An investigation of the use of software development environments in the industry

By

Ping An

Institute of Technology

Linköping University

Submitted to the Department of Computer and Information Science at Linköping University in partial fulfillment of the requirements for the degree of International Master of Science

Department of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden

Linköping 2004

An investigation of the use of software development environments in the industry

by

Ping An

LITH-IDA-EX--04/085—SE

Institute of Technology
Linköping University

2004-05-28

Supervisor: Magnus Bång

Examinator: Henrik Eriksson

Department of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden

Linköping 2004

Abstract

Software engineering tools are being used in the industry in order to improve the productivity and the quality of the software development process. The properties of those tools are being perceived to be unsatisfactory. For example, researchers have found that some problems are due to deficient integration among the tools. Furthermore, a continuing problem is that there is a gap between the IT education and real demand of tool-skills from IT industry. Consequently, knowledge is needed of the properties of software development tools as well an understanding of demanded tool-skill from the industry.

The purpose of this study is to survey commercial software development environment (SDEs) that are used today in professional software engineering and discuss their advantages and disadvantages. A secondary goal of the study is to identify the actual requirements from the industry on the IT-education.

A questionnaire was sent out to 90 software developers and IT managers of 30 IT companies in Sweden. The results of the survey show that IT companies, for most part, use SDEs from commercial software vendors. Respondents report that common problems of the SDEs are the following: bad integration among the tools, problems to trace software artifacts in the different phases of the programming cycle, and deficient support for version control and system configuration. Furthermore, some tools are difficult to use which results in a time-consuming development process.

We conclude that future software development environments need to provide better support for integration, automation, and configuration management. Regarding the required tool-skills, we believe that the IT education would gain from including commercial tools that cover the whole software product lifecycle in the curriculum.

Keywords

Software development environments, Software life cycle, Survey, Software development tools, CASE tools, Integration, Integrated software development environment

Acknowledgements

I would like to express my gratitude to supervisor Magnus Bang at the Department of Computer and Information Science, Linköping University. I also want to acknowledge my examiner Henrik Eriksson and Jalal Maleki, the coordinator of International M. Sc. Program at the Department of Computer and Information Science at Linköping University.

……………………………
Linköping, 2004-05-28

Ping An

Contents

1. Introduction

Various software development tools are being employed in the industry to improve the productivity and quality of the software development process. Computer-aided software engineering tools [Norman & Nunnamaker 1989] are considered to be a crucial intermedium for promotion of software engineering technology transfer between academia and industry. However, the adoption and actual industrial use of these tools still appears to be low. It seems that the tools lack critical properties that may have an impact on its widespread adoption. A possible consequence of this situation is that the quality of the final software products may suffer, and that the software development process becomes unproductive and unstructured. According to Reiss (1996) the most common problems of SDEs are due to unsatisfactory integration among the development tools. Consequently, we need to get a better understanding of problems perceived by actual users of SDEs, to develop better tools. Needed are studies of actual SDEs that are used today in commercial software development process as well as inquiries into the properties of those tools. Additionally, we must to address the IT education so that it better apply to the requirements of the IT industry.

1.1 Purpose and aim

The purpose of this study is to investigate what commercial SDEs and tools that are used in commercial software engineering today and to discuss their advantages and disadvantages of them. The study also aims to close the gap between university-level IT education and actual requirements by the industry. The study will try to answer the following five questions:

Q1. What are the current software development environments that are used by professional software developers?

Q2. What are the advantages and disadvantages of the software development environments?

Q3. Are professional software developers pleased with their software development environments?

Q4. What kind of support is needed in future software development environments?

Q5. What needs to be addressed in the university-level IT curriculum to more appropriately suit the needs of commercial software developers regarding SDEs?

1.2 Outline of the study

The theory and literature about software development engineering is reviewed in the first chapter such as the different phases of software engineering and software development tools aimed to support that process. Second, the methodology and research strategies are discussed. Third, the results of the study are presented. In the final part of the thesis, we present a discussion on the properties of SDEs and the relationship to IT education. Eventually, we conclude the thesis and provide some recommendations to improve the IT curriculum.

1.3 Delimitations

Only commercial software engineering tools have been studied. The scope is limited to the tools that are used in the development and integration phases of the software development life cycle.

2. Background

In this chapter, we review the theory and literature of software development engineering, and discuss the context of software development engineering in relation to tool-usage in each software life cycle.

2.1 The software development life cycle

Institute of Electrical and Electronics Engineers (IEEE) has developed a standard for the software development process (IEEE 1074-1997). Their definition of the software development process has been widely accepted in both industry and academia:

> *The life cycle of software system is normally defined as the period of time that starts when a software product is conceived and ends when the product is no longer available for use. This cycle is based on IEEE Std 610.12-1990 and consists of a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and retirement phase [IEEE Std 1074-1997]*

In this report, we follow parts of the IEEE 1074-1997 standard, and use the following six stages of the development process in our discussion:

- ♦ Software requirement and analysis
- ♦ Software design
- ♦ Software construction
- ♦ Software testing
- ♦ Software configuration management
- ♦ Software documentation development

Software requirement and analysis

In the software requirements phase engineering are concerned with the acquisition, analysis, specification, validation, and management of software requirements. Requirements analysis is the process of analyzing the requirements to detect and resolve conflicts between requirements, discover the bounds of the system and how it must interact with its environment, elaborate system requirements to software requirements [IEEE Std 1074-1997].

Software design

Software design is the activity where software requirements are analyzed in order to produce a description of the internal structure and organization of the system that will serve as the basis for its construction. There are two basic activities: first, the overall software architectural design is specified and the result is descriptions of the top-level organization of the system and its various components. That is, how the system is organized into components as well as the interfaces between these components are described. Second, each component is sufficiently described to allow for it's coding [IEEE Std 1074-1997].

Tools for software requirement and design help analysts to better express users requirements, propose design solutions and analyze information for consistency, completeness and integrity [Almstrum 2004]. The tools support analysts in the following activities:

- Drawing, changing, and manipulation of diagrams
- Generating reports and documentation
- Develop prototypes for the purpose of requirement discovery or verification
- Model and describe a current information system
- Model and describe the requirements for a new information system
- Gather & structure requirements
- Generate stub-code

Software construction

Software engineering: is the development of meaningful software through coding, validation, and testing by a programmer. Software construction is, according to IEEE, the following:

> *Software construction is a fundamental act of software engineering: the construction of working meaningful software through a combination of coding, validation, and testing (unit testing) by a programmer [IEEE 1074-1997].*

Programming tools can help applications programmers and other system implements improve their productivity and quality. These tools are intended for detailed design and systems implementation and help programmers more quickly generate applications software [Almstrum, 2004]. These tools help programmers to test and debug code. Furthermore, they can be used to generate special parts of the system like the graphical user interface and even generate complete applications.

Software testing

Testing conducted in an operational environment is used to determine whether a system satisfies its acceptance criteria (i.e., initial requirements and current needs of its user) and to enable the customer to determine whether to accept the system [IEEE Std 1012-1998]. There are two main tasks that are conducted in this phase:

- Validation

Validation is the demonstration that the software implements each of the software requirements correctly and completely. In other words, the "right product was built" [IEEE Std 1012-1998].

- Verification
  Verification is the activity that ensures the work products of a given phase fully implement the inputs to that phase, or "the product was built right" [IEEE Std 1012-1998].

Examples of testing tools are test generators; test execution frameworks, test evaluation tools, test management tools, and performance analysis tools [Aaby, 2004].

Software configuration management

System configuration management (SCM) is used to handle the different components of a system and in the final phase assemble it to a functional product. SCM is concerned with the identification, organization and control of the software components known as configurable items in a software system under a parallel development environment [Chan & Hung 1997].
SCM tools can support configuration manager to defect, enhancement, issue and problem tracking; version management; release and build [Aaby, 2004].

Software documentation development

IEEE classified the documentation activities into two categories:

1. Implement documentation:
   This activity includes the design, preparation, and maintenance of documentation. Those documents that are identified in the documentation-planned information shall be formulated in terms of audience, approach, content, structure, and graphics.

2. Produce and distribute documentation:
   This activity shall provide the intended audience with the needed information that is collected in the document, as specified in the documentation planned Information. Document production and distribution can involve electronic file management, paper document reproduction and distribution, or other media handling techniques.

Table 1. Activities in the software development process.

Development process	Requirement	Define and Develop Software Requirements
		Define Interface Requirements
		Prioritize and Integrate Software Requirements
	Design	Perform Architectural Design
		Design Data Base (If Applicable)
		Design Interfaces
		Select or Develop Algorithms (If Applicable)
		Perform Detailed Design
	Construction	Create Executable Code
		Create Operating Documentation
		Perform Integration
Integration Processes	Verification and Validation (testing)	Plan Verification and Validation
		Execute Verification and validation Tasks
		Collect and Analyze Metric Data
		Plan Testing
		Develop Test Requirements
		Execute Tests
	Software Configuration Management	Plan Configuration Management
		Develop Configuration Identification
		Perform Configuration Control
		Perform Status Accounting
	Document Development	Plan Documentation
		Implement Documentation
		Produce and Distribute Documentation
	Training	Plan the Training Program
		Develop Training Materials
		Validate the Training Programs
		Implement the Training Program

2.2 Software development environments

Software development environments (SDEs) can be defined as *a suite of computer-aided tools designed to simplify software development and enhance software developers' productivity*. These tools are designed to support different development phases. A key idea of SDEs is that every tool in the environment can communicate, even thought they are developed to support different phases of the software development life cycle. IEEE has defined software engineering environments as:

> *The hardware, software, and firmware used to perform a software engineering effort. Typical elements include computer equipment, compilers, assemblers, operating systems, debuggers, simulators, emulators, test tools, documentation tools, and database management systems [IEEE Std 1348-1995].*

Supporting the development process also means supporting the development and maintenance of all kinds of documents such as requirements specifications, software architecture descriptions, code listings, manuals, and technical documentations. The ultimate goal of such a tool set is to improve the quality of the final product, to support reuse in and across software projects, and, last but not least, to free developers from routine work [Engels et al. 1992].

CASE tools

Starting with the construction of syntax-directed editors for programming languages in the 70's, a technology has emerged that enables building not only editors supporting particular kinds of graphical and textual input, but all sorts of document tools, including static analyzers, incremental compilers, browsers, debuggers, and so on [Engels et al. 1992].

