/citation.cfm?id=130916>

[BRO05] Broersma, M.; "Oracle accused of ignoring vulnerabilities", *Techworld*, 20 July 2005. <http://www.techworld.com/security/news/index.cfm?NewsID=4069>

[BRU00] Bruegger, B. P.; "Which Version Control System?", *TerraLuna: Infrastructures Mailing List*, 28 February 2000. <http://mailman.terraluna.org/pipermail/infrastructures/2000-February/000220.html>

[BUF95] Buffenbarger, J.; "Syntactic Software Merging", Lecture Notes in Computer Science, Vol. 1005, *ICSE SCM-5 Workshops on Software Configuration Management*, Seattle, Springer, Verlag, pp. 153-172, January 1995. ISBN: 3-540-60578-9. <http://portal.acm.org/citation.cfm?id=716256>

[CAN05] Government of Canada; "NSERC – CRSNG", *Minister of Public Works and Government Services Canada*, (current August 2005). <http://www.nserc-crsng.gc.ca/>

[CHA04] Chaparro, D. et al.; "Control version systems", *Edukalibre*, 15 January 2004. <http://www.google.com/search?q=cache:NsRuuvbpmAYJ:edukalibre.org/documentation/control_version_system.ps>

[CHI90] Chikofsky, E. J.; Cross, J. H.; "Reverse Engineering and Design Recovery: A Taxonomy", *IEEE Software*, vol. 7, no. 1, pp. 13-17, January 1990. <http://www.cs.concordia.ca/~comp6431/reading-list/chikofsky90reverse.pdf>

[CLA05] Clair, S. S.; "JPEG File Interchange Format File Format Summary", *C-Cube Microsystems, File Format.Info*, (current August 2005). <http://www.fileformat.info/format/jpeg/egff.htm>

[CMM02] CMMI Product Team; "Capability Maturity Model Integration (CMMI), Version 1.1", *Carnegie Mellon, Software Engineering Institute*, CMU/SEI-2002-TR-012 & ESC-TR-2002-012, 729 pages, March 2002. <http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr012.pdf>

[COL05] CollabNet Inc.; "Tigris.org: Open Source Software Engineering Tools", (current August 2005). <http://www.tigris.org/>

[COM90] CompuServe Inc.; "Graphics Interchange Format – Version 89a – Programming Reference", Columbus, Ohio, 31 July 1990. <http://www.w3.org/Graphics/GIF/spec-gif89a.txt>

[CON04] Connolly, D.; "Overview of SGML Resources", *W3C: World Wide Web Consortium,* 26 March 2004. <http://www.w3.org/MarkUp/SGML/>

[COV00] Cover, R.; "Graph Exchange Language (GXL)", *XML Cover Pages*, 8 September 2000. <http://xml.coverpages.org/gxl.html>

[CRA05] Crawford, M. et al.; "TortoiseCVS: About", *SourceForge.net*, 1 August 2005. <http://www.tortoisecvs.org/>

[CUN05] Cunningham & Cunningham Inc.; "Sub Version", (current August 2005). <http://c2.com/cgi/wiki?SubVersion>

[CUR98] Curbera, F.; Epstein, D.; "XML TreeDiff", *IBM AlphaWorks*, 13 November 1998. <http://alphaworks.ibm.com/tech/xmltreediff>

[DAI02] Daich, Gregory T.; "Document Diseases and Software Malpractice", *CrossTalk – The Journal of Defense Software Engineering*, Software Technology Support Center/SAIC, November 2002. <http://www.stsc.hill.af.mil/crosstalk/2002/11/daich.html>

[DB96] Douglis, F.; Ball, T.; "Tracking and Viewing Changes on the Web", *1996 USENIX Annual Technical Conference*, 1996. <http://www.research.ibm.com/people/f/fdouglis/papers/aide-usenix96.pdf>

[DBC98] Douglis, F.; Ball, T.; Chen, Y. F.; Koutsofios, E.; "The AT&T Internet Difference Engine: Tracking and Viewing Changes on the Web", *World Wide Web*, 1(1): 27-44, January 1998. <http://citeseer.ist.psu.edu/douglis98att.html>

[DEI90] Deimel, L. E.; Naveda, J. F.; "Reading Computer Programs: Instructor's Guide and Exercises", Technical Report CMU/SEI-90-EM-3, *Software Engineering Institute, Carnegie Mellon University*, 173 pages, August 1990. <http://www.sei.cmu.edu/publications/documents/ems/90.em.003.html>

[DES97] DesAutels, P. A.; "SHA1 Secure Hash Algorithm – Version 1.0", *W3C: World Wide Web Consortium*, 1 October 1997. <http://www.w3.org/PICS/DSig/SHA1_1_0.html>

[DEU82] Deutsch, M. S.; "Software Verification and Validation", *Prentice Hall*, 1982. <http://www.amazon.com/exec/obidos/ASIN/0138220727/>

[DEV05] Devine, C.; "SHA-1 Source Code", *www.cr0.net*, (current August 2005).
<http://www.cr0.net:8040/code/crypto/sha1/>

[DIC04] Dickey, T. E.; "BYACC – BERKELEY YACC", 28 April 2004.
<http://dickey.his.com/byacc/byacc.html>

[EAS93] Eastwood, A.; "Firm fires shots at legacy systems", *Computing Canada,*
Vol. 19, Issue 2, p. 17, 1993.

[EAS01] Eastlake, D.; Jones, P.; "US Secure Hash Algorithm 1 (SHA1)", *Network Working Group, The Internet Engineering Task Force*, September 2001. <http://www.ietf.org/rfc/rfc3174.txt>,

[EBE99] Ebert, J.; Kullbach, B.; Winter, A.; "GraX - An Interchange Format for Reengineering Tools".
In Proceedings of the 6th Working Conference on Reverse Engineering (WCRE'99), pp. 89-98,
8 October 1999. <http://www.gupro.de/winter/Papers/ebert+1999.pdf>

[EGY99] Egyed, A.; Kruchten, P.B.; "Rose/Architect: A Tool to Visualize Architecture",
Proceedings of the 32nd Hawaii International Conference on System Sciences, Maui, Hawaii, January 1999.
<http://sunset.usc.edu/~aegyed/publications/RoseArchitect-a_tool_to_visualize_architecture.pdf>

[EGY00] Egyed, A.; "Semantic Abstraction Rules for Class Diagrams",
Proceedings of the 15th IEEE International Conference of Automated Software Engineering (ASE),
Grenoble, France, pp. 301-304, September 2000.
<http://sunset.usc.edu/~aegyed/publications/Semantic_Abstraction_Rules_for_Class_Diagrams.pdf>

[EIB05] Eibl, J.; "KDiff3 – Homepage", *SourceForge.net*, 25 February 2005.
<http://kdiff3.sourceforge.net>

[EIB05s] Eibl, J.; "KDiff3 – Screenshots and Features", *SourceForge.net*, 30 January 2005.
<http://kdiff3.sourceforge.net/doc/screenshots.html#dirmergebigscreenshot>

[EKL99] Eklektix Inc.; "BitKeeper Features: Not quite Open Source", *Linux Weekly News*, 26 March 1999.
<http://lwn.net/1999/features/BitKeeper.phtml>

[EKL02] Eklektix Inc.; "LWN: The BitKeeper non-compete clause", *Linux Weekly News*, 10 October 2002.
<http://lwn.net/Articles/12120/>

[ELL05] Ellson, John; "WebDot HomePage", *AT&T Labs Research*, 20 July 2005.
<http://www.graphviz.org/webdot/>

[ERL00] Erlikh, L.; "Leveraging legacy system dollars for E-business", IEEE *IT Pro*, pp. 17-23,
May/June 2000. <http://www.bridge-quest.com/news/articles/legacydollars.pdf>

[EST00] Estublier, J.; "Software Configuration Management : A Roadmap", *Grenoble University, Dassault Systèmes / LSR*, 9 pages, 4 September 2000.
<http://www.cs.ucl.ac.uk/staff/A.Finkelstein/fose/finalestublier.pdf>

