

The screenshot displays a software development tool interface with several panels:

- PerVisGL - Performance Visualisation with Op...:** A 3D visualization window showing a grid of colored blocks representing different threads. The legend includes: Thread_1042 (yellow), Thread_205 (grey), Thread_83 (blue), Thread_6 (light blue), Thread_5 (purple), Thread_4 (green), Thread_1 (dark blue), Thread_0 (black), and Background_load (dark grey). The FPS is 28.5714 and the best is 134.462.
- RAC (Project management):** A table showing project tasks with columns: Summary, Total hours, Keywords, Status, Hours, Resolution, Type, Billable, Version, and Milestone.
- SVN (Versioning):** A table showing file versions with columns: File name, Version, Author, Repository path, and Type.
- TESTLINK (Test management):** A table showing test results with columns: Name, Status, and Expected r.
- PROBE (Test data gathering):** A table showing test data with columns: Test, Version, Output, and Number.

Summary	Total hours	Keywords	Status	Hours	Resolution	Type	Billable	Version	Milestone
Specify requirement	6.0	-	closed	0	fixed	task	1	-	-
Implement specification	4.0	-	closed	0	fixed	task	1	-	-
Test implementation	6.0	-	closed	0	fixed	task	1	-	-
Create probes	3.0	-	closed	0	fixed	task	1	1.0	milestone1

File name	Version	Author	Repository path	Type
loop.zip	1	jejuho	file:///d:/svn/toolchain_demo/loop.zip	source

Name	Status	Expected r
Ability to test multiple products	passed	
ID for testcases	passed	
Ability to collect test cases into set(s)	passed	

Test	Version	Output	Number
TEST PROJECT	1		1

Juho Eskeli

Integrated tool support for hardware-related software development

VTT PUBLICATIONS 725

Integrated tool support for hardware-related software development

Juho Eskeli



ISBN 978-951-38-7373-8 (URL: <http://www.vtt.fi/publications/index.jsp>)
ISSN 1455-0849 (URL: <http://www.vtt.fi/publications/index.jsp>)

Copyright © VTT 2009

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 1000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 4374

VTT, Bergsmansvägen 5, PB 1000, 02044 VTT
tel. växel 020 722 111, fax 020 722 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O. Box 1000, FI-02044 VTT, Finland
phone internat. +358 20 722 111, fax + 358 20 722 4374

Technical editing Leena Ukaskoski

Keywords development tools, tool integration

Abstract

This thesis presents how the hardware-related software development process can be improved by means of tool integration. Challenges in hardware-related software development are diverse, which is why a multitude of tools is needed during the development. The tools produce data that needs to be managed, but the tools are disconnected. Tool integration provides a means of bringing the data from disconnected tools together into one coherent, easily manageable package.

Research was conducted by initially perceiving hardware-related software development from a systems engineering viewpoint, with a focus on several well-known process models. This was done to understand the kinds of activities that need to be supported by the tools. A workflow concept was introduced as a means to support the development effort of an individual worker. An extensive background study into tool integrations was conducted to understand state-of-the-art tool integration approaches and concepts, and then used to create the foundation for the tool integration.

Hardware-related software development challenges were gathered from literature and industry experiences to reinforce the understanding on needed tool support and to specify the requirements for the tool integration. The main requirements for the tool integration were easy extensibility, which could only be provided via a framework-based solution, and a means to provide data flow from tool to tool while preserving traceability between the data from the tools. Tool requirements for the integration were project management, requirement management, test management, and change management tools. Emphasis was put on tools supporting testing and test analysis.

The tool integration, ToolChain, was implemented in two phases. In the first phase the groundwork for the integration framework was done. Eclipse was chosen as the platform for the integration and plug-ins selected as a means of implementation. In the second phase, tool support focusing on the hardware-related software development aspects was added. Implementations from each phase were validated separately in industry cases. Experiences from these cases are presented in which it is shown how ToolChain can be easily adapted into the target company's environments, and how the tool integration improves the way of working.

Tiivistelmä

Työssä esitetään, miten rautaläheisten ohjelmistojen kehitysprosessia voidaan parantaa työkalu-integraation avulla. Rautaläheisten ohjelmistojen kehitystyön haasteet ovat monimuotoisia, ja siksi kehitystyön avuksi tarvitaan useita työkaluja. Työkalut tuottavat tietoa, jota täytyy hallinnoida, mutta toisaalta työkalut ovat irrallisia, mikä tekee hallinnoinnista hankalaa. Työkaluintegraatio mahdollistaa tietojen koostamisen irrallisista työkaluista yhtenäiseksi, helposti hallittavaksi kokonaisuudeksi.

Tutkimustyö aloitettiin tarkastelemalla rautaläheisten ohjelmistojen kehitystä systeemi-suunnittelun näkökulmasta. Tarkastelu keskittyi yleisesti tunnettuihin prosessimalleihin, ja sen tavoitteena oli selvittää, mitä aktiviteetteja työkalujen tulee tukea. Työnkulut (workflow) esitettiin keinona tukea yksittäisen työntekijän kehitystyötä. Työkaluintegraation nykytila selvitettiin kattavasti mahdollisten lähestymistapojen löytämiseksi, ja tätä tietoa käytettiin työkaluintegraation perustana.

Rautaläheisten ohjelmistojen kehitykseen liittyviä haasteita koottiin kirjallisuudesta ja teollisuuskokemuksista vahvistamaan ymmärrystä tarvittavasta työkalutuesta ja määrittämään vaatimukset työkaluintegraatiolle. Päävaatimuksina työkaluintegraatiolle asetettiin laajennettavuus, minkä mahdollistamiseen kehikko (framework) -pohjainen ratkaisu sopii luontevasti, ja lisäksi tiedon kulku työkalusta työkaluun sekä jäljitettävyyden ylläpitäminen työkaluissa syntyvien tietojen välille. Työkaluvaatimuksina integraatiolle asetettiin projektinhallinta-, vaatimustenhallinta-, testauksenhallinta- ja muutoksenhallintatyökalut. Erityisesti painotettiin testauksen ja testianalyysin työkalutukea.

Työkaluintegraatio, ToolChain, toteutettiin kahdessa vaiheessa. Ensimmäisessä vaiheessa suoritettiin pohjatyö integraatiokehitykselle. Integraatioalustaksi valittiin Eclipse ja Eclipsen liitännäiset (plug-in) integraatioiden toteutuskeinoksi. Toisessa vaiheessa lisättiin työkalutuki, joka painottuu rautaläheiseen ohjelmistokehitykseen. Kunkin vaiheen toteutukset validoitiin erikseen teollisuuskokeilussa. Teollisuuskokeilujen kokemukset esitetään, joista käy ilmi, kuinka ToolChain voidaan helposti ottaa käyttöön kohdeyrityksen kehitysympäristössä ja kuinka työkaluintegraatio helpottaa työskentelyä.

Preface

This thesis was written as part of the TWINS project at the VTT Technical Research Centre of Finland. The TWINS project addresses co-design problems in product development consisting of integrated hardware and software development. TWINS is a jointly-funded project in the Information for European Advancement (ITEA) programme, in which there are 25 partners (from both research and industry) from five countries. Before the writing of this thesis began, research and development of ToolChain started in the ITEA-Merlin project, in which the author participated during 2006–2007. Writing of the thesis began in January 2008 and it was completed in April 2009.

I would like to thank Päivi Parviainen, VTT Technical Research Centre of Finland, for her excellent guidance, without which the writing of this thesis would not have been possible. I thank also Professor Tapio Seppänen and Professor Jukka Riekkilä from the University of Oulu for supervising my thesis.

My colleagues from the TWINS project deserve thanks for the exemplary support and cooperation provided. Especially I would like to thank Jukka Kääriäinen, VTT Technical Research Centre of Finland, for support given during the work.

Oulu, Finland 7th April, 2009

Juho Eskeli

Contents

Abstract	3
Tiivistelmä.....	4
Preface	5
List of symbols.....	8
1. Introduction.....	10
2. The development process.....	12
2.1 Systems engineering.....	12
2.1.1 Waterfall lifecycle.....	12
2.1.2 V-Model	13
2.1.3 Crnkovic model.....	14
2.1.4 Iterative lifecycle	14
2.2 Workflow	15
3. Tool integration.....	17
3.1 Why tool integration is necessary?	17
3.2 Tool integration approaches.....	18
3.2.1 Data integration interfaces.....	20
3.2.2 Tool integration from an application lifecycle management perspective	21
3.3 Existing implementations	21
4. Hardware-related software.....	24
4.1 Challenges	24
4.1.1 Product lifecycle	25
4.1.2 Performance	25
4.1.3 Memory handling	26
4.1.4 Testing.....	26
4.1.5 Timeliness	27
4.1.6 Concurrency	28
4.1.7 Interfaces	29
4.1.8 Heterogeneity	29
4.1.9 Reactivity and responsiveness	30
4.1.10 Predictability	30
4.1.11 Correctness and robustness.....	30
4.1.12 Distributed systems	31
4.1.13 Resource limited target environments	31
4.1.14 Subcontracting.....	31
4.1.15 Managerial challenges.....	32
4.1.16 Summary	32
5. Requirements for tool support.....	33
5.1 ToolChain process model	34
5.1.1 System definition phase	34

5.1.2	Sub-System definition phase	36
5.1.3	Implementation phase	37
5.1.4	Integration & release phase.....	39
5.1.5	Project management	41
5.1.6	Change management	41
5.2	Collected requirements	41
6.	Tool integration design	44
6.1	ToolChain framework	44
6.1.1	Eclipse Architecture	44
6.1.2	ToolChain architecture	46
6.1.3	Connecting tools	47
6.1.4	Data visibility.....	48
6.1.5	Traceability	49
6.1.6	Security and user rights management.....	50
6.2	Hardware-related software development support	51
6.2.1	Overview.....	51
6.2.2	Workflow support.....	53
7.	Tool integration implementation	54
7.1	Implementations in the first phase	54
7.1.1	Tool integrations	56
7.1.1.1	Philips Project Assist Tool	56
7.1.1.2	Open workbench	57
7.1.1.3	Trac integration	57
7.1.1.4	Telelogic DOORS.....	58
7.1.1.5	Open source requirements management tool (OSRMT)	59
7.1.1.6	IBM Rational RequisitePro	60
7.1.1.7	Telelogic Synergy/CM	61
7.1.1.8	Subversion	61
7.1.1.9	SoftFab.....	62
7.1.1.10	Testlink.....	62
7.2	Implementations in the second phase.....	63
7.2.1	PROBE framework integration	63
7.2.2	PERVIS and PERSIM integration.....	64
7.2.3	MVA tool integration	64
7.2.4	Workflow implementation and integration.....	65
7.2.5	Improved traceability view	67
7.2.6	Summary of the integrated tools.....	68
8.	Tool integration trial and validation	69
8.1	Philips case	69
8.2	NSN case	70
9.	Discussion	74
9.1	Integration approach	74
9.2	Tool integrations.....	74
9.3	Process and workflow	75
9.4	Dissemination.....	75
9.5	Validation	76
9.5.1	NSN case	76
9.6	Future work	77
10.	Conclusion.....	79
	References	81

List of symbols

ALM	Application Lifecycle Management
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
CM	Change management
CPU	Central Processing Unit
CSV	Comma Separated Values
DB	Database
DMA	Direct Memory Access
DSP	Digital Signal Processing / Digital Signal Processor
ECF	Eclipse Communication Framework
FIFO	First in, First out
FPGA	Field-Programmable Gate Array
FSA	Finite State Automata
GPL	GNU General Public License
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HW	Hardware
IC	Integrated Circuit
ICT	Information and Communication technologies
IDE	Integrated Development Environment
I/O	Input/Output
JDBC	Java Database Connectivity
JTAG	Joint Test Action Group
MAC	Multiply And Accumulate
MS	Microsoft
MVA	Multivariate analysis
ODBC	Open Database Connectivity

OS	Operating System
OSRMT	Open Source Requirements Management tool
PAT	Philips Project Assist Tool
PCA	Principal Component Analysis
PDE	Plug-in Development Environment
PerSim	Performance Simulation
PerVis	Performance Visualization
PF	Probe Framework
PLC	Product Life Cycle
PM	Project management
RM	Requirement Management
RMA	Rate Monotonic Analysis
RTOS	Real Time Operating System
SDL	Specification and Description Language
SQL	Structured Query Language
SUD	System Under Development
SUT	System Under Test
SVN	Subversion
SW	Software
TC	ToolChain
TM	Test management
UI	User Interface
UML	Unified Modeling Language
URM	User Rights Management
XML	Extensible Markup Language

1. Introduction

Embedded software (SW) can be found practically everywhere. For example most home appliances such as dishwashers, DVD players, and televisions contain embedded software. Embedded software is not limited to only consumer electronics but can also be found in medical devices, avionics, and so on. Embedded software is so ubiquitous today that it is almost easier to name examples of where it does not exist than where it does.

The principal purpose of the software is to implement the various requirements of the appliance together with the underlying hardware (HW) solution. Due to the fact that most embedded systems interact with the real world, where in some cases human lives are at stake, embedded SW development is a considerably complex and challenging endeavour.

When embedded systems are produced for the consumer market, the major part of the total costs originates from mass-producing the hardware components. The software in the system is a one-time cost: once developed the software can be replicated at no additional cost. As a result software developers often need to cope with inferior hardware while attempting to meet specifications. To help in tackling these challenges, a plethora of methodologies and tools have been created as an aid. However, these tools have been built to address specific problems and challenges in the development process; they are by nature disconnected. This lack of interoperability between tools creates additional challenges.

This thesis focuses on hardware-related software (HW rel. SW), which is a part of embedded software development. The differentiating factor is that HW rel. SW is understood as development of software components that are in direct interaction with hardware components (e.g. drivers for hardware components, firmware) whereas embedded software means all software embedded as part of a complete device including hardware and mechanical parts. Activities relating to e.g. desktop software, hardware, or mechanics development are not in the scope of this work.

The research has been conducted by first studying the interoperability of tools, followed by identifying problems experienced in HW rel. SW development, and then by trying to find solutions especially to the problems that can be supported by tools. The interoperability of tools has been studied to create a stable foundation for tool integration. Research into the current state of tool integration has been done, and it is used as background information for the tool integration. Subsequently, special characteristic problems encountered, and general workflow in HW rel. SW development, have been studied from literature and from industry experiences. Furthermore, the tools used during different phases of development have been identified and in addition challenges and problems in the tools are discussed. Lastly, it is examined if some of these challenges can be solved efficiently by improving tool support and the interoperability between the tools.

Challenges and problems in HW rel. SW development need to be addressed in every phase of the development. For this reason it is necessary to perceive development initially from a system-level viewpoint. Systems engineering includes the entire chain of activities from the beginning of the product lifecycle to completion and concerns all the different disciplines participating in system development. In this thesis systems level vision is limited to activities that concern the software development discipline. As an additional limitation the scope begins from the requirements engineering phase and ends in the integration & release phase. Aspects that were defined outside of the given scope are considered as constraints originating from external sources (i.e. inputs/outputs of the boundary phases: requirements engineering and integration & release and maintenance).

After the literature study was complete and industry experiences gathered, the results were analyzed. Based on the analysis an integrated tool support solution was designed and built to address the challenges found in HW rel. SW development. The set of tools for integration was selected based on their integration potential (i.e. the added value of integration) and based on industry feedback. Parts implemented in this thesis are the tool integration framework and the tool integrations themselves, with several exceptions which are mentioned in the text. The tools used by the integration were not implemented as part of this thesis.

Finally the solution was evaluated in industry cases in which HW rel. SW was developed by using the built solution. The purpose of the evaluation was to study how well the built solution operates when faced with real world challenges.

This thesis is structured in the following way: chapter 2 studies the creation of embedded systems software as part of a complex development process. For coordination and support of software development activities many different tools are needed. However, choosing the right tools and tool sets is difficult. The interoperability of tools is also a complicating factor. Tool integration provides a way to improve interoperability between the tools, and it is studied in chapter 3. Chapter 4 examines challenges often experienced in HW rel. SW development. Chapter 5 studies the challenges by project phase, identifies possible tools to aid in the challenges and defines requirements for the integration solution. Chapter 6 is the design of the solution and in chapter 7 the implementation is documented in detail. Chapter 8 presents validation cases of the implementation. Chapter 9 discusses the results of this thesis, and chapter 10 provides a summary of the results. Finally, the references are provided in chapter 11.

2. The development process

Building an embedded system is often a complex endeavour involving engineers from multiple disciplines, including those from hardware and software development. In order to manage this complexity, systems engineering is needed. This section elaborates the system level activities involved in embedded systems development. This is done in order to gain a better understanding of what kinds of activities need to be supported by the various tools. Moreover, to further distinguish the setting in which the various tools need to operate, HW rel. SW development activities are shown as part of the overall process.

In addition, the workflow concept is introduced in this chapter. Workflows and workflow management are issues closely related to development processes. The process description provides the information on what needs to be done, while the workflow description supplies the steps and tasks that are necessary to implement some specific activity of the process.

2.1 Systems engineering

Systems engineering is a discipline that brings together different skills, disciplines, development stages and stakeholders so that complex systems can be built. Keller & Shumate define systems engineering as follows: *Systems engineering is the process used to transform an operational need into a working system that satisfies the requirements* [1].

Systems engineering models were originally created to master the creation of very large and complicated defence and computer systems. Systems engineering provides a comprehensive reference model for the development of new embedded products. However, the models are usually simplified and idealized, and development in real life is more complicated. [2]

The role of systems engineering differs from discipline-specific engineering approaches, such as electronics engineering, software engineering and mechanical engineering [3]. Discipline-specific engineering approaches focus on their own special area, providing mechanisms to support them. Systems engineering provides a framework for the work of other engineering disciplines and it remains independent of discipline and product type.

2.1.1 Waterfall lifecycle

A traditional model used to describe the steps in developing a system is the sequential system development lifecycle. This model identifies the main tasks in the development process and gates defining decisions between the tasks [3]. The model provides the basic principles for application of managerial control to design processes. The development of complex products is dispersed into several concurrent design processes including HW, SW and other discipline-specific processes.

It is the most common lifecycle, and its advantages are simplicity and strong tool support [4]. Figure 1 provides an example of the development process using the waterfall model [5].

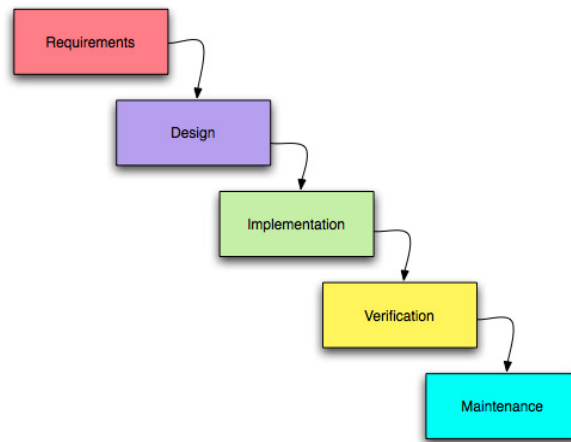


Figure 1. Example of development process using the waterfall model.

2.1.2 V-Model

The traditional development model divides product development into two main sections. The first section is used to define what is to be built (*definition* of requirements / architecture), and to build it. The second section is used to integrate and verify what has been built (*integration / verification*). Another view on system development is to enhance the traditional sequential model by adding horizontal links between the *definition* and *integration / verification* sections. This produces a model called the V-model (figure 2). [2, 3]

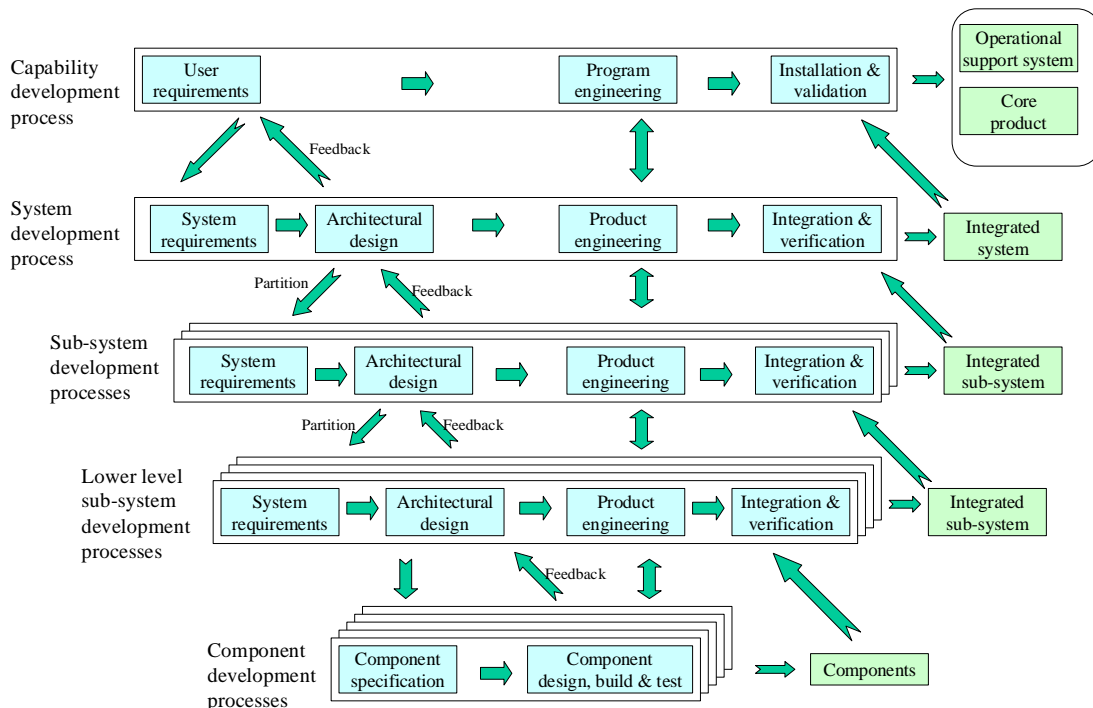


Figure 2. Development approach based on the V-model.

2. The development process

After system-level requirements specification the architectural design divides and assigns system-level requirements into sub-system level entities, which are further specified and divided into smaller entities [6]. Stevens et al. suggest that the customer-supplier relationship applies at each level. For example, a system level process agrees on the specification for a sub-system level and asks for a set of components from a sub-system level. Thus, the system level negotiates with the sub-system level (supplier) and sub-system levels negotiate with their suppliers. [2, 3]

2.1.3 Crnkovic model

Crnkovic et al. present the complex product life cycle (PLC) model which divides PLC into sub-PLCs for SW and HW development and considers activity types during product development. Figure 3 describes the development processes of complex systems and their related lifecycles. [7]

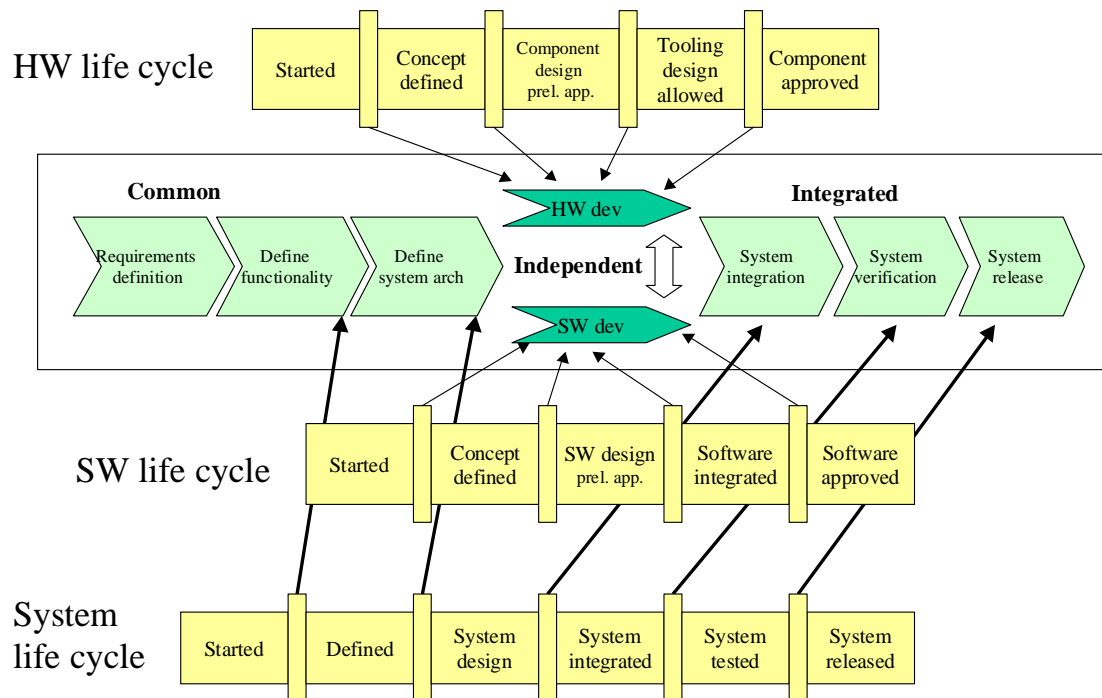


Figure 3. Development process of a complex product with PLCs.