Computer-aided software engineering tools (CASE tools) are used to support the software engineering process. This class of tools includes support activities such as software design, requirements specification, code tracing, code production, testing, document generation [IEEE Std 1348-1995].

González (2004) has stated that the real power of CASE can only be achieved through integration. Ideally, CASE tools should:

- Provide a way to share information between the tools in the environment
- Allow detecting changes in the information elements related
- Provide the control to different versions
- Allow direct access to any of the tools
- Allow integration of procedures and tools in a structure for decomposition
- Keep consistency in the look and interaction with the interface
- Support communication among software engineers
- Keep tools and techniques together to improve the process and product.

Integrated software development environments

Integrated software development environments (ISDE) are a suite of CASE tools, which aim to support the tasks of each software development phase. In Figure 1, we see part of the waterfall model [Pfleeger 2001, Sommerville 2001], and visualize how different CASE tools support each development process in the waterfall.

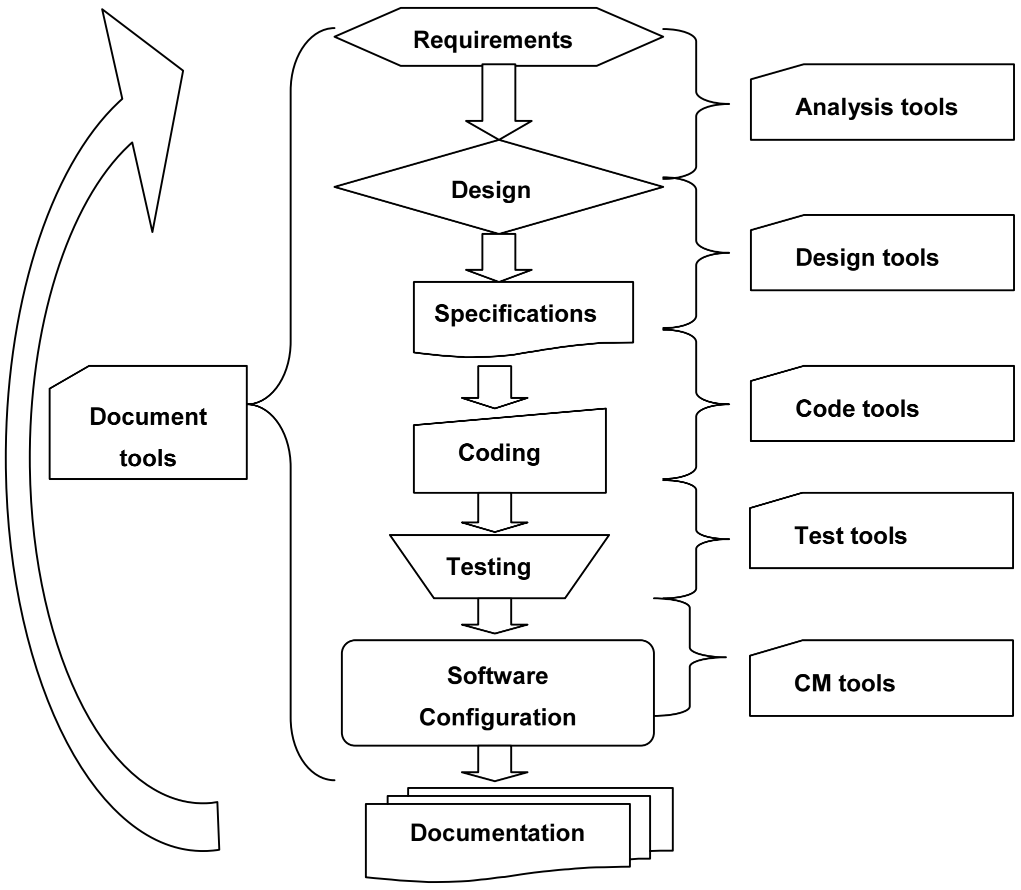


Figure 1. Integrated software development environments

Integration means that tools can communicate and collaborate with other tools used within different development phases. What is most important is that the tools can support at least two consecutive phases of software development, and communicate with the tools that are involved in these consecutive phases. Data generated from a tool in an early phase should be re-used in the following phase by another development tool. For example, an analysis tool could generate data from the requirements phase and the results from that activity could be used as input to the design tools in a later phase. Similarly, documents generated by the tools from each phase, should be available to tools used in the consecutive development phases.

3. Method

In this chapter, we explain the main research strategies and methods used in this work.

3.1 Research Strategy

Qualitative research methods

Qualitative research was developed in the social sciences to enable researchers to understand social and cultural phenomena. Examples of qualitative methods are action research, case study research, and ethnography. Qualitative data sources include observation and participant observation (fieldwork), interviews and questionnaires together with the researcher's impressions and reactions. [MISQ Discovery, Dec. 1998]. Qualitative methods are often used when it is not meaningful to express the collected data in numbers [Lekvall & Wahlbin, 1993].

Surveys

Surveys can be divided into two broad categories: the questionnaire and the interview [Trochim, 2003]. Questionnaires are paper-and-pencil instruments that the respondent completes. Interviews can be said to be data collected and completed by the interviewer based on what the respondent says. It is difficult to draw a firm line between a questionnaire and an interview. For instance, some people think that questionnaires always ask short closed-ended questions while interviews always ask broad open-ended ones. However, questionnaires can also have with open-ended questions (although they do tend to be shorter than in interviews) [Trochim 2003].

3.2 Data collection

Primary and secondary data sources

Data can be classified into two broad types -primary and secondary data. Primary data is information collected directly from the source. Secondary data, on the other hand, is collected through indirect sources (and possible interpreted by others) [Lekvall & Wahlbin 1993]. Primary data can be collected through interviews, experiments, or observations from real social interactions [ibid.]. Secondary data is collected and compiled for other purposes. Examples of secondary data sources are Internet, books, articles and annual reports. Regardless what kind of source is used it is very important to consider precision, validity, reliability and relevance of the data in relation to the purpose and problem statements of the research [Lundahl & Skärvad, 1999].

In this study, we have used a survey to understand the use of software engineering technology used in the development process. The collected data from the survey is considered to be primary data.

Questionnaire design

The questionnaire was divided into two parts. The first part (Section A) included general questions such as the characteristics of individuals and their organizations. The second part (Section B) intended to cover specific topics related to the respondents tools use.

Survey implementation

The survey is limited in companies in Sweden. The respondents’ organizations are commercial IT companies that works in the IT and telecom sectors, primarily. We sent out 90 copies of the questionnaire form. Data collection started in December 2003, and was ended in January 2004.

4. Results

In this chapter, we will present the survey results. The chapter is divided into two parts. Part one discusses the background data of the study such as response rate and characteristics of respondents and organizations. Part two covers software development environments that are used today.

4.1 Response rate and characteristics of respondents

The questionnaire was sent out to 90 software developers and IT managers of 30 IT companies in Sweden. The response rate was 60 percent; we got 51 questionnaires back from 22 companies. The Figure 2 shows the response rate.

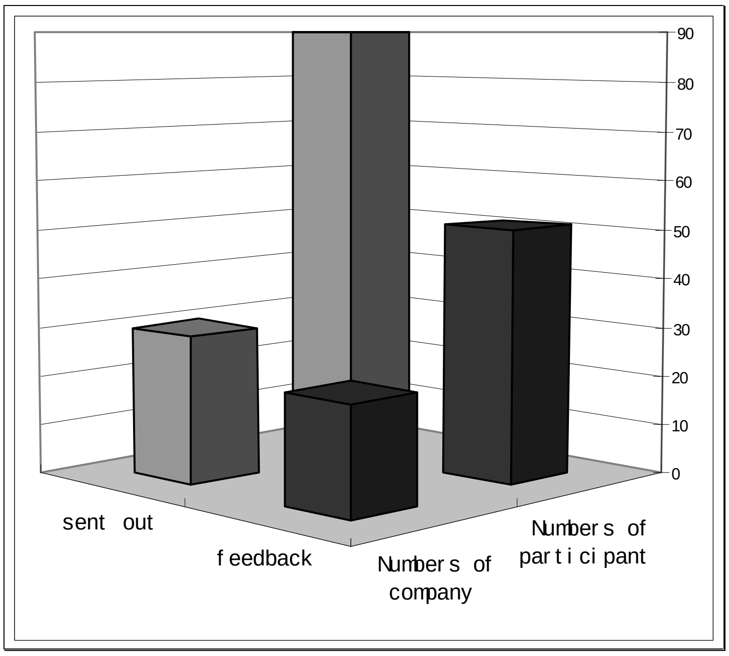


Figure 2. The response rate of the survey

Most respondents were programmers. Figure 3 indicates that more than half of the respondents are programmer. On the other hand, too few people are work as analyst, designer, tester, and technical writer. Most of the respondents have 5 to 10 years of software development experience, and work in the telecom industry.

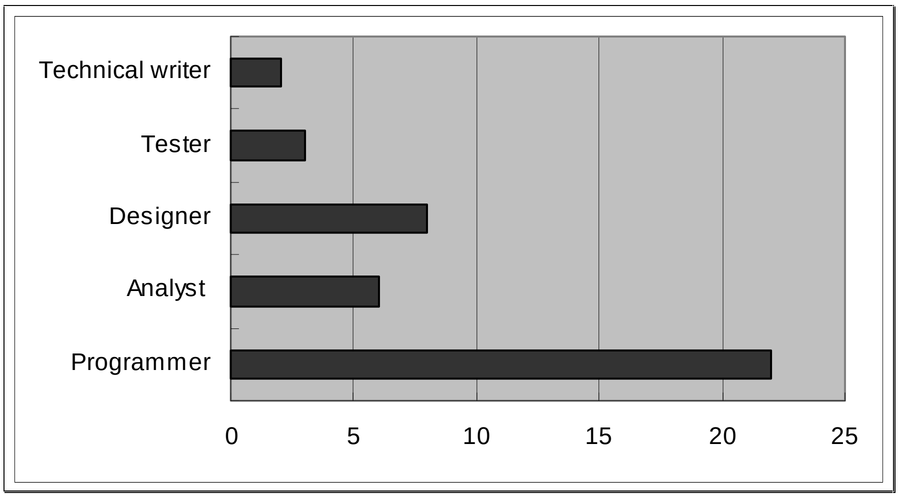


Figure 3. Job title of respondents

4.2 Today's software development environments

We found that the respondents used 57 different tools-suites. Out of these tools, 36 were commercially developed tool-suites, and 21 tools-suites were developed as part of an open source effort. For a description of these tools, see Appendix 2.