[ETT05] Ettrich, Mathias; "KDE Architecture – Binary Compatibility Issues With C++",
(current August 2005).
<http://developer.kde.org/documentation/library/kdeqt/kde3arch/devel-binarycompatibility.html>

[FAN00] Fang, X.; "Software Development Teams as Role Network", *ISFST2000, Proceedings of the International Symposium on Future Software Technology*, Software Research Associates Inc., Software Engineering Lab, Guyang, China, August 2000.
<http://www.sra.co.jp/people/fang/publish/papers/isfst00.pdf>

[FER99] Ferraiolo, Jon et al.; "Scalable Vector Graphics (SVG) Specification – W3C Working Draft 11 February 1999", *SVG Working Group, W3C: World Wide Web Consortium*, 11 February 1999. <http://www.w3.org/TR/1999/WD-SVG-19990211/>

[FER02] Ferenc, R.; "Columbus – Reverse Engineering Tool and Schema", *International Conference on Software Maintenance (ICSM'02)*, Montreal, Canada, October 2002, pp. 172-181. <http://www.inf.u-szeged.hu/~ferenc/research/ferencr_columbus_tool_schema.pdf>

[FIS05] Fish, S.; "Better SCM Initiative: Monotone", *BerliOS*, 23 April 2005. <http://better-scm.berlios.de/monotone/>

[FIS05c] Fish, S.; "Version Control System Comparison", *BerliOS*, 7 May 2005. <http://better-scm.berlios.de/comparison/comparison.html>

[FOK05] Fokus; "BerliOS – The Open Source Mediator", *BerliOS*, (current August 2005). <http://www.berlios.de/>

[FOR02] Forward, A.; Lethbridge, T. C.; "The Relevance of Software Documentation, Tools and Technologies: A Survey", *Proceedings of the 2002 ACM symposium on Document engineering*, University of Ottawa, pp. 26-33, 9 November 2002. ISBN: 1-58113-594-7. <http://portal.acm.org/citation.cfm?id=585065> or <http://www.literateprogramming.com/doceng.pdf>

[FRO02] Froumentin, M.; "Displaying chess games in Web pages using ChessGML and SVG", *W3C: World Wide Web Consortium,* 2002. <http://people.w3.org/maxf/ChessGML/>

[FUZ05w] Fuzellier, M.; "World Wide Web consortium (W3C)", *W3C: World Wide Web Consortium*, 5 August 2005. <http://www.w3c.org/>

[FUZ05h] Fuzellier, M.; "HTTP – Hypertext Transfer Protocol Overview", *W3C: World Wide Web Consortium*, 31 January 2005. <http://www.w3.org/Protocols/>

[GAL97] H. Gall, M. Jazayeri, R. G. Klsch, G. Trausmuth; "Software Evolution Observations Based on Product Release History", *Proceedings of the Conference on Software Maintenance*, pp. 160-166, 1997. <http://citeseer.ist.psu.edu/gall97software.html>

[GAL04] Galbraith, J. et al.; "SSH File Transfer Protocol)", *The Internet Engineering Task Force*, January 2004. <http://www.ietf.org/proceedings/04aug/I-D/draft-ietf-secsh-filexfer-05.txt>

[GAM93] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.; "Design Patterns: Elements of Reusable Object-Oriented Software", *Addison Wesley*, 395 pages, 1993. <http://www.amazon.co.uk/exec/obidos/ASIN/0201633612/>

[GAN00] Gansner, E. et al.; "Graphviz", *AT&T Labs Research*, October 2000. <http://www.research.att.com/sw/tools/graphviz/>

[GER94] Gershon, N.; "Perception to Visualization". In "Scientific Visualization: Advances and Challenges", Rosenblum, L. et al. (eds), *Academic Press*, San Diego, USA, 1994. ISBN: 0-120227742-2.

[GER05] Gernler, A; "OpenSSH", *OpenBSD*, 28 June 2005. <http://www.openssh.com/>

[GIU99] Giulio, A.; Tonella, P.; "Object Oriented Design Pattern Inference", *IEEE International Conference on Software Maintenance*, Oxford, England, pp. 230-240, August 1999. <http://citeseer.ist.psu.edu/tonella99object.html>

[GOI04] Goikhman, Mikhael; Cramer, Enno; "ArchWay – GNU Arch GUI – Screenshots", 31 October 2004. <http://www.nongnu.org/archway/screenshots/full/archang.png>

[GOL99] Goland, Y. et al., "HTTP Extensions for Distributed Authoring – WEBDAV", *The Internet Engineering Task Force*, February 1999. <http://www.ietf.org/rfc/rfc2518.txt>

[GOV00] Govil, M.K. and E.B. Magrab; "Designing for Time-To-Market: Predicting the Effects of Product Design on Production Rate", *Proceedings of the 1999 ASME Design Engineering Technical Conferences*, Las Vegas, Nevada, September 2000. <http://www.google.com/search?q=cache:QSf3WQA4CK0C:www.wbmt.tudelft.nl/pto/research/conferences/Proceedings/ASME99/pdfs/dfm/8926.pdf>

[GUE01] Guenin, T.; "TL 9000: ISO 9000 FOR THE TELECOMMUNICATIONS INDUSTRY", *Proceedings of SWE 2001 National Conference – The Technology Odyssey*, Colorado, Denver, September 2001. <http://www.partnersthroughchange.com/TL9000.pdf>

[GUL05] Gully, S.; "Amaya Home Page", *W3C: World Wide Web Consortium,* 13 July 2005. <http://www.w3.org/Amaya/>

[HAR02] Hardy, Matthew R. B.; Brailsford, David F.; "Mapping and displaying structural transformations between XML and PDF", *Proceedings of the 2002 ACM symposium on Document engineering*, Virginia, USA, pp. 95-102, 9 November 2002. ISBN: 1-58113-594-7. <http://portal.acm.org/citation.cfm?id=585058.585077>

[HAT05] Hatta, M.; "The GNU Operating System", *Free Software Foundation Inc.*, 17 July 2005. <http://www.gnu.org/>

[HAU02] Hausmann, Simon; "Re: KDE 3.0 - the last steps", *KDE Core Devel Mailing List*, 22 March 2002. <http://lists.kde.org/?l=kde-core-devel&m=101680905231662&w=2>

[HEE05] Heesch, D.; "Doxygen", *Doxygen.org*, 21 July 2005. <http://www.stack.nl/~dimitri/doxygen/>

[HEE05a] Heesch, D.; "Doxygen – Diagrams: Doxygen's Internals", *Doxygen.org*, 20 July 2005. <http://www.stack.nl/~dimitri/doxygen/arch.html>

[HEE05b] Heesch, D.; "Doxygen – Documenting the code", *Doxygen.org*, 21 July 2005. <http://www.stack.nl/~dimitri/doxygen/docblocks.html>

[HEE05d] Heesch, D.; "Doxygen – Download Page", *Doxygen.org*, 21 July 2005. <http://www.stack.nl/~dimitri/doxygen/download.html#latestsrc>

[HEE05g] Heesch, D.; "Doxygen – Diagrams: Graph Legend", *Doxygen.org*, 20 July 2005. <http://www.stack.nl/~dimitri/doxygen/examples/diagrams/html/graph_legend.html>

[HEE05h] Heesch, D.; "Doxygen – Helper tools & scripts", *Doxygen.org*, 21 July 2005. <http://www.stack.nl/~dimitri/doxygen/helpers.html>

[HEE05o] Heesch, D.; "Doxygen – Output Formats", *Doxygen.org*, 21 July 2005. <http://www.stack.nl/~dimitri/doxygen/output.html>

[HEE05p] Heesch, D.; "Projects that use Doxygen", *Doxygen.org*, 21 July 2005. <http://www.stack.nl/~dimitri/doxygen/projects.html>

[HEE05q] Heesch, D.; "Doxygen – Frequently Asked Questions", *Doxygen.org*, 21 July 2005. <http://www.stack.nl/~dimitri/doxygen/faq.html>