The process is divided into three main activity types. First, the process contains common activities, which relate to the system level. Second, there are independent activities, which relate to the different disciplines (e.g. HW and SW development). Third, there are integrated activities where information from all processes must be accessible and integrated into common information. [2]

2.1.4 Iterative lifecycle

Iterative lifecycles (figure 4) enhance the waterfall model by trying to tackle the incompleteness problem [8]. This means that a phase (the phases could be the same as those presented in

chapter 2.1.1) cannot truly be complete or correct until its shortcomings and errors are identified in the later phases. Therefore in iterative lifecycles the phases are not executed only once, but likely multiple times. Outputs of the previous iteration can then be used as feedback for the next iteration so that the shortcomings can be corrected. [4]

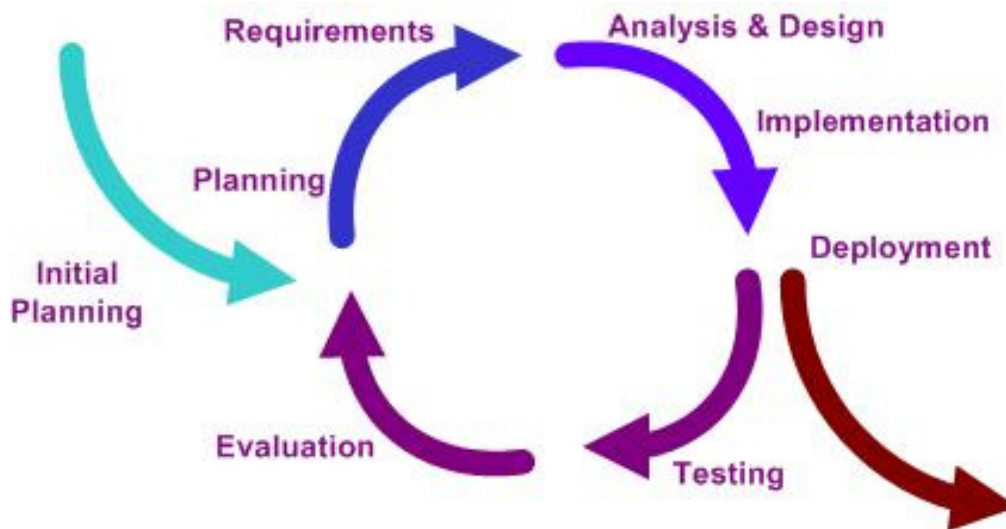


Figure 4. Example of the iterative lifecycle model.

The advantages of iterative lifecycles are that they allow for adjusting the project course during development and early risk reduction, and they also provide support for early testing of analysis models in draft stages. However, because the iterative lifecycle models are more complex, the setback is that such projects are more difficult to plan and control. [4]

2.2 Workflow

This section establishes the definition of workflow as used in this thesis. Workflow management systems and requirements for implementing them are also discussed. Workflow is characterized by a list of activities that need to be done to accomplish a task. There is a difference between workflow and process descriptions: workflow describes the ‘how’ while process is more akin to ‘what’ and ‘why’.

The workflow concept is used in this thesis as a means of providing a detailed work description for a single worker. A detailed work description specifies the interconnections existing between different tools and identifies the working practices with the tools.

Workflow provides guidance as a number of logical steps for an individual worker on how he or she can accomplish the tasks needed to perform the job. In addition, workflow defines the order of steps, or conditions under which steps must be invoked, step synchronization, and information flows [9]. Typically workflows (or workflow tools) have been used in various kinds of form management, e.g. order form management. However, in this thesis workflow is used to provide guidance for design and development work.

At times it may be difficult to distinguish the difference between a process and a workflow. According to the description by Wikipedia a process can be separated from a workflow by the

2. The development process

fact that a process has well-defined inputs, outputs and purposes, while a workflow applies more generally to any systematic pattern of the activity. [10]

The workflow concept has evolved from the notion of process. Workflows widen the scope of process modelling because they support complex control flows, rich process structures and integrations of different systems that cannot be described in process models. [11] Moreover, workflows describe only one process, but a process can consist of many workflows.

Workflow management technology provides methodologies and software to support process modelling as workflow specifications [9]. In this thesis workflow management is understood as an information system that helps in the execution of a workflow of an individual worker. For example, a workflow management system may aid the worker by automatically invoking applications, entering data, and providing assistance on each step of the process.

For functional workflow implementation (i.e. workflow guidance implemented by a workflow management system) it is first necessary to model the process workflows. Modelling of the workflow can begin when the process is understood. Understanding of the process can be achieved for example by interviewing the people who know the process well. All actors (both human and application) that take part in the process task executions and the steps they need to take to complete the task must be solved. It is also necessary to clarify related activities such as data flows, constraints, controls, exceptions and priorities. If needed, it is possible to rationalize the process after it has been modelled, as there may be steps that are no longer necessary. After sufficient understanding of the process is established, a workflow specification can be written. [9]

In practice it is also necessary that the workflow implementation is flexible. Flexible in this context means that it is not necessary to perform steps each time in an identical manner, which is the traditional way in most workflow implementations, but rather that the system should allow for alternative paths to complete the job.

3. Tool integration

Lack of interoperability between tools has been identified as one of the most significant challenges in product development. This is a problem especially in embedded systems development due to the multitude of tools needed and the jungle of existing data dependencies between the tools. Increased interoperability enables for example efficient reporting of project progress, smooth transfer between tasks, and availability of correct information for all. Building tool integration(s) between separate tools is one way to enhance interoperability. However, due to the disconnected nature of tools, the building of the integrations can be difficult and burdensome in many situations. Literature provides various approaches that can be employed to help in the building of tool integrations. However, there seems to be no one correct way of doing things. For example, large tool vendors circumvent integration problems by bundling their tools into one product with shared data. Different tool integration approaches, their relative benefits, and existing integrations are discussed in this chapter.

3.1 Why tool integration is necessary?

Size and complexity in embedded systems software is growing quickly. At the same time, however, the software industry is finding it difficult to deliver quality software at an ever-increasing pace. Increasing the total number of staff does not solve the problem adequately, as the number of capable software engineers is limited and development becomes too costly. Therefore, one way to improve productivity is to use productivity-increasing technologies and work methods that make it easier to excel in embedded systems software development. In practice this means that more development tools are needed. Furthermore, cooperation between partners is necessary, and each partner has its own set of development tools. Data between the different tools (and partners) is related and needs to be managed. Otherwise the results will be less than satisfactory.

Thus, one of the major problems identified in embedded systems software development is the poor interoperability of development tools. Current existing tools are not interoperable or provide integration only to some specific tools. In some cases, manual work is needed to accomplish tool interoperability, which in turn wastes resources and causes errors. [12] There are also totally integrated tool sets, which have their advantages, but they also have limitations such as creating dependency on a particular tool vendor.

Attempts by either the tool users or developers to integrate applications outside of a single-vendor-bundled tool set have generally been limited to various forms of data import, which creates recurring manual administrative effort and related configuration management problems by having data in multiple places. Usually, a small group of dedicated personnel end up acting

as an interface with the system engineering tool set while others continue to do their work in common desktop applications. In practice this has resulted in the fact that the most often used tools for systems engineering tasks are actually Microsoft (MS) Office products despite the efforts of system engineering organizations to move toward purpose-built commercial tools. [12, 14]

Integrated tool environments aim to minimize tool deployment time, standardize the process, and improve the efficiency of projects. Efficiency can thus be improved by rework and disruption avoidance, and by better integration of project functions. [12] Integrated tool environments also enable tracking the consistency of work products and provide transparency in project progress. [13] Moreover the development process becomes easier to understand when the user does not have to work with disconnected tools and has (workflow) guidance provided by the environment.

The European Union has identified tool interoperability as one key challenge. In European Commission Information and Communication technologies (ICT)-call 2007 Objective ICT-2007.3.3: ‘Embedded Systems Design’, tool integrations are specified as one of the target outcomes: *“Suites of interoperable design tools for rapid design and prototyping: integrated tool chains that respond to the needs of industry for designing and prototyping embedded systems. Research will contribute to one or more of: ... (3) open tool frameworks facilitating new entrants and the integration of the tool chain including associated standardization. Key issues include: (i) technology for efficient resource management, (ii) optimising compiler technologies, including parallelisation, taking into account features of the targeted execution platforms and extra-functional requirements; (iii) optimised tools respecting trade-offs when co-developing hardware and software; and (iv) model-driven development.”* [15]

3.2 Tool integration approaches

The number of different tools used in HW / SW development is enormous. Therefore it is not beneficial to integrate all the existing tools. Some tools have greater integration value than others, and this should be considered when planning the integration. [12]

Different tools store their data in their individual repositories. Each tool repository is different in a way that the database is designed to store information for that tool only. For successful integration, it is necessary that the database can be somehow interfaced. The difficulty level of tool interfacing is one factor to consider when choosing which tools to integrate. Different methods exist for interfacing tool-specific data, e.g. Java Database Connectivity (JDBC) and Open Database Connectivity (ODBC) connections, and application programming interface (API). Tool interfaces are studied in more detail in chapter 3.2.1.

Data between tools has to be connected. One step in integration is identifying how the data from different tools relates to each other. For example, test cases from the testing tool are related to requirements from the requirement management tool. Another step is to identify how to use this collected information. For example, requirement coverage by a test cases report could be generated based on this information. One way to begin planning integration is to use the company’s current “manual integration” process as a guide on how the data from tools relate to each other, if at all.

The integration should be designed so that additional restrictions are not needlessly placed on how the data can be mapped between tools. Pederson suggests that ideal data relationship architecture should allow for many-to-many relationships between any managed data elements.

Designing such a relational model is relatively easy, if the data elements can be tracked by a unique identifier or key [12].

There may sometimes be an overlap of functionality between different tools (i.e. data which is similar in nature can be stored by different tools). This should be avoided because having similar data in multiple places causes configuration management issues. Thus, perhaps the simplest solution is to use only one tool for one type of data, and the tool can be chosen for example on the basis of which tool has the most functionality in managing that specific type of data.

The current approaches to tool integration are described in table 1. The descriptions in table 1 are based on [16, 17, 18, 19]. Of these integration approaches, framework-based integration is the most interesting in the scope of this thesis.

Different levels of application integration are identified by Pederson as follows [12]:

Link or Reference: adding a hyperlink or association to launch one application from another. Common to collaboration tools and acts as a navigation aid, but does not support any sort of data association or purposeful navigation within the associated tool

Data association: data is shared between applications. Both applications could use the same common data set for common content or, alternatively, data could be pushed and pulled from one application to another. A weaker form of this sort of integration would be a one-way data push only where the one application can read but not edit data. This is adequate in many cases depending on what the applications in question are intended to do. The interfaces are independent in this scenario.

Interface integration: allows one application to initiate another application's user interface and navigate directly to the desired record. This must be done without having to log onto the associated application or to go through its interface hierarchy to have any value. Otherwise, it is no different to a link or reference and is actually clumsier than simply sharing data in different interfaces.

The desirable level of integration depends on the application and the need. For interface level integration, support needs to be provided by the target application. [12]

Table 1. Current approaches to tool integration.

Approach	Description
Piece-meal	Tools are applied to achieve improvements in specific lifecycle phases, and migration between the development phases is done manually. This approach focuses on specific phases and lacks the view to the overall improvement between the phases
Single-vendor	This approach attempts to improve all lifecycle phases with a full-lifecycle product from a single vendor. The approach requires selecting a vendor with the technical expertise required to effectively support all life cycle phases, creating the risk of locking the organization into a costly, proprietary solution.
Best-in-class	This approach attempts to integrate the best tools of regarded domains, typically from different vendors, for specific lifecycle phases. Best-in-class has two variations, point-to-point integrations and framework-based integrations. Point-to-point integrations, or one-to-one integrations, provide integration built between two defined tools specifically. Point-to-point integrations are adequate only for small numbers of integration endpoints and typically create more complexity in developing and managing tools than they solve. This is the most common type of interface between systems engineering tools. [20] Framework-based integrations attempt to classify tools and provide integration between tool classes based on vendor-neutral interfaces and mechanisms. The framework-based approach provides an integration environment and common look and feel without limiting the choice of tools. [12]

3.2.1 Data integration interfaces

For integrating an application it is necessary that some kind of interface is available for accessing data. The interfaces of the tools have usually been built for some specific purpose (e.g. export of requirements from a requirements management tool in DOC-format, which can later be read in MS Word). Interface(s) can also be added later via modifying the original software.

Possible levels of integration (chapter 3.2) are somewhat related to the available interface types. API integration generally allows for the highest level of integration and application interfacing. The following section briefly introduces the most common types of interfaces, as described by Pederson [12].

Data table association: data is read directly from database tables either directly or through a broker table structure. This method is generally considered to be the direct and simplest but in some cases reading data directly from an application's table structure can be difficult to interpret.

API: programmed interface that writes to or reads from an application. All of the common Microsoft desktop applications have common APIs and so do many other applications. APIs avoid uncertainties regarding table data associations and event triggers.

Static data import/export: in this case a delimited file of some sort, most often MS Excel or comma separated values (CSV), is created in one application and read into another. When the source data changes, the data in the other application will no longer be current. This is the weakest form of data access but it can be programmatically automated and is appropriate for certain applications and processes. It is also the easiest to put in place on a one-time basis.

One has to be careful when using the software's database backend as a point of integration, because it is easy to make the wrong assumptions (especially when reverse engineering) about how the application works internally. If data from the tools is used under these conditions, it is possible that the meaning of the data is interpreted falsely. Moreover, if data is pushed into the application database, this may break the application. One of the dangers is the updating of an application so that the internal structure of the software is modified, which will also break the integration unless the integration is updated accordingly.

For these reasons API-level integration or another equally powerful means of integration should be employed because they guarantee that the application data is used and understood as intended and the application state stays consistent. The integration, however, can be broken in this case too if the API specification is modified without regard to backwards compatibility.

The best integration approach depends on the type of application, on the overall situation, on the resources available for building the integration, and so on. Some of the issues that have to be considered during integration planning are access control, security, and consistency of data. For example, in the case of static data import/export, it is difficult to keep the data consistent, and special means may be needed to guarantee data consistency. However, not all applications in all situations must guarantee data consistency. Furthermore, care has to be taken not to bypass access control and security measures of the application when building integrations. For example, in a situation in which the data table association is used as an integration interface, security and access control schema can be easily bypassed if the same mechanism is not built into the integration.

3.2.2 Tool integration from an application lifecycle management perspective

Another relevant topic regarding tool integration is application lifecycle management (ALM). ALM means the coordination of development lifecycle activities, including requirements, modelling, development, build, and testing, through enforcement of processes that span these activities, management of relationships between development artefacts used or produced by these activities, and reporting on progress of the development effort as a whole [21]. ALM operates with the artefacts produced and used during the lifecycle of SW products by providing visibility into the status of the evolving SW product. A variety of solutions provide mechanisms to represent different types of traceability links between developmental artefacts. However, the interpretation of the meanings of such linkages is often left to the user.

Kääriäinen et al. have specified an ALM framework in an industry case study for evaluating the current state of the ALM solution in the target organization and for detecting ALM elements that possibly need improving. [22] This kind of framework can be used to understand what kinds of requirements need to be implemented by the integrated tool integration solution. Elements of the ALM framework are as follows:

Creation and management of project artefacts: how are different data items created, identified, stored and versioned on various phases of a project lifecycle? All project data should be securely and easily shared with all stakeholders. Team communication should be supported. [22]

Traceability of lifecycle artefacts: how is traceability in a project lifecycle handled? Traceability provides a means to identify and maintain relationships between artefacts and, therefore, facilitates reporting, change impact analysis and information visibility through the product lifecycle. [22]

Reporting of lifecycle artefacts: how does the solution support reporting on a project lifecycle? The solution should facilitate the gathering, processing and the presentation of process and configuration item-related information for an organization. [22]

Process automation and tool integration: How well do the tools support lifecycle processes and what kinds of tool integrations are there? An ALM solution should support the procedures of the project and facilitate fluent data exchange and queries between various development and management tools. [22]

3.3 Existing implementations

The existing implementations of tool integration include company-specific tool integrations (piece-meal or point-to-point) that are not publicly available (and thus not discussed in this chapter), various frameworks, and single vendor solutions for ALM. Solutions offering a complete product development environment including integrated development tools have also appeared. Examples of these implementations are presented in tables 2–3. [13]

3. Tool integration

Table 2. Examples of tool integration implementations.

Name	Type	Description
Borland Open ALM	ALM solution	Several Borland tools (e.g., CaliberRM and Caliber DefineIT, Together, SilkTest, StarTeam) are integrated with each other. CaliberRM integrates also with MS VSTS and Eclipse.
IBM ALM solutions	ALM solution	IBM tool portfolio covers all development lifecycle stages integrating several IBM Rational tools (e.g., RequisitePro, Rational Rose, Rational Software Architect, Rational Functional Tester, Rational Performance Tester and ClearCase CM).
Microsoft Visual Studio Team System (VSTS)	ALM solution	A development platform that supports various phases of the SW development lifecycle. The backbone is the Team Foundation Server, the central point of contact for project and process management. Process guidance is also provided via the Microsoft Solutions Framework (MSF).
Eclipse	Framework	Eclipse provides use mechanisms and rules for integrating tools. The Eclipse Platform offers good support to extend its functionality by plug-ins. Eclipse simplifies tool integration by allowing tools to integrate with the platform instead of each other [18, 23].
MODELBUS	Framework	MODELBUS offers a tool API independent layer of abstraction for exchanging models using Eclipse EMF metamodelling technology. The aim is to simplify access to tool data in a distributed environment based on SOAP middleware. MODELBUS does not offer support for, e.g., management of traceability links across tools.

Table 3. Examples of tool integration implementations, continued.

Name	Type	Description
ALF	Framework	Application Integration Framework (ALF) is part of the Eclipse foundation. The project aims to provide a logical definition of the overall interoperability business process. This technology handles the exchange of information, the business logic governing the sequencing of tools in support of the application lifecycle process, and the routing of significant events as tools interact.
IBM Jazz	Framework	IBM Jazz attempts to build a scalable, extensible team-collaboration platform for seamlessly integrating tasks across the software lifecycle. Jazz is based on Eclipse, and is a kind of middleware layer for linking development assets.
Model-based tool management and integration platform	Framework	The platform supports model integration, where models defined in different tools for different aspects of the same system are related such that they may share and exchange data. The integration platform also enables model management functionalities on a fine-grained level. The approach is based on the Matrix PDM tool and stores a copy of the data in the tool, which then creates consistency problems. [24]
Fujaba	Framework	Mechanism to integrate different tools on the metamodel level. A consistency management system is included, especially for the integration of different or enhanced metamodels. The Meta-Model Extension and Meta-Model Integration patterns enable the integration of data in different scenarios on the meta-model level. [20]
CollabNet	Development environment	CollabNet is a collaborative development environment where the developers and IT project managers collaborate online through CollabNet.

3. Tool integration

As can be seen in the tables above (tables 2–3), many tool integration frameworks and solutions already exist. However, these solutions do not optimally solve the integration problem. For example, the existing ALM solutions are single vendor-specific tool integrations creating dependency on the vendor, regarding for example version support and future developments of the tools. Single vendor solutions also limit the choice of tools; companies cannot choose the individual tools that would fit to the situation and specific purpose the best, but have to consider the development tools as a whole. This results in having to use a set of on-average best tools available for the situation, whereas free selection of individual tools would provide better support when well integrated. The same is true for the development environments. Regarding tool integration frameworks, they are often generic, providing flexibility for the integration, but on the other hand causing the integration having to be planned and built from scratch each time, resulting in a large amount of effort needed. [13]

4. Hardware-related software

This chapter gives an introduction to the problem domain of developing hardware-related software. A definition on what is meant by HW rel. SW is offered, and typical challenges in HW rel. SW development are identified.

HW rel. SW is a sub-category of embedded software. Wikipedia defines embedded software as computer software or firmware that plays an integral role in the electronics with which it is supplied [25]. Embedded software is executed in different kinds of environments such as cars, telephones, and airplanes [26]. HW rel. SW is characterized by consisting of software components that directly interface with the underlying hardware.

Digital signal processing (DSP) is a good example of a situation in which HW rel. SW is needed. In digital signal processor (DSP) programming developers may need to write software in assembly language to take full advantage of the available resources. In order to write efficient and correct implementations of algorithms (e.g. infinite impulse response filter) it is thus necessary to understand how the underlying hardware functions. When the hardware is thoroughly understood, the programmer can partition the algorithm so that no clock cycles are wasted in inefficiencies, e.g. taking advantage of parallel multiply and accumulate (MAC) operations (in the case multiple MAC units exist in the hardware) in the implementation.

Moreover, as embedded systems must interact with the physical world it is also natural that they have features characteristic to them such as power consumption, timing constraints, and reliability requirements. These challenges that HW rel. SW developers have to consider will be studied in more detail in the next section.

4.1 Challenges

More often than not, embedded systems have to interact with the physical world. This fact creates additional challenges for embedded software designers; for example, timeliness, concurrency, liveness, reactivity, and heterogeneity of the system have to be considered.

Some of the constraints relate directly to the nature of the problem domain: for example, when creating real-time systems aspects like schedulability, predictability and robustness need to be taken into consideration. There are also constraints that result from the drive to reduce the system cost to improve the product's market fitness by using cheaper and less powerful components. In real-time and embedded systems there is a need to operate with a minimum memory footprint and with a minimum of support hardware. [4]

All of these factors, and more, are essential to the correctness of the system and will be examined in the following sections. The challenges were identified from literature and from an interview with a DSP expert with years of development experience in the telecoms industry.

4.1.1 Product lifecycle

One of the challenges identified is that there are systems that have long lifecycles: they need to be maintained years or even decades after initial release. These systems can be built so that the hardware portion of the system remains mostly unchanged and maintenance changes are implemented in the software. Therefore, these systems must be designed to have enough computational performance in reserve for future software updates. Software is updated to implement new technologies and is in general more computationally intensive than the current generation technologies for which the system was initially built. It may also be possible that these new technologies have not yet been standardized or are in the draft stage, and it is therefore difficult to predict how much performance is needed in reserve. Another aspect that requires maintenance and sets limitations on the system is the need for backwards compatibility to legacy versions of the product. [27]

Design of the system may begin before all the available components have been finalized; e.g. the DSP processors used by the system are in prototype stage. Thus, the draft status of the specifications makes the design process even more difficult, while validating the understanding of how the hardware functions becomes harder. In addition, integration and validation testing become more difficult and lengthy. [4, 27]

4.1.2 Performance

Performance issues in HW related SW development are plentiful. When evaluating the performance of the system under design one of the issues that has to be addressed is the 'latency budget'. This issue is closely related to timeliness, which is a critical issue identified in the development of real-time systems. Latency budget means that when a signal or trigger arrives, certain tasks have to be done until a deadline, and the time from signal arrival to deadline is divided into different tasks according to some criteria. The division may be performed on the estimation of how much time each task should take. Estimation may be based for example on the group's educated guess as to how much calculation time is needed for the task. There is also the human factor involved that each group working on a task would like to use all available time budgets. [27]

Optimizations may be needed if the latency budget is overrun. Optimizations should be focused on the task that takes proportionally the longest time to execute, because there might be the possibility for the greatest time saving. However, it is sometimes difficult to decide when to optimize: now or later. [27]

Interrupt behaviour is an important factor relating to platform performance and latency budgets. Two important scenarios should be mapped that will reveal how the system performs under load: the worst case scenario in which the amount of interrupts peak and the average situation. Studying these situations could yield surprising results; it may be possible that the system is not able to handle an interrupt 'avalanche' formed in the worst case scenario. This is also an example where the system performance is affected by the system architecture design: if external interrupts are used excessively in a multi-processor setup it could have a paralyzing effect on system performance. Namely if there is a processor dedicated for control tasks, and it is on the receiving end of the interrupt avalanche, then different approaches need to be explored. [27]

4. Hardware-related software

Power consumption is a performance issue that needs to be addressed, and it is especially important in mobile embedded devices. However, in some cases heat generation may be a bigger issue than power consumption. In devices where the heating surface area is small it may be difficult to cool down the device adequately without novel methods. In addition, in some devices like mobile phones, there is a certain limit for heat generation (so that the mobile phone does not burn the consumer). [27]

4.1.3 Memory handling

Various memory issues are prominent in the development of embedded systems. Embedded systems come with many different types of memories of varying sizes and performance characteristics. The engineer must carry out his/her work with a minimum amount of memory to decrease the manufacturing costs of the device, while making out the most of what is at his/her disposal. Activities that need to be performed may include things like deciding on how to use caching, choosing what types of memory to use and for which purposes, for example, when to use the faster memory found on the central processing unit (CPU) and when to use the slower general purpose memory. [27] Furthermore, memory allocation may not be automatic, but a real-time operating system (RTOS) can be used to handle the allocation. [4]

4.1.4 Testing

Testing aims to improve the reliability and performance of the system under development (SUD). Some of the things that need to be tested in embedded systems include memory leaks, manufacturing faults, and software errors. Numerous different testing methodologies exist: white box-, black box-, positive-, negative-, stress testing, and so on. These testing methodologies can be employed in different testing activities such as unit- and regression testing. Testing consumes time, and no system can be entirely covered by tests; in complicated systems there are simply too many variables to cover all the possible scenarios. Enough time should be reserved for testing, and tests should be designed so that they cover the most important aspects of the system under test (SUT). [27]

For testing embedded systems specialized testing gear is sometimes needed, but this does not come cheap. The budget may allow for only a limited amount of testing beds and thus imposes restrictions on test execution, ultimately further emphasizing the importance of test planning. [27]

Embedded systems present challenges to testability. The system may initially support different interfaces and methods of debugging and testing such as the Joint Test Action Group (JTAG) interface, extension for logic analyzer, memory dump, debug messages, boot loader which performs error checking, and so on. However some, if not most, of these capabilities are often discarded and removed from the final product to save costs or for other reasons. Corporations like to protect their intellectual property by any means necessary and therefore make reverse-engineering of the system as difficult as possible. For example disabling the JTAG interface, which is a powerful tool for debugging, may be such a step. Removing of the debug interface does not come without a cost though. Maintaining the product with an extended lifecycle becomes more problematic when these testing and debugging means are reduced or removed. Great care should be taken during system development to weigh which testing and

diagnostic capabilities to include in the device. The consideration should be made case by case; testing capabilities left in the device consume limited resources and are not necessary in every application. [27]

There are also several other issues related to testing. It is important how tests are handled and stored because in embedded systems there is more variation in a testable system and its components than anywhere else. Therefore a proper version management system is essential, so that tests can be replicated and additional errors caused by issues such as false testing parameters, different compilers used for compiling, and script versions used for running tests, can be eliminated. [27]

Testing uncovers bugs. In reality, bugs are everywhere; even the testing software and test platforms contain bugs. Processors, operating systems (OS), compilers, development environments, integrated circuits, and software in general contain bugs. Testing is even more important if the SUD contains several components which have not yet matured. It is not rare that seemingly trivial changes introduce problems that are difficult to trace: for example changing the manufacturing process of the integrated circuit could cause timing or other problems even though both processes should produce identically functioning integrated circuits (IC). [27]

The importance of unit testing cannot be emphasized enough. When software needs to be modified, and if the change is made by a different team or person that originally created the code, unit tests try to guarantee that the software block works as intended after modifications. Thus, during integration testing effort can be better spent concentrating on finding problems related to integration, because it can be guaranteed beyond a reasonable doubt that the software works as intended. Unit tests need to be checked after software modifications. Unit tests can also be used to help in understanding the purpose of the code block. [27]

Running unit tests in a cross-compiler environment has some characteristic difficulties. Because software is developed in a desktop environment, the code cannot likely be executed in a desktop environment, or it can be too slow if it is run in the emulator. In such a case a setup needs to be planned regarding how the unit tests can be executed, possibly in a target environment where the execution is swift. [27]

4.1.5 Timeliness

Timeliness is a constraint placed upon certain action; action that is completed before a set deadline is considered timely. Action can begin in response as triggered by an event or by due-time, and it must then complete within a defined time limit after beginning execution. Timeliness is a factor that has to be considered when ensuring correctness of an action. [4]

Deadlines can be defined as ‘hard’ or ‘soft’. If an action misses a hard deadline it leads to a system failure, because this is what a hard deadline means: deadlines *must* be met in all cases. If an action completes after a hard deadline, it is then considered useless. To ensure that a system meets hard deadlines the following modelling concerns are important to consider: execution times, deadlines, arrival patterns, synchronization patterns and time sources. In the case of soft deadlines, a system allows an action to miss a deadline and is designed to operate correctly even in such circumstances. The system may respond in such cases for example with reduced performance. [4, 28]

If an embedded system uses a processor that relies on techniques such as branch prediction, speculative instruction execution and elaborate caching schemes, its reliability may be difficult to guarantee. This is because it is difficult to analyze how such a system performs under load. [26]

4.1.6 Concurrency

When two or more computational processes execute simultaneously, the processes are called concurrent. Pseudo-concurrent processes execute in the same execution unit, in which case there is only one process executing at any given time and the executing process is switched by the operating system. True concurrency is the execution of multiple processes assigned over several execution units. However it makes little difference whether the execution is truly concurrent or not; common concurrency issues remain. Challenges in concurrency relate to things such as scheduling of concurrent threads, arrival patterns of events, synchronization of threads, and controlling the use of shared resources. [4]

Different strategies exist for scheduling concurrent threads. They differ at least in implementation complexity and in performance quality. The most common are:

- First in, First out (FIFO) run-to-completion event handling
- Non-pre-emptive task switching
- Time-slicing round robin
- Cyclic executive
- Priority-based pre-emption.

