



Egil Trygve Baadshaug
egiltryg-at-stud.ntnu.no

Eilev Hagen
eilev-at-stud.ntnu.no

Kris-Mikael Krister
krismika-at-stud.ntnu.no

Eirik Benum Reksten
eirikben-at-stud.ntnu.no

Daniele Giuseppe Spampinato
spampina-at-stud.ntnu.no

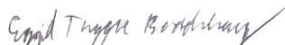
Ketil Sandanger Velle
ketilsan-at-stud.ntnu.no

November 22, 2007

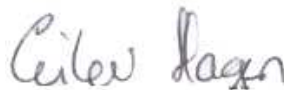
Preface

This report is the result of a project assigned by the course *TDT4290 Customer Driven Project*, completed autumn 2007. The project team was composed of six students attending the Norwegian University of Science and Technology (NTNU), Department of Computer and Information Science (IDI). The task for our project team was to create a modeling tool useful for application developers that want to avoid software vulnerabilities in their creations. Our customer was SINTEF represented by Per H. Meland and Jostein Jensen. We would like to thank them both for their high spirit and continuous support. We would also like to thank our supervisors, Renate Kristiansen and Basit Ahmed, for the valuable help and assistance they have contributed during the project.

Trondheim, November 2007



Egil Trygve Baadshaug



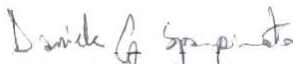
Eilev Hagen



Kris-Mikael Krister



Eirik Benum Reksten



Daniele Giuseppe Spampinato



Ketil Sandanger Velle

Contents

List of Figures	vii
List of Tables	x
Listings	xii
1 Project introduction	1
1.1 Terminology	1
1.2 Report structure	2
1.3 Project results	3
2 Project directive	5
2.1 Introduction	5
2.2 Project mandate	5
2.3 Concrete project work plan	11
2.4 Organization	16
2.5 Templates and standards	18
2.6 Version control	21
2.7 Documentation of project work	22
2.8 Quality assurance	22
2.9 Overall test plan	23
3 Preliminary studies	25
3.1 Introduction	25
3.2 Business requirements	26
3.3 Current situation	26
3.4 Sources of information	28
3.5 Security modeling techniques	34
3.6 Desired solution	48
3.7 Market investigations	51
3.8 Alternative solutions	52
3.9 Evaluation criteria	52
3.10 Evaluation	53
3.11 Choice of solution	58
4 Requirements specification	59
4.1 Introduction	59
4.2 Overall domain description	61
4.3 Functional requirements	64
4.4 Non-functional requirements	86
4.5 Summary	92

5 Design	93
5.1 Introduction	93
5.2 System architectural design	93
5.3 Detailed description of components	97
5.4 User interface design	104
5.5 Requirements not designed	106
5.6 Summary	108
6 Realization	111
6.1 Introduction	111
6.2 Coding Conventions	111
6.3 Process of generating the application	113
6.4 Summary	124
7 Testing	127
7.1 Introduction	127
7.2 Overall test plan	127
7.3 Specific test plan	129
7.4 Test results	145
7.5 Tracking of tests	153
7.6 Summary	154
8 Installation and user documentation	157
8.1 Introduction	157
8.2 Installation guide	157
8.3 The SeaMonster user manual	158
8.4 Contents of the enclosed CD	165
9 Evaluation	169
9.1 Introduction	169
9.2 The process	169
9.3 Involved parties	178
9.4 The course	179
9.5 The project task	180
9.6 Fulfillment of project goals	180
9.7 Further work	182
9.8 Summary	182
10 Project conclusion	185
Bibliography	187
A Project directive appendix	195
A.1 Involved parties	195
A.2 Abstract for the FP7-ICT project SHIELDS (215995)	197
A.3 Concrete project work plan	198
A.4 Documents and standards	207
A.5 Common ground rules	215
B Pre-study appendix	219
B.1 Screenshots from different programs	219

C	Specific requirements appendix	223
C.1	Functional Requirements	223
C.2	Non-Functional Requirements	226
C.3	Documentation Requirements	227
C.4	Process Requirements	229
D	Design appendix	231
D.1	Detailed class diagram	231
D.2	SeaMonster XMI schema	233
E	Testing appendix	241
E.1	Early system test	241