Popular platforms for software development

The respondents state that the Microsoft is the most popular brand, which could indicate that the Microsoft Windows is the leading platform in the software development industry. Additionally, more and more developers appear to use open source software such as GNU, which are mainly based on Unix or Linux platform.

Used programming languages

We found that C and C++ are most widely used programming languages today.
Figure 4 shows the distribution of the programming languages. As we can see, Java is also a popular language, however, still not as common as the C family. Although not present in the figure, we also found that programmers use other languages such as Cobol, Fortran, JAVA Script, HTML/XML, squeak, Ada, PHP (scripting), ANTLR, and Python, etc.

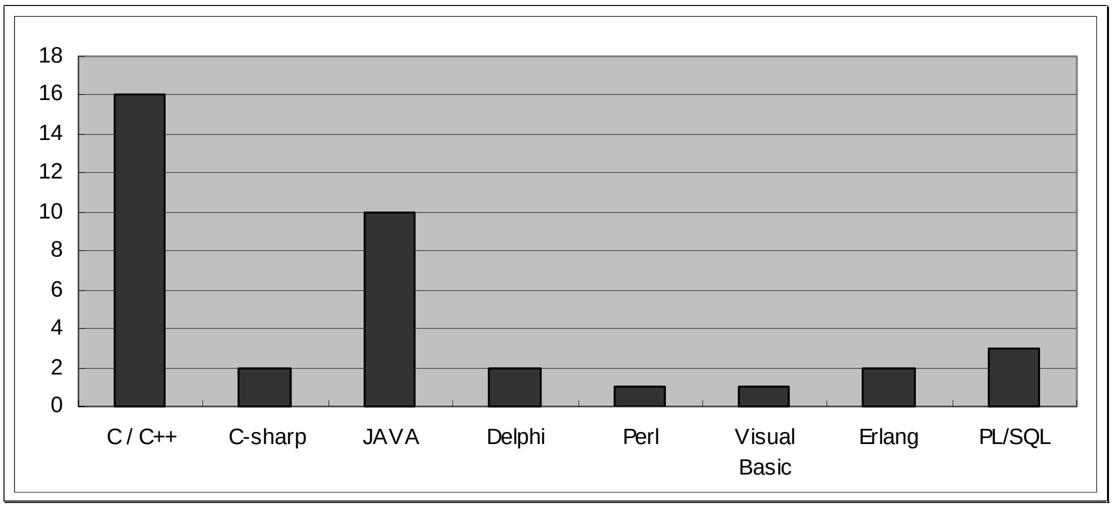
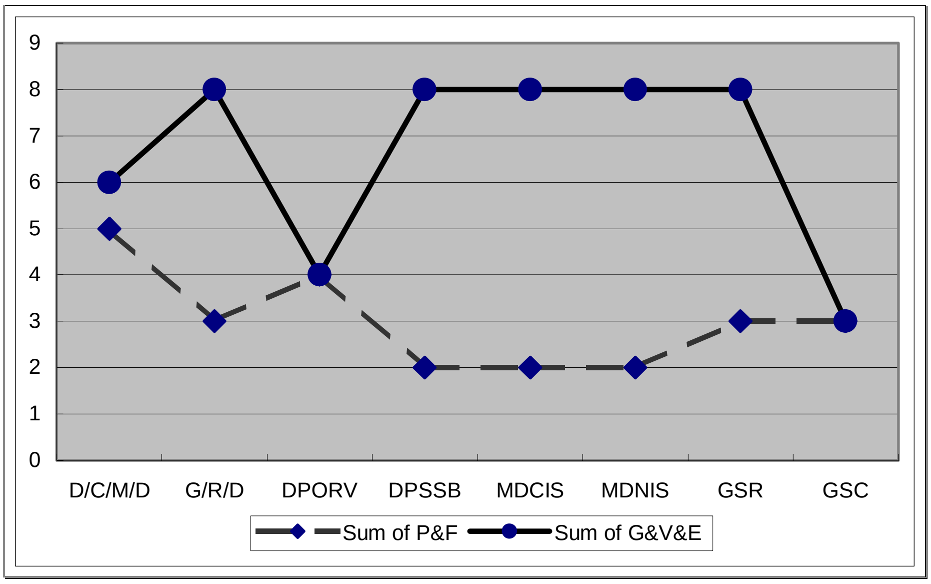


Figure 4. Programming languages used by respondents

Requirement analysis and design tools

Regarding tools for requirement analysis and design, we found that our respondents generally find these tools to be very supportive. As shown in Figure 5-1, the requirement analysis and design tools aid developers to define the project scope and system boundaries. They also state that the tool have excellent capability to model and describe a current/new information system. RADT are lack of the functions of developing prototypes for the purpose of requirement discovery and verification, as well as many defects in groupware &collaboration. Figure 5-2 shows that the developers find the tools easy to use. The distance between the two points in two lines respectively, the larger the better



SUM of G&V&E: very good and excellent levels

SUM of P&F: Poor and Fair levels

Figure 5-1. The perceived capabilities of requirement analysis and design tools

D/C/M/D: Drawing, changing, and manipulation of diagrams

G/R/D: Generating reports and documentation

DPORV: Develop prototypes for the purpose of requirement discovery or verification

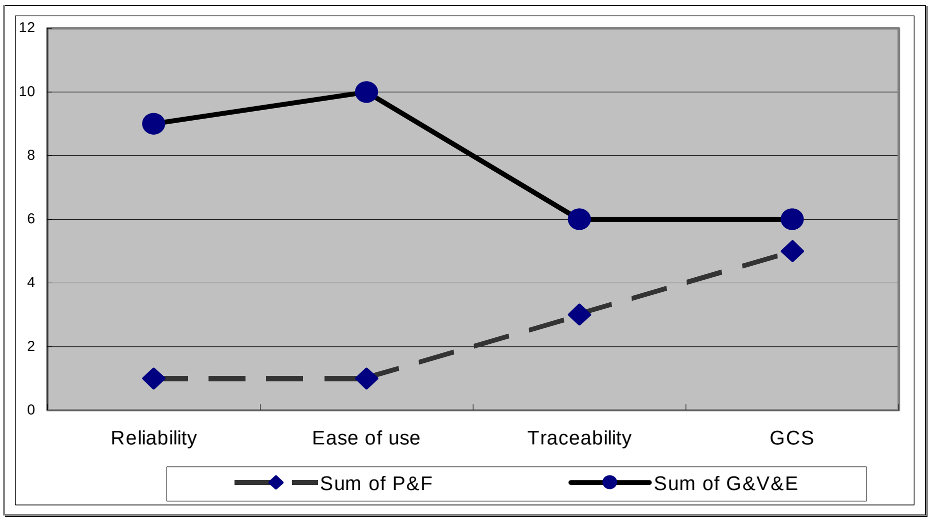
DPSSB: Define project scope and system boundaries

MDCIS: Model and describe a current information system

MDNIS: Model and describe the requirements for a new information system

GSR: Gather & structure requirement

GSC: Generate stub-code



SUM of G&V&E: very good and excellent levels

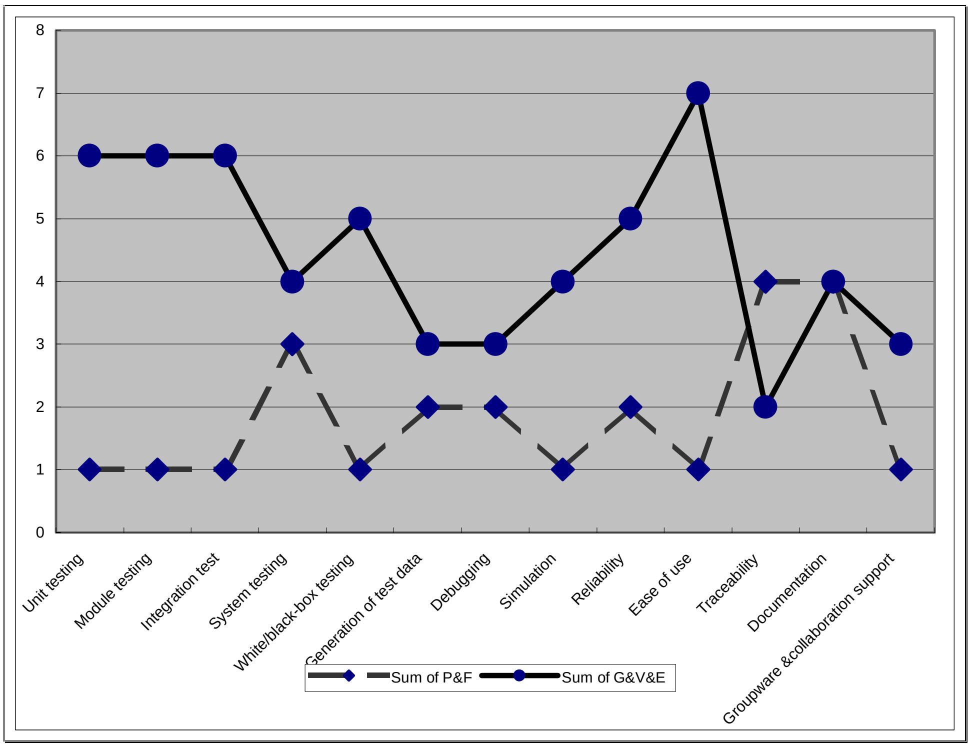
SUM of P&F: Poor and Fair levels

GCS: Groupware &collaboration support

Figure 5-2. The perceived capabilities of requirement analysis and design tools

Test tools

Developers are generally displeased with their test tools, in particular, tools for system testing. However, as Figure 6 show, the respondents are pleased with their tools for unit testing, module testing, and integration testing. The test tools are easy to use, but traceability capability is a significant lack. The distance between the two points in two lines respectively, the larger the better