In addition, a RTOS can be used to control the task execution and prioritization of a multitasking system. [4]

In reactive embedded systems threads can be scheduled according to incoming events. The arrival patterns of the events may be either periodic or aperiodic. A periodic pattern occurs at a fixed rate with the possible addition of a small variation. Aperiodic events have different timing characteristics such as irregular, bursty, bounded, bounded average rate and unbounded. Arrival patterns have to be analyzed, for example, to determine buffer sizes so that no events are lost. [4]

Thread rendezvous patterns deal with issues related to tasks running in different threads and exchanging information. Tasks are run in different threads to separate the timing of execution from other tasks. However, if there is a need to share data or control information there is also a need for task synchronization. Messaging between tasks can be performed for example using synchronous or asynchronous function calls, waiting, timed, and balking calls. Rendezvous of cross-threaded messaging is important to characterize so that deadlines will always be met. RTOSs provide a variety of techniques for inter-process communication. [4]

Sharing resources is a problem when multiple tasks compete for a shared resource. For example one task may be in the midst of reading a value from a shared resource, task switching occurs, and another task writes a new value in the shared resource. Subsequently the first task would get incorrect results. To prevent this, access to shared resources can be serialized for example by using semaphores or queues. Thus the correctness of the value is guaranteed by ensuring that only one task at a time accesses the value. [4]

Semaphores, threads and processes provide tools for managing concurrency, but because of their low level of abstraction complex compositions built using these tools become too hard to understand for most. [26]

4.1.7 Interfaces

Challenges exist in trying to model the real world situation from the software/hardware perspective using different abstraction mechanisms. One of these abstractions is called procedure and is best described as finite computations that accept arguments and after execution produce results. However, procedures are far from an ideal match for embedded systems. In some cases there may be need for a unit that acts more like a process than a procedure. Process is a continuous computation that transforms a stream of input data into a stream of output data. For example in a mobile GSM phone there is a Viterbi decoder which may be implemented in a dedicated signal processor, and it acts more like a process than procedure, transforming a stream of data. It is not an easy task to package such a system so that it can safely share computing resources with other computations. [26]

One more reason why building concurrent embedded systems is so challenging is because it is extremely difficult to characterize aggregate systems composed of processes and threads. For example, an aggregate of two processes is not any known component, at least not a process. In addition, there is a need to include properties such as concurrency and dynamics in programming language definitions. [26]

The issues mentioned are challenges that stem from trying to model the real world into a software/hardware perspective using the abstractions that are available, but there are also other types of challenges relating to interfaces. For example, it is characteristic to an embedded system to have low level control of the hardware. The software needs an interface for the hardware, and this interface is called the driver. The driver is usually provided with the hardware, but the hardware in question may need a custom driver because the hardware is not mass manufactured, the RTOS used does not have a suitable driver for the hardware, or the driver exists but has poor performance qualities. [4]

Multi-processor environments create additional complications for interface design. In real-time systems it is critical to convey the signal of a time-critical event to the target component as quickly as possible. Several challenges need to be addressed such as data-bus structure, when to use direct memory access (DMA), where and when to use direct signals (I/O ports, serial buses etc.) between components and so on. Often it is not clear what kind of architecture would yield the best results and comparing different alternatives is difficult. In such situations the educated guess of an experienced designer may be the only tool that is available. [27]

4.1.8 Heterogeneity

Embedded systems are by nature heterogeneous. Because these devices can be built for nearly any purpose, a variety of implementation technologies and styles exists. Embedded systems are also a mix of hardware and software: the software is written specifically for the hardware in question. [26] For example a first generation system may use a different CPU than the next generation system, and if the new CPU is not backwards compatible, this may lead to major rewriting of software components. Different software versions or implementations might also be used for the same underlying hardware along the product family.

Because embedded systems operate in various (real-world) environments, they are also exposed to a variety of stimuli. These stimuli are called external events; events need to be

handled, according to some constraints. Timeliness constraints for these events vary by event type, and possibly the events are handled in an ad-hoc way in the real-time software. [26]

In addition, embedded systems developers often need to combine multiple programming languages, mainly because there is no common model that is the solution to everything. This is an indication of the fact that embedded system developers use the tools that best fit the task in hand. [26]

The multi-processor environment mentioned in chapter 4.1.7 provides an example of a heterogeneous embedded system. The system may mix application-specific integrated circuits (ASIC), field-programmable gate arrays (FPGA), DSPs and other ICs. Finding a suitable architecture and interconnections for the components while keeping the various quantitative and qualitative requirements in mind is a challenging task. [27]

4.1.9 Reactivity and responsiveness

Lee characterizes a reactive system according to its ability to react continuously to its environment at the speed of the environment [26]. By definition most real-time systems are reactive systems. They connect to monitoring and/or control hardware and operate under real-time constraints, and are possibly safety-critical. Most of the challenges faced by reactive systems originate from their operational environment, the real world. The surroundings of the device may be unpredictable but the device must still respond correctly to events when they occur. [4, 26]

Various models and equations have been created to model the physical world. These models have limitations when they produce valid results. When the engineer uses these models and equations and violates limitations such as linearity or considers elements independent when in reality they are not, problems occur. [4]

Reactive systems can be modelled using finite state automata (FSA). FSAs are mathematical models and thus have strict rules. States in the automata have firm and distinct boundaries. This has possible indications in control systems where discontinuity between states could lead to problems stabilizing the system. [4]

4.1.10 Predictability

One aspect characteristic to many real-time systems is that they must be predictable. A predictable system's response characteristics are known in advance. This is expected of safety-critical and high-reliability systems, such as those where human safety has to be considered. Response characteristics can be determined for example by static mathematical analysis methods such as rate monotonic analysis (RMA). Measures that can be taken to guarantee predictable behaviour in embedded systems can be disabling pre-emption or using control algorithms with known behaviour. [4]

4.1.11 Correctness and robustness

System correctness and robustness are design considerations that have to be addressed in the development of embedded systems. Correct action is logically valid and is performed in a

timely fashion. [28] A robust system is designed to operate correctly even when parts of the system fail. Robust system design is especially important in military and hospital applications. Achieving correctness and robustness in complex systems is not a trivial task. System designers must somehow design the system to cope with all the anticipated failure scenarios but also scenarios that were not initially anticipated. [4]

4.1.12 Distributed systems

Embedded systems may be distributed across many processors where the processors may be physically on the same board or in different locations. In distributed computing additional challenges that need to be considered include coordination of tasks running on different processors, managing boot-up processes and design of communication interfaces between separate processors. [4]

Distributed computing can be used to solve high availability requirements and different performance requirements. A system with redundant components allows parts of it to fail gracefully and thus can be designed to provide high availability safely. Redundancy can be captured and represented by using architectural design patterns. [4]

Distributed computing can help solve performance requirements that could not be met with a single component or that would otherwise be too costly. Parallel processing systems are a form of distributed computing. Not all calculations can be effectively divided for parallel computing, but in cases where this can be done performance can be scaled up by adding more computational units. It may also be the case that a single processor type is not adequately suited for all types of calculations, and then a specialized computational unit can be assigned to perform those computations. ASICs and similar specialized processing units could be used in these cases. Determining how to divide different computational tasks to different processors and choosing the best processors for these tasks can be challenging. [27]

4.1.13 Resource limited target environments

In embedded systems smaller CPUs and less memory lower manufacturing costs. Therefore the embedded-system developer often needs to cope with the push to reduced hardware component costs, physical size, heat production, and weight by reducing the number, size, and capability of the target computing platform. [4]

Embedded system developers need to use cross-compiler tools, which are more complicated to use than the ones available for desktop software development. Debugging embedded systems is also more difficult. Some of the embedded systems hardware capabilities are difficult or impossible to simulate on a workstation. The developer has to come up with alternative means to execute and test the code, and this consumes time and effort. [4, 27]

4.1.14 Subcontracting

Embedded systems are currently so complex that a single vendor cannot manufacture everything itself. Hardware and software components need to be subcontracted. Choosing a subcontractor for a component has various challenges such as price and performance of the component,

contractor reputation and reliability. The customer of the SUD can also have requirements for subcontracting: which vendor and/or model of component to use, or questions as to why a certain component was chosen when other platform manufacturers use a different component from another vendor. [27]

4.1.15 Managerial challenges

HW related SW development faces some additional challenges besides those that are commonly known such as physical distance and cultural differences in teams. When multiple teams with different disciplinary backgrounds work together on the same system, it is not always clear which changes or issues should be communicated to others. This is an issue that can be resolved to a degree with more transparency in communication (like group meetings), but even then it is not always clear or understood what other teams are working on and what measures should be taken. [27]

4.1.16 Summary

As can be seen, building embedded systems poses many different and difficult challenges for embedded systems engineers. Some of the problems addressed (e.g. timeliness, concurrency, reactivity and responsiveness) are directly related to real-time systems. There are also difficulties that all embedded systems developers share in general such as the need to cope with less powerful hardware due to the drive for cheaper manufacturing costs and the handling of complicated interfaces. Performance and memory issues are also prominent in embedded systems development. Furthermore, debugging and testing embedded systems is considered difficult because of a lack of sophisticated tools and because embedded platforms complicate testing. Activities related to these challenges are studied in the next chapter and requirements are defined for tool support which will try to address some of these challenges.

5. Requirements for tool support

This chapter introduces the ToolChain (TC) process model (figure 5). The basic structure of the model was created with the help of the common process models introduced in chapter 2. Activities related to the HW rel. SW development challenges (chapter 4) are identified and possible tool support is discussed. The activities are then mapped to the TC process model. Finally, the collected requirements for TC are presented. The requirements are gathered from the TC process model, by selecting general requirements for tool integration and by focusing the tool support on certain activities and challenges.

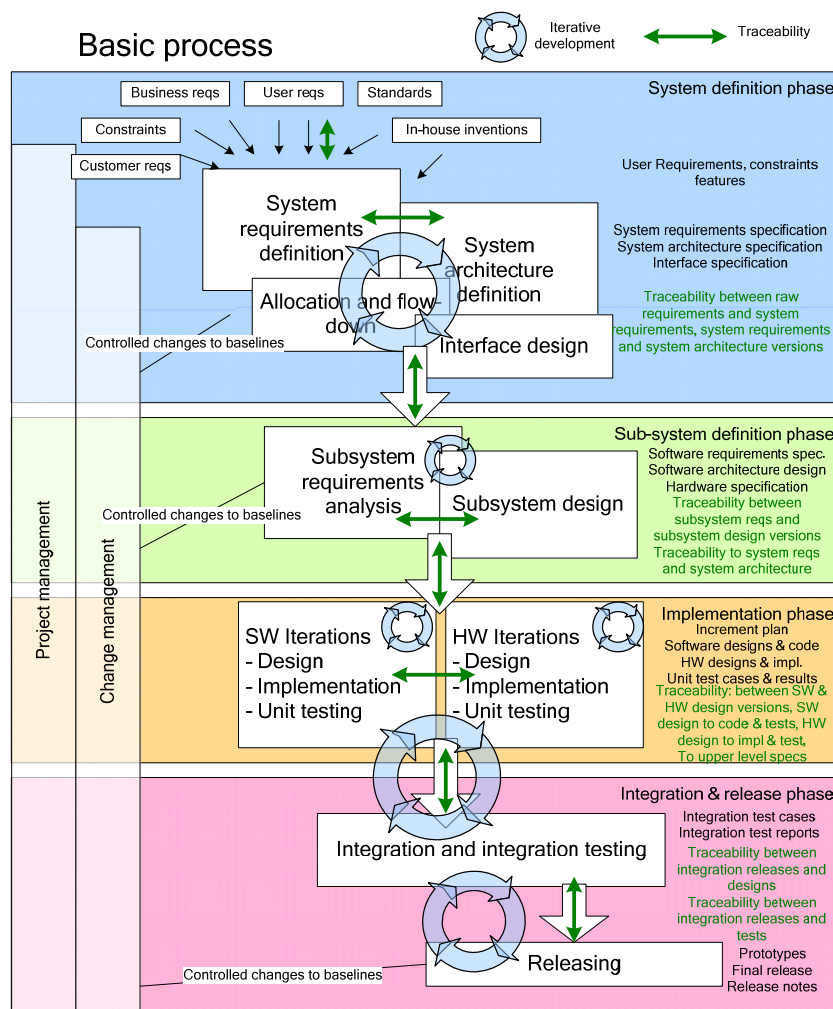


Figure 5. The ToolChain process model.

5.1 ToolChain process model

The challenges in HW rel. SW development are plentiful but need to be solved. A single challenge may need to be addressed in multiple development phases, each time from a different perspective, with different tools. As a result, the development of HW rel. SW is highly complicated to perform. Developers need to use many different tools and methods. The TC process model (figure 5), presented in this chapter, was created by composing phases from well-known models (chapter 2). The model does not present a complete PLC; instead it begins from system definition and ends in the integration & release phase. The model presents activities relevant to HW rel. SW development. The activities are related to the challenges identified in chapter 4 and are organized by process phase (e.g. activities related to system definition).

5.1.1 System definition phase

The system definition phase (tables 4–7) begins with requirements gathering. The requirements are gathered from various sources as indicated in figure 5. The process of how these inputs are gathered is not further elaborated, as it is outside of scope of this thesis. When the first set of requirements is defined, system architecture definition can begin. The system architecture and system requirements definitions proceed in iterations. This work also produces interface descriptions. Once a reasonably stable set of system requirements and architecture are defined, the process moves to the sub-system definition phase.

Table 4. System definition activities.

1.System definition	
<i>1.1 Requirements gathering</i>	
Increment plan	<p>Purpose of the increment plan is to provide a timetable for implementation and to provide steps for each increment. The first increment is planned in more detail than the following increments, which are planned at the end of the preceding increment, in less detail.</p> <p>Focus in the first increments is mostly on requirements definition, architecture definition, and less in implementation and testing. Focus will gradually shift towards implementation and testing as the project matures. Each increment should aim to build components that can be tested; e.g. an increment could consist of analyzing use cases, implementation, integration, and testing.</p> <p>Tasks should be planned so that different disciplines produce results in time for other disciplines; e.g. a series of HW prototypes is scheduled and developed, and the HW development schedule is leading because of the realization throughput time (critical path). The SW development iterations are tuned as much as possible to serve the HW releases.</p> <p>The plan should also include release dates for prototypes and tasks relating to building those prototypes (such as integration of HW and SW components via cooperation). The increment plans can be implemented in the project management (PM) tool.</p>

Table 5. System definition activities, continued.

<i>1.1 Requirements gathering</i>	
Collecting raw requirements	<p>Candidate requirements for the system are collected from various sources (e.g. customer, standards, constraints, etc.) as shown in figure 5. The design constraints of the product are addressed (e.g. manufacturing costs) as requirements.</p> <p>The candidate requirements are prioritized and selected for implementation. Preliminary analysis is performed on requirements that will be implemented. More detailed analysis and description of requirements continues in the requirements definition phase.</p> <p>Requirement management (RM) software (e.g. Telelogic DOORS) can be used to store raw requirements. Especially important is that the origin of the requirements is preserved.</p>
<i>1.2 Requirements definition</i>	
System requirements definition, including non-functional requirements	<p>System requirements are derived from raw requirements. The requirements are analyzed and detailed until enough understanding of the requirements is gained and future work can be based on them. The requirements should be defined so that they can be understood unambiguously, no matter the background of the reader. Typically natural language requirements with use cases are the result of this work.</p> <p>RM software can be used to help in defining system requirements from raw requirements. Traceability to original raw requirements should be maintained. System requirement baselines (accessible to all project partners) should be maintained in the change management (CM) tool (e.g. Subversion) or by the RM tool if supported.</p>

Table 6. System definition activities, continued.

<i>1.2 Requirements definition</i>	
System requirements feasibility evaluation	All requirements are not feasible for implementation, but that is not necessarily known beforehand. Simulation (e.g. Simulink models in MATLAB) or prototyping can be used to evaluate the feasibility of implementation of system requirements.
System tests planning	<p>System tests are defined to validate functional and non-functional requirements of the system. Test planning can be done by using test management (TM) software. PM software can be used for responsibilities and timetable allocation of testing activities. Tool integration is necessary to maintain traceability between test cases and requirements, but also between test cases and tasks.</p> <p>Tool integration between RM, TM, and PM tools makes it possible to verify that requirements have been covered by tests, and that the testing activities proceed on schedule.</p>
<i>1.3 System architecture definition</i>	
Definition	System architecture can be defined by using suitable modelling language such as unified modeling language (UML) or specification and description language (SDL). Extensive software support exists for both of these languages, e.g. MS Visio for UML.
Evaluation of system architecture performance	<p>The purpose of the system architecture evaluation is to reveal how successful the allocation of requirements to sub-systems has been in the architecture definition. Measurements are performed against non-functional requirements.</p> <p>System architecture performance can be estimated via simulation tools such as Simulink/MATLAB or by purpose-specific tools such as PerSim, which is built for estimating system performance when upgrading from single processor architecture to multi-processor architecture.</p>

Table 7. System definition activities, continued.

<i>1.3 System architecture definition</i>	
Allocation of requirements to sub-systems	<p>The system requirements are allocated to the sub-systems defined in the architecture (i.e., to those sub-systems that are responsible for implementing the requirement. The requirements are then analyzed and detailed from the sub-system viewpoint, allocated to the sub-sub-systems, etc. until an adequate level is reached where design can begin.)</p> <p>Traceability between requirements in and between different levels (system, sub-system, sub-sub-system, HW-SW) needs to be maintained in the RM tool. Baselines of requirements allocation, system architecture and system requirements versions need to be maintained in the CM tool and RM tool. Tool integration between CM and RM tools is used to maintain relations between different versions of system architecture and requirements.</p>
<i>1.4 Interface design</i>	
Interface definition	<p>Modelling software (UML, SDL, etc.) can be used to help in defining interfaces. With the help of communication software (e.g. WIKIs) the amount of misunderstandings between different disciplines can be reduced. Tool integration between interface definitions stored in the CM tool and requirements from the RM tool can be used to maintain interface relations to corresponding architecture versions and requirement versions.</p>

5.1.2 Sub-System definition phase

In the sub-system definition phase (tables 8–9), the sub-system architectures are defined based on the system architecture, and the system requirements allocated to the subsystem are analyzed. Software requirements are defined as part of the sub-system requirements. Sub-systems are defined in several levels, until sufficient detail for starting design is reached.

Table 8. Sub-system definition activities.

2.Sub-system definition	
Communication between HW and SW teams	<p>Visibility in architecture and interface definition work for HW and SW teams is important, so misunderstandings can be reduced. Communication software can be used as a means to synchronize the work of different disciplines during interface definition. Tool integration can provide the needed visibility in its integrated environment, to where interface and architecture definitions are fetched from the CM tool.</p>
<i>2.1 Sub-system requirements analysis</i>	
Sub-system requirement definition	<p>Software and testability requirements are synthesized from the system level requirements into the sub-system perspective. RM software can be used for software- and testability requirements definition, and to maintain traceability between system requirements and sub-system requirements (e.g., HW).</p>
Sub-system requirements feasibility evaluation	<p>Simulation (e.g. Simulink models in MATLAB) or prototyping can be used to evaluate feasibility of sub-system requirements</p>

Table 9. Sub-system definition activities, continued.

<i>2.2 Sub-system design</i>	
Sub-system architecture definition	In sub-system design stage implementation is divided into hardware and software modules. Verification needs to be done such that all functionality is covered after division and that there is no overlap between HW and SW implementation. Responsibilities are assigned accordingly. Allocation/division approaches its final form by iteration, until design can begin. Modelling software (e.g. UML tools like MS Visio) can be used to define the sub-system architecture. CM software is used to maintain sub-system architecture versions. Modelling software (e.g. UML tools like MS Visio) can be used for partitioning and a PM (e.g. TRAC) tool for assigning responsibilities. Tool integration provides the needed visibility in architecture definition and related project tasks.
Evaluation of sub-system architecture performance	Sub-system architecture performance can be measured via simulation software such as Simulink/MATLAB or by prototyping.
Design of testability features	Consider if it is necessary to include testability features into the final product. If so, system performance may need to be up-scaled to support testability features.

5.1.3 Implementation phase

In the implementation phase (tables 10–12), the software and hardware development is done in increments and iterations, including design, coding & implementation, and unit testing for the increments. First an increment plan is made that is updated throughout the implementation phase. When a first set of iterations that can be integrated is ready the integration work starts. Simultaneously the next increment's implementation can start.

In this phase unit, regression and other testing practices are used to verify correct implementation. Errors, such as memory leaks, manufacturing faults, and software errors, can be identified by employing rigorous testing methods. Writing unit tests during implementation helps in identifying errors, but also in understanding the purpose of the code block. Special tools may be needed to set up a unit testing environment on a cross-compiler platform. Tools such as JTAG interface, logic analyzers, and memory dumps can be very helpful in solving problems during implementation.

Prototyping with evaluation boards is one way to test the different performance characteristics (CPU utilization, memory consumption, interrupt behaviour) of the platform. Another way is to evaluate the system performance against the performance requirements via simulators and emulators. Optimizations may be necessary if the performance constraints are not met. Optimization efforts should be focused to sections where optimization yields the most improvement.

Table 10. Implementation activities.

3. Implementation in increments	
Communication between HW and SW teams during phases of implementation	Communication between HW and SW teams is of crucial importance during implementation to avoid misunderstandings. Communication software can be used for this purpose. Tool integration between communication-, PM-, and CM-software can also be used so that teams can share a view into the project progress and produced items.
<i>3.1 SW design</i>	
Software design	Software can be designed for example by using UML, and in some cases it is possible to synthesize software directly from models. Another example is to use a register level description in digital circuit design and then generate VHDL code from the description, which in turn can be used to program FPGAs. Yet another possibility is to use Simulink models in MATLAB which can be turned into C-code with suitable extensions. Communication software can be used in this phase for coordinating cooperation between HW and SW teams. CM software can be used to store the design versions.
<i>3.2 SW implementation</i>	
Implementation pitfalls in HW/SW-software	Maturity of the HW platform used is one of the aspects that is worth considering in the SW implementation phase. Co-operation between HW and SW teams is necessary to resolve possible issues. Project management software can be used to monitor how the project is progressing. Additionally, communication software can be used to aid cooperation between HW and SW teams.