List of Figures

2.1	The Unified Process life cycle.	12
2.2	The organization chart for the project.	17
3.1	Increase in discovered software flaws.	27
3.2	Example of an item from the CVE list.	29
3.3	Example of an item from the CWE list.	30
3.4	Mapping example between XCCDF and OVAL.	32
3.5	Vulnerability cause graph example.	36
3.6	A graphical attack tree.	37
3.7	A textual attack tree.	37
3.8	An attack tree using boolean values and logical ANDs.	38
3.9	An attack tree with costs linked to the nodes.	39
3.10	Exploit graph: a mobile code attack.	41
3.11	A simple use case diagram for a online bookstore.	43
3.12	Use case diagram with the “extend” and “include” relationship.	43
3.13	A misuse case diagram of a web shop.	44
3.14	Misuse case diagram of a web shop in more detail.	45
3.15	UMLsec use case diagram.	45
3.16	Security pattern class diagram.	47
3.17	Security pattern sequence diagram.	48
3.18	The notation used in security activity graphs.	49
3.19	Vulnerability cause graphs and security activity graphs.	49
3.20	Example of a security activity graph.	50
3.21	Example of a security activity graph.	50
4.1	Relationships between the software security entities.	62
4.2	An overall perspective of the SeaMonster application.	62
4.3	Use case diagram for “Create a new model”.	65
4.4	Use case diagram for “View/Modify an existing model”.	66
4.5	Use case diagram for “Open a model”.	66
4.6	Use case diagram for “Save a model”.	67
4.7	Use case diagram for “Model a security entity”.	68
4.8	Use case diagram for “Copy and paste between models”.	68
4.9	Use case diagram for “Consult CWE/CVE DB’s”.	69
4.10	Use case diagram for “Print a model”.	70
4.11	Global use cases diagram.	72
4.12	Sequence diagram for scenario 1.	74
4.13	Sequence diagram for scenario 2.	76

4.14	Sequence diagram for scenario 3.	77
4.15	Sequence diagram for scenario 4.	78
4.16	Sequence diagram for scenario 5.	79
4.17	Misuse case of “Open a model”.	86
5.1	Global use cases diagram.	94
5.2	Top-level GMF architecture.	95
5.3	Basic GMF development process.	96
5.4	Top level class diagram of the applications data model.	98
5.5	Class diagram for the AttackTree part of the applications data model.	99
5.6	Class diagram for the VCG part of the applications data model.	101
5.7	Class diagram for the Misuse Case part of the applications data model.	101
5.8	System plugins.	103
5.9	System deployment diagram.	104
5.10	An example of how the main board will appear.	105
5.11	Typical Eclipse look and feel.	106
5.12	Notation for the AND nodes in attack trees.	106
5.13	Set of symbols of the VCG notation.	107
5.14	Set of symbols of the Misuse Case notation.	107
5.15	Set of symbols of the Attack Tree notation.	107
6.1	Mapping definition model for AttackTree plugin.	115
6.2	Graphical definition model for AttackTree plugin.	117
6.3	Graphical definition model for AttackTree plugin.	120
6.4	Content of the <code>seaMonster</code> plugin.	123
6.5	Content of the <code>seaMonster.edit</code> plugin.	123
6.6	Content of the <code>seaMonster.xxx</code> plugins.	124
6.7	Communication flow for converting an event to a command.	125
7.1	The V-model.	130
7.2	The work flow of the testing.	132
7.3	Acceptance test: attack tree.	143
7.4	Acceptance test: vulnerability cause graph.	144
7.5	Acceptance test: misuse case diagram.	144
8.1	Main functionality in SeaMonster.	158
8.2	Structure overview of SeaMonster.	159
8.3	Example of a vulnerability cause graph.	159
8.4	Example of an attack tree.	160
8.5	Example of a misuse case.	161
8.6	The sub menu for “New”.	161
8.7	Initializing a new attack tree diagram.	162
8.8	How to save a diagram.	163
8.9	How to open a diagram.	164
8.10	How to print a diagram.	164
8.11	How to export a diagram.	165
8.12	How to add notes to a diagram.	166
8.13	How to copy an element in a diagram.	166