[HEE05s] Heesch, D.; "Doxygen – Diagrams: Getting started", *Doxygen.org*, 20 July 2005. <http://www.stack.nl/~dimitri/doxygen/starting.html>

[HEE05t] Heesch, D.; "Doxygen – Troubleshooting", *Doxygen.org*, 21 July 2005. <http://www.stack.nl/~dimitri/doxygen/trouble.html>

[HIE05] Hietaniemi, J.; "Comprehensive Perl Archive Network", *Comprehensive Perl Archive Network*, 2 April 2005. <http://www.cpan.org/>

[HOA05c] Hoare, G.; "Certificates – monotone documentation", 17 July 2005. <http://www.venge.net/monotone/docs/Certificates.html>

[HOA05d] Hoare, G.; "monotone: distributed version control", 17 July 2005. <http://www.venge.net/monotone/>

[HOA05f] Hoare, G.; "freshmeat.net: Project details for monotone", *FreshMeat.net*, 19 July 2005. <http://freshmeat.net/projects/monotone/>

[HOA05i] Hoare, G.; "monotone documentation", 17 July 2005. <http://www.venge.net/monotone/monotone.html>

[HOA05s] Hoare, G.; "monotone – Summary[Savannah]", *Savannah*, 30 July 2005. <http://savannah.nongnu.org/projects/monotone/>

[HOA05v] Hoare, G.; "Versions of files – monotone documentation", 17 July 2005. <http://www.venge.net/monotone/docs/Versions-of-files.html>

[HOL99] Holz, E.; "Application of UML within the Scope of new Telecommunication Architectures", *Humboldt-Universität zu Berlin – Institut für Informatik*, March 1999. <http://www.informatik.hu-berlin.de/~holz/Literatur/UML.Mannheimtxt.fm5.pdf>

[HOLG02] Holger, M. et al.; "Leveraging SVG in the Rigi Reverse Engineering Tool", *University of Victoria, Computer Science Department*, 17 July 2002. <http://www.svgopen.org/2002/papers/kienle_weber_mueller__rigi_reverse_engineering/>

[HOLT00] Holt, R. C. et al.; "GXL: Towards a Standard Exchange Format", WCRE 2000, *University of Waterloo, Department of Computer Science*, November 6, 2000. <http://citeseer.ist.psu.edu/holt00gxl.html>

[HOLT02] Holt, R. C. et al.; "GXL – Graph eXchange Language", *University of Waterloo, Computer Science Department*, 17 July 2002. <http://www.gupro.de/GXL/>

[HUF90] Huff, S.; "Information systems maintenance". *The Business Quarterly*, Vol. 55, pp. 30-32, 1990.

[HYD05] Hydro-Québec; "Hydro-Québec", *Government of Québec*, 11 April 2005. <http://hydroquebec.com/>

[IAW01] IBM Alphaworks; "XML Diff and Merge Tool", 27 March 2001. <http://www.alphaworks.ibm.com/tech/xmldiffmerge>

[IBM05] International Business Machine Corporation; "alphaWorks – Emerging technologies", 5 August 2005. <http://www.alphaworks.ibm.com>

[ICS99] IEEE Computer Society; "Recommended Practice for Architectural Description of Software-Intensive Systems", P1471, *IEEE Computer Society*, 1999. <http://www.win.tue.nl/~mchaudro/sa2004/ieee1471.pdf>

[JAC99] Jackson, Q. T.; "PAISLEI: Towards Grammar-Only Parsing of C++", 1999. <http://www.google.com/search?q=cache:qWxludPjIt8J:members.shaw.ca/qjackson/writing_editing/articles/PAISLEIGrammarOnly.html>

[JAM05] Jamieson , K.; "GNU make – Using Implicit Rules", *Massachusetts Institute of Technology*, 21 may 2005. <http://web.mit.edu/6.033/labdoc/make_10.html>

[JEU02] Jeuring, Johan; Hagg, Paul; "Generic Programming for XML Tools", *Utrecht University, Institute of Information and Computing Sciences*, 12 pages, 27 May 2002. <http://www.cs.uu.nl/~johanj/publications/gp4xml.pdf>

[JOH75] Johnson, S. C.; "YACC – Yet Another Compiler Compiler", CSTR 32, *Bell Laboratories*, Murray Hill, NJ, 1975.

[JOH05] Johnson, J. et al.; "GNU Standard C++ Library v3 – GNU Project – *Free Software Foundation (FSF)*", *The GCC Team, Free Software Foundation Inc.*, 11 July 2005. <http://gcc.gnu.org/libstdc++/index.html>

[JOR94] Jorgensen, P.C.; Erickson, C.; "Object-Oriented Integration Testing", *Communications of the ACM*, vol. 37, no. 9, pp. 30-38, September 1994. <http://portal.acm.org/citation.cfm?id=182989>

[KAN86] Kantor, B. et al.; "Network News Transfer Protocol", *The Internet Engineering Task Force*, February 1986. <http://www.ietf.org/rfc/rfc977.txt>

[KAZ97] Kazman, R.; Carriere, S. J.; "Playing Detective: Reconstructing Software Architecture from Available Evidence", *Technical Report CMU/SEI-97-TR-010*, Carnegie Mellon Software Engineering Institute, Pittsburgh, Pennsylvania, October 1997. <http://www.sei.cmu.edu/architecture/ASE.pdf>

[KDE05] KDE e.V.; "K Desktop Environment – Conquer your Desktop!", (current August 2005). <http://www.kde.org/>

[KEL01] Keller, R.K. et al.; "The SPOOL Approach to Pattern-Based Recovery of Design Components", Published by Erdogmus, H. and O. Tanir, *Advances in Software Engineering – Evolution, Comprehension, and Evaluation*, Springer-Verlag, 2001. <http://www.iro.umontreal.ca/~keller/Publications/Papers/2002/cbook-2002-dpr.pdf>

[KIN04] King, B.; "GCC-XML", *Kitware*, 5 May 2004. <http://www.gccxml.org/>

[KNA99] Knapen, G.; Lague, B.; Dagenais, M.; Merlo, E.; "Parsing C++ despite missing declarations", Program Comprehension, 1999. *Proceeding of the 7th International Workshop on 5-7 May 1999*, pp. 114-125, 7 May 1999. <http://portal.acm.org/citation.cfm?id=520033.858245>

[KNI98] Knight, C.; "Visualisation for Program Comprehension: Information and Issues", Computer Science Technical Report 12/98, *University of Durham, Department of Computer Science*, 23 pages, October 1998. <http://vrg.dur.ac.uk/papers/getpaper.php3?id=7> or <http://www.dur.ac.uk/~dcs3crk/workfiles/documents/Lit_Survey_Tech_Reports/Tech_Report_12-98.ps.gz>

[KOE96] Koenig, A.; "1997 C++ Public Review Document: Working Paper for Draft Proposed International Standard for Information Systems – Programming Language C++", *American National Standards Institute (ANSI)*, December 1996. <http://anubis.dkuug.dk/jtc1/sc22/open/n2356/>

[KOL00] Kollman, R.; "IDEA Homepage", *University of Bremen Database Systems Group*, 2000. <http://dustbin.informatik.uni-bremen.de/projects/idea/>

[KOL02] Kollman, R. et al.; "A Study on the Current State of the Art in Tool-Supported UML-Based Static Reverse Engineering", *9th IEEE Working Conference on Reverse Engineering (WCRE '02)*, Richmond, Virginia, pp. 22-32, October 2002. <http://www.se.eecs.uni-kassel.de/se/fileadmin/se/publications/KSSSZ02.pdf>

[KOL05] Kölker, J.; "Sed – GNU Project", *Free Software Foundation Inc.*, 12 May 2005. <http://www.gnu.org/software/sed/sed.html>

[KOP97] Koppler, R.; "A Systematic Approach to Fuzzy Parsing", *Software – Practice and Experience*, vol. 27, no. 6, June 1997, pp. 637-649. <http://citeseer.ist.psu.edu/koppler96systematic.html>