Table 11. Implementation activities, continued.

<i>3.2 SW implementation</i>	
Software debugging	Software needs to be debugged in problem situations. Various types of tools exist for supporting debugging activities. One possibility is to use a JTAG interface and integrated development environment (IDE) for software debugging by e.g. executing code line by line while observing the CPU register values.
Code optimization	Static code analyzers can be used to locate problematic components in software, and adjustments can then be performed accordingly. Compilers also offer various compile-time optimization possibilities.
Prototyping	Prototyping gives valuable feedback on how the software operates on a real platform, but also feedback to the HW team if changes are necessary. The prototyping schedule and tasks should be assigned as early as possible. To test HW prototype functionality it may be necessary to add testability features for the implementation that are removed from the final product. Prototyping is done in cooperation between the HW and SW teams, and feedback is given both ways. CM and communication software can be used for coordination of prototyping activities. CM stores software and hardware baselines which are integrated in prototyping.

Table 12. Implementation activities, continued.

<i>3.2 SW implementation</i>	
Builds	<p>A build can be made one or more times per day, for different purposes. Depending on the need, the build can consist of specific software modules or of a complete software package. Builds are needed for integration, testing, and releasing.</p> <p>New features should be added in such way that when bugs are introduced in the build, it is later possible to revert back to the working baseline. Builds can be baselined in CM software, where notes are kept of changes to the previous build.</p>
<i>3.3 SW unit testing</i>	
Unit testing	Several unit testing frameworks exist for widely used programming languages. For example, JUnit can be used when developing in Java language. Similar frameworks exist for other programming languages as well.
Testing of non-functional requirements.	<p>Non-functional requirements can be verified via e.g. internal instrumentation of the software, which collects measurements from the target and manages them, or testing can be done externally to the whole device.</p> <p>Test cases can be managed in the TM tool (e.g. Testlink), and tool integration between test instrumentation and the TM tool can be used to trace test data to test cases.</p>

5.1.4 Integration & release phase

In the integration and release phase (tables 13–16) the increments from the implementation phase are put together and tested. Part of the phase is producing prototypes of the product and finally a final release.

Table 13. Integration & release activities.

4. Integration & release	
<i>4.1 Integration and integration testing</i>	
Time allocation	<p>Most of the problems occur during integration of HW and SW components. Therefore it is important to reserve enough time for integration and testing activities. Incremental integration is much easier than “big-bang” integration.</p> <p>Testing activities can be coordinated via PM- and TM software support. With the help of tool integration of PM and TM tools, testing tasks are traced to test cases.</p>
Integration	The SW and HW components produced from a finished increment are integrated together. Continuous integration is easier than big bang integration because problems present themselves during integration which is performed after each increment.
Requirements coverage verification	Tool integration between RM and TM tools can be used to verify that requirements have been covered by tests and the tests have been completed successfully.

Table 14. Integration & release activities, continued.

<i>4.1 Integration and integration testing</i>	
Testing environment	<p>Test data produced by the test environment should be made available to all parties, so that defects become easier to track. It is also important to ensure that the test environment and tests can be replicated at a later date if needed. In a HW/SW project there are so many changing variables that if proper care is not taken, replicating the tests can be difficult or downright impossible.</p> <p>TM software can be used to store test parameters and test results. The TM program should be able to manage all necessary information relating to the testing environment, so that the environment (or version of it) can be duplicated at a later date if needed. It may be necessary to use CM software to store additional information on the test environment. Tool integration between TM and CM tools can be used to maintain traceability between test cases and related test environment configuration information.</p>
Integration tests	<p>Integration tests consist of planning of test cases, execution of tests and result inspection. The integration tests are planned according to an increment plan: the main focus areas of the increment are the first priority for testing. Focus is on interfaces- and system level (when possible) testing. Different test strategies that may be employed are black box testing (most common), positive testing, and negative testing.</p> <p>Some testing tools provide test coverage metrics for deciding when the system has been tested enough. The TM tool can be used for test case handling. If test cases reveal problems that need to be corrected, communication and PM tools can be used to inform the relevant parties. Tool integration can be used to gather all relevant information from TM, CM, PM and communication tools into an integrated environment.</p>

Table 15. Integration & release activities, continued.

<i>4.1 Integration and integration testing</i>	
Prototyping and simulation	<p>Prototyping and simulation is a way of getting results on system development progress as early as possible; the earlier the problems are located and necessary adjustments can be made, the better. Simulation software (e.g. MATLAB, PerSim/PerVis) or prototypes can be used for this purpose.</p>
<i>4.2 Releasing</i>	
Releases	<p>Planning of releases is done as part of the increment plan. A release is put together from the working baseline. Releases can be of a part of the system or a complete system. Releases are also used for different purposes such as internal use and customer version (alpha-, beta-, and final-release).</p> <p>Releases are baselined in the CM system including notes. Release notes should mention differences to the previous build (i.e. what has been added, known problems, and what is lacking), so that it is possible to revert back to the previous working baseline in case of problems.</p>
Release testing	<p>Tests that were defined during system test planning are tested here. Release validation criteria are derived from the increment plan: functionality, amount of defects under a certain limit, etc. After this release is tested and the results are published, measures are taken to make corrections in the next release. Tool integration with CM, PM and TM tools can be used to control release testing and the following activities.</p>

Table 16. Integration & release activities, continued.

<i>4.2 Releasing</i>	
Controlled correction of faults in baselines	<p>When a defect is uncovered via testing, the defect is analyzed and its correction prioritized according to its impact on system performance/functionality. Ultimately a change control board makes the decision if the defect is corrected or not. If necessary, a responsible person is assigned to correcting the fault. Otherwise the ticket for the defect is closed. The ticket is also closed when the defect is fixed.</p> <p>Assigning fault to the correct person may be challenging, because it is not always known beforehand where the defect is located. In this case the responsible person for fixing the defect may change one or multiple times.</p> <p>Tool integration to bug tracking software (e.g. Bugzilla), CM, and PM tools can be used to manage bugs and assign tasks to relevant personnel. The CM system stores the defect information for different baseline versions. This is useful in a situation where the defect has been fixed, but the correction has accidentally not been included in the new release. Thus, the fix can be located with the help of integration and be included in the new release.</p>

5.1.5 Project management

Project management issues have to be addressed in every phase of the process. In HW/SW development people come from different backgrounds: electrical engineering, mechanical engineering, software engineering, and so on. These factions may have difficulties in identifying and understanding problems faced by the other factions, mainly because their backgrounds differ. Thus, communicating information which has a possible impact on other groups becomes challenging: it is not always known whether the information is already known by others or that the information should be relayed to others. By employing more transparency in communication this challenge can be relieved slightly. Transparency can be provided via communication tools or by coordination of group meetings between different disciplines.

5.1.6 Change management

CM practices are used in all phases of the process. CM support is needed for baselines and for storing artefacts produced in different development phases of the process. Without CM, the project can spiral out of control as the amount of different artefacts and artefact versions grows: for example trying to determine which software version is related to which hardware prototype can become difficult. Different approaches can be used for CM: manual documentation, tools which maintain their own data (i.e. a TM program that stores test cases and related information), and a dedicated CM tool for storing information produced by other tools.

5.2 Collected requirements

This chapter presents the collected requirements of tool integration (tables 17–19) which were extracted from the challenges and activities presented in tables 4–16. The requirements fall into two categories, requirements related to supported tools and requirements related to tool integration.

5. Requirements for tool support

Because of the high number of tools that could be identified in tables 4–16 (e.g. PM, RM, CM, TM, simulation, and modelling), it is not clearly purposeful to integrate everything. Which tools, therefore, should be integrated? To answer the question several workshops were arranged with industry partners during the first phase of the tool integration development. The partners were asked what types of tools would benefit most from integration. Results of the inquiry revealed multiple core tool types, which can be seen in the requirements (table 17).

Furthermore, during the second phase of tool integration development, several interesting tools were developed for industrial needs by the project partners. These tools aid in the HW rel. SW testing and support instrumentation of test data from embedded devices running Linux OS, test data storage and management, analysis of test data, and performance simulation. These tools also address some specific identified HW rel. SW development challenges (chapter 4.1): e.g. difficulties in debugging embedded devices due to lack of data or challenges resulting from a lack of debugging interfaces.

Table 17. Collected requirements of tool integration.

Requirement name	Requirement specification
Tool support	PM, CM, TM, and RM tool support is needed in integration, because these are the most commonly used tools according to the industry inquiry. They are also tools that have data with many relations to each other. [29] Tools to support test data instrumentation from embedded devices with Linux OS, test data storage and management, analysis of test data, and performance simulation are needed in the integration.
Traceability between different development artefacts	Traceability relations are maintained by the tools themselves but it is also necessary to maintain relations between artefacts from different tools by means of tool integration.
Data flow from tool to tool	It should be possible to transfer data from tool n to tool $x/y/z$. This means that there can be arbitrary amount of $(n * m)$ bi-directional connections between the tools.
Data visibility	Data from tools is visible in the integration environment.
Reporting facilities	Possibility of generating reports from data, e.g. requirements test coverage, change impact analysis, etc. This is an additional requirement, which will be implemented if the schedule allows.

Table 18. Collected requirements of tool integration, continued.

Requirement name	Requirement specification
Framework-based tool integration solution	A framework which provides a point of integration for the tools. The framework's purpose is to provide a common look and feel to the integrated tools (i.e. to act as a kind of dashboard). The framework should also provide resources for guiding how the tools (and data from tools) can be integrated into the framework and means for defining practical interactions between the tools (e.g. traceability between requirements and test cases).
Security	The tool integration should provide at least equally good security as the disconnected tools provide without integration (i.e. the security is not weakened by the integration).
User rights management	User rights management (URM) provides varying levels of access to the integration environment. URM keeps track of authentication information for tools, so that the user does not have to manually login to the tools. Because users of tool integration act in different roles and each role has specific needs and requirements, URM should reflect this by allowing for configuration of different roles (e.g. different levels of information provided to the user depending on whether he is stakeholder in company A or company B). URM is an additional requirement.
Launching of external tools	It should be possible to launch tools from the integrated environment.

Table 19. Collected requirements of tool integration, continued.

Requirement name	Requirement specification
Interchangeability of the integrated tools	<p>Tools of the same type can be changed to other tools. E.g. multiple RM tools are supported: DOORS, RequisitePro, etc. This requirement can be implemented on two alternative levels. Level a. can be considered better in terms of integration.</p> <p>Tools of the same type can be used interchangeably, i.e. partner A uses tool xyz for requirements management and partner B is using tool zyx, while both partners can share the same requirements.</p> <p>Multiple different tools of the same type are supported, but each partner has to use the same tool for the same purpose.</p>
Extensibility of tool integration	New tools can be integrated into the tool integration by providing a suitable interface. It should be possible for the users to create integration for their own tool(s).
Workflow guidance	Workflow guidance should be available in the integration environment. It should be possible to define and execute & follow workflows with the given set of tools.

6. Tool integration design

This chapter presents the design of an integrated tool support solution for hardware-related software development called ToolChain. The design of TC is performed against the requirements collected in the previous chapter. The design is split into two sections, the first part of which documents the design and implementation of TC that began early in 2006 and continued until autumn 2007 [13, 29]. The first section presents more general aspects of the integration (i.e. foundation), while the second section focuses on the design and development of topics related to hardware-related tool support. The development of the second section lasted from autumn 2007 until the beginning of 2009.

6.1 ToolChain framework

The primary requirements (tables 17–19) for the tool integration are to support data flow from tools, to maintain traceability between development artefacts, and to provide an integrated environment for tools. By arranging workshops with industry partners and by discussing the requirements of the integrated tool environment, the decision was made that Eclipse would serve well as a foundation for the tool integration. Eclipse was chosen because it is widely used in the software development industry, and it provides facilities out-of-the-box which are needed for integration [29].

Eclipse allows for freedom from vendor lock-in because Eclipse is open-source, and development can thus be done without limitations. As a related design choice, tools from various vendors were selected for integration to avoid vendor lock-in. Furthermore, because Eclipse is a framework-based solution, development of point-to-point integrations between the various tools can be avoided. Additionally, by using Eclipse, implementation time can be reduced, because Eclipse already implements many features that will be needed by the integration (i.e. the plug-in extension mechanism). In addition, tool integration theory and existing tool integration solutions as presented in chapter 3 were used as an aid during the design of TC.

6.1.1 Eclipse Architecture

The Eclipse Platform is built on a mechanism for discovering, integrating and running modules called plug-ins. A tool provider can write a tool as a separate plug-in that operates on files in the workspace and surfaces its tool-specific user interface in the workbench. When Platform is launched, the user is presented with an integrated development environment composed of a set of available plug-ins. The quality of the user experience depends much on how well the tool integrates to the Platform and how well various tools work with each other. [30]

Eclipse provides a core of services for controlling a set of tools working together to support programming tasks. Tool builders contribute to the Eclipse platform by wrapping their tools in pluggable components, called Eclipse plug-ins, which conform to Eclipse's plug-in contract. The basic mechanism of extensibility in Eclipse is that new plug-ins can add new processing elements to existing plug-ins. Furthermore Eclipse provides a set of core plug-ins to bootstrap this process. [31]

A plug-in is the smallest unit of the Eclipse Platform function that can be developed and delivered separately. A plug-in is a part of the Eclipse Platform but it can be thought of as a separate Java application that can be distributed separately and can be attached to Eclipse by using a specified interface. [30]

A plug-in in Eclipse is a component that provides a certain type of service within the context of the Eclipse workbench. A component means an object that may be configured into a system at system deployment time. The Eclipse runtime provides an infrastructure to support the activation and operation of a set of plug-ins working together to provide a seamless environment for development activities. Within a running Eclipse instance, a plug-in is embodied in an instance of some plug-in runtime class, or plug-in class, for short. [31]

A small kernel called the Platform Runtime handles the start-up and, actually, all of the Eclipse Platform's functionality is located in the plug-ins. Eclipse Platform Runtime handles start up when the plug-ins installed are discovered, extensions and extension points are matched up, and a global plug-in registry is built. Each plug-in has its own Java class loader and they are only activated if needed. This procedure helps avoiding long start-up times. [30]

Every plug-in has a manifest file that declares the interconnections to other plug-ins. The interconnection model declares the extension points and extensions to the other plug-ins. An extension point is a named entity for collecting contributions. As can be seen in figure 6, the plug-ins can also be integrated among themselves. Normally, a small tool is written as a single plug-in, but a complex tool may contain several plug-ins which declare its functionality. [31]

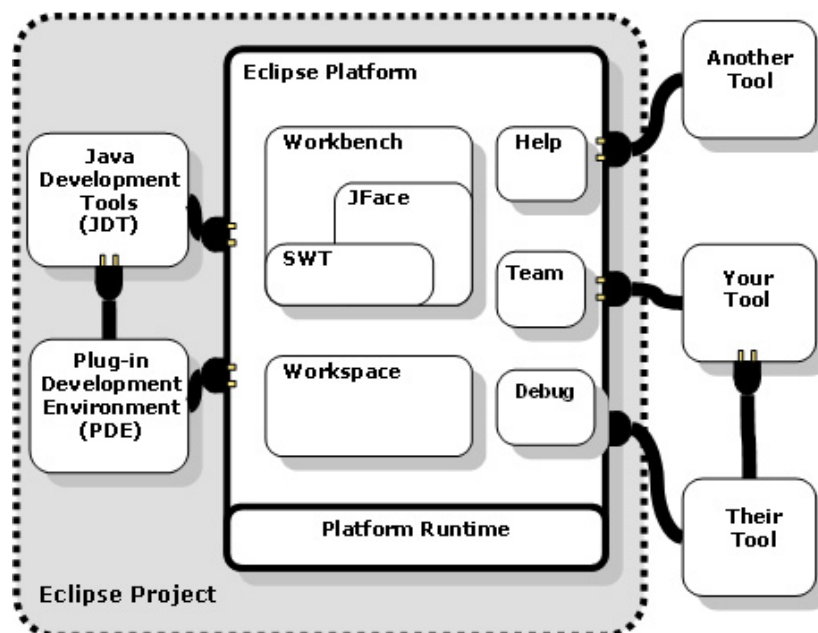


Figure 6. The Eclipse Architecture.

The development of the plug-ins in Eclipse is well guided. Eclipse includes a Plug-in Development Environment (PDE) that contains a wizard to start plug-in projects with the basic functionalities and interfaces to the Platform. PDE also includes a wizard for creating installable JAR files, thereby making every plug-in quite easy to distribute and install on the other parties' desktops. [31]

6.1.2 ToolChain architecture

The main architectural decision behind TC is to use Eclipse and its facilities as the foundation and to have several Eclipse plug-ins as the means of connecting the different tools into the same environment. For this reason each tool that is integrated into the TC framework has its own plug-in. Figure 6 presents how the tools connect to Eclipse via plug-ins. Additionally, tools wishing to connect to the TC framework must conform to the TC interface definition. This approach guarantees that TC does not commit itself on the matter of how the particular plug-in gets the data it needs, i.e. the plug-in developer can create a plug-in for a tool as planned and subsequently integrate the plug-in into ToolChain by implementing the interface. [13] Connecting of the tools is described in the next chapter.

The TC infrastructure operates so that each user will need Eclipse and TC plug-ins installed. For each tool used in the integration it is necessary that data access between the plug-in and the tool is provided for by some means. Additionally, the users will need to have a connection to a centralized TC traceability database. As mentioned, one of the requirements was that it would be possible to use different tools from the same type (e.g. RM). This requirement was however de-prioritized due to the tight schedule so that each TC user will need to use the same tools for the same purpose (e.g. the same RM tool). However, it is not always necessary for each user to have the complete set of tools (i.e. it is in some cases possible to have only a subset of tools in use, thus saving in overall licensing costs). Figure 7 shows a multi-user, multi-tool TC environment. In the figure there are two users with slightly different tool sets, and dashed lines of the tools represent the fact that instances of the same tool share information.

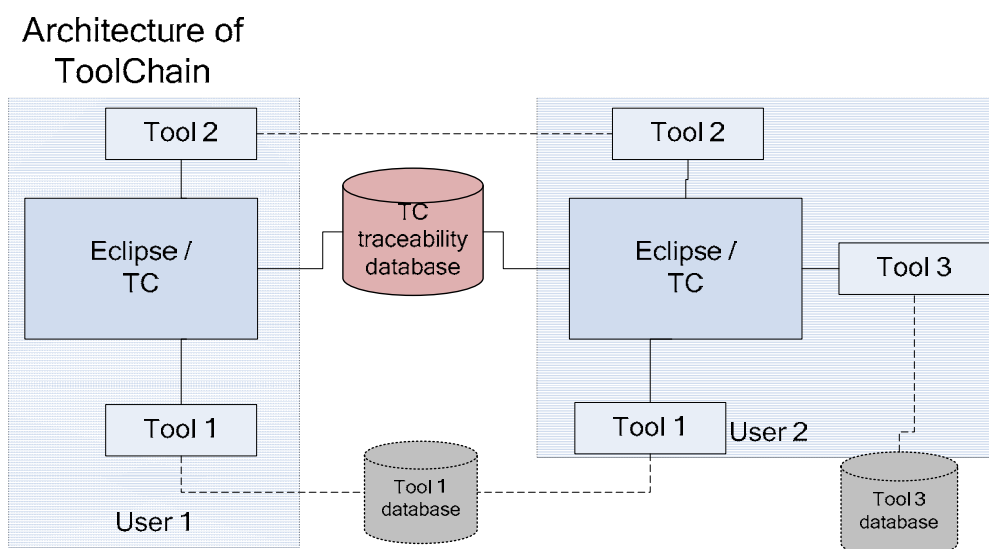


Figure 7. Example infrastructure of TC in a multi-user / multi-site environment.

After tool(s) have been connected to the TC framework (figure 8), data from the various tools becomes available for use. The data flows from connected tools are directed into the Traceability-view plug-in, which provides the benefits of tool integration by allowing inspection of data from tools and observation of overall status in a centralized way. The links between different tools are stored in a traceability database. TC does not store any information originating from the tools in its database other than what is needed for traceability. This means that if the user has access to the tools via tool-specific plug-ins / TC integrations, it is possible to see the complete picture regarding project status in “real-time” via TC. Due to this design choice, difficult data replication and synchronization tasks can be avoided, which means that it will also be much easier to implement. However, providing access to tools in a multi-site environment has some challenges: the information systems infrastructure needs to be designed so that each instance of TC is able to connect to each tool repository that the TC instance needs. This can be a very complex setting when multiple tools are used, and if the tool repositories are not managed in a centralized location.

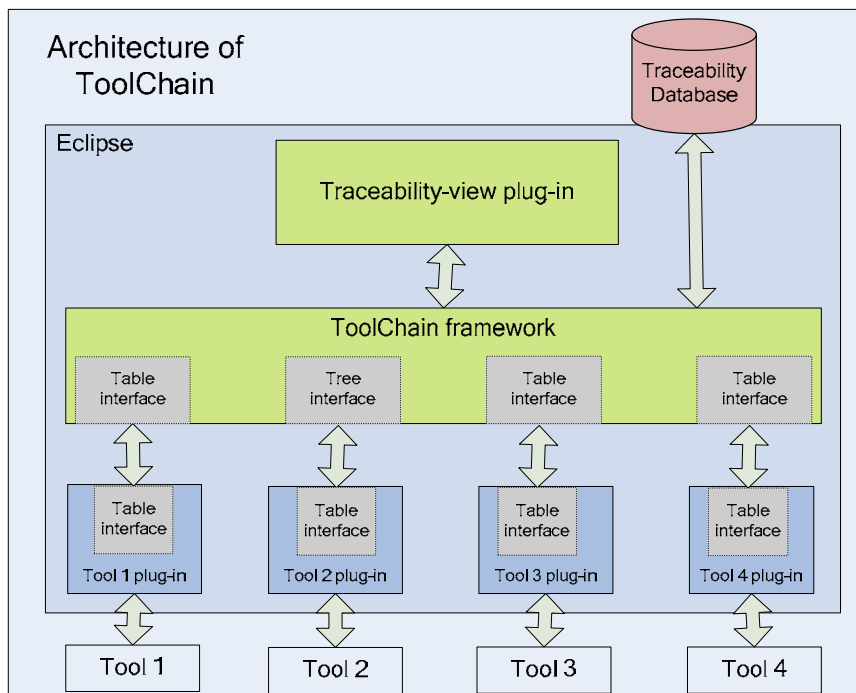


Figure 8. TC architecture with connections to tools elaborated.

6.1.3 Connecting tools

As previously mentioned, TC does not commit on how the tools integrating into TC handle their data. For data sharing between multiple users, it is necessary that the tools employ some means of data delivery and synchronization between the various instances of the tools. Usually the tools store information in a centralized repository or by some other means synchronize the information between instances of the tool. If the tool cannot handle synchronization of instances, TC does not try to resolve this issue from the technical point of view but does it rather by dictating the use of a certain instance: delivery and synchronization of the used instance is up to the tool users.

Tool specific plug-ins (or tool integrations) connect to the tool in question by any means that are supported (e.g. JDBC and API). It is up to the plug-in developer to create this interconnection between the tool and the plug-in. The developer has also the freedom to specify the level of integration needed, what kind of data is handled, etc. The plug-in does not necessarily need to be built from scratch, because some tools already have Eclipse plug-ins. These plug-ins can in some cases be used to connect the tool to the TC framework. For a successful connection, it is necessary that the data from the plug-in can be extracted and connected to the TC interface. In practice, this most often means that the source code is needed for the plug-in.

From the TC point of view, the purpose of the tool specific plug-ins is to act as a gateway between the TC framework and the tools. For a plug-in to connect to TC it must first conform to one of the two TC specified interfaces: table- or tree-format. These formats dictate how the information from a tool is represented (or structured) in the TC user interface (UI). It is up to the plug-in provider to transform the data from the tool into a TC accepted format.

In addition to conforming to a TC interface format, plug-ins must extend the ToolChainMain class, whose purpose is to activate selected tools when TC is launched. By extending the ToolChainMain class plug-ins register (or advertise) themselves as part of the TC. This guarantees that data flow and interaction between the disconnected plug-ins becomes possible. In addition to conforming to the TC interface and extending the TC base class, it is not necessary for a plug-in provider to implement any other functionality in the plug-in in order to connect to TC.

6.1.4 Data visibility

The information from the tools is gathered directly from tool-specific plug-ins after the request is made. Usually this means that the information is “real-time” because it is not cached or stored in the TC database, but rather requested directly from the tools. However, as already mentioned, it is up to the tool plug-ins to dictate how the information is handled.

After the information from the tools is available in the TC framework, some means must be employed to facilitate data visibility. One way is to provide a dashboard-like interface in the tools for users. The dashboard plug-in in TC gathers information from other plug-ins connected to the TC framework and provides project information in a clear and concise manner. The dashboard plug-in also employs traceability as a means of handling the relations between the development artefacts. The plug-in handles this by allowing the user to drag & drop development artefacts against each other and thus creating a relation between the two. Relations are stored in a dedicated traceability database.

The dashboard-view allows for inspecting traceability of development artefacts from one perspective, e.g. what artefacts are related to artefacts from the requirement tool. In this case the perspective would be that of the requirement tool. For legacy reasons (decisions made during the first phase of development) the perspective is currently fixed to only one, that of the requirement management tool. However, this is not a limitation by design, but rather by implementation. In an optimal situation the dashboard-view should allow the user to inspect traceability from whichever tool’s perspective. Inspection of artefacts operates in a way that when the user clicks on a development artefact, its detailed information is provided in one section, and other sections provide a list of traced artefacts and some brief information about them.