9.1	Timeline with important events marked.	170
9.2	Cause and effect diagram that shows the most notable result of the problems during the project.	173
9.3	Estimated hourly distribution of work in the different phases. . .	176
9.4	Real hourly distribution of work in the different phases.	176
9.5	Graph showing total weekly used hours on the project by the team.	177
9.6	Cause and effect diagram for a successful project.	183
A.1	The legend used in the Gantt diagrams.	198
A.2	Estimated plan of work in form of a Gantt diagram.	199
A.3	Work plan in form of a Gantt diagram after iteration one. . . .	200
A.4	Work plan for the Project directive phase in form of a Gantt diagram.	202
A.5	Work plan for the Preliminary study phase in form of a Gantt diagram.	202
A.6	Work plan for the Requirements specification phase in form of a Gantt diagram.	203
A.7	Work plan for the Testing phase in form of a Gantt diagram. . .	203
A.8	Risk matrix.	208
A.9	Flow chart of decision making within the team.	216
B.1	Screenshot from the Coras tool.	220
B.2	Screenshot from the Microsoft threat modeling tool.	220
B.3	Screenshot from the Microsoft threat modeling tool.	221
B.4	Screenshot from Isograph's AttackTree+.	221
B.5	Screenshot from Amenaza' SecurITree.	222
D.1	Detailed class diagram for the MisuseCase part of the applications data model.	232

List of Tables

2.1	Estimated and used workload distribution.	10
2.2	Milestones for the project.	11
2.3	Iterations and phases during the project.	13
2.4	Activities during the planning phase.	13
2.5	Activities during the preliminary studies phase.	14
2.6	Activities during the requirements specification phase.	14
2.7	Activities during the design phase.	14
2.8	Phase responsible for the realization phase.	15
2.9	Activities during the testing phase.	15
2.10	Phase responsible for the user and installation guide phase.	15
2.11	Activities during the project evaluation phase.	16
2.12	Activities during the presentation and demonstration phase.	16
2.13	Phase responsible for the conclusion phase.	16
3.1	Cropped CCE entry example of minimum password length in a system.	30
3.2	Cropped result from the OVAL Interpreter application.	32
3.3	Exploit graph table regarding Figure 3.10.	42
3.4	Evaluation criteria grades.	52
3.5	Evaluation criteria for termination of the project.	54
3.6	Evaluation criteria for buying an existing solution.	54
3.7	Evaluation criteria for further development of an existing solution.	55
3.8	Evaluation criteria for developing a new application.	55
3.9	Advantages and disadvantages regarding each source of security related information.	56
3.10	Advantages and disadvantages regarding each security modeling technique.	57
4.1	Software security entities and their related models.	61
4.2	Users and their activities.	63
4.3	Use case for “Create a new model”.	65
4.4	Use case for “View/Modify an existing model”.	66
4.5	Use case for “Open a model”.	67
4.6	Use case for “Save a model”.	68
4.7	Use case for “Model a security entity”.	69
4.8	Use case for “Copy and paste between models”.	70
4.9	Use case for “Consult CWE/CVE DB’s”.	70
4.10	Use case for “Print a model”.	71

4.11 Functional requirements group EF.	80
4.12 Functional requirements group IGF.	81
4.13 Functional requirements groups ATF, VCGF, MCF and MF.	84
4.14 Functional requirements group SF.	85
4.15 Use case for “Open a model”.	87
4.16 Use case for “Input validation”.	87
4.17 Functional requirements group SECF.	88
4.18 Implementation non-functional requirements.	88
4.19 Usability non-functional requirements.	89
4.20 Platform non-functional requirement.	89
4.21 Packaging non-functional requirements.	90
4.22 Legal non-functional requirement.	91
4.23 Business requirements traceability matrix.	92
5.1 Table of requirements not taken into account during the design phase.	109
7.1 Responsibilities for each test group.	128
7.2 Entity tests.	134
7.3 System test: Attack tree modeling.	136
7.4 System test: VCG modeling.	137
7.5 System test: Misuse case diagram modeling.	138
7.6 System test: General functionality.	139
7.7 System test: Platform independence test.	140
7.8 System test: Security.	140
7.9 System test: Linking of views.	141
7.10 Usability test.	142
7.11 Acceptance test.	143
7.12 Consequences of test failure.	146
7.13 Usability test comments from usability tester one.	150
7.14 Usability test comments from usability tester two.	151
7.15 Tracking of tests.	154
A.1 Contact information for the customer.	196
A.2 Contact information for the supervisors.	196
A.3 Contact information for the project team.	196
A.4 Risk analysis	206
C.1 Functional requirements.	226
C.2 Non-Functional requirements.	227
C.3 Documentation requirements.	228
C.4 Process requirements.	230
E.1 System test: Attack tree.	242