[KOS96] Koskimies, K., et al.; "SCED: A Tool for Dynamic Modelling of Object Systems", Report A-1996-4, *University of Tampere*, Tampere, Finland, July 1996. <http://www.cs.uta.fi/reports/pdf/A-1996-4.pdf>

[KOS98a] Koskimies, K., et al.; "Automated Support for Modeling OO Software", *IEEE Software*, vol. 15, no. 1, January/February 1998, pp. 214-223. <http://portal.acm.org/citation.cfm?id=624623.625817>

[KOS98b] Koskimies, K., et al.; "SCED", *Tampere University of Technology, Institute of Software Systems*, December 1998. <http://www.cs.tut.fi/~tsysta/sced/>

[KOS04] Koskinen, J.; "Software Maintenance Costs", University of Jyväskylä, Information Technology Research Institute, ELTIS, Finland, 28 September 2004. <http://www.cs.jyu.fi/~koskinen/smcosts.htm>

[KRA96] Kramer, C.; Prechelt, L.; "Design Recovery by Automated Search for Structural Design Patterns in Object Oriented Software", *3rd Working Conference on Reverse Engineering (WCRE '96)*, Monterey, California, November 1996, pp. 208-216. <http://page.mi.fu-berlin.de/~prechelt/Biblio/wcre96.pdf>

[KUH00] Kuhn, B. M.; "Ghostview – GNU Project – Free Software Foundation", *Free Software Foundation Inc.*, 15 April 2000. <http://gnu.planetmirror.com/software/ghostview/ghostview.html>

[KUM01] Kumar, D.; Curtis, B.; "Curtis Unveils New PCMM Version: QAI India to Implement Version 2.0 of the People CMM", *Software Dioxide*, 1st August 2001, 2 pages. <http://www.softwaredioxide.com/Channels/Experts/PCMM_2_QAI.htm>

[KUN05] Küng, S.; "tortoisesvn.tigris.org", *Tigris.org*, 15 July 2005. <http://tortoisesvn.tigris.org/>

[LEM05] Lemarie, Y.; "Services : Training : Evolution of UML", *Objecteering Software*, (current August 2005). <http://www.objecteering.com/services_training_evu.php>

[LEV05] Levert, C.; "grep – GNU Project", *Free Software Foundation Inc.*, 2 May 2005. <http://www.gnu.org/software/grep/grep.html>

[LIE78] Lientz, B.P., E.B. Swanson, and G.E. Tompkins; "Characteristics of Application Software Maintenance", *Communications of the ACM*, Vol. 21, No. 6, June 1978, pp. 466-471. <http://portal.acm.org/citation.cfm?id=359511.359522>

[LIE80] Lientz, B.P. and E.B. Swanson; "Soare Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations", *Addison-Wesley*, 1980. <http://citeseer.ist.psu.edu/context/10360/0>

[LIE81] Lientz, B.P.; Swanson, E.; "Problems in application software maintenance", *Communications of the ACM*, Vol. 24, Issue 11, pp. 763-769, 1981. ISSN: 0001-0782. <http://portal.acm.org/citation.cfm?id=358790.358796>

[LIL05] Lilley, C.; Jackson D.; "Scalable Vector Graphics (SVG)", *W3C: World Wide Web Consortium*, 6 August 2005. <http://www.w3.org/Graphics/SVG/>

[LIN01p] Lindholm, Tancred; "A 3-way Merging Algorithm for Synchronizing Ordered Trees – the 3DM merging and differencing tool for XML", *Helsinki University of Technology, Department of Computer Science, Laboratory of Information Processing Science*, 205 pages, 13 September 2001. <http://www.cs.hut.fi/~ctl/3dm/thesis.pdf>

[LIN01s] Lindholm, Tancred; "The "3DM" XML 3-way Merging and Differencing Tool", *Helsinki University of Technology, Department of Computer Science, Laboratory of Information Processing Science*, 13 September 2001. <http://www.cs.hut.fi/~ctl/3dm/>

[LIN04] Lindholm, Tancred; "A Three-way Merge for XML Documents", *Proceedings of 2004 ACM symposium on Document engineering*, Milwaukee, USA, pp. 1-10, 30 October 2004. ISBN: 1-58113-938-1. <http://www.hiit.fi/fuego/fc/doceng04-pc.pdf> or <http://portal.acm.org/citation.cfm?id=1030397.1030399>

[LIN05p] Lindholm, T.; "BerliOS Developer: Project Info - 3DM XML diff and merge tool", *BerliOS*, (current August 2005). <http://developer.berlios.de/projects/tdm/>

[LIN05w] Lindholm, T.; "The '3DM' XML 3-way Merging and Differencing Tool", *BerliOS*, 1 June 2002. <http://tdm.berlios.de/3dm/doc/index.html>

[LOR03] Lord, Tom; "[arch-users] GNU, doc foo, short-term plans, hacking suggestions, money", *GNU Arch Mailing List*, 12 July 2003. <http://lists.alug.org.uk/announce/2003q3/000109.html>

[LOR05] Lord, Tom; "FrontPage – Arch Wiki", *Free Software Foundation Inc.*, 5 August 2005. <http://wiki.gnuarch.org/>

[LOV02] Love, Robert; Andrews, J.; "Interview: Robert Love", *KernelTrap.org*, 16 July 2002. <http://kerneltrap.org/node/336>

[LUC00] Di Lucca, G.A., Fasolino, A.R.; U. de Carlini; "Recovering Use Case Models from Object-Oriented Code: A Thread-Based Approach", *Proceedings of 7th IEEE Working Conference on Reverse Engineering (WCRE '00)*, Brisbane, Australia, November 2000, pp. 108-117. <http://portal.acm.org/citation.cfm?id=837093>

[MAL01] Malloy, B.; "Keystone: A Parser and Front-End for ISO C++", *Clemson University, Computer Science Department*, July 2001. <http://www.cs.clemson.edu/~malloy/projects/keystone/keystone.html>

[MAR03] Martynov, I.; "search.cpan.org – Data::Dumper – stringified perl data structures, suitable for both printing and eval", Version 2.121, *Comprehensive Perl Archive Network*, 24 August 2003. <http://search.cpan.org/~ilyam/Data-Dumper-2.121/Dumper.pm>

[MAT02] Matzko, S. et al.; "Reveal: A Tool to Reverse Engineer Class Diagrams", *Proceedings of the 40th International Conference on Technology of Object-Oriented Languages and Systems*, Sydney, Australia, February 2002, pp. 13-21. <http://www.cs.clemson.edu/~malloy/papers/tools02/paper.pdf>

[MCC87] McCormick, B .H., DeFanti, T. A.; Brown, M. D. (editors); "Visualization in Scientific Computing", *Computer Graphics*, Vol. 21, No. 6, November 1987.