The UI design approach for TC was to make the tool-specific (and other) plug-ins have a common look and feel because the user interfaces offered by different tools vary greatly in look and feel. This can be confusing to the user and will raise the learning curve considerably. The added benefit of using an Eclipse environment for tool integration is that when the plug-ins are implemented by using the Java SWT-class libraries the look and feel in all integrations is similar and there is only one UI to learn.

6.1.5 Traceability

Traceability in TC is used to present how the development artefacts from the different tools map to each other. The database structure implementing the essentials for traceability is quite simple, consisting of only two tables: 'Items' and 'Traceabilitylinks'. The structure differs from the previous version of TC, presented in [13], in a way that it allows for an unlimited number of different tools and does not set any restrictions on linkages between artefacts from different tool types.

The 'Items' table is used to store generic information on the traced artefact in the database: id, name, path, and type. The id field is an identifier for the item in the tool. The name field is the name of the artefact. Path (if any) is the relative location of the artefact in the tool. The type field is used to indicate the type of tool from which the artefact originates. The number of needed fields in this table is rather limited because more detailed artefact information is collected directly from the tool integration plug-in as needed, thus removing the difficulty of replication and synchronization of data that would result from storing the information in the traceability database. The fields are only used to create a unique identifier for the artefact so that each artefact from each tool can be uniquely identified. Figure 9 shows an example of a traceability scenario where unique identifiers are used to connect artefacts from the different tools.

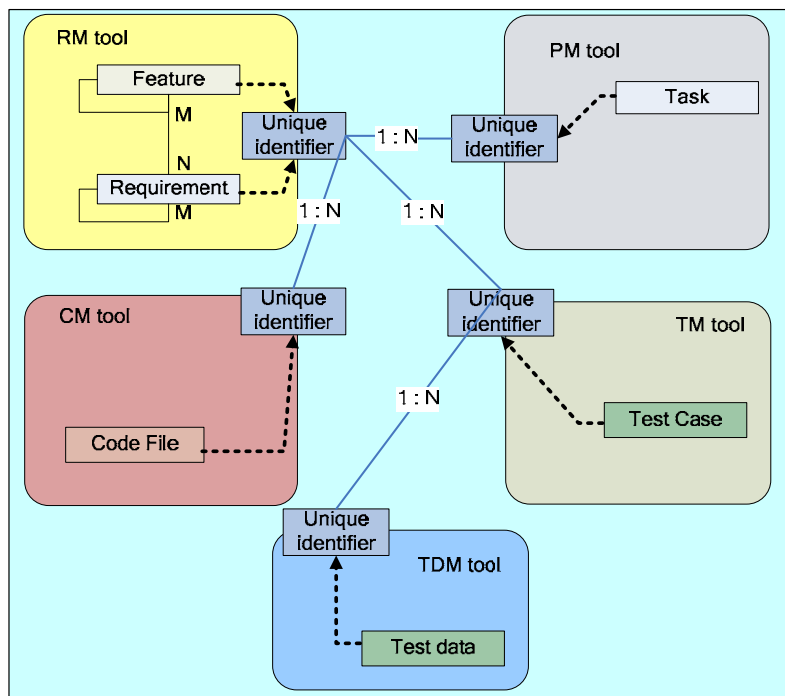


Figure 9. Traceability of artefacts between tools in TC.

The 'Traceabilitylinks' table consists of two foreign keys: 'suid' (source id) and 'tuid' (target id). These foreign keys are used as reference rows (i.e. artefacts from tools) from the 'Items' table and thus creating a linkage between the two. These foreign keys form together a primary key for the 'Items' table. Two-way linking of artefacts is possible with this database structure, e.g. to see test cases that are related to a requirement, or requirements that are related to a test case. Figure 10 illustrates the TC database model.

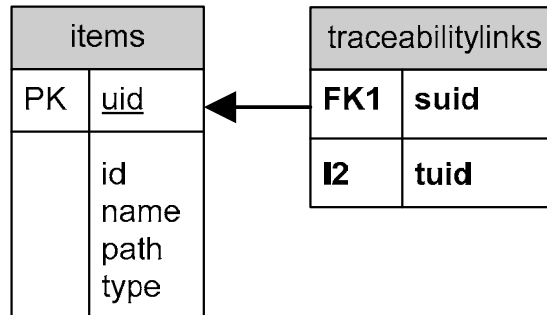


Figure 10. The ToolChain database structure.

Two basic operations are needed for handling traceability links: adding of links and removal of links. When creating a link, due to foreign key constraints in the 'Traceabilitylinks' table, it is first necessary to add information of both tool artefacts into the 'items' table. Then the link can be created. When removing a link, because it is possible that a row in the 'items' table is referenced by one or more row(s) in the 'Traceabilitylinks' table, a row from the 'Items' table is allowed to be removed only after no row in the 'traceabilitylinks' table is referencing it.

Traceability is implemented in the dashboard-like traceability view of TC. The view allows the creating of links between artefacts via a drag & drop mechanism and removal of links by clicking on the linked item and selecting 'remove'. The view also provides means of inspecting information on linked artefacts.

6.1.6 Security and user rights management

TC does not employ its own security or user rights management schema but instead uses those of the tools with which it integrates. This is due to the heterogeneous nature of the tool integration where the tools originate from different vendors. What this means in practice is that when building tool integration (i.e. the Eclipse plug-in for the tool), whatever means are used to connect to the tool, the tool's own authentication and security measures, are used. This is usually possible only if the tool supports integration, for example via API. If direct database integration is used, security and user rights management are bypassed, in which case the same mechanism has to be built into the Eclipse plug-in (if needed). Otherwise integration may provide access to material that should be limited to certain user groups.

Because the amount of tools in integration can be rather large, it could be beneficial to have some kind of authentication system for TC that stores the different configuration parameters and user accounts for a set of tools per user or per project basis. However, this exercise is outside of the scope of this work.

6.2 Hardware-related software development support

The second phase of TC development tries to improve hardware-related software development support via an enhanced tool set and by tight integration of the tool set into the existing integration environment, which was described in the previous section. Furthermore, an experimental workflow system is designed into the integration as an aid for the development work.

During the second phase of TC development, industry and research objectives were primarily focused upon improving the instrumentation of embedded systems, management of the instrumented information, and using the instrumented data for various purposes like analysis of the SUT behaviour and performance. The tool set consists of the following components: Probe Framework (PF), a performance visualization and simulation tool (PerVis & PerSim), multivariate analysis (MVA) tool, and a test management tool. The test management tool was integrated in the first phase of development but is mentioned here because it is an important part of the tool set. By using this tool set and the TC framework, integration will be built which allows for an improved testing process, compared to disconnected tools. The following section provides an overview of the whole system, some details about the components of the system, and finally a more detailed explanation of the integration from the TC point of view.

6.2.1 Overview

As previously mentioned, the system consists of the following components: Probe Framework, Eclipse ToolChain, test case management software, test analysis, and performance simulation. Figure 11 provides a rough overview of the operation of the system.

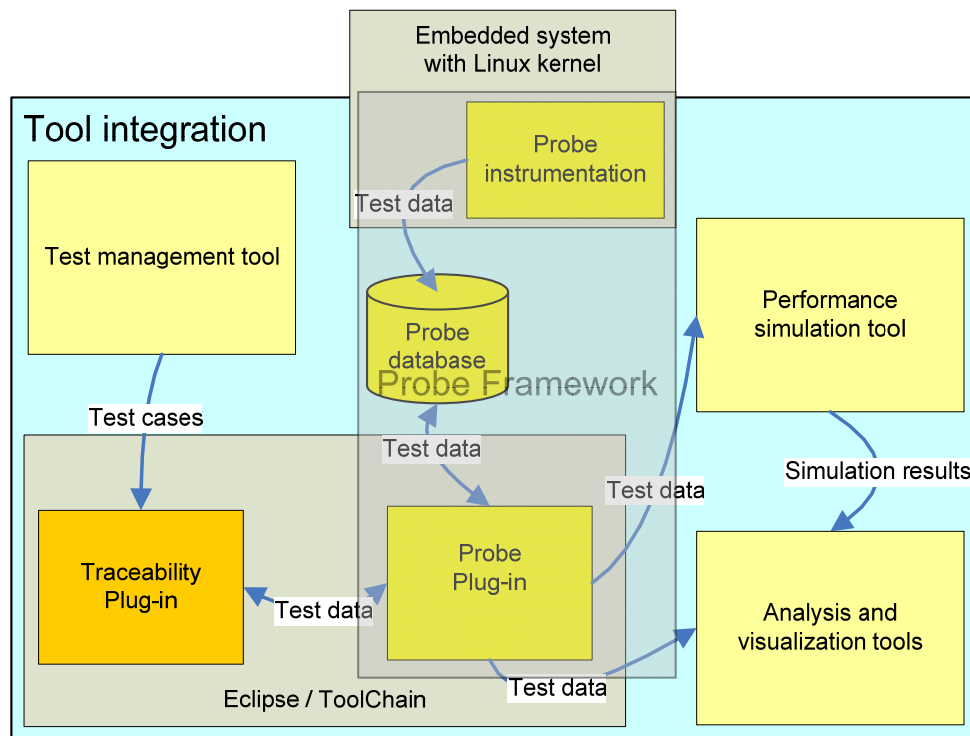


Figure 11. HW rel. SW development support in TC.

As can be seen in figure 11, the Probe Framework consists of three separate parts: *instrumentation*, *database*, and *interface for the database (i.e. Probe Plug-in)*. Probe Framework instrumentation is designed to gather data from systems running Linux OS. It has been developed by Markku Pollari, VTT. Instrumentation is possible in both kernel and user modes. Kernel instrumentation is done using SystemTap while various possibilities exist for user mode instrumentation, such as inserting custom probes into the application source code. Probe instrumentation takes care of handling the test data generated during execution of a test case on the embedded system, by sending the collected data to the *probe database*. [32]

The probe database provides the following means of importing the collected test data into the database: file based import and import over a network using TCP/IP-protocol. The imported data can be structured in binary- or CSV-format. The probe database provides also two means of export: file based or the TCP/IP based export that can be used with TC. Probe database export needs to be guided with an extensible markup language (XML) based file. The file guides the probe database to export the data that is needed and in the format specified by the file. TC uses the XML-format over TCP/IP to query the probe database for test data. Replies are sent from the probe database to the TC in the format specified by the query. Currently queries support only listing and fetching of test data; however in the future version it is possible to import test data to the probe database through the probe plug-in. The probe database has been developed by Juha Vitikka, VTT. [33]

Test data is stored in the probe database in a tree structure: the root node is the project (it is possible to have many projects, and thus many root nodes), under the project is a test case and under the test case are the test runs. Each test run generates a set of data. A set of data consists of columns (columns are attributes like CPU utilization) and rows which signify measurement values for the columns. [33]

TC probe database integration makes it possible to select a certain set of data with specific columns and rows and to direct the set for further processing. As can be seen in figure 11, further processing can mean test data analysis (PerVis or MVA tool) or performance simulation (PerSim). Furthermore, the probe database plug-in integrates with the traceability view, which makes it possible to assign traceability for the test data. Traceability can be assigned for example between a test case from the test management tool (integrated with TC) and between a probe database test case, which is a different entity from the test case from the test management tool. In practice this means that the test management tool stores the test case specification, description and results (i.e. passed / failed), while the probe database stores the measurements of the test case. By assigning traceability between the two, the testing process can be better managed.

TC integration between the analysis tools PerVis and MVA makes it possible to visualize the test results. Each of the tools is designed for a specific purpose: PerVis is meant for visualizing application thread behaviour, that is, how concurrent threads are executed in the temporal domain [34]. PerVis/PerSim has been developed by Marko Jaakola, VTT. The MVA tool is designed for more general purpose analysis of test cases and is the property of NSN. The MVA tool employs principal component analysis (PCA) as a means of visualizing the correlation between various columns (measurement attributes). By visualizing the correlation it is possible to deduce how and if the attributes correlate to each other. Visualization also provides a means to detect erroneous or irregular behaviour. [35] TC integration to these tools allows the selection of analysis parameters and execution of the analysis from the integrated environment with the data set selected from the probe plug-in.

TC integration with the simulation tool (PerSim) provides a means of analyzing the multi-core performance of a single-core system from the data set. The data set is collected from the single core system with the test application and is stored in the probe database. TC relays the data set and simulation parameters to PerSim, after which PerSim generates a load model from the data set, a profile of how the application run on the single-core system would behave in a multi-core environment. After the simulation is complete, the load model is stored locally in a temporary folder. The multi-core profile is then visualized using PerVis. [34] Because simulation is time consuming, a future version of the probe database integration makes it possible to store simulation results under the test data set. Thus, if the simulation results are needed more than once, time can be saved by extracting the load profile through the probe management plug-in, rather than redoing the simulation.

6.2.2 Workflow support

The primary purpose of the workflow support system in this thesis is to guide the user during the development process: how to use the tool set and TC in combination to complete various tasks. The requirements for the workflow support system require workflow implementation in an integrated environment and a way of defining workflows for the system. Eclipse provides a means to implement these requirements via its cheat sheet system. The cheat sheet system was originally designed to guide a new user in an unfamiliar development environment (i.e. Eclipse). For example, a cheat sheet could be used to guide the user through all the steps needed to create, compile, and run a simple Java program. Cheat sheets use a task-based system in which a certain activity is divided into tasks / sub-tasks. Each task takes the user through a series of steps that need to be done to complete the task. Some of the actions can be automatically performed by the cheat sheet, such as automatically launching a tool. [36]

The first version of the workflow support system that will be implemented via the cheat sheet system will be quite simple, but is adequate for prototyping and gathering experience on how the workflow support should work in the integrated environment. For a workflow system understanding on development workflow(s) needs to be gathered in some kind of format (e.g. a block diagram describing steps, tools, actions, and data flow). The gathered workflows will then be modelled into the cheat sheet system. This is a non-formal approach to specifying workflows but adequate in the scope of this exercise.

After the workflow has been modelled into the cheat sheet system, the system can offer guidance to the user to accomplish some specific activity (e.g. testing of the system). For example, the activity can be split into sub-sections like test case specification, test execution, and test analysis. These sub-sections are elaborated on such a level by the cheat sheet system that the user will immediately know what the meaning of the sub-section activity is, and what steps are needed to succeed in the activity by using the provided tool set.

In the first version of the workflow support system, workflows will be locally stored and delivered coupled with the TC plug-ins. However, in the future version it would be good if modifications to workflows could be made by users and the modified versions could be stored for further usage in a central workflow repository or similar system. This would also allow for inspection of the work progress.

7. Tool integration implementation

This chapter documents the development of ToolChain, which was done in two phases as mentioned earlier. The first version, the result of the first phase, provides a general framework for tool integration and some tool integrations. The purpose of the first version was to prove the feasibility of the approach, not to develop perfect tool integration. As a result, the complexity of the developed plug-ins vary; some of the plug-ins provide only simple functionality while the others are more complex. However, all of the plug-ins are presented for the purpose of completeness. The second phase of development focuses on the hardware-related software development support and its implementation in TC.

7.1 Implementations in the first phase

The development of ToolChain started from learning how to create and implement Eclipse plug-ins. The first tool integrated into the Eclipse environment was IBM Rational RequisitePro. The tools integrated in the first prototype were, for requirements management, IBM Rational RequisitePro, for project management Philips Project Assist Tool (PAT), for configuration management Telelogic Synergy/CM, and for test management Philips SoftFab. [29]

After the tool-specific plug-ins were finished, integration between the tools was created. The goal of the integration was to increase traceability during the project lifecycle. Items from the RM tool (RequisitePro) were selected to be “the integration point” to where the other tools’ traceability links should be linked. Initially creation of links was handled differently and separately for each tool: e.g. creating a link between project tasks (PAT) and requirements (RequisitePro) was performed in the plug-in made especially for linking these items, while on the other hand, a link between code files (Synergy/CM) and requirements was made in yet another plug-in. Thus the first prototype was not according the architecture described in figure 8. [13, 29]

The first prototype proved the concept of tool integration via Eclipse plug-ins. The next step in ToolChain development was to implement a more generic tool integration solution. The generalization work was done in close cooperation with the project consortium; the status of the development was presented in bi-monthly workshops and plans were made for future development directions. [13]

Development work continued by developing integrations to TC for alternative tools in PM, CM, TM, and RM tool types. This was done to study the differences in integration of the tools and thus to discover generalization possibilities for the integration. Another aim was to prove that the ToolChain concept is robust with respect to changing tools; interoperability of tools should be maintained when plugging out a tool and replacing it by another. The following

additional tools were integrated to the tool chain using the same point-to-point method as in the first prototype solution: Telelogic DOORS, OSRMT (Open Source Requirements Management Tool), Open Workbench, Subversion and Testlink. These tools were chosen based both on the industry partners' input and on the decision to use open source tools. [13]

As a result, experience from building the additional tool integrations allowed for specification of generic interfaces for PM, TM, CM and RM tools for ToolChain. Furthermore, this enabled the changeability of the tools and also easy integration of other similar tool(s) into the ToolChain via TC's tool integration interface. [13]

The final version of ToolChain from the first phase of development supports three PM tools (Open Workbench, PAT, and Trac), three RM tools (DOORS, OSRMT, and RequisitePro), two CM tools (Subversion, and Synergy/CM) and two TM tools (Testlink, and SoftFab). Adding other PM, RM, CM or TM tools to the ToolChain is easy via the tool integration interface. With ToolChain's tool selector feature any combination of these tools can be taken into use; however if the tools are later changed, the traceability database must be cleared and therefore all the traceability data will be lost. It is also possible to create a completely open source tool set from the available tool integrations of the first phase TC. [13]

The traceability view of TC (figure 12) gathers important information from the selected tools. Users can specify traceability between different development artefacts in this view by drag & dropping. After traceability is specified, the data is immediately available to the other TC users, who can inspect the existing relations and create new ones if necessary. [13]

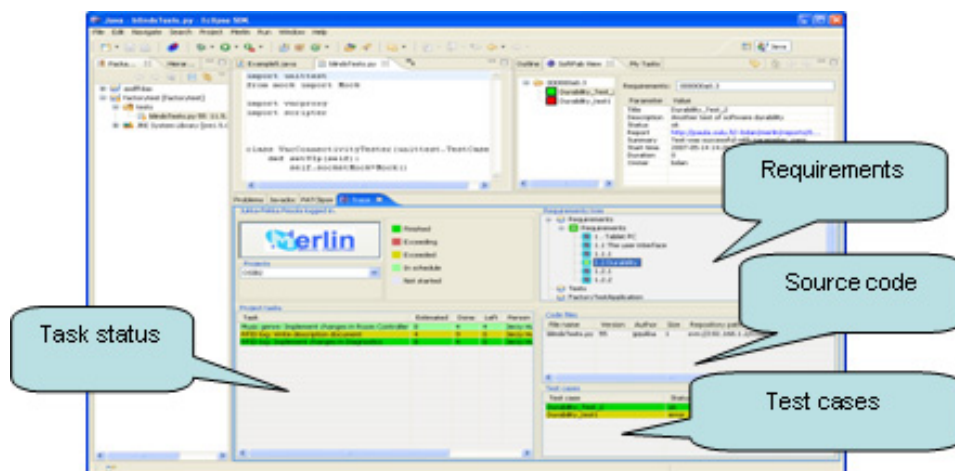


Figure 12. The ToolChain requirement traceability perspective.

The traceability links are created by using the drag & drop mechanism, e.g., tracing a code file to requirement is done by selecting a requirement and then dragging code files from the CM tool view and dropping them into the code files window in the Traceability View. The dragged items come from the tools' own plug-ins. The same mechanism works for all the existing and new tools. This mechanism makes it significantly easier to add new tools to TC; in the first prototype changing tools and defining relationships between their artefacts had to be done by modifying the source code and the traceability database structure. [13]

In the final version of the first phase, all the plug-ins handle their own user interface themselves but the traceability view of the TC synchronizes the views. Thus, all views are

always up to date for all TC users, regardless if they have the underlying tool installed on their computer, or wherever they are located. [13]

7.1.1 Tool integrations

This chapter discusses the tool integrations that are also part of the TC tool integration solution. The tool integrations presented here connect to the TC framework via their tool specific plug-ins. Tools of the following types were selected for integration in the first phase based on industry feedback and prioritization of tools: RM, PM, CM, and TM. [13, 29]

7.1.1.1 Philips Project Assist Tool

The Philips Project Assist Tool is a project management tool internally developed by Philips. PAT has a web browser based front-end, developed on Ruby on Rails running on a MySQL database. The integration called PatClipse was the second tool attached into the TC. The plug-in gets the essential data from the Project Assist Tool's MySQL database via a JDBC driver. [37]

The plug-in tries to follow the original user interface layout as closely as possible. The project structure is shown in tree format, from increments to tasks. The burndown graphs that show how the projects and increments are proceeding are calculated and drawn the same way as they are in PAT. The working hours can also be updated similarly to PAT, via a form. [37]

There are even some improved features in PatClipse that the Project Assist tool does not directly support. PatClipse provides a coloured view of the process, in which it can be seen if the increments are running late or the estimated time limits for the tasks have been exceeded (figure 13). Furthermore, once the user has logged in to PatClipse, the user can view the tasks assigned to him/her in the so-called My Tasks view. PatClipse also monitors if changes have been made in the task descriptions or estimated amount of work and notifies the user of the changes and points out the differences. [37]

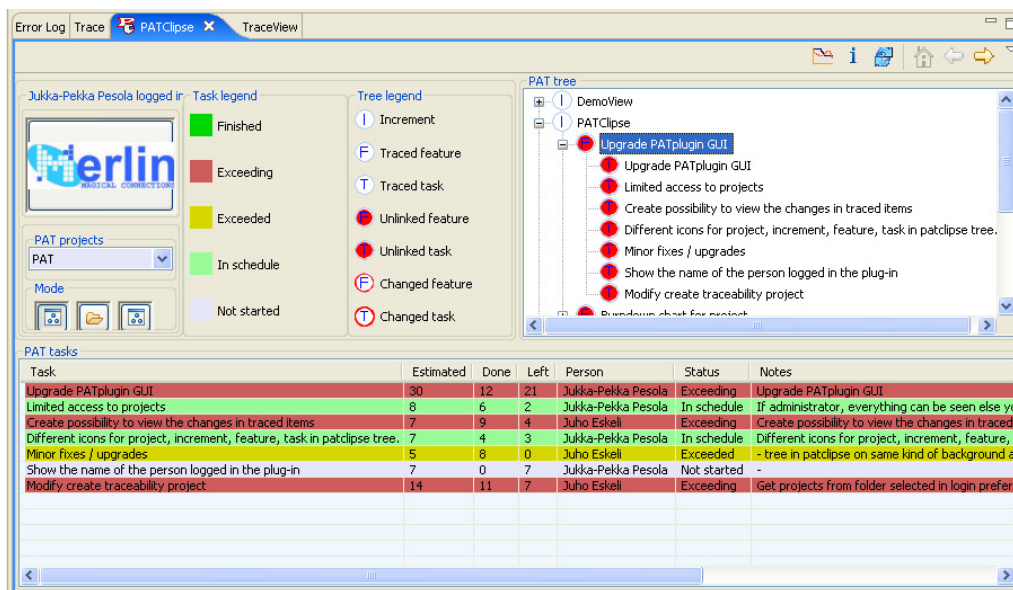


Figure 13. Project outlook in TC's PatClipse plug-in.

7.1.1.2 Open workbench

Open Workbench is an open source desktop application that can be used for project scheduling and management. Open Workbench is a free alternative to Microsoft Project. [38] Open workbench stores project information as XML files, and for this reason sharing project information between project members has to be done manually, or the XML file could be stored (as read only) on a network shared file system to make sharing easier. With the Open workbench plug-in, the projects can be opened in the Eclipse environment by browsing to the XML file's location in the file system. After the file is loaded, it is parsed via a Xerces XML-reader. The parsed data is then displayed in the plug-in. [37]

The open workbench project view (figure 14) shows the projects (one or more) and their structure in a tree. Brief information about the projects or individual tasks is shown in the table. Tasks' statuses are indicated by different colours in the table. The plug-in is quite simple in nature and was developed for proof of concept purposes only. [37]

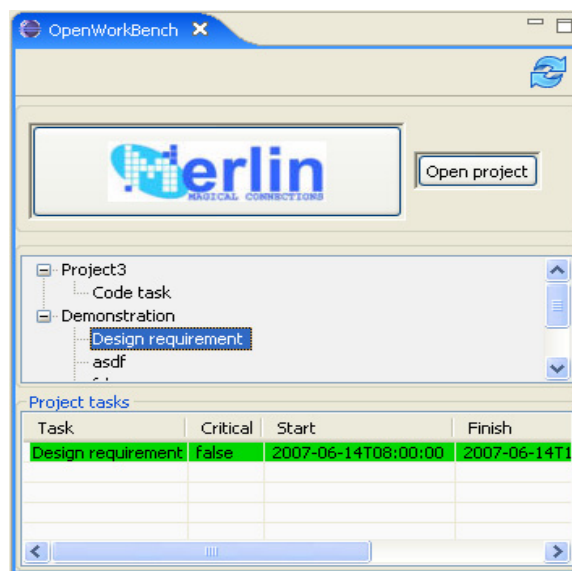


Figure 14. TC plug-in for open workbench, project view.

7.1.1.3 Trac integration

Trac is a web-based open source bug tracking and project management system for software development projects. Trac has been developed in Python programming language, and it uses structured query language (SQL) backend with support for multiple different database engines. Trac also provides API over hypertext transfer protocol (HTTP) for integrations. [39]

At the time of the integration's implementation, there was an Eclipse plug-in for Trac, but the plug-in provided access only to Trac wiki-pages, which was not enough. Because the plug-in was open source, it was used as the basis for a new plug-in which allowed for inspection of Trac tickets per project. Tickets can be thought of as tasks related to some activity. Most often tickets are related to bugs in Trac but can be used for other purposes as well.