Listings

2.1	File structure.	19
2.2	Report folder structure in checked out form.	20
5.1	A method called when the user wants to make “component 2” a predecessor of “component 1”.	100
6.1	Code example: Algorithm for cycle detection in attack trees. . .	112
A.1	Report structure including files.	211
D.1	SeaMonster XMI schema.	233

Chapter 1

Project introduction

Software is compromised by security vulnerabilities, and the amount is continually increasing. Security issues are easily forgotten before the release of the software, and this could lead to severe damage from attacks while vulnerable on a large scale of computers. Consider the following words by Andrew Jaquith about an analysis done on software security [51].

“We found security flaws in 70 percent of the defects we analyzed. After we excluded flaws that were of low business impact or were not easily exploitable, nearly half (47 percent) of the remaining serious defects could have been caught - and fixed inexpensively - during the design stage.”

Why are such evidently trivial mistakes overlooked? Is it possible to reduce the amount of such attack points in software during the design stage, or at least before the release date? Is it possible for security experts to easily explain their valuable research to software developers? These questions will all be addressed during this report, and are the main task for the project.

The work process includes an extensive and broad study on security modeling methods, and different security related databases. This is later on narrowed down to a more specialized selection to be included in a final implementation made to address the problems stated. The study also contains instances relevant for the application implemented, that are outside of the current scope, but should be considered for further work of this project. This approach was requested by the customer so the team could get familiarized with security modeling and best practices available, and to prepare the team members for the actual implementation phase. Several of the sources are also connected in different ways, and as the knowledge of software security was slender throughout the team, the team agreed with the customer regarding the extensive study.

1.1 Terminology

Non-trivial words and expressions not discussed in this report, are described in the glossary on page 193, and are shown as *emphasised* text the first time they are written. Terms or concepts in several words are written as extended form the first time used, but shortened down to an acronym when referred to later on.

Some of the acronyms are rewritten in extended form where this was considered suitable. In addition to this, there are several often used terms throughout the report, and the most important ones are explained below.

The “security” concept is mentioned repeatedly in this report. No wonder of course, since its strictly related to the problem to be solved, and the topics of discussion. The scope is not limited to software security, since many of the methods and subjects mentioned are general and not limited to computers or any digital items at all. However, the focus is directed against software, and the context when reading about security in this report are software and computer related issues. To apply security to software, you have to strengthen and sustain the following three properties [60].

1. Confidentiality, that refers to preventing access by unauthorized users, and limiting information access to the authorized users of the system.
2. Integrity, that refers to the trustworthiness of a resource, arranging that no inappropriate changes are done to the resource, and the source of the resource is from the origin specified.
3. Availability, that refers to the possibility to access the system, or resource, at the planned times.

When any of these are missing or vulnerable, the security could be threatened, and the system might got a weak spot requiring attention.

Vulnerabilities are defects or weaknesses in a system not addressing certain actions, resulting in a possible security breach.

Modeling is a term used throughout the entire report since its highly relevant to the current task’s nature. When referring to “model” or “modeling” in the report, the context is always within modeling security related issues using diagrams if not explicitly stated otherwise.

Threats are used to signify an action with the possible outcome of setting an object or entity in source of danger. A threat arises from a vulnerability, can be exploited by an attack, and can be mitigated by a countermeasure.

Attacks are malicious actions targeting a system, object or entity with the intention of crippling the security. Attacks are often targeting vulnerable areas, to increase the probability of success. When modeling threats and attacks, defenders of the system are usually modeling threats to find vulnerable spots in need of attention. The offenders on the other hand, are instead usually modeling attacks, to locate ways to break the security.