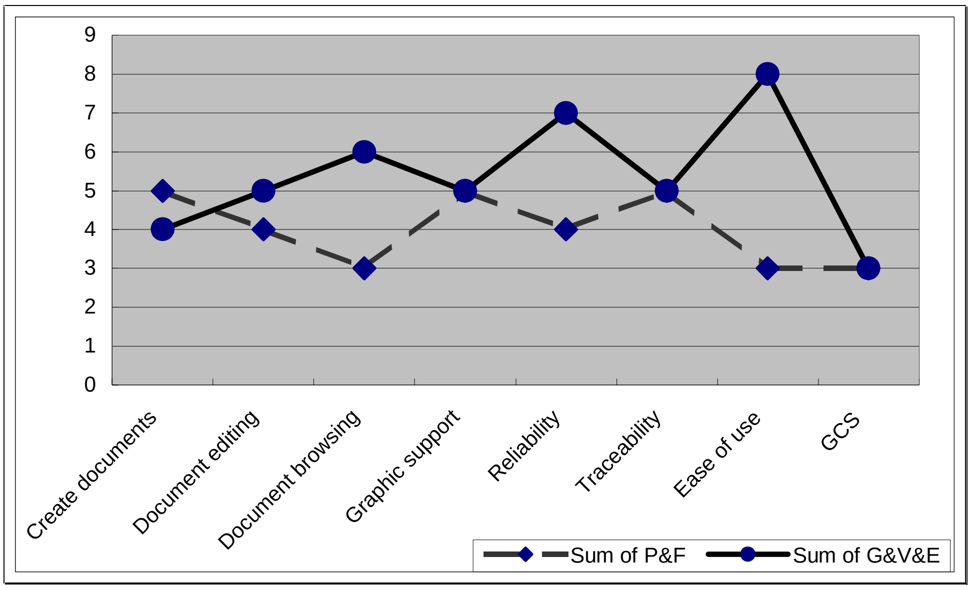
SUM of G&V&E: very good and excellent levels

SUM of P&F: Poor and Fair levels

Figure 6. The perceived capabilities of the test tools

Documentation tools

There are few specially designed tools for documentation of software projects. Figure 7 shows that the documentation tools are ease of use, but they lack basic functions to visualize the software project and means to trace between software artifacts. The tools do not support appropriately links between code and the related documentation. The tools are insufficient for editing and browsing complex and large documents. The distance between the two points in two lines respectively, the larger the better.



SUM of G&V&E: very good and excellent levels

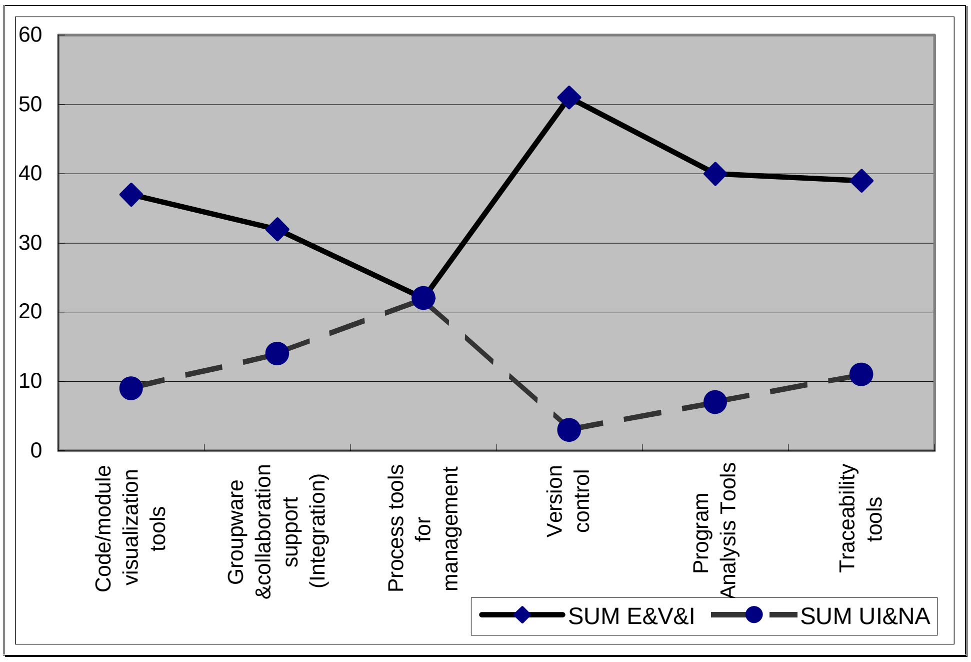
SUM of P&F: Poor and Fair levels

Figure 7. The perceived capabilities of the documentation tools

Needed support and user satisfaction

We asked what support the developers needed in the future. Figure 9 indicates that version control is a highly needed feature of future SDEs. Needed are also program analysis tools, traceability tools, and code/module visualization tools. Moreover, we found that the respondents seem not to be aware of the importance of groupware and collaboration support, which is surprising. Furthermore, several respondents state that they need some special features such as model-based development, code generation, performance measurements, and more speedy environments.

In the figure, the distance between the two points in two lines respectively, the larger the better.



SUM of E&V&I: Perceived by the respondents as being extremely important, very important and important.
SUM of UI&NA: Unimportant and NA

Figure 9. Tools needed in the future

The developer are generally pleased with their tools but not fully content. Figure 9 shows the distribution of the user satisfaction.

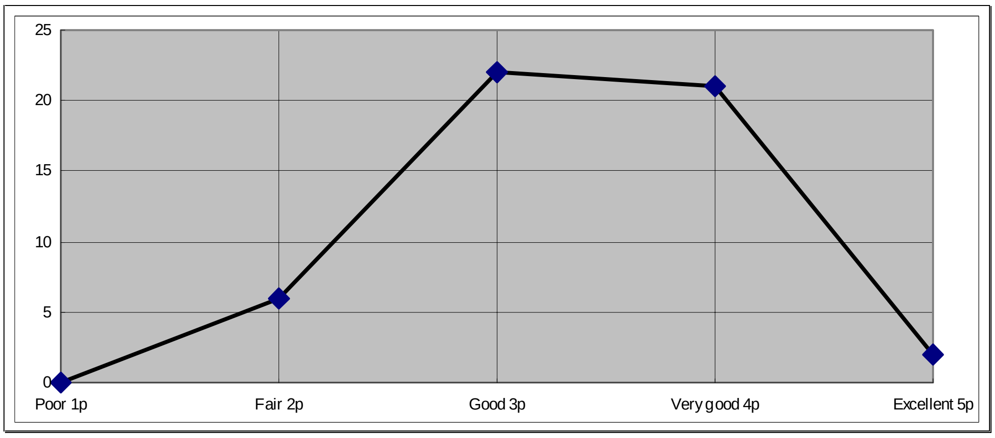


Figure 10. Degree of user satisfaction for software development environments

5. Discussion

This chapter discusses what SDEs that are used today, and their advantages and disadvantages as it were perceived by respondents. We also discuss requirements on future the SDEs as well as pondering issues of university-level IT education to better suit the needs of the IT industry.

5.1 Commercial software development tools used today

Requirement analysis and design tools

Most commercial requirement analysis and design (RAD) tools are well known to the respondents, such as tool-suits, IBM Rational Rose. The designers think that current RAD tools provide first-rate support to go from requirements to actual code, and the tools also aid the traceability process sufficiently. For most part, these tools also provide several ways to visualize the software project, which make it easier for the developers to focus on architecture and design. They have comprehensive packages for requirement analysis; however, they lack specific functionality for gathering requirements. Additionally, code and stub generation as well as simulation functionality are limited, and SDEs are complex and powerful systems, which makes them difficult for new users to learn.

Construction tools

Tools for code construction (CC) tools, naturally, are aimed at programming and coding. Examples of such tool are IBM WebSphere Studio Application Developer and Microsoft Visual Studio. Most of the respondents report that today's CC tools are quite good, such as editors, compilers, and debuggers. They endow sufficient functionality for different programming tasks. They are flexible and intuitive to use, and this means that programmers can work efficiently with different projects simultaneously. Some tools provide rich features for rapid GUI development, code generation and swift navigation. However, the users report that such advanced features are difficult to use.

Test tools

The study surveyed the properties of tools that support the testing phase. Examples of such tools are IBM Rational Team Unifying Platform, IBM Rational TestManager, and Borland Optimizeit Enterprise Suite. Many testers state that their testing tools are inadequate, and that there is significant demand to improve the test tools. Current test tools are easy to integrate and to use with other tools from different vendors. However, using these tools is very time-consuming. For example, it takes a lot of effort to perform a full test iteration of a system and to fit it to the environment to the right conditions. This problem makes these kinds of test systems expensive to adapt.

Documentation tools

Software engineers use well-known tools like Microsoft Word or Adobe Acrobat to document their efforts. Since these tools are developed for general writing, they are not suitable for professional software documentation. For example, they lack basic functions to visualize the software project and means to trace and browse between software artifacts. That is, the tools do not support links between code, the related documentation, and the design diagrams. Furthermore, the tools they use are insufficient when it comes to edit and browse complex and large documents. To conclude, most technical writers think that there are very few good tools for integrated documentation.

A note on open source software development tools

Currently, free software development tools are being developed and distributed. For example, the Eclipse project at IBM has generated a widely acclaimed suit of tools. More and more developers have started to use open source software development tools. Our study substantiates this claim. We found that respondents use open source tools, at least partially. The reason why they adapt open source software is that the source code is available but the tools are also free. However, the open source tools have many disadvantages such as bugs. Additionally, such tools are often immature and do not support industry-standards like the rational unified process (RUP). Furthermore, they are poor at visualizing code, and provide limited support for modeling, version control, and refactoring.
To conclude, open source development tools seem to be immature and, at present, not sufficiently good for commercial adapt in the software industry. Nonetheless, the respondents regard open source software development tools as important software development tools in the future.

5.2 Requirements on future tools

General problems of today’s SDEs are largely due to integration problems, issues of automation, and configuration management. Let us discuss these issues in detail, and how we might improve the tools.

5.2.1 Tool integration and integrated development environments

Figure 8 shows that developer consider integration between tools to be a very important property as well as means to support collaboration between group members. Interestingly, although they say that they have problems with their integration processes and tools, they seem to be unaware that today’s IDEs can solve many of such problems. Thus, it can be assumed that respondents have limited knowledge on current IDEs and their capabilities, and also software development processes such as the rational unified process [Kruchten 2000].