[MCZ01] Mascolo, C.; Capra, L.; Zachariadis, S.; Emmerich, W; "xmiddle: A Data-Sharing Middleware for Mobile Computing", *University College London, Department of Computer Science*, 30 pages, 30 August 2001. <http://www.cs.ucl.ac.uk/staff/l.capra/publications/mcze02.pdf>

[MER05] Meranda, D.; "PNG Documentation – Version 1.2.5", *SourceForge.net*, 12 July 2005. <http://www.libpng.org/pub/png/pngdocs.html>

[MEY05a] Meyer, Benjamin; "ark.png", *TrollTech AS*, (current August 2005). <http://trolls.trolltech.no/~bmeyer/pictures/ark.png>

[MEY05d] Meyer, Benjamin; "kdf.png", *TrollTech AS*, (current August 2005). <http://trolls.trolltech.no/~bmeyer/pictures/kdf.png>

[MEY05h] Meyer, Benjamin; "khexedit.png", *TrollTech AS*, (current August 2005). <http://trolls.trolltech.no/~bmeyer/pictures/khexedit.png>

[MEY05r] Meyer, Benjamin; "kregexp.png", *TrollTech AS*, (current August 2005). <http://trolls.trolltech.no/~bmeyer/pictures/kregexp.png>

[MEY05t] Meyer, Benjamin; "ktimer.png", *TrollTech AS*, (current August 2005). <http://trolls.trolltech.no/~bmeyer/pictures/ktimer.png>

[MCV02] McVoy, L.; Andrews, J.; "Interview: Larry McVoy", *KernelTrap.org*, 28 May 2002. <http://kerneltrap.org/node/222>

[MIL56] Miller, G.A.; "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information", *The Psychological Review*, Vol. 63, 1956, pp. 81-97. <http://www.well.com/user/smalin/miller.html>

[MIL05] Miller, P.; "Aegis 4.20", *SourceForge.net*, (current August 2005). <http://aegis.sourceforge.net/>

[MIL05c] Miller, P.; "Aegis: A Project Change Supervisor – CVS Transition Guide", *SourceForge.net*, 14 pages, 28 January 2005. <http://aegis.sourceforge.net/cvs-comparison.pdf>

[MIL05p] Miller, P.; "Aegis is Only for Software, Isn't It?", *SourceForge.net*, 18 pages, August 1996. <http://aegis.sourceforge.net/auug96.pdf>

[MIL05q] Miller, P.; "Aegis Frequently Asked Questions (FAQ)", *SourceForge.net*, 28 January 2005. <http://aegis.sourceforge.net/aegis-4.20.faq>

[MIL05w] Miller, P.; "Aegis 4.20 – Web Interface", *SourceForge.net*, (current August 2005). <http://aegis.sourceforge.net/webiface.html>

[MIT03] MIT Software Design Group; "Software Design Group – Superwomble", *MIT Laboratory for Computer Science,* (current March 2003). <http://sdg.lcs.mit.edu/projects/superwomble.htm>

[MOA90] Moad, J.; "Maintaining the competitive edge", *Datamation,* pp. 61-66, 1990.

[MOF01] Moffitt, J.; "A Critique of the BitKeeper License", *Massachusetts Institute of Technology*, 2001. <http://www.mit.edu/afs/athena/user/x/i/xiphmont/Public/critique.html#id2708519>

[MOR05] Morton, Andrew; "The Linux Kernel Development Process", *Peggy Aycinena*, Palo Alto, USA, 19 May 2005. <http://www.aycinena.com/index2/index3/archive/andrew%20morton%20&%20the%20linux%20kernel.html>

[MUL94] Müller, H.; Wong, K; Tilley, S. R.; "Understanding Software Systems Using Reverse Engineering Technology", 62nd Congress of L'Association Canadienne Française pour l'Avancement des Sciences, *University of Victoria, Department of Computer Science*, 9 pages, 17 May 1994. <http://rigi.cs.uvic.ca/downloads/papers/pdf/ussuret.pdf>

[MUL00] Müller, H.A., et al.; "Reverse Engineering: A Roadmap", *Proceedings of the Conference on the Future of Software Engineering*, Limerick, Ireland, June 2000, pp. 47-60. <http://www.cs.ucl.ac.uk/staff/A.Finkelstein/fose/finalmuller.pdf>

[MUL04] Müller, H.A.; "Rigi Group Home Page", *University of Victoria, Computer Science Department*, December 2004. <http://www.rigi.csc.uvic.ca/index.html>

[MUR01] Murray-Rust, P.; Rzepa, H. S.; "Chemical Markup Language. A Position Paper", 10 April 2001. <http://www.xml-cml.org/information/position.html>

[MUR01j] Murray-Rust, Peter; Rzepa, Henry; "CML Mols Viewer", 23 July 2001. <http://www.xml-cml.org/jumbo3/jumbo3-JS/jumbo.html>

[MYE79] Myers, G. J; "The Art of Software Testing", *John Wiley & Sons Inc.*, New York, USA, 177 pages, 1979. ISBN: 0-4710-4328-1. <http://portal.acm.org/citation.cfm?id=539883>

[MYE86] Myers E. W.; "An O(ND) Difference Algorithm and its Variations", *Algorithmica* vol. 1 no. 2, pp. 251-266, 1986. <http://www.xmailserver.org/diff2.pdf>

[NCS98] National Center for Supercomputing Applications; "CGI: Common Gateway Interface", 21 January 1998. <http://hoohoo.ncsa.uiuc.edu/cgi/intro.html>

[NEE00] Neelakanth, N.; "Flex – GNU Project – Free Software Foundation (FSF)", *Free Software Foundation Inc.*, 29 April 2000. <http://www.gnu.org/software/flex/>

[NET96] Netscape Communications Corporation; "JavaScript Guide", 1996. <http://wp.netscape.com/eng/mozilla/3.0/handbook/javascript/>

[NGU04] Nguyen, Binh; "ktimer - What is ktimer", *About.com*, 30 October 2004. <http://linux.about.com/cs/linux101/g/ktimer.htm>

[NOR05] North, S.; "The DOT Language", *AT&T Labs Research*, (current August 2005). <http://www.graphviz.org/pub/scm/graphviz2/doc/info/lang.html>

[OGP04] Open Group Base, "The Open Group Base Specifications Issue 6", *IEEE Std 1003.1, 2004.* <http://www.opengroup.org/onlinepubs/009695399/basedefs/tar.h.html>

[OMG03a] Object Management Group; "OMG – Object Management Group", March 2003. <http://www.omg.org/>

[OMG03b] Object Management Group; "OMG Unified Modeling Language Specification", March 2003. <http://www.omg.org/docs/formal/03-03-01.pdf>

[OMG05] Object Management Group Inc.; "UML™ Resource Page", 19 January 2005. <http://www.uml.org/>

[OMG05w] Object Management Group; "Introduction to OMG's Unified Modeling Language (UML)", 26 May 2005. <http://www.omg.org/gettingstarted/what_is_uml.htm>

[ORI98] Oriogun, P.; "Software – Problems and Expectations", *School of Informatics and Multimedia Technology at University of North London*, September 1998. <http://homepages.unl.ac.uk/~oriogunp/softprob.htm>

[ORL05] Orlowski, A.; "Tridgell demos Bitkeeper interoperability", *The Register*, 21 April 2005. <http://www.theregister.co.uk/2005/04/21/tridgell_bitkeeper_howto/>

[OUT04] Ousterhout, John et al.; "BitKeeper – The Scalable Distributed Software Configuration Management System – Screen shots", *BitMover Inc.*, 2004. <http://www.bitkeeper.com/gifs/csettool.gif>

[PAR94] Parnas, D. L.; "Software aging", *International Conference on Software Engineering.* In *Proceedings of the 16th international conference on Software engineering* Sorrento, Italy, pp. 279-287, 1994. ISBN: 0-8186-5855-X. <http://portal.acm.org/citation.cfm?id=257788>

[PAR95] Parr, T. J.; Quong, R. W.; Cohen, W.; Dietz, H.; "Index of /1.33/workshop95", *PCCTS Workshop*, 1995. <http://web.archive.org/web/20010822193518/www.antlr.org/1.33/workshop95/>

[PAR05] Parenteau, A. et al.; "CvsGui", *SourceForge.net*, 24 July 2005. <http://www.wincvs.org/>

[PAV05] Pavlov, Igor; "7z Format", *SourceForge.net*, 7 August 2005. <http://www.7-zip.org/>

[PEE05] Peeters, F.; "Linus and Bitkeeper", *Linux Kernel Mailing List*, 12 April 2005. <http://www.linuxsa.org.au/pipermail/linuxsa/2005-April/078690.html>

[PER05] Perens, Bruce; "Open Source Initiative – The Open Source Definition – Version 1.9", *Open Source Initiative*, (current August 2005). <http://www.opensource.org/docs/definition.php>

[PES98] Pesch, R.; "Concurrent Versions System (CVS)", *Free Software Foundation Inc.*, 7 November 1998. <http://web.archive.org/web/20030618164634/www.gnu.org/manual/cvs-1.9/ html_mono/cvs.html>