1.2 Report structure

The next chapter in the report is the project directive, regulating the administrative parts of the project and include guidelines and work methods used in the team. Following the project directive is the preliminary studies chapter that contains research on the topics covered by the task, the state of the art possible solutions, and modeling techniques. The chapter contains the main

documented background information of the problem, and serves as a bridge to the next chapter, requirements specification. This includes the system requirements stated explicitly, and also shows the links to the business requirements. The following chapter, the design, includes how the system to be works, and indicate how to reach the requirements. Next up is the realization chapter, which includes the concepts and principles of the automatic code generation the frameworks supply. The needed extra functionality that had to be coded in the traditional way to reach the stated requirements are also found in this chapter. The following chapter is the testing chapter where all the testing documentation for the software is presented. Then follows a brief chapter on installation and user documentation, a chapter on project evaluation and finally the conclusion chapter including a section on further work.

1.3 Project results

The final result of the project is this report and an implementation, or a prototype, used to model security related issues. The application is fully working, but can be extended where this is needed, hence the term “prototype”.

Chapter 2

Project directive

2.1 Introduction

This chapter contains an overall description of the project, the partition, distribution, and structure of the work related to the project. The chapter also lists initiatives taken to ensure adequate monitoring of progress throughout the project, as well as routines for communication between the involved parties. These sections together are called “The project directive”.

2.1.1 Purpose

The purpose of the project directive is to work as a guidance for team members during the project. It is a dynamic document, meaning changes will be made to it throughout the entire span of the project.

2.2 Project mandate

This section contains information about the project’s background, objectives, and constraints, It also presents some high level requirements, in addition to a description of how the project should be coordinated, controlled, and directed.

2.2.1 Project name

The name of the project is “SeaMonster”, based on initial letters from the words security modeling.

2.2.2 Project sponsor

The sponsor of the project is SINTEF. From now on the term “customer” will be used to signify the sponsor of the project. Contact information of the customer can be found in Appendix [A.1](#).

2.2.3 Partners

The following entities are in some way involved in the project.

- The project team is the main working power of the project. The work includes producing all deliveries in the course, such as the final report and the implemented application. Contact information of all the team members can be found in Table A.3 in Appendix A.1.
- The customer states the requirements and requests for the application and the report. The representatives from SINTEF also attend weekly meetings with the project team. Customer contact information can be found in Table A.1.
- The supervisors are monitoring the progress of the project. They answer questions and give the project team weekly input. Contact information of the supervisors can be found in Table A.2.
- Travis Schau, a BFA graduate from the University of Michigan School of Art & Design, provided artistic design assistance to the project. The “SeaMonster” logo was in its entirety made by him.

2.2.4 Scope for the project

Since about year 2000, there has been a continuous growth of *vulnerabilities* in software. These vulnerabilities, which can be major security issues, are normally solved by using patches after the release of the software. Instead of focusing on software security, developers tend to merely rely on firewalls and antivirus applications. They often have little or no experience with security issues, and security experts who are familiar with these issues, often lack programming experience. Published articles and reports concerning software vulnerabilities are often considered long and too cumbersome by developers. For this reason, there is a need for tools that can model vulnerabilities, threats, and attacks, in a way that can help the communication between security experts and developers. By improving this flow of knowledge, it is likely that security will be better implemented at early stages of software development than before.

2.2.5 Result goals

The goals of the project are stated below.

- RG.01** Deliver a general view of how security modeling is done today, including existing vulnerability databases, as well as modeling methods and tools used to describe causes of software attacks and their countermeasures.
- RG.02** Deliver a working prototype used for modeling possible security issues both in existing and planned software.
- RG.03** It should be possible to use the delivered prototype from this project for further development in future projects.
- RG.04** Contribute with background material to the FP7-ICT project SHIELDS (215995)¹.

¹See Appendix A.2 for the complete abstract for this project.

RG.05 Contribute with background material to the working group ISO/IEC JTC1/SC 27 “IT Security techniques”².

RG.06 Expect the number of common security vulnerability occurrences, being the basic reason for most of the threats of today’s IT systems (*viruses*, *worms*, Cross Site Scripting (XSS)³, *phishing*), to decrease notably when the implemented application is actively used during development to get an overview of vulnerabilities, causes and countermeasures. This will also have an educational effect.

2.2.6 Measurement of project effects

The following list presents concrete, measurable goals that are stated by the customer.

M.01 Reduce the time it takes to make a security model by 20%.

M.02 Reduce the time spent on vulnerability lookup during development by 50%.

M.03 At least an average of 10 monthly downloads of the application from the repository.

M.04 A foundation for at least two new student project assignments (2008-2009).

M.05 Create material for at least one scientific publication after the project has been completed.