7. Tool integration implementation

Trac also provides out of the box integration to Subversion which allows, for example, tracing of the ticket to (buggy) source code. However this integration is separate from the TC integration: it is a point to point integration developed between Trac and Subversion by Trac developers.

The TC integration connects to Trac via the HTTP-API as previously mentioned. Implementation of the connectivity was implemented in the previously created 3rd party plug-in and was used as the basis for the TC/Trac integration. The plug-in operates by asking the user address of the project (e.g. <http://127.0.0.1:1234/trac/mp3>), username, and password. The information is stored locally, so it is not necessary to update the information each time. After the necessary information for connection has been provided, the user connects to the Trac project by pressing the connect button, after which project tickets (i.e. tasks) are fetched and shown in the plug-in. Figure 15 shows the built plug-in in operation.

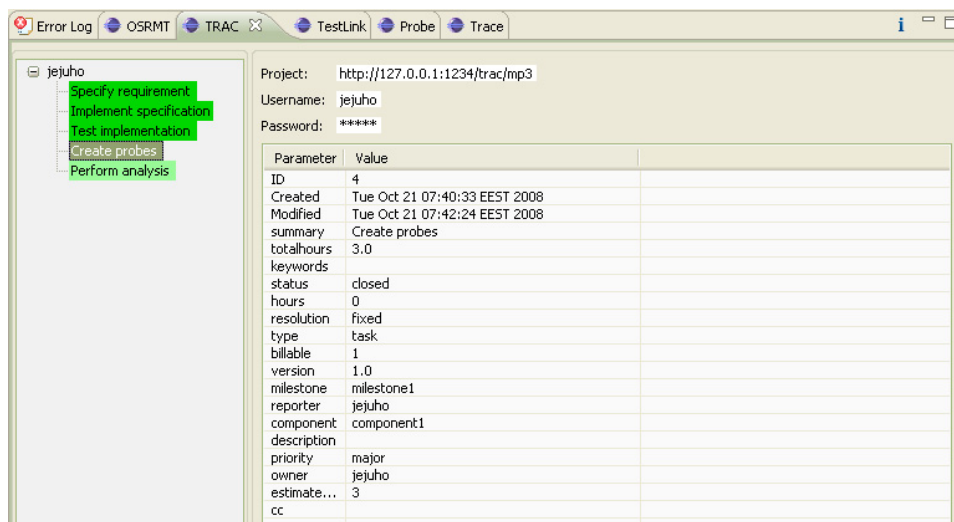


Figure 15. TC's Trac plug-in showing tickets and ticket information.

7.1.1.4 Telelogic DOORS

Telelogic DOORS is complex requirement management software originally developed by Telelogic, now acquired by IBM [40]. The DOORS integration (figure 16) was implemented for the OSIB industry case (chapter 8.1).

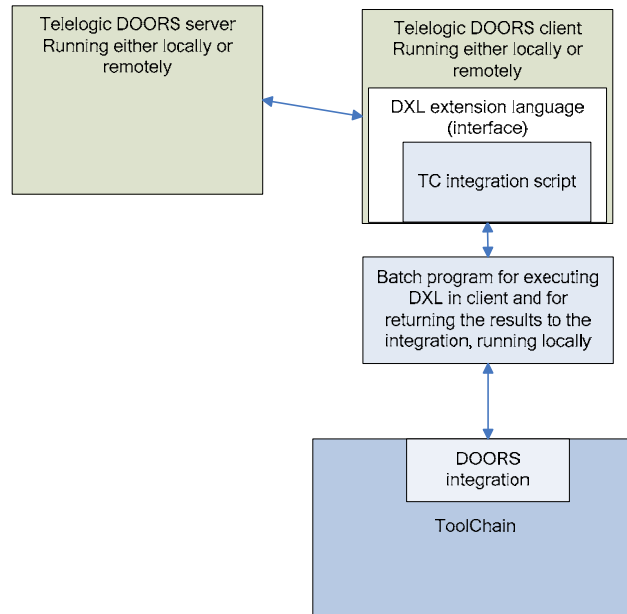


Figure 16. Doors integration of ToolChain.

The DOORS clients communicate with the DOORS server. DOORS provides a means of connecting directly to the server via the API-library, which is available in C-language; however this method was tried out briefly but discarded due to difficulties resulting from inadequate documentation. Another alternative to this is to use the DXL extension language provided by DOORS for integration. For this approach it is necessary to run the DXL scripts in the DOORS client. The purpose of the DXL scripts is to perform query and update operations needed by the integration in the DOORS environment. It is possible to connect to the DOORS client to Eclipse/TC via TCP/IP-protocol and execute DXL-based queries from there on. In short, commands and queries originate from the Eclipse/TC environment, which calls the batch program, developed in C, which then calls the DXL scripts running in the DOORS client. The DOORS client then fetches the information from the DOORS server. Ultimately, information is propagated all the way back to the Eclipse/TC DOORS plug-in. The method is overly complex and its limitation is that the client has to be running when the integration is used, but this was the best approach that could be developed in the set timeframe. The DOORS plug-in is one of the more complex tool integrations developed for TC. [37]

7.1.1.5 Open source requirements management tool (OSRMT)

OSRMT is an open source requirements management tool [41]. The OSRMT plug-in was developed to increase the amount of RM tools available with ToolChain. OSRMT does not natively provide other means of integration other than the standard database connectivity. However, because the source code is available for the tool, a different approach (from the other integrations) was trialled in this case. Originally the idea was to separate the basic operations, such as list the projects, list requirements, get requirement information, etc., from the source code and to use these operations to build the plug-in, but this turned out to be rather difficult because the OSRMT tool itself is quite complex and the code base is quite large. Therefore, the

7. Tool integration implementation

plug-in for OSRMT was developed by disconnecting the original used interface, made in Java-AWT, from the source code and then “placing” the separated functionality under the Eclipse user interface. As a result, the plug-in operates identically to the original tool, and only the graphical user interface (GUI) differs (figure 17). [37]

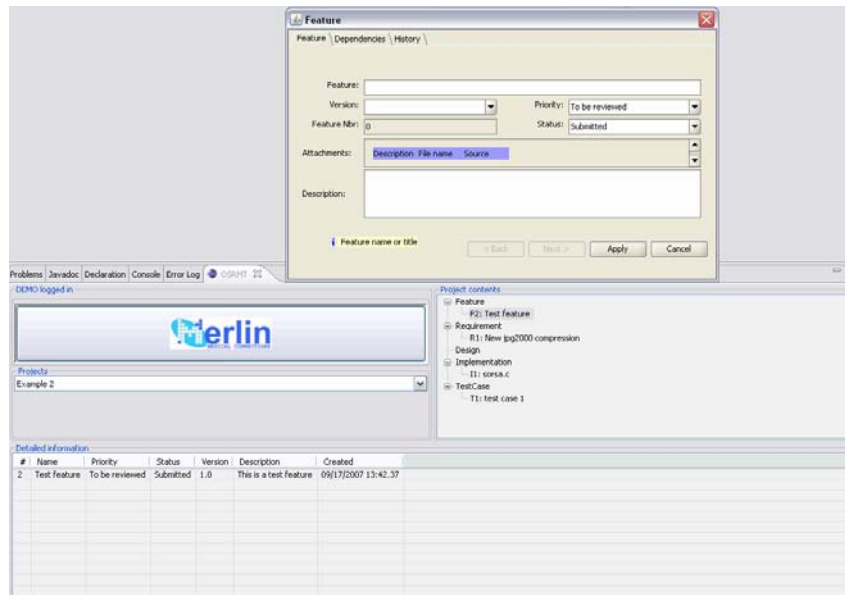


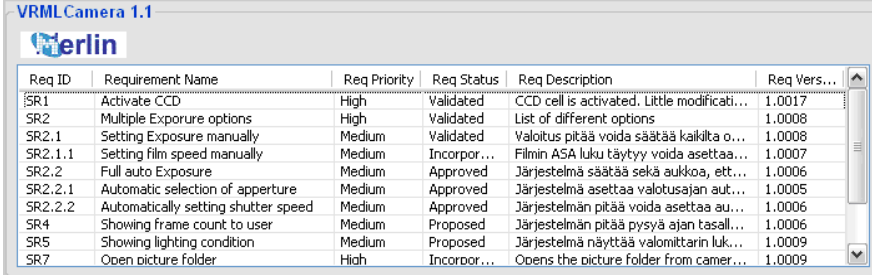
Figure 17. Creating a new feature in the OSRMT plug-in.

However, it was later realized that this approach is sub-optimal due to the effort needed to understand the source code of the tool itself, and the integration method causes the tool to be too tightly coupled with the integration. Tight coupling can be problematic when the tool is updated, which results in the fact that the plug-in integration has to be updated accordingly. In retrospect, it might have been easier to connect directly to the underlying database, but this was not initially done because the OSRMT database structure was incomprehensible (even with the help of the source code). [37]

Information from OSRMT is shown in TC by using the tree interface. Implementing the integration interface for OSRMT was rather quick due to the fact that the developer was already very familiar with how the OSRMT tool operates on the source code level. All that was needed was some glue code to finish the integration. The OSRMT plug-in is perhaps the most complicated plug-in of TC due to the nature of its implementation (the plug-in is the tool itself). [37]

7.1.1.6 IBM Rational RequisitePro

Rational RequisitePro is a requirements management tool made by IBM [42]. The ReqPro plug-in was the first integration developed for ToolChain. Development of the ReqPro plug-in was done by Samuli Heinonen, VTT [43]. The plug-in gets its data from the Rational RequisitePro's MS Access 2000 database via a JDBC driver. The ReqPro plug-in shows the information about project requirements in a table view (figure 18). The information is in the same format as the Rational RequisitePro tool client shows and consists of requirements id, name, priority, status, description and version. [37]



Req ID	Requirement Name	Req Priority	Req Status	Req Description	Req Vers...
SR1	Activate CCD	High	Validated	CCD cell is activated. Little modificati...	1.0017
SR2	Multiple Exposure options	High	Validated	List of different options	1.0008
SR2.1	Setting Exposure manually	Medium	Validated	Valoitus pitää voida säätää kaikilta o...	1.0008
SR2.1.1	Setting film speed manually	Medium	Incorpor...	Filmin ASA luku täytyy voida asettaa...	1.0007
SR2.2	Full auto Exposure	Medium	Approved	Järjestelmä säätää sekä aukkoa, ett...	1.0006
SR2.2.1	Automatic selection of aperture	Medium	Approved	Järjestelmä asettaa valotusajan aut...	1.0005
SR2.2.2	Automatically setting shutter speed	Medium	Approved	Järjestelmän pitää voida asettaa su...	1.0006
SR4	Showing frame count to user	Medium	Proposed	Järjestelmän pitää pysyä ajan tasall...	1.0006
SR5	Showing lighting condition	Medium	Proposed	Järjestelmä näyttää valomittarin luk...	1.0009
SR7	Open picture folder	High	Incorpor...	Opens the picture folder from camer...	1.0009

Figure 18. TC's ReqPro plug-in, RequirementsTable view.

7.1.1.7 Telelogic Synergy/CM

Telelogic Synergy/CM is a commercial, task-based version management tool. Telelogic has been acquired by IBM. [44] A commercial Eclipse plug-in exists for Synergy/CM, but it was not tested due to high licensing costs. Furthermore, no plug-in was developed for TC for Synergy/CM.

Synergy/CM stores its information in an IBM Informix database. Integration between TC and Synergy/CM was created by reverse engineering the information stored in the Informix database. Reverse engineering the database structure was somewhat complicated and involved parsing information from the tuples due to the rather interesting design approach taken by the original database developers.

The Informix JDBC driver is used to connect to the tool database. The integration was created in a way that direct relation is made between the file version stored in Synergy/CM and the requirement from TC. The tool's task-based approach is not used in the integration. The integration was developed during the early stages of the first phase implementation and was not transferred to the "generalized" traceability view, mainly due to high licensing costs of the tool and expiration of the tool license. However, it would be quite easy to integrate Synergy/CM again into the TC, provided that the tool itself was available. [37]

7.1.1.8 Subversion

Subversion (SVN) is an open source version management tool, designed to be a replacement for CVS, which is widely used in the open source community [45, 46]. Subversion has an existing Eclipse IDE integration called Subclipse [47]. It was decided that this plug-in would be used, instead of implementing a separate plug-in, because it offers "reference" integration into Subversion. [37]

Because of the existing integration of Subversion for Eclipse it was only necessary to get the most essential status information from Subversion. The status information includes for example version number, creation date, last modified date and modifier. The integration between the tool chain and Subversion was performed by using the Subversion API. SVN provides two interfaces: JavaHL (JNI) and SVNKit (pure java). Of these two, JavaHL was used because it proved to be more suitable for purposes of this integration. [37]

The integration works in the following manner: 1) the user selects a project stored in SVN from the package explorer, 2) the user creates a link between the requirement and SVN stored file by dragging the chosen file from the package explorer into the TC. This creates a SVN path

7. Tool integration implementation

for the file (e.g. <http://test.com/project/trunk/file.c>) when creating the link into the traceability database. Now if the user wants to see requirement-specific information, all the files linked to the requirement and related information is displayed by fetching the information from the Subversion database via the Subversion API (figure 19). [37]

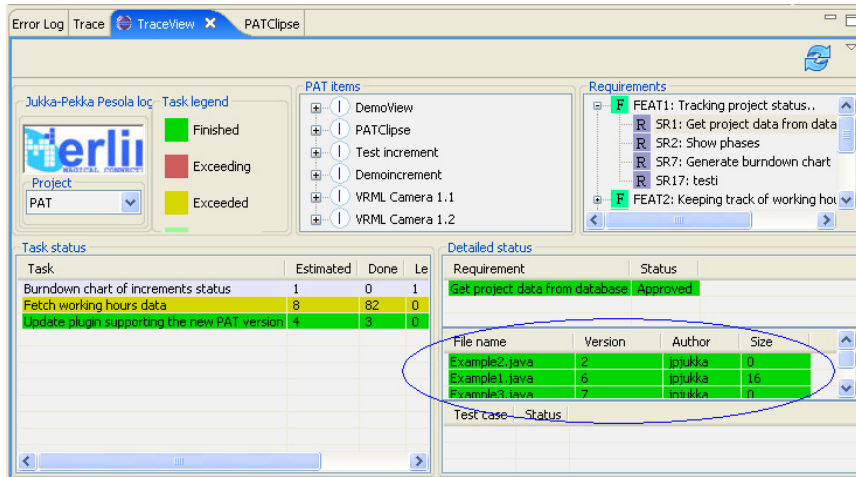


Figure 19. Subversion integration, SVN stored files linked to requirement.

7.1.1.9 SoftFab

SoftFab is a testing and test management tool internally developed and used by Philips [48]. The SoftFab plug-in for Eclipse was developed by the University of Oulu. The plug-in provides test case and report information for requested requirement(s) (figure 20).

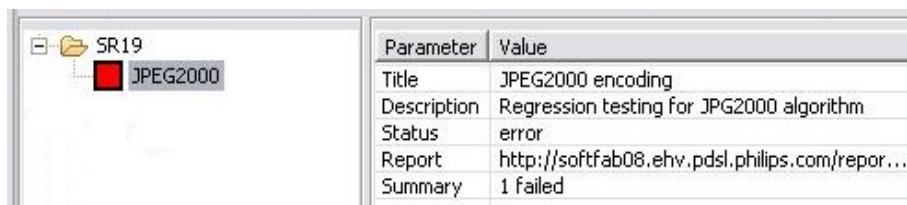


Figure 20. The SoftFab plug-in.

The actual test execution is performed by the SoftFab server, so the plug-in only fetches the testing information from the server. The plug-in communicates with the SoftFab server in XML over a HTTP connection. It is capable of operating in standalone mode, but can also pass test case status information to TC when requested. [37]

7.1.1.10 Testlink

Testlink is open source, web-based test management software [49]. The Testlink plug-in fetches information directly from Testlink's MySQL database via JDBC connectivity and shows the information in the Testlink view. In the Testlink plug-in the user is able to navigate to test cases

by selecting a project from the combo box, followed by a test suite which has the test cases under it. The plug-in shows the most essential information from test cases such as status, build, etc. [37]

7.2 Implementations in the second phase

7.2.1 PROBE framework integration

The Probe framework provides tools for test data instrumentation from embedded Linux devices and management of instrumented test data by means of a probe database [50]. Probe integration from the TC point of view consists of creating a plug-in for interfacing with the probe database. The Probe framework structure has been previously described in chapter 6.2.1. The main purpose of the Probe plug-in is to supply the user a means of managing test data by fetching a list of test data stored in the Probe database and by providing means for further processing of the data. Figure 21 shows how the Probe plug-in displays test set data in a table.

	PacketSize	Frame/s	PacketTransmitted	PacketReceived	AverageLatency	PacketLoss	TransmitThroughput	ReceivedThroughput
Max	10	10	5		100			
Min								
In limits (%)	0	0	0	N/A	42			83
Above limits (%)	100	100	100	N/A	58			17
Below limits (%)	N/A	N/A	N/A	N/A	N/A			N/A

	PacketSize	Frame/s	PacketTransmitted	PacketReceived	AverageLatency	PacketLoss	TransmitThroughput	ReceivedThroughput
66	500	2500	2500	31.599	0	0.264		0.264
66	13950	69750	69750	31.615	0	7.3656		7.3656
	27400	137000	136999	31.6139	1	14.4672		14.4671
	40850	204250	204248	31.6133	2	21.5688		21.5686
	54300	271500	271498	31.6152	2	28.6704		28.6702
	67750	338750	338747	31.6158	3	35.772		35.7717
	81200	406000	405997	31.615	3	42.8736		42.8733
	94650	473249	473245	31.6149	4	49.9751		49.9747
	108100	540500	540495	31.6153	5	57.0768		57.0763
	121550	607749	607745	31.9741	4	64.1783		64.1779
66	135000	674999	674994	32.5854	5	71.2799		71.2794
222	500	2500	2500	69.5187	0	0.888		0.888
222	13950	69750	69749	69.5414	1	24.7752		24.7748
222	27400	137000	136998	69.5388	2	48.6624		48.6617
222	40850	204250	204246	69.5402	4	72.5496		72.5482
222	54300	258264	258257	108.2329	7	91.7354		91.7329

Figure 21. Probe DB plug-in showing test set data.

The plug-in operates by sending queries in XML format over a TCP/IP connection to the Probe database, and by parsing the comma separated return values. The basic sequence of operations done when the plug-in is launched and when the connection is made is as follows:

1. Get projects
2. For each project get project versions
3. Get test cases for each project version
4. Get test sets for each test case (i.e. different executions of the same test case)
5. Get output types for each test case (i.e. attributes, or columns).

The plug-in will then continue to build a tree structure (project version-test case-test set) of the fetched data. The plug-in also allows inspection of test data values by clicking on a specific test

set in the tree. The test data is shown in a table with statistics of the test data. The test data is fetched via specific queries. If upper and lower limits have been specified for the output types (i.e. columns), statistics include percentages of rows that were above the upper limit, below the lower limit, and inside the limits. The plug-in also allows updating the upper and lower limits of the test case in a table and thus resulting in re-calculation of the statistics. Values can be updated in the Probe database via a TCP/IP connection by sending an XML query.

7.2.2 PERVIS and PERSIM integration

PerVis & PerSim have been developed for visualization and simulation of single-core & multi-core load models. The probe database (DB) integration into TC was created to make further processing of the test data easier. To begin the analysis, the user needs to choose the test data set by selecting the test set, rows of the test set that will be used in processing, and output types (i.e. columns, attributes, etc.). Selection of the test data set is shown in figure 21. However, the measurements for PerVis & PerSim are stored in custom format in the Probe DB, under only one output type. In this case each measurement row provides information on thread task switches with time information. To fetch the data from the Probe DB, the Probe plug-in creates a specific XML query which is then sent to the Probe DB, and the results of the query are then stored in a file which will be passed to either PerVis for visualization or to PerSim for simulation based on the selection by the user in the Probe plug-in.

It is also possible to specify visualization parameters for the PerVis tool in the Probe DB plug-in, in the PerVis tab (figure 21). The specifiable parameters are visual resolution, device clock rate, start tick, and end tick. The parameters are passed to the PerVis tool together with the location of the data set file on tool execution.

The PerSim simulation works in the same manner. The Probe plug-in has its own tab for specifying simulation parameters. The parameters for simulation are: number of processors, cycles-per-instruction in simulation, cycles-per-instruction in measurement, print sampling cycles, and simulation granularity. The parameters of the simulation are passed to the PerSim tool in the same manner as with the PerVis tool.

The tools are launched from TC as separate system processes. In the case of the PerSim tool, outputs from the tool and progress of the simulation are monitored in the PerSim tab. When simulation is finished, it is possible to visualize the results of the simulation with PerVis. The results are stored in a specific path & file from where the PerVis tool can locate them. The tool integration does not yet provide means for storing simulation results in the Probe DB, which would be beneficial because running simulations is time consuming and sequential simulation executions overwrite each other. If this is implemented in the future, multiple simulation results (with varying parameters) could be stored under the Probe DB test set.

7.2.3 MVA tool integration

The MVA tool can be used to visualize test data to support the analysis process. As in the case of PerVis & PerSim integrations, the MVA tool integration with the TC Probe DB plug-in works in a similar manner. Initially the user selects a test set (or in some cases multiple test sets), several output types (at least three), and multiple rows of data for analysis. The test data is

stored in the Probe DB: each output type represents a real value, instead of the PerVis/PerSim “binary string” which in fact represents multiple output types. This kind of data representation makes it easier to distinguish the real meaning of the data; however in case of the “binary string” it is not very meaningful to separate the output types, because the tool always needs to process complete strings. Selection of a test set with rows and outputs is shown in figure 21. To fetch the data from the Probe DB, the Probe plug-in creates a specific XML query which is then sent to the Probe DB, and results of the query are then stored in a file which is passed to the MVA tool.

The Probe plug-in allows for specifying analysis parameters for the MVA tool in the Probe DB plug-in. Parameters can be specified in the MVA tab in the plug-in (figure 21). Some notable specifiable parameters (from the TC integration point of view) are usage of upper and lower limits and usage of markers and indexes (useful when analysis is performed on multiple test sets, to differentiate the sets and values). These parameters affect how the CSV file, which contains the data set, is created by the plug-in. The data format for the CSV file is as follows:

1st row: column names

2nd row: column values or upper limit (if upper limit parameter is selected)

3rd row: column values or lower limit (if the lower limit parameter is selected)

1st column: column 1 values or marker description (if markers are used)

2nd column: column 2 values, or column 1 values (if markers are used), or index values (if marker/index parameter is used)

3rd column: column 3 or 2 or 1 values (based on the preceding selections).

The fact that the CSV file containing the test data needs to be built differently based on the selection means that the XML query for the Probe DB needs to be formatted according to the selected parameters. The CSV file contents are initially generated by the Probe DB, which takes care of the markers, indexes, and column values. All these are comma separated, each row representing measurement values of columns (i.e. output types like CPU utilization, memory consumption, etc.). The Probe-plug-in appends output type names on the first row of the CSV file, inserts upper and lower limits to the following rows (if needed) and subsequently appends the Probe DB generated columns/rows to the end of the file. If multiple test sets were selected, each test set is sequentially appended after each other in the CSV, after being imported from the Probe DB.

The MVA tool parameters are written to their own file. When the MVA tool is launched for test set(s), the location of the configuration- and CSV data files are passed as parameters to the tool, which is launched as standalone software in a separate process. After a while, the MVA tool will pop up with the contents of the test set(s) and specifications made by the parameters.

7.2.4 Workflow implementation and integration

Eclipse cheat sheets are used in this thesis as means of implementing workflow support in the integrated environment. Cheat sheets (workflows) are defined in XML. Eclipse cheat sheets provide two alternative approaches to definition: simple and composite cheat sheets. A simple

7. Tool integration implementation

cheat sheet consists of one task and many steps. A composite cheat sheet consists of many tasks with many steps. Composite cheat sheets are more suitable for defining workflows. The format for the simple and complex cheat sheets is defined in Eclipse documentation (cheat sheet content file format). [36]

Cheat sheets are composed of tasks, sub-tasks, and steps (in this order). All steps need to be taken to complete a task (or sub-task), but steps can be skipped or redone. Tasks can also have dependencies between themselves, such as one task having to be completed before another can be started. Each task guides the user through steps to achieve a certain goal (figure 22). Tasks can include hyperlinks and references to further documentation; this can be a useful feature when guiding the user through a real development process.

The screenshot shows the Eclipse IDE interface. On the left, the 'Probe' view is active, displaying a table of performance metrics. The 'Data' table has the following columns: PacketSize, Frame/s, PacketTransmitted, PacketReceived, AverageLatency, PacketLoss, TransmitThroughput, and Received. The table contains 22 rows of data. On the right, the 'Trace' view is active, showing a 'ToolChain workflow support' tree and a 'Define analysis' panel. The 'Define analysis' panel has a 'Select data for analysis' section with instructions: 'Select "Data"-tab in "Probe"-view. The table shows test data for selected test case. Rows can be chosen for analysis by selecting the desired rows from table.' and 'In "Probe"-view select output types (e.g. cpuLoad, memory consumption etc.) from under test case for analysis (multiple can be selected if CTRL is pressed while left clicking)'. There is also a 'Choose analysis method' section with instructions: 'Go to either "Multivariate analysis" step or "Performance simulation" depending on your situation.' and a 'Click when complete' checkbox.

Figure 22. Example of the TC cheat sheet workflow support mechanism.