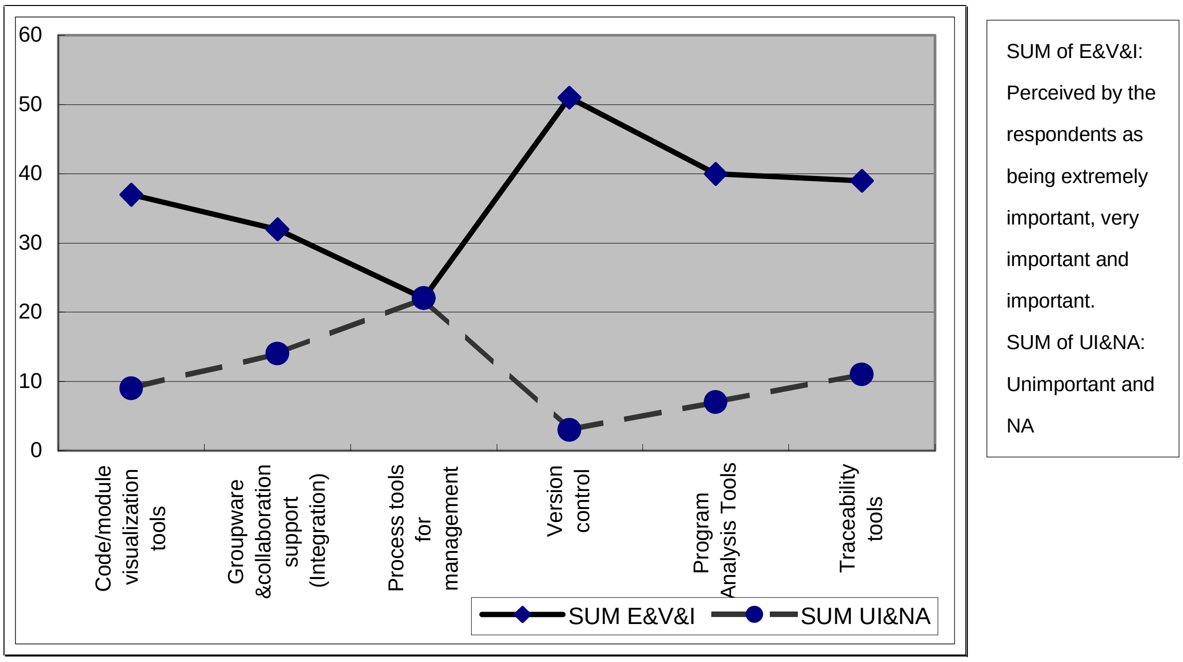


Figure 8. Tools needed in the future

When we asked what tools they need in the future, the majority of participants state that their ultimate goal is a complete, integrated, and automated “all-in-one” tool-suite that supports the whole software development lifecycle. For example, this requirement is shown in the questionnaire in two ways: First, respondents report on communication and collaboration problems between project personnel. From the developers’ point of view, they feel that few developers use the same professional functions in the tools in the same stage. It is difficult to share information with others during development, e.g. to communicate views and visualization possibilities. Second, interoperability difficulties exist among the tools, which results in problems of version control and modeling, and hampers the requirements analysis process.

From our point of view, future tools, in order to work effectively, need to address the whole software life cycle and should provide better means to integrate different software artifacts, and must also support collaboration between project members. Additionally, we need better integration among the tools so that they can communicate and share files (that is, share common project resources such as code and documents across tools and environments). This is also an important requirement to support collaboration between different personnel. This problem can be solved if the vendors adopt common standards like XML to let tools easily share files and information in the different stages of the development process. Regarding collaboration, a solution could be to integrate groupware like Lotus Notes into the development environments, to facilitate communication about the software artifacts in the team.

5.2.2 Automation support and the CASE tools

Many tools seem to lack automation support such as automatic workflow management and generation of manuals. On the other hand, advanced features can be too sophisticated. For instance, some tools are provide too much automation and this can result in that users looses control the processes. For example, automatic code generators can create code that is difficult to understand. Consequently, such tools are not always easy to use. Also, such advance features can have a steep learning curve.

Several companies, in our study, used a number of CASE tools to aid them in their work. For example, automated analysis and design tools, automated testing tools, and automated configuration tools. We found that the project members believed that the CASE tools were critical to their projects' success. Generally, they believe that the CASE tools could be improved, easier to use, and also customizable to suit different projects.

5.2.3 System configuration management

System configuration management tools (SCM) are used to administrate all configurable software items of which could be description of items, linkages among them, and to track interdependencies.

In our study, most of the project personnel consider SCM as an essential technology in the construction phase. Furthermore, the respondents state that they need better support for version control (VC tools) that are part of the SCM environments. Version control is the ability to store multiple versions of the same file under controlled, restricted-access conditions [Buckley 1994]. In our study, the respondents think that a major deficiency of today's VC tools is due to bad integration, which results in poor interoperability among the different components and modules. That is, they need one tool for managing the version control of all software artifacts of a project. As we have mentioned earlier, configuration manager think that it is important that the different SCMs could communicate. Moreover, version control and system configuration become a more and more important task because the developed systems are getting more complex. Finally, we found that many software developers seem to be unaware of the many possibilities of modern SCM tools. They know that SCM tools can be used for version control, but seem to be uninformed of that these tools can provide transaction management, change request tracking, system modeling, derived object pools, and problem reporting.

To conclude, most project personnel found that current tools for version control (VC) and SCM are clumsy and inefficient. Full-blown version control systems and mature SCM tools are considered to be highly needed in the near future.

Concerning today's SDEs, needed are more mature tools that approach the integration problems and provide better automation and configuration management.

5.3 Recommendations for the IT education

To properly motivate the student, it is necessary to provide meaningful real-world tasks and tools. However, it is often difficult to create such natural tasks within a university environment [Linders & vanCleemput 1974].

Many respondents in our study recommended typical software and tools-suits and also design methodology that they think should be part of the IT education. Technically, designers recommended improvements to the education on the pre-coding phase, such as modeling and design tasks. For example, a widely used design methodology in the industry is the Rational Unified Process (RUP) [Jacobson 1994]. They think that the curriculum should include courses on RUP. On the practical side, they also recommended ways to think and deal with reasoning, branches and labels on files in big software projects. Regarding tools, they think that tools for code browsing, code documentation, refactoring, and configuration management such as intelliJ IDEA, Eclipse, VCS and others should be included in the courses as a natural part. Additionally, many respondents think that some courses should approach and cover all aspects of the software development lifecycle (from initial design to maintenance tasks) and also include tools that can support such complex development activities.

To conclude, developers seem to have deficient knowledge on commercially-used methodologies and development processes such as RUP when they graduate. Furthermore, to achieve the skills and methodology of it, we do need to improve our education and IT Course. Moreover, the IT education would gain from putting more emphasis on early phases such as modeling and design (for large software projects). This requirement particularly applies to the non civil engineer courses (that is, not only the advanced computer science curriculum). Moreover, to improve the students' technical skills of configuration management and version control system one could include some lessons and tools on that in the education.

5.4 Study limitations

Every study has its limitations. The data set could potentially be misleading because many respondents work within the telecom area. This could result in that special demands where proposed in the questionnaire. These companies have to manage large software projects and this fact could appear in the answers. Additionally, the survey was conducted in Sweden, which could have some effects on the final result.

The study was focusing on commercial tools exclusively. However, many respondents discussed also open source software development tools. We omitted these tools in our analysis. The primary investigator needs a better understanding of the qualitative research paradigm. Such knowledge could be used to improve the study to include additional interviews in order to understand the respondents’ problems more deeply.

6. Conclusion

This study surveyed the use of commercial software development tools in the IT industry. This chapter concludes the study, provides a set of requirements on future development tools, and gives suggestions to improve the university-level education on software engineering tools.

6.1 Main findings

At present, according to our questionnaire, the most used integrated commercial environments are generally from vendors like IBM, Microsoft, Borland, and Sun Microsystems. For instance, IBM Rational Rose is the most accepted integrated development environment. Popular tools that support parts of the software development cycle are IBM WebSphere Studio Application Developer, IBM TestManager, and IBM Rational SCM solutions (includes IBM Rational ClearCase and IBM Rational ClearQuest). Documentation tools are mainly provided by Microsoft (e.g., the Office environment). Other tools that are used for documentation are Adobe and Visio. Moreover, respondents report that they are using open source tools for different tasks of the development phase. Open source communities like GNU org, Erlang org, Squeak org, Eclipse org, Junit org, CVS org, Apache org, and Mozilla org mainly provide open source software.

Professional software developers are not fully satisfied with their current software development environments although their systems provide good support in many respects. Generally, the programming and coding tools are perceived to be very good. The apparent disadvantages are related to bad integration among different tools, which results in difficulties in using different tools for a project. Additionally, respondents believe that documentation tools are insufficient in many respects, in particular, they lack support for visualizing, and relating documents to the actual software artifacts (i.e., a traceability problem).

A general problem of today’s programming practice is that different tools are used in different phases of the programming life cycle. For most part, these tools cannot communicate with each other, which makes it difficult to integrate various software artifacts into a functional system. Thus, we conclude that the future software development environments need to better support integration, automation, and configuration management. It would gain from more mature tools and standards to facilitate the integration process.

Regarding the required IT education, we saw that developers need more knowledge of professional software development process per se. The different phases and how it is done in practice. For example, respondents would like to have courses on industry standards and methods like the Rational Unified Process. Furthermore, IT curriculum would gain from including also commercial tools in the courses, not only the open source tools so frequently used in the education today. Additionally, the view is that courses should focus on the entire product life cycle, not only single development phases but also have commercial tools that cover the whole software product lifecycle. Moreover, we should put more emphasis on the early phases of the product life cycle, that is, modeling and design. Additionally, courses on techniques and methodologies for configuration management and version control should be beneficial to the industry. An overall conclusion is that integrated software development environment that use established development processes used in the industry (e.g., RUP) should be included in the curriculum.