[PET05] Petersen, Jesper K.; "KDE Repository – markup – SVN: tags/3.4.1/kdeutils/kregexpeditor/editorwindow.h rev.417278", *The KDE Source Repository*, 23 March 2005. <http://websvn.kde.org/tags/KDE/3.4.1/kdeutils/kregexpeditor/editorwindow.h?rev=417278&view=markup>

[PFE01] Pfeiffer, Carsten; Kulow, Stephan; "c't 5/2001, page 242 – KDE 2.0 – The Omnivore – KDE's flexible I/O architecture ", 13 pages, 19 July 2001. <http://www.heise.de/ct/english/01/05/242/>

[PIG97] Pigoski, T. M.; "Practical Software Maintenance", *John Wiley & Sons*, 1997. <http://www.amazon.com/exec/obidos/tg/detail/-/0471170011>

[PIN02] Pinzger, M. et al.; "Revealer: A Lexical Pattern Matcher for Architecture Recovery", *9th Working Conference on Reverse Engineering (WCRE'02)*, Richmond, Virginia, pp. 170-178, October 2002. <http://www.infosys.tuwien.ac.at/Cafe/doc/mp-revealer.pdf>

[PKW05] PKWARE Inc., "APPNOTE.TXT – .ZIP File Format Specification – v6.2.1", 40 pages, 1 April 2005. <http://www.pkware.com/company/standards/appnote/appnote.txt>

[POR88] Port, O.; "The software trap – automate or else", *Business Week*, Vol. 3051, Issue 9, pp. 142-154, 1988.

[POS85] Postel, J. et al.; "File Transfer Protocol (FTP)", *The Internet Engineering Task Force*, October 1985. <http://www.ietf.org/rfc/rfc959.txt>

[PRI05] Price, D. R.; "CVS – Open Source Version Control", *Free Software Foundation Inc.*, 27 July 2005. <http://www.gnu.org/software/cvs/>

[PRO05] Proulx, B.; "Bison – GNU Project – Free Software Foundation (FSF)", *Free Software Foundation Inc.*, 2 July 2005. <http://www.gnu.org/software/bison/bison.html>

[PUS05] Pushok Software Co. Ltd.; "CVS SCC API plugin as the replacement for Microsoft SCC (sourcesafe)", 12 July 2005. <http://www.pushok.com/soft_cvs_proxy.php>

[QUE05] Government of Québec; "Le Fonds québécois de la recherche sur la nature et les technologies", *Le Fonds québécois de la recherche sur la nature et les technologies*, 22 July 2005. <http://www.fcar.qc.ca/>

[RAD90] Radatz, J. et al., "IEEE Standard Glossary of Software Engineering Terminology", IEEE Standard 610.12-1990 (Revision and redesignation of IEEE Standard 792-1983), Standards *Coordinating Committee of the IEEE Computer Society, IEEE Standards Board*, 84 pages, 28 September 1990. <http://standards.ieee.org/reading/ieee/std_public/description/se/610.12-1990_desc.html>

[RAG99] Raggett, D.; "HTML 4.01 Specification", *W3C: World Wide Web Consortium*, 24 December 1999. <http://www.w3.org/TR/html4/>

[RAN79] Randell, B., "Software engineering in 1968", International Conference on Software Engineering, Proceedings of the 4th international conference on Software engineering, Munich, Germany, pp. 1-10, 1979. <http://portal.acm.org/citation.cfm?id=802915>

[RAT00] Rational Software; "Rational Rose 2000e, Rose Extensibility User's Guide", March 2000. <http://www.google.com/search?q=cache:RV_7U1fgmTUC:wwwrational.com/support/documentation/manuals/docset149/ Rational%20Rose/Documentation/Rose_REI_guide.pdf>

[RAT02] Rational Software; "Rational Rose Datasheet: Model-Driven Development with UML", June 2002. <http://www.rational.com/media/products/rose/D185H_Rose.pdf>

[RAT05] Rational Software; "Rational Software", *Rational Software – IBM Division*, (current August 2005). <http://www.rational.com/>

[RAY01] Ray, S. R.; "The Future of Software Integration: Self-integrating Systems", *Vanderbilt Workshop, New Visions for Software Design & Productivity: Research & Applications*, National Institute of Standards & Technology, Gaithersburg, USA, 14 December 2001. <http://www.nitrd.gov/subcommittee/sdp/vanderbilt/position_papers/steven_ray_the_future_of_software.pdf>

[REA03] Reasoning Inc.; "Automated Software Inspection: A New Approach to Increased Software Quality and Productivity", 2003. <http://www.reasoning.com/pdf/ASI.pdf>

[REF05d] Refsnes, H.; "Introduction to DTD", *W3Schools*, (current August 2005). <http://www.w3schools.com/dtd/dtd_intro.asp>

[REF05h] Refsnes, H.; "Introduction to DHTML", *W3Schools*, (current August 2005). <http://www.w3schools.com/dhtml/dhtml_intro.asp>

[RES99] Rescorla, E. et al.; "The Secure HyperText Transfer Protocol", *Network Working Group, The Internet Engineering Task Force*, August 1999. <http://www.faqs.org/rfcs/rfc2660.html>

[RES05] Reser, B.; "subversion.tigris.org", 5 July 2005. <http://subversion.tigris.org/>

[RIC99] Richner, T.; Ducasse, S.; "Recovering High-Level Views of Object-Oriented Applications from Static and Dynamic Information", *IEEE International Conference on Software Maintenance*, Oxford, England, pp. 13-22, August 1999. <http://www.iam.unibe.ch/~scg/Archive/Papers/Rich99aRecoveringViews.pdf>

[RIC00] Richardson, Debra; "ICS 121: Software Methods and Tools", *Lecture Notes, University of California, School of Information and Computer Sciences, Irvine*, 4 March 2000. <http://www.ics.uci.edu/~djr/classes/ics121/PresentationsPDF/Topic1.pdf> or <http://computing.breinestorm.net/software+crisis/>

[RID05] Riddell, Jonathan; "KDE's Switch to Subversion Complete", *KDE Dot News*, 5 May 2005. <http://dot.kde.org/1115285369/>

[RIL03] Rilling, J.; "CONCEPT", *Concordia University, Computer Science Department*, August 2003. <http://www.cs.concordia.ca/CONCEPT/>

[RIL03c] Rilling, J.; "Lecture Notes: Reverse Engineering", *Concordia University, Computer Science Department*, August 2003. <http://www.cs.concordia.ca/~comp6911/html/lecture_notes.html>

[RIV92] Rivest, R.; "The MD5 Message-Digest Algorithm", *Network Working Group, MIT Laboratory for Computer Science and RSA Data Security Inc.*, April 1992. <http://www.faqs.org/rfcs/rfc1321.html>

[RIV00] Riva, C.; "Reverse Architecting: An Industrial Experience Report", *7th IEEE Working Conference on Reverse Engineering (WCRE '00)*, Brisbane, Australia, pp. 42-51, November 2000. <http://ieeexplore.ieee.org/iel5/7172/19289/00891451.pdf?arnumber=891451>

[ROB91] Robertson, P. K.; "A Methodology for Choosing Data Representations, *IEEE Computer Graphics and Applications*, Vol. 11, No. 3, pp. 56-68, May 1991. <http://portal.acm.org/citation.cfm?id=617658>

[ROE05] Roelofs, G.; "(PNG) Portable Network Graphics: Home Site", *SourceForge.net*, 17 July 2005. <http://www.libpng.org/pub/png/>

[ROU97] O'Rourke, Tom; Meszaros, Gerard; "Patterns in software architecture", Atlanta, Georgia, pp. 125-127, January 1997. ISBN: 1-58113-037-6. Addendum to the 1997 ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. <http://portal.acm.org/citation.cfm?id=274567.274593>