2.2.7 Limitations

The project can not span more than the scheduled time, due to the fact that the deadline is absolute. Since all team members have at least two other courses in addition to this one, no one can be expected to use more than half of the weekly resources (around 24 of 48 hours) on this project.

2.2.8 Tool selections

Eclipse

The Eclipse platform [26] was chosen as the development Integrated Development Environment (IDE)⁴. The reasons for this are stated below.

- Eclipse is one of the most diverse Java development environments (Java will be the main programming language for this project).
- The Eclipse Graphical Modeling Framework (GMF) [25] and the Eclipse Modeling Framework [27] will be used as a basis for the application, which requires using Eclipse as the development IDE.

²The concrete documents and papers for this group are restricted but see <http://www.standardsinfo.net/isoiec/index.html> for more information.

³An attack targeting vulnerable web pages dynamically generated from information supplied.

⁴Application used to lessen the work of writing code.

L^AT_EX-compiler and BibT_EX-addon

L^AT_EX [57] is a typesetting system for producing reports with decent layout. It is platform independent, and by splitting up the document in several files, it supports working on the same document by several people at the same time. The add-on BibT_EX [28], is to be used to handle references in a proper manner.

SmartDraw

SmartDraw [84] is a utility for drawing a large variety of modeling types, such as Gantt diagrams, cause and effect diagrams [75], and UML diagrams. The software can be used to produce L^AT_EX-friendly output, such as simple images in vector graphics. The application is of commercial nature, and it requires a Windows platform to run. It is used several places in the report, including Chapter 3, Chapter 9 and Appendix A.

MagicDraw UML

MagicDraw UML [72] is a CASE tool supporting the UML 2.0 standard, code engineering for multiple programming languages (Java, C++, C# and others) as well as for data modeling. The tool has teamwork facilities and supports integration with several IDE's (e.g. Eclipse). It is used to model the UML sequence and use case diagrams shown in Chapter 4.

Visual Paradigm for UML

Visual Paradigm for UML [99] is a UML CASE tool supporting the latest version of UML. The tool presents also other capabilities not interesting to the aim of the project, for instance business process modeling. The version used is the free Community Edition, for non-commercial purposes. It is used for some UML models in Chapter 5.

Inkscape

Inkscape [46] is a cross-platform vector graphics editor with capabilities similar to Adobe Illustrator [1], CorelDraw [19], and Xara X [101]. The editor uses the W3C standard Scalable Vector Graphics (SVG) file format [89], making it easy to convert to L^AT_EX-friendly images. It is easy to edit nodes, perform complex path operations, trace bitmaps and much more. Due to the fact that it is open source, it is favoured before similar commercial tools. The editor is used to draw some of the figures in Chapter 4.

Gnuplot

Gnuplot is a command-line driven function plotting utility [37]. It is platform independent and open source. The program can be configured to produce graphs written in L^AT_EX-syntax, and is used to draw graphs in Chapter 3 and Chapter 9.

vim

Due to vim's [98] highly configurable nature, built in \LaTeX syntax highlighting and platform independency, this will be the main tool for editing plain text and writing report documents.

LEd

The \LaTeX -editor, LEd [58], is a free environment for rapid \TeX and \LaTeX document development used by some members of the team.

Google calendar and spreadsheet

The team use a shared Google account [39] giving access to the same calendar and spreadsheet document. Every member of the team adds their lectures and other similar events making them unavailable. The calendar is used to decide when to schedule internal meetings or work periods for the team. The spreadsheet is shared and updated in a similar way. It has been created to document the work done by the individual team members.

km's pie chart application

Several programs were evaluated and tested before accepting the look of the pie charts needed in Chapter 9. None of the tested programs were considered usable by the responsible for this phase, so a drawing tool based on the JFreeChart framework was implemented. The source code can be found at <http://folk.ntnu.no/krismika/kmpie>⁵.

SuperDuper to-do list

As the project management got harder, and the report got more complex, the team wanted to structure the responsible for different tasks in a better fashion, adding strict deadlines to the tasks. A PHP script was created to keep track of tasks, distribute responsibilities, and use as a scribbling pad for what was to be done. The list is located at <http://folk.ntnu.no/krismika/todo>, and the source code can be found at <http://folk.ntnu.no/krismika/todo/todo.php.txt>.