To model a workflow it is first necessary to somehow capture the basic steps of the work. One way of capturing the workflow is to follow the execution of work while taking notes. The captured workflow can be modelled for example as a block diagram, by using MS Visio or MS PowerPoint. The example workflow presented below follows the phases of the process model shown in figure 5:

1. Requirements definition (OSRMT)
2. Test case creation (TestLink)
3. Test execution & gathering of test data (Probe framework)
4. Test analysis:
 - a. Performance simulation – PerSim,
 - b. Performance visualization – PerVis.

After the workflow has been modelled on a general level, each step needs to be studied in more detail. The details can include e.g. steps needed to perform requirement definition in the OSRMT software. After the workflow has been modelled in adequate detail, the workflow can

be translated to Eclipse cheat sheets. However, it is up to the user to decide how to model the workflows as cheat sheets.

Eclipse provides a cheat sheet editor for defining cheat sheets, or cheat sheets can be created manually by using a text editor. The Eclipse cheat sheet editor is rather crude and simple, so in practice defining complex workflows by using either the Eclipse editor or by a text editor is a rather slow process. The maintainability and ease of updating of the workflows is also questionable. Furthermore, the system provides limited interaction possibilities for the users: adding comments or modification of the workflow during execution is not possible. In the current implementation cheat sheets are distributed with the TC plug-ins, but in the improved version the cheat sheets could be deployed, for example, through the Subversion integration of the TC which would make delivering up-to-date workflows to large user groups easier and automatic.

7.2.5 Improved traceability view

The traceability view of the TC maps together information gathered from the integrated tool set. In the traceability view, information from the RM tool has a central role. Information from all the other tools is either directly or indirectly related (test data from PF is related to test cases, which are related to requirements) to the requirements. Thus, it is possible to maintain traceability between requirements and related items by drag & dropping items from the tools' plug-ins to slots reserved for these tools in the traceability view.

The traceability view provides a list of requirements and shows in parentheses the amount of traced items for the requirements (e.g. 4/1/0, which means that there are four tasks, one file stored in CM and no test cases for the requirement). If a requirement that has traced items is clicked on, the traceability view will fetch information for the selected requirement into a field reserved for requirement information, and show traced items from various tools in their own reserved fields with additional information on the items. This fetching of information is done by querying the tool-specific plug-ins, more specifically the interface they implement to integrate with the TC.

In the current implementation traceability is only possible between requirements and related items, with the exception of the test data (PF) – test case (TM tools) relation, but some work has already been done to overcome this limitation. Figure 23 shows the complete Eclipse/TC environment with the traceability view in the bottom half of the screenshot.

Usage of traceability has been improved in the second phase of the implementation so that the traceability operations allow for an arbitrary link structure between items from different tools. However, work remains to update the traceability view to allow for observing and defining the traceability from other perspectives than from RM, because in some cases it could be useful to see, for example, what items are related to a specific test case. The possibility to generate a graph of the traceability structure would also be a powerful tool for representing the linkages between the items.

Figure 9 shows the integration and example use of four different tools in TC. For each tool the entities that are connected to another entity in a certain relationship are presented in the figure. This linking of the connections is based on the entities' unique identifiers, which can be for example id strings. Based on these unique identifiers traceability links are created and stored in to the traceability database by the Traceability view's drag & drop mechanism. As a database back-end, TC uses the open source database engine MySQL for storing the traceability information.

7. Tool integration implementation

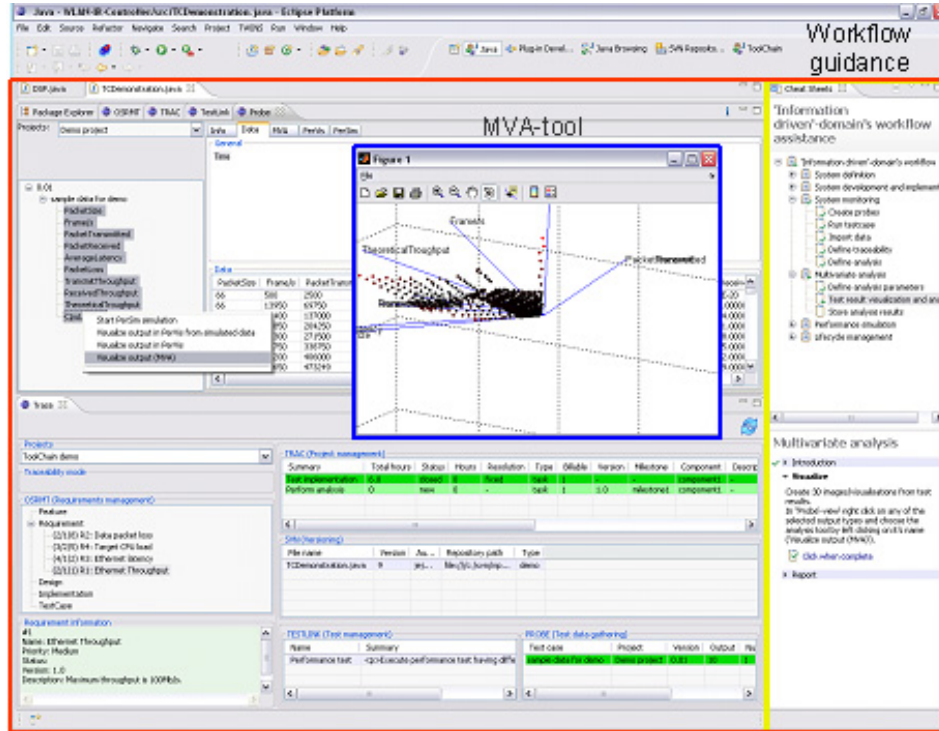


Figure 23. Overview of TC/Eclipse environment with workflow system.

7.2.6 Summary of the integrated tools

The following table (table 20) lists all the tools that have been integrated into the ToolChain. Support for some of the tools provided in phase 1 were deprecated in phase 2, mainly because of the high licensing costs of the tools making further development impossible.

Table 20. List of TC integrated tools.

Tool	Type	License	Phase
Philips Project Assist Tool	PM	Internal	1
Open workbench	PM	Open source	1 & 2
TRAC	PM	Open source	1 & 2
Microsoft Project	PM	Commercial	2
IBM Rational RequisitePro	RM	Commercial	1 & 2
Telelogic DOORS	RM	Commercial	1 & 2
OSRMT	RM	Open source	1 & 2
Telelogic Synergy/CM	CM	Commercial	1
Subversion	CM	Open source	1 & 2
SoftFab	TM	Internal	1
TestLink	TM	Open source	1 & 2
Probe framework	Testing	Open source	2
PerVis & PerSim	Analysis / Simulation	Internal	2
MVA tool	Analysis	Internal	2

8. Tool integration trial and validation

This chapter deals with validation and experiences of using ToolChain. As mentioned previously, development of ToolChain has been performed in two phases. Therefore, the validation of the TC has also been done separately for the two phases. The result of the first phase of development, Merlin ToolChain, has been released under GNU General Public License (GPL) version 2. It has been made available for download at SourceForge, which is a portal for distributing open source software [51]. It has been downloaded over one hundred times at time of writing. Both versions of the TC have been validated in industry cases, and the results are presented in this chapter. Development of the second version was performed on a non-GPL licensed branch of the TC, and no decision on its public release has been made.

8.1 Philips case

This validation case of the first version of ToolChain was carried out in cooperation between Philips and VTT. The trial was conducted in the OSIB project, realized by Philips Applied Technologies. The OSIB project aims to provide Integrated Ambient Experience™ for a new hotel chain. The software is designed and developed for several hardware subsystems interconnected through well-defined software interfaces. The sub-systems of the OSIB solution are as follows: [13]

- Moodpad: advanced remote control with a touch screen and hardware buttons. Moodpad is used to enable the user interface to the hotel room (controlling the light, TV, Venetian blinds, sunscreen curtain, room climate).
- Room Controller: implements the hotel room logic and interacts with the Moodpad, room TV, climate control system, RFID door lock.
- Ambient Experience server: sub-system functioning as a gateway between hotel rooms and the external components: Property Management System of the hotel chain and the remote hotel diagnostics centre.

All the devices communicate either via Ethernet or wireless (Wi-Fi) connections with each other. The project uses an agile way of working, where development is done in increments of two weeks and requirements are selected for each increment, together with the customer. [13]

The goals for the case from the ToolChain development viewpoint were to evaluate the usability and usefulness of ToolChain in real-life product development. The aim was to gain experience from

setting up a tool chain in an industry context, from its use in practice, as well as from the adaptability of the selected concepts to the needs appearing during the use of ToolChain. [13]

From the case project viewpoint, the goals for using the tool chain were to improve traceability and visibility of the project progress while not interrupting the product development work. [13]

The case was the first time that the ToolChain was tried out in a real-life setting, so the technical improvements of ToolChain during the case were significant. First ToolChain was adapted to the Philips environment by integrating the tools they had in use to the ToolChain. The toolset used in the case was Project Assist Tool, Telelogic DOORS, Subversion, and Philips SoftFab. In practice, this meant developing a new plug-in for DOORS, modifying the existing Subversion plug-in, and updating the PAT and SoftFab plug-ins because the versions used by the project differed from those used in the development of the ToolChain. [13]

The installation of ToolChain in the case project environment was done in the duration of a week when the ToolChain developers from VTT visited Philips to set up the ToolChain. Some compatibility problems were encountered at first, but they were handled quite easily as they were mainly configuration problems of the tools. Some additional features considered essential by the case project were also added to the ToolChain during the installation. [13]

The results of the trial showed that ToolChain could be adapted and set up for the industrial project environment relatively quickly: only a week for setup was needed. Furthermore, the results revealed that even though the ToolChain implementation of the first phase was only a research prototype, the setup period could be shortened in the future. [13]

In the trial OSIB project members would use the ToolChain in everyday development work. During this period the OSIB project members wrote down all the encountered problems and ideas for improvement. This feedback was given to the ToolChain developers bi-weekly and urgent problems were handled immediately. The feedback was also analyzed and prioritized in the bi-monthly workshops with industry partners. Some examples of the improvements made for the ToolChain based on the feedback include: development of the tool selector that enables selecting the tools that are in use, improving usage of available space in the user interface, and showing more information to the user. The database software was also changed to an open source alternative from a commercial one during this development period. The new version of ToolChain was sent to Philips after finalization. [13]

In conclusion, during the evaluation period many new features were added to ToolChain and as a result TC became more robust for practical use. The case also proved that TC can be used in daily operations in a natural way without complicating things, but rather making working easier. [13]

8.2 NSN case

The result of the second phase of development, integrated tool support for hardware-related software development, was validated in an industry case. In the case, a tool set consisting of Eclipse and TC plug-ins, Trac, OSRMT, Subversion, Testlink, Probe framework, and Multivariate Analysis tool was used. The purpose of the case was to evaluate the ease of installation and functionality of the tool integration. In the case a workflow was formulated containing the steps necessary to evaluate multi-core DSP processor performance using MVA

and ToolChain. A multi-core DSP board and its in-house developed TCP/IP stack Ethernet performance was measured using a commercial load tester by Agilent. The CPU load was measured by using an on board software hook.

The performance requirements for the system were as follows:

- Ethernet throughput, minimum throughput is 100Mb/s
- Data packet loss, maximum packet loss is 5%
- Ethernet latency, maximum latency is 100 ms
- Target CPU load, maximum target CPU load is 15%.

The performance requirements were defined in the requirement management software, which in this case was OSRMT. The test case was then created in test management software Testlink to reflect the performance requirements. TC was used to create traceability between the performance requirements and the test case. Figure 24 shows the traceability in TC.

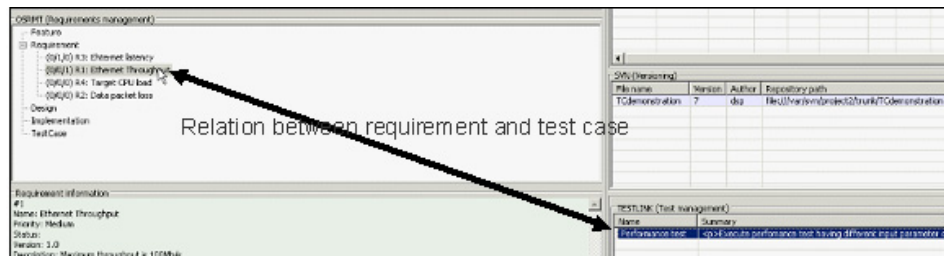


Figure 24. Traceability of requirement and test case in TC.

The test case was subsequently executed by following the test case definition in Testlink. In practice this meant using the Agilent network transmitter and loader to generate load for the DSP board's TCP/IP stack with a loop and measuring the results. The measurements from Agilent were then exported into a CSV format and imported into the *probe database*. Figure 25 shows the imported test data in TC. Traceability between the test case and test data from the test run was updated in TC, as shown in figure 26.

The screenshot shows the ToolChain Probe plug-in interface. On the left, a tree view shows 'sample data for demo' with sub-items like PacketSize, Frame/s, PacketTransmitted, PacketReceived, AverageLatency, PacketLoss, TransmitThroughput, ReceivedThroughput, TheoreticalThroughput, and CpuLoad%. On the right, a data table is displayed with the following columns: PacketSize, Frame/s, PacketTransmitted, PacketReceived, and AverageLat. The table contains 10 rows of data.

PacketSize	Frame/s	PacketTransmitted	PacketReceived	AverageLat
66	500	2500	2500	31.599000
66	13950	69750	69750	31.615000
66	27400	137000	136999	31.613900
66	40850	204250	204248	31.613300
66	54300	271500	271499	31.615200
66	67750	338750	338747	31.615800
66	81200	406000	405997	31.615000
66	94650	473249	473245	31.614900

Figure 25. ToolChain Probe plug-in with test data.

8. Tool integration trial and validation

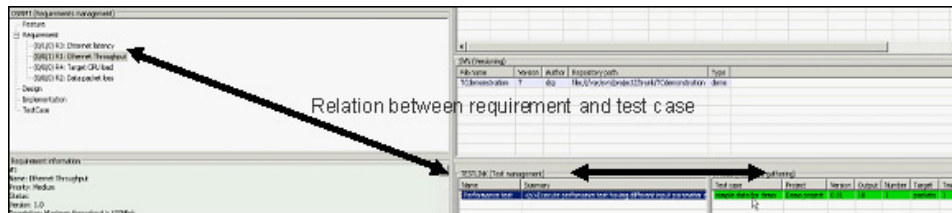


Figure 26. Traceability of requirement, test case, and test data.

Analysis of the test case was performed using the TC probe plug-in. The test data set, its attributes (e.g. packet size, frames/s, latency) and rows (i.e. measurement values) were selected in the plug-in. Furthermore, analysis parameters for the Multivariate Analysis tool were specified in the plug-in. After data set selection and analysis parameters were defined, the analysis tool was executed from the TC. Figure 27 shows the analysis parameters and visualization results for the data set.

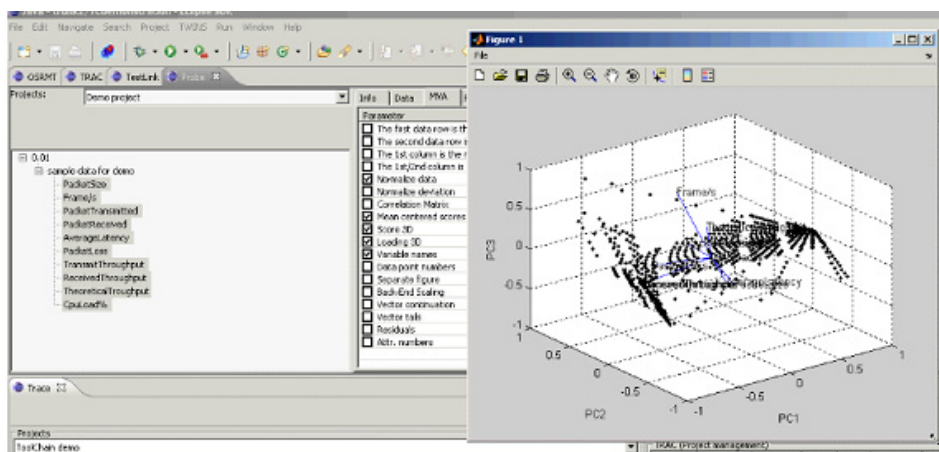


Figure 27. Specifying analysis parameters for MVA tool and visualization of results.

From the resulting visualization, it is possible to calculate the failure rate and thus gain knowledge of the TCP/IP stack performance. In this case the results show a high failure rate. Results of the analysis are stored under the test case in Testlink. In this case the test case failed because the failure rate was too high for the software in question. TC notifies the user that the requirement has not been validated because all the test cases have not passed.

The preceding scenario was defined in the TC workflow system, so that a user unfamiliar with the tool set and TC integration could perform the steps from importing the data into the probe database all the way to inputting the analysis results into Testlink. The complete picture of the TC/Eclipse environment with *probe plug-in*, *traceability view*, and *workflow guidance* can be seen in figure 23.

After the execution of the case, the users of the system were asked to provide feedback on the tool integration. *The users were asked the following general questions:*

- Experiences with installation and taking the TC into use
- Usage experiences with the TC
- Strong and weak points of the TC
- Optimal solution (i.e. “perfect” integration / tool set).

The following feedback was provided on the questions: the users reported experiencing some difficulties during installation, mainly in installation of the open source tools and due to the fact that the tool documentations were scattered (i.e. each open source tool had its own instructions) and locating them was time consuming. As a result, the installation took some time. They reported that if the installation and taking TC into use were to happen in a tighter schedule, more support would be needed.

Initially, the users considered the idea of tool integration of Eclipse and separate tools as complex and complicated. However they then reported that the tools used in the case (OSRMT, Trac, Testlink, MVA) were quite easy to learn to use; though some of them were easier than others. Previous demonstrations of TC and documentation provided for the first phase TC implementation were considered helpful. They also reported that support was provided and available via e-mail as needed. Furthermore, workflow support made it easier for new users to get used to TC, and workflows could be updated relatively quickly if needed. They suggested that a complete user guide would prove helpful when beginning to use TC, but also reported that the manual for the phase one implementation (i.e. Merlin manual, available from SourceForge) was helpful.

In conclusion, they consider the usage of the TC as quite straightforward; the Eclipse environment and the TC plug-ins are easy to understand, and traceability with its drag and drop implementation is easy to use. TC also provides quite good selection of different tools: project management, configuration management, requirement management, and test management. However, especially in the given setting, tool support of test automation is needed. Furthermore, there is a need for a more extensive tool set than what is provided by the current implementation. Moreover, all of the features available in the tools are not used. There also seems to be a need for fine-tuning many of the TC plug-ins; some of the output from plug-ins was considered too “rough”. Workflow support was considered as an added bonus, but automation would be needed in workflows.

9. Discussion

In this thesis it was studied how an integrated tool support solution could be used to aid in hardware-related software development. This section discusses the choices made, lessons learned, and how successful the results were. The discussed topics include: tool integration approach, tool integrations, traceability support, process and workflow support, dissemination work done on TC, validation cases, and future work.

9.1 Integration approach

Design of the TC began with the architectural decision on how the various tools would connect to each other. Existing literature on tool integration with the help of industry experiences and knowledge of existing tool integration implementations (chapter 3) were used to guide the research and design process. With this help, a framework-based solution, more precisely Eclipse, was chosen. This turned out to be a good decision, in the sense that the tool integration built on the Eclipse framework is now running smoothly. Furthermore, it is clear that a framework-based solution offers good potential for further development, as shown by the two-phase implementation of the TC: even after the first phase of implementation, completely different kinds of new tools and mechanisms such as workflow support could be easily added to the second phase implementation.

When Eclipse was initially selected, it was still in its infancy, but already gathering momentum. Today, the Eclipse ecosystem is expanding at tremendous pace, with new tools (e.g. tools for modelling, communication, and language-specific implementations) and features added at a steady rate. Eclipse is also widely used in the information industry and is well known to large user groups. TC will directly benefit from these aspects as there will be an increased amount of tools that can be easily integrated to work with or form part of TC.

9.2 Tool integrations

In the first phase of development, a set of core tools (PM, RM, CM, TM) was integrated into the TC. These tools aim to fulfil the basic needs of a software development project. Challenges in HW rel. SW development are plentiful; however TC's support in terms of tools for these challenges is currently limited (see table 20 for the list of tools). HW rel. SW development tool support in TC was realized in the form of tools for testing, test result analysis, and for estimating the performance of different processor architectures. As mentioned previously, it is not practical to integrate every available tool; thus the tools chosen were selected based on industry feedback on which tools would provide the most added value. In TC's case, the

complete set of tools (phase 1 & phase 2) consists of tools from multiple vendors. TC also provides the possibility of expanding the tool set via its integration interface, which provides TC users the freedom of adding their own tools. This is a completely different approach to that of proprietary tool vendors who build their tool integrations consisting entirely of their own tools.

TC consists of plug-ins with varying levels of refinement. Some of the plug-ins are suitable for only maintaining traceability in the traceability view, while some of the plug-ins offer tighter integration into the tools (e.g. OSRMT integration which allows performing many of the tool's functions in the Eclipse environment, or the HW rel. SW integrations created in the second phase of the implementation which provide seamless integration of testing, analysis, and performance simulation into the TC environment). The varying levels of refinement of the plug-ins stem from the fact that the plug-ins were used to study how to integrate the tools into TC.

9.3 Process and workflow

The systems engineering viewpoint was used as a starting point in this thesis to understand the big picture of the HW rel. SW development process, and to understand what kinds of activities need to be supported by the tools. Furthermore, various process models were inspected to provide better understanding on needed support. Specific challenges related to HW rel. SW development were also studied from literature and from experiences gathered, in order to specify requirements of the tool integration solution (whose main purpose is to support the HW rel. SW development process). The workflow concept was introduced in order to provide support to users on a personal level.

In practice, workflow and process support was thus realized partly via the tools provided by the integration (chapter 9.2) and in part as workflow support as provided via the Eclipse cheat sheet system. Before workflows could be supported by the system, it was necessary to somehow capture the workflow. As part of the NSN case (chapter 8.2), a performance testing scenario was modeled into the Eclipse cheat sheet system. The modelling method used was fairly ad hoc; block diagrams were used to capture the steps in the process and were then detailed to the tool level by taking notes on what the needed actions were. Subsequently, the workflows were defined in the cheat sheet system as well as possible. The purpose of this thesis was not to build a workflow system, but to study how workflow support could be used to improve tool integration in a way that efficiency is maximized. One way for workflow support to do this is to provide support for tools and to give guidance on how to use the tools together. Thus, the developed workflow system is fairly simple, but nevertheless the chosen approach seems to be working fairly well as indicated by the positive feedback given on workflow support of TC in the NSN case.

9.4 Dissemination

Two scientific articles have been written on the development of the TC: a conference paper by Heinonen et al. [29] and a conference paper by J.-P. Pesola, J. Eskeli, P. Parviainen et al. [13]. The author of this thesis has participated in the writing of the latter paper as one of the main contributors. The history of TC development has been published in the previously mentioned

papers, all the way to the first stage of TC implementation (including the Philips validation case) as discussed in this thesis. The author of this thesis also plans on publishing a conference article on the new research that has been discussed in this thesis.

The new research that was performed as part of this thesis includes addition of several new tools and tool types in the TC framework, proving further its adaptability to different situations. Workflow support was also introduced as a concept that can be used to guide developers working with multiple tools in an integrated environment to complete certain activities. Furthermore, the NSN validation case was presented in which the current state of implementation was tested.

The result of first version implementation has been published as an open source tool in SourceForge. However, it has not yet been decided when or how the result of the second phase of implementation will be published.

9.5 Validation

TC was implemented in two phases. For this reason, TC was validated separately in two different cases. The Philips case has previously been presented and analyzed in detail in [13]; therefore the focus is on the NSN validation case in this chapter. However, the results from the Philips case are important and show that TC can be installed in the target environment rather quickly and TC operates naturally in day-to-day work while providing improved visibility into project progress.

Shifting from a work environment without tool integration into one with tool integration is not, however, without complications. If the users are not already familiar with the Eclipse environment, the initial adaptation to using the system may take some time, and it may be challenging to find the time in today's fast-paced research & development work. One great challenge related to tool integrations is how to lower the adaptation threshold for tool integration in a company. TC tackles this challenge by providing the possibility for companies to use their own tools, as opposed to single vendor tool integration solutions that dictate certain tool sets. Thus, via the TC method the original way of working is disturbed as little as possible.

9.5.1 NSN case

The limitation of this case compared to the Philips case was that there were fewer users participating in the trial and the use scenario was artificial in a way that the performance analysis on the DSP platform had already been done but was then replicated using the TC tool set and integration. Thus, the case provides a rather static usage of TC and does not provide a complete overview of the TC capabilities. However, the case also provided useful feedback on the usage of TC in an industry setting, which can be used to further develop the TC integration.

In this case, one of the users reported difficulties in installation of the various tools used in the tool integration. However, more often than not, this is not a problem because the tools have already been installed and taken into use, as the main idea of TC is to connect the tools already in use and thus provide additional visibility and traceability to the development. Installation of the TC itself went relatively smoothly, with some support needed from TC developers. Comparison on ease of installation should be made on the scale of similar tool integration solutions, not on the scale of how difficult it is to install, for example, a web browser on a

desktop computer. Tool integrations are not taken into use everyday, and when done, adequate time must be preserved for preparation. In this case there were some difficulties due to the fact that documentation for the second phase implementation was not available at the time of delivery of TC to the case environment.

Scattered documentation of the tools was reported as a problem. This is a common problem in heterogeneous tool integration environments. It can be alleviated via workflow support. Workflow support can provide instructions on tools used in the workflows by for example providing hyperlinks to the tool documentation in the workflow description.