[ROY70] Royce, W. W.; "Managing the development of large software systems", *Proceedings of IEEE WESCON*, San Francisco, USA, pp. 1-9, August 1970. Reprinted in *Proceedings of the 9th International Conference on Software Engineering*, pp. 328-338, 1987. ISBN: 0-89791-216-0. <http://portal.acm.org/citation.cfm?id=41801>

[RSA04c] RSA Security; "RSA Security – 4.1.3.10 What are certificates?", 2004. <http://www.rsasecurity.com/rsalabs/node.asp?id=2277>

[RSA04w] RSA Security; "RSA Security – 3.6.5 What are SHA and SHA-1?", 2004. <http://www.rsasecurity.com/rsalabs/node.asp?id=2252>

[RUS02] Russell, S.; "A Novices Tutorial on Subversion", *Germane Software*, 2002. <http://www.germane-software.com/~ser/R_n_R/subversion.html>

[SAI03] Saito, Yaz; "Octopy is a graphical frontend to tla", *HP Labs*, 21 November 2003. <http://www.ysaito.com/octopy/index.html>

[SAM90] Sametinger, J.; "A tool for the maintenance of C++ programs", Software Maintenance, 1990. *Conference Proceedings of 26-29 November 1990*, pp. 54-59, 29 November 1990. <http://www.se.jku.at/publications/pdf/TR-SE-90.02.pdf>

[SAN99] Sand, Espen; "KDE Repository – markup – KDE: kdeutils/khexedit/hextoolwidget.h v1.1", *The KDE Source Repository*, 2 August 1999. <http://webcvs.kde.org/kdeutils/khexedit/hextoolwidget.h?rev=1.1&view=auto>

[SAN99h] Sand, Espen; "KHexEdit - Hex editor for the KDE desktop environment", December 1999. <http://home.online.no/~espensa/khexedit/>

[SAN03] Sand, Espen; "KDE Repository – markup – KDE: kdeutils/khexedit/hextoolwidget.h v1.6", *The KDE Source Repository*, 7 March 2003. <http://webcvs.kde.org/kdeutils/khexedit/hextoolwidget.h?rev=1.6&view=auto>

[SAU04] Sauter, T.; "GNU arch – GNU Project – Free Software Foundation (FSF)", *Free Software Foundation Inc.*, 19 november 2004. <http://www.gnu.org/software/gnu-arch/>

[SAV05] Savannah; "Welcome [Savannah]", *Free Software Foundation Inc.*, (current August 2005). <http://savannah.nongnu.org/>

[SCH01] Schimanski, Stefan; "KDE Repository – markup – KDE: kdeutils/ktimer/ktimer.h v1.1", *The KDE Source Repository*, 19 March 2001. <http://webcvs.kde.org/kdeutils/ktimer/ktimer.h?rev=1.1&view=auto>

[SCH03] Schimanski, Stefan; "KDE Repository – markup – KDE: kdeutils/ktimer/ktimer.h v1.3", *The KDE Source Repository*, 24 October 2003. <http://webcvs.kde.org/kdeutils/ktimer/ktimer.h?rev=1.3&view=auto>

[SCH04h] Schroder, C.; "Harmonize RCS with Monotone", 6 October 2004. <http://www.enterprisenetworkingplanet.com/netsysm/article.php/3418181>

[SCH04r] Schroder, C.; "RCS in the Key of Monotone", *Enterprise Networking Planet,* <http://www.enterprisenetworkingplanet.com/netsysm/article.php/3421371>, 13 October 2004.

[SEG02] Software Engineering Group; "Fujaba Homepage", *University of Paderborn*, December 2002. <http://www.uni-paderborn.de/cs/fujaba>

[SEI02] Software Engineering Institute; "The Dali Architecture Reconstruction Workbench", *Carnegie Mellon University*, September 2002. <http://www.sei.cmu.edu/ata/products_services/dali.html>

[SEL04] Selic, B.; "UML 2.0 Superstructure specification", *Object Management Group*, October 2004. <http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-02.pdf> or <http://www.uml.org/#UML2.0>

[SEN94] Senay, H.; Ignatius, E.; "A Knowledge-Based System for Visualization Design", *IEEE Computer Graphics and Applications*, Vol. 14, No. 6, November 1994, pp. 36-47. <http://portal.acm.org/citation.cfm?id=617929>

[SGI00] SGI, "Download STL source code", 8 June 2000. <http://www.sgi.com/tech/stl/download.html>

[SHA02] Shasha, D.; Wang, Jason; Giugno, Rosalba; "Algorithmics and Applications of Tree and Graph Searching", *PODS 2002*, *Courant Institute*, *NYU*, 70 slides, 2002. <http://www.cs.nyu.edu/cs/faculty/shasha/papers/pods2002.ppt>

[SHA03] Shaikh, M. et al.; "Version Management Tools: CVS to BK in the Linux Kernel", *Department of Information Systems*, *London School of Economics*, 17 August 2003. <http://opensource.mit.edu/papers/shaikhcornford.pdf>

[SHU96] Shull, F.; Melo, W.L.; Basili, V.R.; "An Inductive Approach for Discovering Design Patterns from Object Oriented Software System", *Technical Report CS-TR-3597*, University of Maryland, College Park, Maryland, October 1996. <http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/CS-TR-3597.pdf>

[SIC00] Sick, O.; "Cover Pages: Chess Markup Language (ChessML)", 7 October 2000. <http://xml.coverpages.org/chessML.html>

[SIL00] Silverstein, D.C.; "Writing Larger Programs", *University of California at Berkeley*, March 2000. <http://www.csua.berkeley.edu/~dans/TAing/CS3/Sp00/handouts/week4.html>

[SIN05] Singer, T.; "[SmartCVS]", *SmartCVS*, 1st August 2005. <http://www.smartcvs.com/>

[SLE05] Sleepycat Software Inc.; "Sleepycat Software: Products: Berkeley DB v4.3.28", (current August 2005). <http://www.sleepycat.com/products/db.shtml>

[SMI88] Smith, R.; "mod-gnu-diff/ChangeLog", *mod-gnu-diff archives*, 3 November 1988. <http://paxutils.progiciels-bpi.ca/showfile.html?name=admin/torture/mod-gnu-diff/ChangeLog>

[SMI94] Smith, R.; "gnu/diffutils/diff3.c", *Free Software Foundation Inc.*, Tux.org, 1994. <http://www.tux.org/pub/sites/ftp.bitmover.com/gnu/diffutils/diff3.c>

[SNE00] Snelting, G.; Tip, F.; "Understanding Class Hierarchies Using Concept Analysis", *ACM Transactions on Programming Languages and Systems*, Vol. 22, No. 3, May 2000. <http://portal.acm.org/citation.cfm?id=353940>

[SPA05] Spaans, J.; "LKML: Headers for Root index", *ISP Services B.V.*, (current August 2005). <http://lkml.org/lkml/>

[STA02] Stallman, R.; Andrews, J.; "Linux: Richard Stallman On GNU/Linux And BitKeeper", *KernelTrap.org*, 21 May 2002. <http://kerneltrap.org/node/204>

[STL04a] STLport Consulting, "STLport v4.6.2", 10 April 2004. <http://www.stlport.org/archive/STLport-4.6.2.tar.gz>

[STL04j] STLport Consulting, "STLport v3.2.1", 10 January 2004. <http://www.stlport.com/archive/STLport-3.2.1.tar.gz>

[SUN02] Sundstrom, P.; "hDiff: Version 2.1.0", *Ginini Technologies Software*, 16 August 2002. <http://www.ginini.com/software/hdiff/>

[SUN04c] Sun Microsystems, Inc.; "javac – Java programming language compiler", June 2004. <http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/javac.html>

[SUN04d] Sun Microsystems Inc.; "Javadoc Tool Home Page", Febuary 2004. <http://java.sun.com/j2se/javadoc/>

[SYS00a] Systä, T., et al.; "Analyzing Java Software by Combining Metrics and Program Visualization", *Proceedings of the Conference on Software Maintenance and Reengineering (CSMR'99)*, Brisbane, Australia, pp. 214-223, November 2000. <http://www.cs.tut.fi/~tsysta/papers/40_systa.pdf>