6.2 Future work

This study did not focus on open source software development tools. However, our view is that studies of such tools would be beneficial. The IT industry is very interested into adopting open source software into their software development processes. This is probably due to that these tools are inexpensive. However, this study has shown that these are many problems with open source tools, in particular, related to the integration among the tools. Consequently, knowledge is needed on new methods and technologies for integrating the open source tools and software artifacts. Therefore, open source is also my main avenue for further research.

References

Buckley F.J., *Implementing a software configuration management environment,* IEEE Computer, Volume: 27, Issue: 2, Feb. 1994.

Bell D., *Software Engineering, A programming approach,* Pearson Addison Wesley, 3rd edition, 2000.

Chikofsky E.J.; Rubenstein B.L., *CASE: reliability engineering for information systems,* IEEE Software, Volume: 5, Issue: 2, March 1988.

Chan A.K.F, Hung S-I, *Software configuration management tools, Software Technology and Engineering Practice,* 1997. Proceedings. Eighth IEEE International Workshop on incorporating Computer Aided Software Engineering, 14-18 July 1997.

Conner WM, De Jong KA, *The academic/industry gap in systems programming and operating systems,* Proceedings of the 10th SIGCSE technical symposium on Computer science education, ACM SIGCSE Bulletin, Volume 11 Issue 1. January 1979.

Dart SA, *The Past, Present, and Future of Configuration Management,* published in the Proceedings of the IFIP World Congress, Madrid Spain, September 1992.

Engels G, Lewerentz C, Nagl M, Schäfer W, Schürr A, *Building integrated software development environments. Part I: tool specification,* ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 1 Issue 2, April 1992.

Granger MJ, Pick RA, *The impact of computer-aided software engineering on student performance*, Proceedings of the 22nd SIGCSE technical symposium on Computer science education, ACM SIGCSE Bulletin, Volume 23, Issue 1 March 1991.

Hapke M, Jaszkiewicz A, Kowalczykiewicz A, Weiss D, Zielniewicz P, *OPHELIA Software Development Tools Integration Technology*, Poznan University of Technology, 2000

Kurt C. Wallnau, Peter H. Feiler, *Tool Integration and Environment Architectures,* SEI/CUM press, May 1991.

Kruchten P, *The Rational Unified Process: An Introduction,* Addison-Wesley, 2000.

Linders JG, van Cleemput W.M., *Design Automation in a Computer Science*

Curriculum, 2.6 "The Need for Interaction Between Industry and Universities". ACM press, January 1974

Lawrence Pfleeger S, *Software Engineering: Theory and practice,* Prentice Hall, 2nd Edition, 2001

Myers MD, *Qualitative Research in Information Systems*, published in MISQ Discovery, June 1997

Norman, R.J., and Nunamaker, J.F. *Integrated Development Environments: Technological and Behavioral Productivity Perceptions.* Hawaii International Conference on Systems Sciences 1989. ACM press.

Sommerville I, *Software Engineering,* Addison-Wesley, 2000.

Wallnau KC, *Issues and Techniques of CASE Integration with Configuration Management,* CMU/SEI press, 1992.

IEEE Standards

IEEE Std 1074-1997	IEEE Standard for Developing Software Life Cycle Processes
IEEE Std 610.12-1990	IEEE Standard Glossary of Software Engineering Terminology
IEEE Std 830-1998	IEEE Recommended Practice for Software Requirements Documentation
IEEE Std 1012-1998	IEEE Standard for Software Verification and Validation
IEEE Std 1348-1995	IEEE Recommended Practice for the Adoption of Computer-Aided Software Engineering (CASE) Tools
IEEE Std 828-1998	IEEE Standard for Software Configuration Management Plans
IEEE Std 830-1998	IEEE Recommended Practice for Software Requirements Specifications
IEEE Std 1016-1998	IEEE Recommended Practice for Software Design Descriptions
IEEE Std 1063-2001	IEEE Standard for Software User Documentation
IEEE Std 1219-1998	IEEE Standard for Software Maintenance

Electronic Sources

http://cs.wwc.edu/~aabyan/ (Anthony Aaby, Professor of Computer Science, Walla Walla College class home page)
http://www.cs.utexas.edu/users/almstrum/cs370/tlee/r4.htm (Dr. Vicki L. Almstrum, professor at the University of Texas at Austin, class home page)
http://user.it.uu.se/~almstrum
http://www.cse.psu.edu/~lambert/420/big/node92.html
http://pathbridge.net/chikofsky
http://www.andrew.cmu.edu/user/conzalez/Teaching/ISW2/caseintro.html (Cleotilde (Coty) González, Assistant Professor, Department of Social and Decision Sciences, Carnegie Mellon University, class home page)
http://www.apa.org/
http://www.refactoring.com
http://www.isworld.org/surveyinstruments/surveyinstruments.htm
http://trochim.human.cornell.edu/kb/ (William M. Trochim, Cornell University)
http://www.ibm.com
http://www-306.ibm.com/software/rational/
http://www.microsoft.com/office/prodinfo.mspx

http://www.telelogic.com/
http://msdn.microsoft.com/vstudio/
http://msdn.microsoft.com/vbasic/
http://msdn.microsoft.com/vcsharp
http://www.borland.com
http://www.borland.com/starteam/
http://developers.sun.com/prodtech/javatools/index.html
http://otn.oracle.com/tech/pl_sql/index.html
http://www.borland.com/optimizeit
http://www.adobe.com/products/main.html
http://java.sun.com/j2se/javadoc/index.jsp
http://www.gnu.org/
http://www.xemacs.org
http://www.erlang.org
http://www.squeak.org
http://www.eclipse.org
http://www.junit.org/index.htm
http://www.cvshome.org
http://www.apache.org
http://www.php.net/
http://www.mozilla.org
http://www.ultraedit.com
http://www.doxygen.org
http://docpp.sourceforge.net

Appendix 1. Survey Questionnaire

Section A. General Questions

Participant Name:	____
Organization:	____
Address:	____
Email Address:	____
Business phone:	____

1. **What is your job title?**

Programmer	☐	Trainer	☐	Product manager	☐
Analyst	☐	Technical writer	☐	Manager	☐
Designer	☐	Consultant	☐	Project leader	☐
Tester	☐	Other (specify): __________	☐		

2. **How many years of software development experience do you have?**

1-2years ☐ 2-5 years ☐ 5-10years ☐ 10-15 years ☐ over 15 years ☐ NA ☐

3. **What type of applications do you develop?**

Telecom	☐	Chemical/health	☐	Scientific	☐
Transportation	☐	Electronic	☐	Aerospace	☐
Finance/bank	☐	Other (specify): __________	☐		

4. **What type of organization do you work within?**

Commercial ☐ Government ☐ Academic ☐ Other (specify):_________ ☐

5. Do you use? (Brand of the software development tools)

	Frequently	Often	Sometime	Seldom	Never
Microsoft	☐	☐	☐	☐	☐
IBM	☐	☐	☐	☐	☐
Intel	☐	☐	☐	☐	☐
Macromedia	☐	☐	☐	☐	☐
Borland	☐	☐	☐	☐	☐
SUN	☐	☐	☐	☐	☐
Apple	☐	☐	☐	☐	☐
Oracle	☐	☐	☐	☐	☐
Others: ______________	☐	☐	☐	☐	☐

6. What supports do you looking for in the future?

	Extremely Important	Very Important	Important	Unimportant	NA
Code/module visualization tools	☐	☐	☐	☐	☐
Groupware &collaboration support (Integration)	☐	☐	☐	☐	☐
Process tools for management	☐	☐	☐	☐	☐
Version control	☐	☐	☐	☐	☐
Program Analysis Tools	☐	☐	☐	☐	☐
Traceability tools	☐	☐	☐	☐	☐
Others:_________________ ______________________	☐	☐	☐	☐	☐

7. Describe the main problems that you experience with your software development tools?

Problem1:

____

Problem 2:

____

Problem 3:

____

8. **Overall, how do you rate the programming environments/tools that you are using in your work?**

Poor	Fair	Good	Very good	Excellent
☐	☐	☐	☐	☐

9. **What software tools and methodologies/knowledge are missing, in your view, in the university-level IT courses? (Free text)**

Section B. Software Development Environments in software development process

Instruction:

Indicate your role in the software development team.

Leave out the parts that do not apply to your role.

For example, if you are a programmer, you can directly go to Part 2, Page 7.

Analyst & Designer….	☐ ➔	**Part 1.Tools for Requirement Analysis/Design...**	**Page 5**
Programmer……………	☐ ➔	**Part 2. Tools for Implementation & Coding ……**	**Page 7**
Tester…………………	☐ ➔	**Part 3. Tools for Testing………………………….**	**Page 9**
Technical Writer……….	☐ ➔	**Part 4. Tools for Software Documentation………**	**Page 11**

1. *Tools for Requirement Analysis & Design*

1.1 List the main tool used in the requirements analysis phase.

Name	Brand	Version	Platform

1.2 Rate the level of support this tool provides.

	Poor	Fair	Good	Very good	Excellent	NA
Drawing, changing, and manipulation of diagrams	☐	☐	☐	☐	☐	☐
Generating reports and documentation	☐	☐	☐	☐	☐	☐
Develop prototypes for the purpose of requirement discovery or verification	☐	☐	☐	☐	☐	☐
Define project scope and system boundaries	☐	☐	☐	☐	☐	☐
Model and describe a current information system	☐	☐	☐	☐	☐	☐
Model and describe the requirements for a new information system	☐	☐	☐	☐	☐	☐
Gather & structure requirements	☐	☐	☐	☐	☐	☐
Generate stub-code	☐	☐	☐	☐	☐	☐
Reliability	☐	☐	☐	☐	☐	☐
Ease of use	☐	☐	☐	☐	☐	☐
Traceability	☐	☐	☐	☐	☐	☐
Groupware &collaboration support	☐	☐	☐	☐	☐	☐
Others (specify): ______________	☐	☐	☐	☐	☐	☐

1.3 What are the main advantages and disadvantages of this tool (listed above)?

Advantages: ____

____

Disadvantages:____

____

1.4 Have you been using other tools for requirements analysis & design? If so, what tools?

____

Thank you very much for your participation!