In the case one of the users reported that the development environment consisting of TC and various tools seemed complicated at first, but also makes note that workflow support helps in adapting to the new environment. The users also reported difficulties in using the tools. However, the main focus in this thesis and in the built solution was on the integration, not on the ease of usability, of the integrated tools. Furthermore, in most situations, users are already familiar with the tools and the only new element for them would be to learn the use of Eclipse/TC integration. However, in the case of a new employee it would be beneficial for workflow support to teach the new user on how to use the tools.

Some critique was presented on the roughness of the plug-ins of the tools and on the lack of support for certain kinds of tools (e.g. test automation) in the TC. However, as has previously been mentioned, TC's purpose is not to be a complete solution but rather a proof of concept that proves the feasibility of the tool integration approach.

9.6 Future work

This section discusses future work related to matters discussed in this thesis. As previously mentioned, the research that was done during the second phase of implementation, focusing on hardware-related software support (e.g. the new tools and workflow system), has not yet been published. The author of this thesis plans to publish these results in a conference article as soon as possible.

As to the future development of the TC, adding reporting facilities is one possibility; the validation cases revealed need for reporting facilities in tool integration, and it was also defined as a requirement for the tool integration. Reporting has not yet been added because it was initially prioritized as of lower importance than the actual proof of concept of tool integrations and because further study is needed on what kinds of reports should be generated.

Reporting generates formatted sheets of information for some specific need or purpose. As a starting point for implementation, static reports could be used; e.g. if the user wants to see the status of the requirement test coverage, the system could check for each requirement the traced test cases, then check test case statuses, and finally generate a report (e.g. PDF) where the results are summarized and important details highlighted. Reporting could be further improved when full understanding of the needed support is available (e.g. based on industry feedback).

TC currently provides communication only in the form of data visibility. As a means of improved communication, the Eclipse Communication Framework (ECF) could be used. ECF provides a means of implementing e.g. real-time shared editing in the Eclipse environment or real-time communication [52].

Feedback received during the first phase of implementation suggested that providing a web-based UI for TC in addition to the Eclipse UI would increase TC's usability potential. This way, TC would become more platform independent.

In the current implementation of TC the traceability view is limited to a requirement centric view, i.e. everything is inspected from the point of view of requirements. This could be further developed, because in some situations it may be necessary to inspect things from different viewpoint, e.g. what requirements relate to this test case.

To begin with, the workflow support in TC could be improved by upgrading the used Eclipse version to Eclipse Ganymede, which provides an improved cheat sheet editor [53]. Furthermore, as an improvement to the workflow modelling scenario, a formal modelling language could be used (e.g. state charts) to capture the workflow which would make updating of the workflows easier. The cheat sheet system could be improved by adding automation into the workflow that could perform mundane tasks in the tools automatically, or by allowing the user to choose an action from the cheat sheet system that would then orchestrate the necessary actions in the tools.

As mentioned in the requirements (tables 17–19), TC could support use of multiple tools of same type concurrently (e.g., two partners have different RM tools in use, but need to share the requirements with each other). In the current implementation, this is not supported. As a first step in this direction, the Eclipse community provides project task management integration in the form of the Mylyn framework, with which it is possible to see tasks from Bugzilla, Trac, and JIRA in Eclipse [54]. This would in theory allow use of these PM tools simultaneously. At least the current Trac plug-in could be discarded in favour of Mylyn, which provides much neater facilities for task management than the current Trac plug-in. To support the traceability in the drag & drop form, Mylyn could be modified to support TC. Furthermore, these kinds of possibilities signify why choosing Eclipse as an integration framework was a good choice: the open source community is constantly providing new tools and integrations to Eclipse, for free.

Installation of TC is currently done manually by copying the needed files (i.e. plug-ins, configuration files, etc.) to specific directories as pointed out by the documentation. The installation could be improved by creating an installer wizard that automates most of the installation process and prompts the user only when needed.

10. Conclusion

Development of hardware-related software poses various challenges, and for solving these challenges a multitude of tools have been developed. Furthermore, this multitude of tools needs to be used seamlessly during development, but more often than not, the tools are disconnected which makes use needlessly difficult because the consistency of data in the tools needs to be managed manually. This thesis focused on studying if some of the HW rel. SW development challenges could be solved efficiently by means of improving tool support and the interoperability between the tools.

The research process began with the aim of forming a general overview of HW rel. SW development. As a first step, HW rel. SW development was perceived from a systems engineering viewpoint and various well-known process models were studied. This was done to understand what kinds of activities need to be supported by the tools. The workflow concept was introduced to support the development process.

The tool integration concept and various tool integration approaches, mechanisms, and existing implementations were then studied. This was done to establish the rationale for tool integration and to form a concrete background for the requirements and design of the integration solution.

The next step was to study the challenges experienced in HW rel. SW development. The challenges were gathered from literature and individual experiences of a professional developer. The challenges were then mapped to the overall process, also called the TC process model. The challenges were then used to specify requirements for the tool integration solution. Requirements were divided into two main categories: tool support and tool integration requirements. The tools supported by the integration were selected based on their integration potential and based on industry feedback. Basic requirements for tool integration were to provide traceability between development artefacts, data flow from tool to tool, and improved visibility into project data. An architectural decision was also made at this point that a framework-based solution would be used which would allow easy integration of tools.

As a next step, design of TC was done where the TC framework solution was formulated. TC is based on an Eclipse framework, which is a Java open source IDE. TC consists of tool-specific plug-ins and of a traceability view plug-in that implements the core functionality (i.e. traceability, data visibility, etc.). The tool specific plug-ins connect to tools used in the integration via means such as JDBC and API. Traceability information is stored in the centralized traceability database. TC allows addition of new tools via plug-ins that implement the TC tool interface. HW rel. SW development support of TC focuses on improving the test data gathering from embedded systems, management of the test data, and using the gathered data in analysis of SUT behaviour and in analysis of the SUT performance. Furthermore, workflow support was

designed to aid TC users during work. The workflow system guides the user on how to use the tools and integration to accomplish certain activities. A task-based Eclipse cheat sheet system is used for the implementation of the workflow.

TC was implemented in two phases. The first phase consisted of creating the TC framework on the Eclipse platform and integrating the tools from PM, CM, TM, and RM categories, with a total of nine tools integrated in this phase. The first phase of the implementation built the foundation of TC, including the plug-in based approach, integration interface for new tools, traceability and data visibility implementation into traceability view plug-in, etc. In the second phase of the implementation the tools for HW rel. SW development support were added: Probe Framework, PerVis & PerSim, and the MVA tool. In addition work was done on improving the TC traceability model and workflow support in the form of the Eclipse cheat sheet system.

Validation of TC was also performed in two phases: a validation case for each phase. In the first phase TC was validated in Philips' OSIB project, and results of this case pointed out how easily TC could be installed and adapted to the target company's environment. TC also worked in daily operations without complications, while improving the traceability and data visibility in the project. The first phase implementation was also released as open source in SourceForge where it has been downloaded well over one hundred times.

The validation case for the second phase showed no big surprise in the sense that the installation of TC went rather smoothly. Some critique was given on the difficulty of configuring individual tools used in TC. This is not the fault of TC but rather the consequence of the nature of the tools used in the integration. After the case, requests were also made for support for certain types of tools which would have improved the tool chain's usability in the given scenario (i.e. test automation). Workflow support in TC was considered promising.

According to the background study that was performed prior to the design of TC on tool integrations, tool integration frameworks, and on related literature, it seems that the integration approach used by TC is the first of its kind. TC has also generated much interest on various occasions where it has been presented (in e.g. conferences and seminars). As the validation cases show, TC seems to be answering to the needs of industry partners in a way that it is easy to take into use and can be adapted to different situations. With TC it is possible to choose one's own set of tools – even a tool set consisting of purely open source tools is possible. This leads to the fact that with TC there is no risk of (expensive) vendor lock-in to proprietary tool sets. However, TC is not a complete product in the same sense as the proprietary tool sets (e.g. MS TFS), but rather a proof of concept solution.

Development of TC has also been a learning experience in a way that much more is now known on how to design and create tool integrations, for example what kind of data should be linked together and from which tools, and how the integration should be performed. One of the main challenges in tool integration thus seems to be in deciding which data to link together (everything can be integrated, but one may ask if it is worth it) and on resolving how the tool integration could use this information for added value. In future, research done on TC for this thesis will be published in a conference paper presenting the new aspects. In order to improve TC adding new features such as reporting, communication, and new tools is also possible. Further research into the workflow system and its role in tool integration seem to be interesting alternatives, too.

References

- [1] Keller, M. & Schumate, K. (1992) Software specification and design, A disciplined approach for real-time systems. John Wiley & Sons. 405 p.
- [2] Ronkainen, J., Kääriäinen, J. & Abrahamsson, P. (2003) Technical report, VTT.
- [3] Stevens, R., Brook, P., Jackson, K. & Arnold, S. (1998) Systems Engineering: Coping with Complexity. Pearson Education. 374 p.
- [4] Douglass, B. (1999) Doing hard time: developing real-time systems with UML, objects, frameworks, and patterns. Addison Wesley. 714 p.
- [5] (read 27.02.2009) Wikipedia: Waterfall model. URL: http://en.wikipedia.org/wiki/Waterfall_model.
- [6] Kotonya, G. & Sommerville, I. (1998) Requirements Engineering: Process and Techniques. John Wiley & Sons. 282 p.
- [7] Crnkovic, I., Asklund, U. & Dahlqvist, A. (2003) Implementing and Integrating Product Data Management and Software Configuration Management. Artech House, London. 338 p.
- [8] (read 27.02.2009) Wikipedia: Iterative and incremental development. URL: http://en.wikipedia.org/wiki/Iterative_and_incremental_development.
- [9] Georgakopoulos, D. & Hornick, M. (1995) An overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. Distributed and Parallel Databases, Vol. 3, pp. 119–153.
- [10] (read 27.02.2009) Wikipedia: Workflow. URL: <http://en.wikipedia.org/wiki/Workflow>.
- [11] Chan, D.K.C. & Leung, K.R.P.H. (1997) Software Development as a Workflow Process. In: the Proceedings of Joint 1997 Asia Pacific Software Engineering Conference (APSEC'97), December 2–5, Hong Kong SAR, China.
- [12] Pederson, J. (2006) Creating a tool independent system engineering environment. In: IEEE Aerospace Conference, 4–11 March. 8 p.
- [13] Pesola, J.-P., Eskeli, J., Parviainen, P., Kommeren, R. & Gramza, M. (2008) Experiences of Tool Integration: Development and Validation. In: Mertins, K., Ruggaber, R., Popplewell, K. & Xu, X. (eds.). Enterprise Interoperability III – New Challenges and Industrial Approaches. Springer. Pp. 499–510.
- [14] Industrial survey of tools used in requirements engineering and management, architecture design, and product information management (PLM/PDM,CM), VTT, 2007.
- [15] EUROPEAN COMMISSION, ICT – INFORMATION AND COMMUNICATION TECHNOLOGIES, Work Programme 2007.
- [16] Kanwalinder, S. (1993) Tool Integration Frameworks -- Facts and Fiction. In: IEEE Proceedings of the National Aerospace and Electronics Conference 2. Pp. 750–756.
- [17] Nghiem, A. (read 27.02.2009) Web Services Part 6: Models of Integration. URL: <http://www.awprofessional.com/articles/article.asp?p=28713&seqNum=2>.
- [18] (read 27.02.2009) Eclipse: Application Lifecycle Framework. URL: <http://www.eclipse.org/proposals/eclipse-almiff/index.php>.

- [19] (read 27.02.2009) Digital, Framework-Based Environment Design Center, Version 2.0, SPD 56.03.00. URL: <http://h18000.www1.hp.com/info/SP5603/SP5603PF.PDF>.
- [20] Burmester, S., Giese, H., Niere, J., Tichy, M., Wadsack, J., Wagner, R., Wendehals, L. & Zuendorf, A. (2004) Tool integration at the meta-model level: the Fujaba approach. *International journal on software tools for technology transfer*, Springer, Vol. 6, No. 3, pp. 203–218.
- [21] Schwaber, C. (2006) *The Changing Face of Application Life-Cycle Management*. Forrester Research Inc., August 18.
- [22] Kääriäinen, J. & Välimäki, A. (2008) Impact of Application Lifecycle Management – A Case Study. In: Mertins, K., Ruggaber, R., Popplewell, K. & Xu, X. (editors) *Enterprise Interoperability III – New Challenges and Industrial Approaches*. Springer. Pp. 55–67.
- [23] Amsden, J. (read 24.03.2009) Levels Of Integration, Five ways you can integrate with the Eclipse Platform. URL: <http://www.eclipse.org/articles/index.html>.
- [24] El-khoury, J., Redell, O. & Torngren, M. (2005) A tool integration platform for multi-disciplinary development. In: 31st EUROMICRO Conference on Software Engineering and Advanced Applications, August 30 – September 3. Pp. 442–449.
- [25] (read 27.02.2009) Wikipedia: Embedded Software. URL: http://en.wikipedia.org/wiki/Embedded_software.
- [26] Lee, E.A. (2002) Embedded Software. In: Zelkowitz, M. (ed.). *Advances in Computers*, Vol. 56, Academic Press, London.
- [27] Interview of Jussi Ronkainen on embedded software challenges, VTT, 2008.
- [28] (read 27.02.2009) Wikipedia: Real-time computing. URL: http://en.wikipedia.org/wiki/Real-time_computing.
- [29] Heinonen, S., Kääriäinen, J. & Takalo, J. (2007) Challenges in collaboration: tool chain enables transparency beyond partner borders. In: *Proceedings of 3rd International Conference on Interoperability for Enterprise Software and Applications I-ESA 2007*, March 28–30.
- [30] (read 27.02.2009) Eclipse foundation home pages. URL: <http://www.eclipse.org/>.
- [31] (read 27.02.2009) Notes on Eclipse plug-in architecture. URL: http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html.
- [32] Pollari, M. (2009) *A Software Framework for Improving the Testability of Embedded Real-time Systems*. Master's thesis. University of Oulu, Department of Electrical and Information Engineering, Oulu.
- [33] Vitikka, J. (2008) *Supporting database interface development with application lifecycle management solution*. Master's thesis. University of Oulu, Department of Electrical and Information Engineering, Oulu.
- [34] Jaakola, M. (2008) *Performance simulation of multi-processor systems based on load reallocation*. Master's thesis. University of Oulu, Department of Electrical and Information Engineering, Oulu.
- [35] Tuuttila, P. (2006) Test result visualisation and analysis with principal component analysis. In: *Proceedings of Estiem vision of cycles seminar*, February 3–4, Finland, Oulu.
- [36] (read 27.02.2009) Documentation on Eclipse cheat sheets. URL: http://help.eclipse.org/ganymede/topic/org.eclipse.platform.doc.isv/guide/ua_cheatsheet.htm.
- [37] Heinonen, S., Pesola, J.-P. & Eskeli, J. (2007) *Merlin ToolChain technical specification*, VTT.

- [38] (read 20.03.2009) Open Workbench project management tool. URL: <http://www.openworkbench.org/>.
- [39] (read 20.03.2009) Trac project management tool. URL: <http://trac.edgewall.org/>.
- [40] (read 20.03.2009) DOORS requirements management tool.
URL: <http://www.telelogic.com/products/doors/>.
- [41] (read 20.03.2009) OSRMT requirements management tool.
URL: <http://sourceforge.net/projects/osrmt/>.
- [42] (read 20.03.2009) RequisitePro requirements management tool.
URL: <http://www-01.ibm.com/software/awdtools/reqpro/>.
- [43] Heinonen, S. (2006) Requirements Management Tool Support for Software Engineering in Collaboration. Master's thesis. University of Oulu, Department of Electrical and Information Engineering, Oulu.
- [44] (read 20.03.2009) Telelogic Synergy change management tool.
URL: <http://www.telelogic.com/corp/Products/synergy/>.
- [45] (read 20.03.2009) Subversion version control system. URL: <http://subversion.tigris.org/>.
- [46] (read 20.03.2009) Concurrent Versions System.
URL: http://en.wikipedia.org/wiki/Concurrent_Versions_System.
- [47] (read 20.03.2009) Subclipse integration for Subversion. URL: <http://subclipse.tigris.org/>.
- [48] Spanjers, H. (read 20.03.2009) Philips SoftFab testing system. URL: <http://www.topic.nl/nl/cm-workshop/presentations/Hans%20Spanjers%20-%20Philips%20Applied%20Technologies.pps>.
- [49] (read 20.03.2009) Testlink test management tool. URL: <http://testlink.org/wordpress/>.
- [50] (read 20.03.2009) Probe Framework for test data management and instrumentation.
URL: <http://sourceforge.net/projects/noen/>.
- [51] (read 20.03.2009) Merlin ToolChain integration framework.
URL: <http://sourceforge.net/projects/merlintoolchain>.
- [52] (read 20.03.2009) Eclipse Communication Framework. URL: <http://www.eclipse.org/ecf/>.
- [53] (read 20.03.2009) Eclipse Ganymede website. URL: <http://www.eclipse.org/ganymede/>.
- [54] (read 20.03.200) Eclipse Mylyn website. URL: <http://www.eclipse.org/mylyn/>.

Author(s) Eskeli, Juho		
Title Integrated tool support for hardware related software development		
Abstract <p>This thesis presents how the hardware-related software development process can be improved by means of tool integration. Challenges in hardware-related software development are diverse, which is why a multitude of tools is needed during the development. The tools produce data that needs to be managed, but the tools are disconnected. Tool integration provides a means of bringing the data from disconnected tools together into one coherent, easily manageable package.</p> <p>Research was conducted by initially perceiving hardware-related software development from a systems engineering viewpoint, with a focus on several well-known process models. This was done to understand the kinds of activities that need to be supported by the tools. A workflow concept was introduced as a means to support the development effort of an individual worker. An extensive background study into tool integrations was conducted to understand state-of-the-art tool integration approaches and concepts, and then used to create the foundation for the tool integration.</p> <p>Hardware-related software development challenges were gathered from literature and industry experiences to reinforce the understanding on needed tool support and to specify the requirements for the tool integration. The main requirements for the tool integration were easy extensibility, which could only be provided via a framework-based solution, and a means to provide data flow from tool to tool while preserving traceability between the data from the tools. Tool requirements for the integration were project management, requirement management, test management, and change management tools. Emphasis was put on tools supporting testing and test analysis.</p> <p>The tool integration, ToolChain, was implemented in two phases. In the first phase the groundwork for the integration framework was done. Eclipse was chosen as the platform for the integration and plug-ins selected as a means of implementation. In the second phase, tool support focusing on the hardware-related software development aspects was added. Implementations from each phase were validated separately in industry cases. Experiences from these cases are presented in which it is shown how ToolChain can be easily adapted into the target company's environments, and how the tool integration improves the way of working.</p>		
ISBN 978-951-38-7373-8 (URL: http://www.vtt.fi/publications/index.jsp)		
Series title and ISSN VTT Publications 1455-0849 (URL: http://www.vtt.fi/publications/index.jsp)		Project number 6086
Date December 2009	Language English, Finnish abstr.	Pages 83 p.
Name of project TWINS		Commissioned by VTT
Keywords development tools, tool integration		Publisher VTT Technical Research Centre of Finland P.O. Box 1000, FI-02044 VTT, Finland Phone internat. +358 20 722 4520 Fax +358 20 722 4374

Tekijä(t) Eskeli, Juho		
Nimeke Integroitu työkalutuki laiteläheiseen ohjelmistokehitykseen		
Tiivistelmä Työssä esitetään, miten rautaläheisten ohjelmistojen kehitysprosessia voidaan parantaa työkaluintegraation avulla. Rautaläheisten ohjelmistojen kehitystyön haasteet ovat monimuotoisia, ja siksi kehitystyön avuksi tarvitaan useita työkaluja. Työkalut tuottavat tietoa, jota täytyy hallinnoida, mutta toisaalta työkalut ovat irrallisia, mikä tekee hallinnoinnista hankalaa. Työkaluintegraatio mahdollistaa tietojen koostamisen irrallisista työkaluista yhtenäiseksi, helposti hallittavaksi kokonaisuudeksi. Tutkimustyö aloitettiin tarkastelemalla rautaläheisten ohjelmistojen kehitystä systeemi-suunnittelun näkökulmasta. Tarkastelu keskittyi yleisesti tunnettuihin prosessimalleihin, ja sen tavoitteena oli selvittää, mitä aktiviteetteja työkalujen tulee tukea. Työnkulut (workflow) esitettiin keinona tukea yksittäisen työntekijän kehitystyötä. Työkaluintegraation nykytila selvitettiin kattavasti mahdollisten lähestymistapojen löytämiseksi, ja tätä tietoa käytettiin työkaluintegraation perustana. Rautaläheisten ohjelmistojen kehitykseen liittyviä haasteita koottiin kirjallisuudesta ja teollisuuskokemuksista vahvistamaan ymmärrystä tarvittavasta työkalutuesta ja määrittämään vaatimukset työkaluintegraatiolle. Päävaatimuksina työkaluintegraatiolle asetettiin laajennettavuus, minkä mahdollistamiseen kehikko (framework) -pohjainen ratkaisu sopii luontevasti, ja lisäksi tiedon kulku työkalusta työkaluun sekä jäljitettävyyden ylläpitäminen työkaluissa syntyvien tietojen välille. Työkaluvaatimuksina integraatiolle asetettiin projektinhallinta-, vaatimustenhallinta-, testauksenhallinta- ja muutoksenhallintatyökalut. Erityisesti painotettiin testauksen ja testi-analyysin työkalutukea. Työkaluintegraatio, ToolChain, toteutettiin kahdessa vaiheessa. Ensimmäisessä vaiheessa suoritettiin pohjatyö integraatiokehitykselle. Integraatioalustaksi valittiin Eclipse ja Eclipsen liitännäiset (plug-in) integraatioiden toteutuskeinoksi. Toisessa vaiheessa lisättiin työkalutuki, joka painottuu rautaläheiseen ohjelmistokehitykseen. Kunkin vaiheen toteutukset validoitiin erikseen teollisuuskokeilussa. Teollisuuskokeilujen kokemukset esitetään, joista käy ilmi kuinka ToolChain voidaan helposti ottaa käyttöön kohdeyrityksen kehitysympäristössä, ja kuinka työkaluintegraatio helpottaa työskentelyä.		
ISBN 978-951-38-7373-8 (URL: http://www.vtt.fi/publications/index.jsp)		
Avainnimeke ja ISSN VTT Publications 1455-0849 (URL: http://www.vtt.fi/publications/index.jsp)		Projektinumero 6086
Julkaisu-aika Joulukuu 2009	Kieli Englanti, suom. tiiv.	Sivuja 83 s.
Projektin nimi TWINS		Toimeksiantaja(t) VTT
Avainsanat development tools, tool integration		Julkaisija VTT PL 1000, 02044 VTT Puh. 020 722 4520 Faksi 020 722 4374

VTT PUBLICATIONS

- 708 Satu Innamaa. Short-term prediction of traffic flow status for online driver information. 2009. 79 p. + app. 90 p.
- 709 Seppo Karttunen & Markus Nora (eds.). Fusion yearbook. 2008 Annual report of Association Euratom-Tekes. 132 p.
- 710 Salla Lind. Accident sources in industrial maintenance operations. Proposals for identification, modelling and management of accident risks. 2009. 105 p. + app. 67 p.
- 711 Mari Nyysönen. Functional genes and gene array analysis as tools for monitoring hydrocarbon biodegradation. 2009. 86 p. + app. 59 p.
- 712 Antti Laiho. Electromechanical modelling and active control of flexural rotor vibration in cage rotor electrical machines. 2009. 91 p. + app. 84 p.
- 714 Juha Vitikka. Supporting database interface development with application lifecycle management solution. 2009. 54 p.
- 715 Katri Valkokari. Yhteisten tavoitteiden ja jaetun näkemyksen muodostuminen kolmessa erityyppisessä verkostossa. 2009. 278 s. + liitt. 21 s.
- 716 Tommi Riekkinen. Fabrication and characterization of ferro- and piezoelectric multilayer devices for high frequency applications. 2009. 90 p. + app. 38 p.
- 717 Marko Jaakola. Performance Simulation of Multi-processor Systems based on Load Reallocation. 2009. 65 p.
- 718 Jouko Myllyoja. Water business is not an island: assessing the market potential of environmental innovations. Creating a framework that integrates central variables of internationally successful environmental innovations. 2009. 99 p. + app. 10 p.
- 719 Anu Tuominen. Knowledge production for transport policies in the information society. 2009. 69 p. + app. 52 p.
- 720 Markku Hänninen. Phenomenological extensions to APROS six-equation model: non-condensable gas, supercritical pressure, improved CCFL and reduced numerical diffusion for scalar transport calculation. 2009. 60 p. + app. 54 p.
- 721 Aku Itälä. Chemical Evolution of Bentonite Buffer in a Final Repository of Spent Nuclear Fuel During the Thermal Phase. 2009. 78 p. + app. 16 p.
- 722 Kai Hiltunen, Ari Jäsberg, Sirpa Kallio, Hannu Karema, Markku Kataja, Antti Koponen, Mikko Manninen & Veikko Taivassalo. Multiphase Flow Dynamics. Theory and Numerics. 2009. 113 p. + app. 4 p.
- 723 Riikka Juvonen. DNA-based detection and characterisation of strictly anaerobic beer-spoilage bacteria. 2009. 134 p. + app. 50 p.
- 724 Paula Jouhten. Metabolic modelling and ¹³C flux analysis. Application to biotechnologically important yeasts and a fungus. 2009. 94 p. + app. 83 p.
- 725 Juho Eskeli. Integrated tool support for hardware-related software development. 2009. 83 p.