[SYS00b] Systä, T.; "Understanding the Behavior of Java Programs", *7th IEEE Working Conference on Reverse Engineering (WCRE '00)*, Brisbane, Australia, pp. 214-223, November 2000. <http://www.cs.tut.fi/~tsysta/papers/129_systa.pdf>

[TAY03] Taylor, C.; "Observation: all", 31 July 2003. <http://www.taylors.org/cim/observed/030731/TortoiseCVS/screenshot1.png>

[THO04] Thomas, Mark; "The Y Window System – Download Y", 21 February 2004. <http://www.y-windows.org/downloads.html>

[THO05] Thompson, H. et al.; "W3C XML Schema", *W3C: World Wide Web Consortium,* 28 June 2005. <http://www.w3.org/XML/Schema>

[TIC85] Tichy, W. F.; "RCS: a system for version control", *Software Practical Experience*, Vol. 15, No. 7, pp. 637-654, 1985. <http://portal.acm.org/citation.cfm?id=4202>

[TOG01] TogetherSoft; "Together Control Center 5.5", 2001. <http://www.togethersoft.com/>

[TRI98] Tripp, L. L. et al., "IEEE Std. 830-1998 (Revision of IEEE Std 830-1993)", *IEEE Recommended Practice for Software Requirements Specifications*, 25 June 1998. Software Engineering Standards Committee of the IEEE Computer Society. ISBN: 0-7381-0332-2. <http://web.nps.navy.mil/~nschneid/is3020/PDF/830-1998.pdf>

[TRO05] Trolltech AS; "Trolltech – Qt Product Overview – single source C++ cross-platform application development for Windows, Linux, Mac", (current August 2005). <http://www.trolltech.com/products/qt/>

[TRO05i] Trolltech AS; "Qt 2.3.10 Toolkit – group index – I/O Classes", (current August 2005). <http://doc.trolltech.com/2.3/io.html>

[TRO05n] Trolltech AS; "Qt 3.3.4: Non-GUI Classes", (current August 2005). <http://doc.trolltech.com/3.3/tools.html>

[TRO05t] Trolltech AS; "Qt 2.3.10 Toolkit – group index – The Tool Classes", (current August 2005). <http://doc.trolltech.com/2.3/tools.html>

[TOR95] Torvalds, L.; "Re: CVS, Linus, and us", *Linux Kernel Mailing List*, 11 December 1995. <http://www.uwsg.iu.edu/hypermail/linux/kernel/9602/0800.html>

[TOR05] Torvalds, L.; "Re: Kernel SCM saga...", *Linux Kernel Mailing List*, 7 April 2005. <http://lkml.org/lkml/2005/4/8/9>

[TSP05] The Software Patch; "The Software Patch - Download site for patches, upgrades, service packs and hardware drivers", (current August 2005). <http://www.softwarepatch.com/>

[TRI05] Tridgell, A.; "SourcePuller 0.1", *SourceForge.net*, 21 April 2005. <http://sourceforge.net/project/shownotes.php?group_id=137038&release_id=322303>

[TUR05] Turner, D.; "GNU General Public License – GNU Project - Free Software Foundation (FSF)", *Free Software Foundation Inc.*, 7 June 2005. <http://www.gnu.org/copyleft/gpl.html>

[UHL96] Uhl, J. and H. Müller; "What is Rigi?", *University of Victoria, Computer Science Department*, July 1996. <http://www.rigi.csc.uvic.ca/rigi/manual/node3.html>,

[UNI05] Unicode Inc.; "What is Unicode?", 19 May 2005. <http://www.unicode.org/standard/WhatIsUnicode.html>

[UNI05i] Unicode Inc.; "Unicode 3.0, Chapter 1 – Introduction", 19 May 2005. <http://www.unicode.org/book/uc20ch1.html>

[USC05] US-CERT.; "Technical Cyber Security Alerts", *United States Computer Emergency Readiness Team, U.S. Department of Homeland Security,* (current August 2005). <http://www.us-cert.gov/cas/techalerts/index.html>

[VAS05] VA Software Corporation; "SourceForge.net: Welcome", *OSTG: Open Source Technology Group Inc.*, (current August 2005). <http://www.sourceforge.net/>

[WAL02] Wallington, J. P.; "Comparing and Merging Files: Overview ", *Free Software Foundation Inc.*, 9 May 2002. <http://www.gnu.org/software/diffutils/manual/html_node/Overview.html>

[WAN05] Wang, Yuan; "X-Diff: An XML Diff Tool", *University of Wisconsin Madison*, 4 February 2005. <http://www.cs.wisc.edu/~yuanwang/xdiff.html>

[WAT04] Watters, P. A.; "Software Configuration Management: ITEC823 – Week 12", 3 June 2004. <http://www.comp.mq.edu.au/units/itec823/lectures/Week11%20-%20Config%20Management.pdf>

[WEI98] Weintraub, D.; "The Not-So-Official ClearCase Page: Definitions", *Weintraub World*, 11 November 1998. <http://clearcase.weintraubworld.net/cc.def.html>

[WHI02] Whittaker, J. A.; "How to Break Software: A Practical Guide to Testing", *Addison-Wesley*, 178 pages, May 2002. ISBN: 0-2017-9619-8. <http://www.amazon.com/exec/obidos/tg/detail/-/0201796198/>

[WIK05d] *Wikipedia*; "diff", (current August 2005). <http://en.wikipedia.org/wiki/Diff>

[WIK05n] *Wikipedia*; "non-disclosure agreement", (current August 2005). <http://en.wikipedia.org/wiki/Non-disclosure_agreement>

[WIK05p] *Wikipedia*; "patch (Unix)", (current August 2005). <http://en.wikipedia.org/wiki/Patch_(Unix)>

[WIL05] Williams, C. C.; Hollingsworth, J. K.; "Automatic Mining of Source Code Repositories to Improve Bug Finding Techniques", *IEEE Transactions on Software Engineering*, Vol. 31, Issue 6, pp. 466-480, June 2005. <http://ieeexplore.ieee.org/iel5/32/31460/01463230.pdf?arnumber=1463230>

[WIN03] Winer, Dave; "XML-RPC Specification", *UserLand Software Inc.*, 30 June 2003. <http://www.xmlrpc.com/spec>

[WON98] Wong, K.; "Rigi User's Manual – Version 5.4.4", *University of Victoria, Computer Science Department*, 30 June 1998. <http://ftp.rigi.csc.uvic.ca/pub/rigi/doc/rigi-5.4.4-manual.pdf>

[WOO99] Woods, S.G., S.J. Carrière, and R. Kazman; "The Perils and Joys of Reconstructing Architectures", *SEI Interactive, Software Engineering Institute*, September 1999. <http://www.sei.cmu.edu/news-at-sei/columns/the_architect/1999/September/architect-sep99.pdf>

[WOO00] Wood, L. et. al.; "Document Object Model (DOM) Level 1 Specification (Second Edition)", *W3C: World Wide Web Consortium*, 2000. <http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/>

[WZI05] WinZip International LLC; "Winzip - Increased Zip File Capacity", (current August 2005). <http://www.winzip.com/wzdic.htm>

[YAM04] Yamanaka, A.; "CVS-SSH2 Plug-in for Eclipse", *JCraft Inc.*, *SourceForge.net*, 5 September 2004. <http://www.jcraft.com/eclipse-cvsssh2/>

[YAS05] Yasskin, J.; "What is Arch? – Arch Wiki", *Free Software Foundation Inc.*, 19 January 2004. <http://wiki.gnuarch.org/moin.cgi/What_20is_20Arch_3f>

[ZHA95] Zhang K and Shasha D.; "Chapter 14. Approximate tree pattern matching", 28 pages, 1995. Published in "Pattern matching in strings, trees and arrays", A. Apostolico and Z. Galil (editors), *Oxford University Press*, pp. 341-371, 1997. <http://citeseer.ist.psu.edu/shasha95approximate.html>