2. Tools for Implementation & Coding

2.1 Do you use? (Programming language)

	Frequently	Often	Sometime	Seldom	Never
C / C++	☐	☐	☐	☐	☐
C-sharp	☐	☐	☐	☐	☐
JAVA	☐	☐	☐	☐	☐
Delphi	☐	☐	☐	☐	☐
Perl	☐	☐	☐	☐	☐
Visual Basic	☐	☐	☐	☐	☐
Erlang	☐	☐	☐	☐	☐
Cobol	☐	☐	☐	☐	☐
Fortran	☐	☐	☐	☐	☐
Other (specify)	☐	☐	☐	☐	☐

____

2.2 What tools do you use for the following activities?

	Name of main tool	NA
Editing code	____________________	☐
Compiling/linking	____________________	☐
Profiling	____________________	☐
Static code analysis	____________________	☐
Debugging	____________________	☐
Other tools used in the programming phase	____________________	☐

2.3 What are the main advantages and disadvantages of your programming environment (listed above)?

Advantages: ____

____

Disadvantages:____

____

Thank you very much for your participation!

3. Tools for Testing

3.1 List the main tool used in the testing phase.

Name	Brand	Version	Platform

3.2 Rate the level of support this tool provides.

	Poor	Fair	Good	Very good	Excellent	NA
Unit testing	☐	☐	☐	☐	☐	☐
Module testing	☐	☐	☐	☐	☐	☐
Integration test	☐	☐	☐	☐	☐	☐
System testing	☐	☐	☐	☐	☐	☐
White/black-box testing	☐	☐	☐	☐	☐	☐
Generation of test data	☐	☐	☐	☐	☐	☐
Debugging	☐	☐	☐	☐	☐	☐
Simulation	☐	☐	☐	☐	☐	☐
Reliability	☐	☐	☐	☐	☐	☐
Ease of use	☐	☐	☐	☐	☐	☐
Traceability	☐	☐	☐	☐	☐	☐
Documentation	☐	☐	☐	☐	☐	☐
Groupware &collaboration support	☐	☐	☐	☐	☐	☐

3.3 What are the main advantages and disadvantages of this tool (listed above)?

Advantages: ____

____

Disadvantages: ____

____

3.4 Have you been using other tools for testing? If so, what tools?

____

Thank you very much for your participation!

4. Tools for Software Documentation

4.1 List the main tool used in the documentation phase.

Name	Brand	Version	Platform

4.2 Rate the level of support this tool provides.

	Poor	Fair	Good	Very good	Excellent	NA
Create documents	☐	☐	☐	☐	☐	☐
Document editing	☐	☐	☐	☐	☐	☐
Document browsing	☐	☐	☐	☐	☐	☐
Graphic support	☐	☐	☐	☐	☐	☐
Reliability	☐	☐	☐	☐	☐	☐
Traceability	☐	☐	☐	☐	☐	☐
Ease of use	☐	☐	☐	☐	☐	☐
Groupware &collaboration support	☐	☐	☐	☐	☐	☐

4.3 What are the main advantages and disadvantages of the main tool?

Advantages: ____

____

Disadvantages: ____

____

4.4 What other documentation tools do you prefer to create / edit / browse / generate / software documents? (Multi-choice)

Ms word	☐	ROBODoc	☐	C-DOC Professional	☐	ISA's Panorama C/C++	☐
Adobe Acrobat	☐	Doc++	☐	DocJet	☐	Object Outline	☐
AutoDuck	☐	ProgDoc	☐	Author-IT	☐	TwinText LXR	☐
Frame Maker	☐	Javadoc	☐	Ada Browse	☐	Cocoon	☐
Visio	☐	Doxygen	☐	Others: ________	☐		

Thank you very much for your participation!

Appendix 2. List of tools

This list is collected from the answer of the questionnaire. The specifications of each product are directly reference from the product official websites.

Part 1. Commercial tools

Adobe

Ex. Acrobat / Frame Maker

Borland Enterprise Studio for C++

Accelerate your C++ development for mobile devices

Borland Enterprise Studio For Java

Complete e-business implementation platform

Borland Enterprise Studio For Mobile

Accelerate your mobile application lifecycle

Borland JBuilder

JBuilder X speeds EJB, ™ Web, Web Services, XML, mobile, and database application development with two-way visual designers and rapid deployment to leading J2EE™ application servers, including BEA Web Logic, IBM® WebSphere, Sun ONE, Oracle9i, ™ Sybase EAServer, JBoss, and the integrated Borland® Enterprise Server. Power productivity with innovative Struts and Web Services designers, UML™ code visualization, refactoring, enterprise unit testing, and support for multiple version control systems. Build in quality with integrated Borland® Optimizeit™ Suite performance tools. Leverage unparalleled flexibility afforded by the Open Tools architecture.

Borland StarTeam

Borland® StarTeam® is an automated configuration and change management system that puts control of the development process in the hands of project teams. By providing users with access to all project assets through a central repository supported by customizable workflow and process management, StarTeam facilitates team communication and collaboration. A true enterprise solution, StarTeam delivers more than file version control: with StarTeam, project teams can benefit from a customized solution offering an integrated environment for managing requirements and changes, tracking defects and threaded discussions, and managing the tasks required by effective project management.

Borland Optimizeit Enterprise Suite

Optimizeit Enterprise Suite includes all components of Optimizeit Suite plus Optimizeit Request Analyzer. Optimizeit™ Request Analyzer provides advanced profiling techniques that allow developers to analyze the performance behavior of code across J2EE application tiers. Using Optimizeit™ Request Analyzer developers can efficiently prioritize the performance of

JDBC, JMS, JNDI, JSP™ RMI, and EJB™ web requests so that trouble spots can be proactively isolated earlier in the development lifecycle. Precise drill-down capabilities accelerate the time-to-resolution of performance problems, diagnosing and locating the root cause right down to the offending line of source code. It's available only as part of Borland Optimizeit Enterprise Suite. Uncovers and corrects production-environment performance problems in the test lab, before deployment.

IBM Rational RequisitePro

A powerful, easy-to-use, and integrated product for requirements and use case management that promotes better communication, enhances teamwork and reduces project risk.

IBM Rational Rose Data Modeller

A visual modeling tool that makes it possible for database designers, analysts, architects, developers and anyone else on your development team to work together, capturing and sharing business requirements, and tracking them as they change throughout the process.

IBM Rational Rose XDE Modeller

It enables architects and designers to practice model-driven development with the Unified Modeling Language (UML). Users can produce platform independent models of software architecture, business needs, reusable assets and management-level communication.

IBM Rational Rapid Developer

Combines architect RAD modeling and design with automatic code construction across all tiers. Offers a highly productive development environment with broad applicability to developers of varying backgrounds and skills targeting a wide range of deployment technologies.

IBM Rational Ada Developer

Part of our traditional languages solution, it provides support for applying modern software practices with Ada-based development projects throughout the software lifecycle.

IBM Rational PurifyPlus

A complete set of runtime analysis tools designed for improving application reliability and performance.

IBM Rational Rose Technical Developer

Supports the most advanced modeling constructs, including model execution and fully executable code generation, resulting in the highest levels of productivity.

IBM Rational Rose XDE Developer

Offers software designers and developers a rich set of model-driven development and runtime analysis capabilities for building quality software applications.

IBM WebSphere Studio Application Developer (WSAD)

WSAD is an open comprehensive development environment for building, testing and deploying on demand e-business applications.

IBM Rational Test RealTime

Across-platform solution for component testing and runtime analysis, it was designed specifically for those who write code for embedded, real-time, and other types of cross-platform software products.

IBM Rational PurifyPlus

A complete set of runtime analysis tools designed for improving application reliability and performance.

IBM Rational Rose XDE Developer Plus

It offers software designers and developers a rich set of model-driven development and runtime analysis capabilities for building quality software applications. It offers complete visual design and development environments that address the needs of organizations targeting both J2EE-based and. NET-based systems.

IBM Rational Functional Tester for Java and Web

It virtually eliminates script maintenance by creating resilient, reusable test scripts — in Java — with ScriptAssure™.

IBM Rational Robot

It automates functional, regression and configuration testing for a wide range of application types, including .NET.

IBM Rational Performance Tester

Uncovers and corrects production-environment performance problems in the test lab, before deployment.

IBM Rational Team Unifying Platform

Integrates all the testing activities for one application with centralized test management, defect tracking, and version control.

IBM Rational TestManager

It is used to manage all testing activities for an application: manual, regression, functional, performance, and runtime analysis. Rational TestManager can be accessed by all members of a project team, and make it easy to share test coverage information, details of defects, and reports. Rational TestManager ships with IBM Rational Functional Tester for Java and Web, Rational Robot, and Rational Performance Tester. Because of its value to development teams, it is also included in the IBM Rational Team Unifying Platform.

IBM Rational SCM solutions

Solutions for simplifying and managing change including version control, software asset management, and defect and change tracking. SCM enables development teams to capture, control and securely manage software changes and assets. Rational SCM solutions integrate with industry-leading integrated development environments (IDEs), including IBM WebSphere® Studio, the open source Eclipse platform and Microsoft .NET, providing developers with instant access to change information and code anytime, anywhere.

IBM Rational ClearCase

Change Management Solution: Integrated software configuration management for medium to large development projects.

IBM Rational ClearQuest

It can flexible defect and change tracking across the project lifecycle.

Java Sun - JavaDoc

JavaDoc is a tool for generating API documentation in HTML format from doc comments in source code.

Java Studio Standard

It is an integrated development environment (IDE) for Java technology. Java Studio Standard is also the recommended upgrade for Sun ONE Studio 4, Enterprise Edition for Java and Community Edition users.

Microsoft Office

Ex. Word, Visio, Excel, FrontPage

Microsoft® Visual C# .NET

Microsoft Visual C# .NET 2003 is the comprehensive toolset for creating XML Web services and Microsoft .NET—connected applications for Microsoft Windows® and the Web.
This robust development package, which uses the component-oriented C# development

language, offers beginning and intermediate developers with C++ or Java experience a modern language and environment for creating next-generation software. Visual C# .NET 2003 delivers superior functionality for streamlining business processes, including: Rapid design, development, and deployment support for creating and consuming Web services.
Form designers and visual controls for creating rich Windows-based applications.
Authoring tools and services for building powerful Microsoft .NET server-based solutions.
Migration tools for converting Java-based projects to the Microsoft .NET development environment.
With Visual C# .NET 2003, developers can build solutions for the broadest range of clients, including Windows, the Web, and mobile or embedded devices. Using this elegant programming language and tool, developers can leverage their existing C++ and Java-language skills and knowledge to be successful in the .NET environment.

PL/SQL

PL/SQL is Oracle's procedural extension to industry-standard SQL. PL/SQL naturally, efficiently, and safely extends SQL. Its primary strength is in providing a server-side, stored procedural language that is easy-to-use, seamless with SQL, robust, portable, and secure. Thus, it offers a platform for robust, high-performing enterprise applications, not only for our Fortune 500 customers, but also for Oracle Applications, which have over 4 million lines of code.

Telelogic DOORS/ERS

The world's leading requirements management tool, is a multi-platform, enterprise-wide system designed to capture, link, trace, analyze and manage changes to information to ensure a project's compliance to specified requirements and standards.

Telelogic TAU

It is an integrated family of software development and testing tools that provides a unique visual development environment. This simplifies, automates and accelerates the production of real-time and other advanced software.

Visual Studio .NET

Visual Studio .NET 2003 is the comprehensive tool for rapidly building Microsoft .NET–connected applications for Microsoft Windows® and the Web, dramatically increasing developer productivity, and enabling new business and enterprise opportunities. Explore the features in each of the three Visual Studio .NET 2003 editions: Enterprise Architect, Enterprise Developer, and Professional. Developers can use Visual Studio .NET to: Build the next-generation Internet; Create powerful applications quickly and effectively; Span any platform or device. Visual Studio .NET is the only development environment built from the ground up to enable integration through XML Web services. By allowing applications to share data over the Internet, XML Web services enable developers to assemble applications from new and existing code, regardless of platform, programming language, or object model.

Visual Basic .NET 2003

Visual Basic .NET 2003 provides the easiest, most productive language and tool for rapidly building applications for Microsoft Windows® and the Web. Ideal for existing Visual Basic developers as well as new developers in the Microsoft .NET development environment, Visual Basic .NET 2003 delivers enhanced visual designers, increased application performance, and a powerful integrated development environment (IDE) to get you on the fast track to application development.

Part 2. Open source software

2.1 GNU

GNU Smalltalk Language

GNU Smalltalk is a free (or Open Source) implementation that closely follows the Smalltalk-80 language as described in the book Smalltalk-80: the Language and its Implementation by Adele Goldberg and David Robson. GNUS mall talk runs on most versions of Unix or Unix like systems (GNU/Linux, Free BSD, etc). There is even a version for commercial operating systems like MS-NT.

Emacs - Extensible, real-time editor

Extraordinarily powerful text editor with additional features including content sensitive major modes, complete online documentation, highly extensible through Lisp, support for many languages and their scripts through its multilingual extension, and a large number of other extensions available either separately or with the GNU Emacs distribution. Runs on many different operating systems regardless of machine.

XEmacs

It is a highly customizable open source text editor and application development system. It is protected under the GNU Public License and related to other versions of Emacs, in particular GNU Emacs. Its emphasis is on modern graphical user interface support and an open software development model, similar to Linux. XEmacs has an active development community numbering in the hundreds, and runs on Windows 95 and NT, Linux and nearly every other version of Unix in existence.

Gcc - GNU Compiler Collection

The GNU Compiler Collection is a full-featured ANSI C compiler with support for K&R C, as well as C++, Objective C, Java, and Fortran. GCC provides many levels of source code error checking traditionally provided by other tools (such as lint), produces debugging information, and can perform many different optimizations to the resulting object code.

G++ - C++ compiler

G++ is the traditional nickname of GNU C++, a freely redistributable C++ compiler. It is part of gcc, the GNU compiler suite, and is currently part of that distribution.

GNU Pascal - Pascal compiler of the GNU Project

GNU Pascal is the Pascal language compiler of Project GNU. It is a 32/64-bit compiler that is linked against the GCC backend and will be integrated in to GCC in the long run. It runs on all operating systems compatible with GCC, and can act as a native or a cross compiler between those systems. It implements ISO Pascal, Borland Pascal 7.0, and parts of other Pascal standards or de-facto standards.

Make - Generates executables and other non-source programs

Make examines a set of related files, determines which of them are out of date, and runs just the commands necessary to bring them back up to date. Make is typically used to compile and link programs, but it can be useful in many other situations as well.

Gprof

The GNU Profiler

Dbg - C++ debugging utilities

The dbg library is a set of C++ utilities to facilitate modern debugging idioms. It has been designed to support defensive programming techniques in modern C++ code. It integrates well with standard library usage and has been carefully designed to be easy to write, easy to read and very easy to use.

Ddd - Graphical front end for command line debuggers

Works with command-line debuggers such as GDB, DBX, WDB, Ladebug, JDB, XDB, the Perl debugger, or the Python debugger. In addition to the usual front-end features such as viewing source text, DDD has an interactive graphical display, where data structures are displayed as graphs

Gdb - GNU Debugger

GDB lets you to see what is going on 'inside' another program while it executes--or what another program was doing at the moment it crashed.

Strace - Debugging tool

Strace is a system call tracer: it traces all system calls made by another process or program. Since the program to be traced does not need to be recompiled before tracing, strace is useful for programs for which the source is not readily available.

Valgrind - Memory debugger

Valgrind finds memory-management problems by checking all reads and writes of memory are checked, and intercepting all calls to malloc/new/free/delete. As a result, Valgrind can detect problems like the use of uninitialised memory, reading/writing memory after it has been free'd, reading/writing off the end of malloc'd blocks, reading/writing inappropriate areas on the stack, memory leaks, and passing of uninitialised and/or unaddressible memory to system callsValgrind - Memory debugger

2.2 Erlang

Erlang is a programming language designed at the Ericsson Computer Science Laboratory. Open-source Erlang is being released to help encourage the spread of Erlang outside Ericsson. Erlang/OTP is a development environment for building distributed real-time high availability systems with short time to market requirements. Erlang/OTP is available in two versions: a licensed one with full support and an Open Source version with the entire source code free of charge.

2.3 Squeak

Squeak is a full-featured implementation of the Smalltalk programming language and environment based on (and largely compatible with) the original Smalltalk-80 system. Squeak has very powerful 2- and 3-D graphics, sound, video, MIDI, animation and other multimedia capabilities -- and one of the most impressive development environments ever created. It also includes a customizable framework for creating dynamic HTTP servers and interactively extensible Web sites. The entire Squeak system is open source software, distributed freely with a liberal license.

2.4 Eclipse

Eclipse is a kind of universal tool platform - an open extensible IDE for anything and nothing in particular.

2.5 Junit

JUnit is a regression-testing framework written by Erich Gamma and Kent Beck. The developer who implements unit-tests in Java uses it. CppUnit is a C++ unit-testing framework. It's started its life as a port of JUnit to C++ done by Michael Feathers

2.6 CVS

CVS is the Concurrent Versions System, the dominant open-source network-transparent version control system. CVS is useful for everyone from individual developers to large, distributed teams: Its client-server access method lets developers access the latest code from anywhere there's an Internet connection. Its unreserved checkout model to version control avoids artificial conflicts common with the exclusive checkout model. Its client tools are available on most platforms.

2.7 Apache

The Apache Software Foundation provides support for the Apache community of open-source software projects. The Apache projects are characterized by a collaborative, consensus based development process, an open and pragmatic software license, and a desire to create high quality software that leads the way in its field. We consider ourselves not simply a group of projects sharing a server, but rather a community of developers and users.

2.8 PHP Open source

PHP is a widely used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

2.9 Mozilla

Mozilla was the original code name for the product that came to be known as Netscape Navigator, and later, Netscape Communicator. Later, it came to be the name of Netscape Communications Corporation's dinosaur-like mascot. Now, we intend to use the name Mozilla as the generic term referring to Internet client software developed through our open source project. Netscape Communications Corporation holds trademarks on the names Netscape, Navigator, and Communicator; it has not yet been decided what, if any, restrictions Netscape will place on the use of those names. However, a generic term for browsers is still needed, and ''Mozilla" is as good a name as any. So, Mozilla is a set of technologies, but not a specific (in biologic terms, Mozilla is a genus; a particular product is a species). And mozilla.org (pronounced Mozilla Dot Org or The Mozilla Organization) is the group of people who coordinate the project.

LINKÖPINGS UNIVERSITET	**Avdelning, Institution** Division, Department Institutionen för datavetenskap 581 83 LINKÖPING	**Datum** Date 2004-05-28

Språk Language Svenska/Swedish X Engelska/English	**Rapporttyp** Report category Licentiatavhandling X Examensarbete C-uppsats D-uppsats Övrig rapport	**ISBN**
		ISRN LITH-IDA-EX--04/085--SE
URL för elektronisk version http://www.ep.liu.se/exjobb/ida/2004/dt-d/085/		**Serietitel och serienummer** Title of series, numbering **ISSN**

Titel	An investigation of the use of software development environments in the industry
Författare Author	Ping An

Sammanfattning
Abstract
Software engineering tools are being used in the industry in order to improve the productivity and the quality of the software development process. The properties of those tools are being perceived to be unsatisfactory. For example, researchers have found that some problems are due to deficient integration among the tools. Furthermore, a continuing problem is that there is a gap between the IT education and real demand of tool-skills from IT industry. Consequently, knowledge is needed of the properties of software development tools as well an understanding of demanded tool-skill from the industry.
The purpose of this study is to survey commercial software development environment (SDEs) that are used today in professional software engineering and discuss their advantages and disadvantages. A secondary goal of the study is to identify the actual requirements from the industry on the IT-education.
A questionnaire was sent out to 90 software developers and IT managers of 30 IT companies in Sweden. The results of the survey show that IT companies, for most part, use SDEs from commercial software vendors. Respondents report that common problems of the SDEs are the following: bad integration among the tools, problems to trace software artifacts in the different phases of the programming cycle, and deficient support for version control and system configuration. Furthermore, some tools are difficult to use which results in a time-consuming development process.
We conclude that future software development environments need to provide better support for integration, automation, and configuration management. Regarding the required tool-skills, we believe that the IT education would gain from including commercial tools that cover the whole software product lifecycle in the curriculum.

Nyckelord
Keyword
Software development environments, Software life cycle, Survey, Software development tools, CASE tools, Integration, Integrated software development environment