SourceForge

SourceForge [73] is a free online web page for supporting software development. SourceForge will be the external release site for this project, and it has functions like *bugtracking* and *wiki*. This will be used by the team, and it is open for viewing by the users of the implemented software. SeaMonster SourceForge page can be found at <http://sourceforge.net/projects/seamonsster>.

⁵A Gantt diagram creator also made for this project is included. However, this was not used to create the charts shown in the final report

Phase	Team est.	Person est.	Team usage	Person usage	Ratio
Project management	150	25	158	26	105%
Lectures and self-study	150	25	87	15	58%
Planning	120	20	107	18	89%
Pre-study	330	55	306	51	93%
Requirements specification	200	33	341	57	171%
Design	150	25	133	21	89%
Realization	200	33	194	32	97%
Testing	80	13	117	20	146%
Project evaluation	70	12	59	10	84%
Presentation & demonstration	78	13	13	2	16%
Total	1528	255	1515	252	99%

Table 2.1: Estimated and used workload distribution.

SeaMonster modeling tool

Some of the figures in the preliminary studies chapter have been made using the application created in this project. At first, the figures were created using different general purpose modeling tools, but as the project evolved, they were replaced by images created using the SeaMonster application.

2.2.9 Duration

The project starts 2007-08-28⁶ and the final presentation to the customer will be 2007-11-22.

2.2.10 Planned effort

The team consists of six persons. Each person is expected to work an average of 24 hours per week. This sums up to a workload of 1728 hours for the entire project, but since the team was missing two persons most of the time the first three weeks, 200 hours were subtracted from the total amount of workload.

Workload during phases

See Table 2.1 for the estimated workload together with the actual used amount (excluding the last highly intensive week). Person estimation is team estimation divided on six. The same applies to person usage. The evaluation of this table can be found in Chapter 9.

Milestones

See Table 2.2 for a list of the project milestones. In addition to these planned dates, the actual completion of each milestone is also included. Five of the phases were not delivered for approval due to time constraints. More evaluation of this table can be found in Chapter 9.

⁶The ISO-8601 standard [47] is used to format dates in this report.

Milestone	Date planned	Date finished
Project directive document approved	2007-09-21	2007-09-25
Pre study document approved	2007-10-05	2007-10-16
Iteration 1 complete	2007-10-07	2007-10-16
Requirements frozen	2007-10-20	2007-10-20
Requirements specification document approved	2007-10-23	2007-10-30
Design document approved	2007-10-26	-
First document deliverance	2007-11-01	2007-11-01
Realization document and User documentation and installation guide document approved	2007-11-02	-
Testing document and Evaluation document approved	2007-11-09	-
Iteration 2 complete	2007-11-10	2007-11-12
Presentation document approved	2007-11-16	-
End of project	2007-11-22	2007-11-22

Table 2.2: Milestones for the project.

2.3 Concrete project work plan

This section describes how the team is planning to work through the span of the project. Due to a limited time frame, there is no room for substantial mistakes. By using an iterative method of working, it is possible to limit the loss of time if the project was to drift off in the wrong direction. The waterfall model [11], that could be a feasible work model, is based on finishing one phase before moving on to the next phase. This complicates possible unforeseen events, added requirements, and changes in the phases of the project, and it could easily lead to problems. The team's approach to the software development is more flexible. In particular, looking at the literature, it refers to the UP (Unified Software Development Process or simply Unified Process [79, 32]) model. UP transforms an object-oriented system development to an iterative and incremental process model. The basic principle is that every software project should be decomposed to controlled iterations that supply incremental versions of the product. Such increments could be builder increments (if they add some new use case to the product) or perfective increments (if they refine what already exists). An UP life cycle could be described as a sequence of iterations from the conception of the project until its end, see Figure 2.1 for a general example. Most of the iterations produce a releasable version of the product along with the available documentation. Usually, iterations are collected in stages that are characterized by the amount of effort spent on the several activities, such as requirements specification, design, and so on. Normally, four stages are identified: inception, elaboration, construction and transition. Inception is the starting phase of the project in which a feasibility study is performed. During the elaboration phase the project team is expected to capture a healthy majority of the system requirements. However, the primary goals of elaboration are to address known risk factors and to establish and validate the system architecture. The expected output is an architectural model intended as a baseline. In the construction

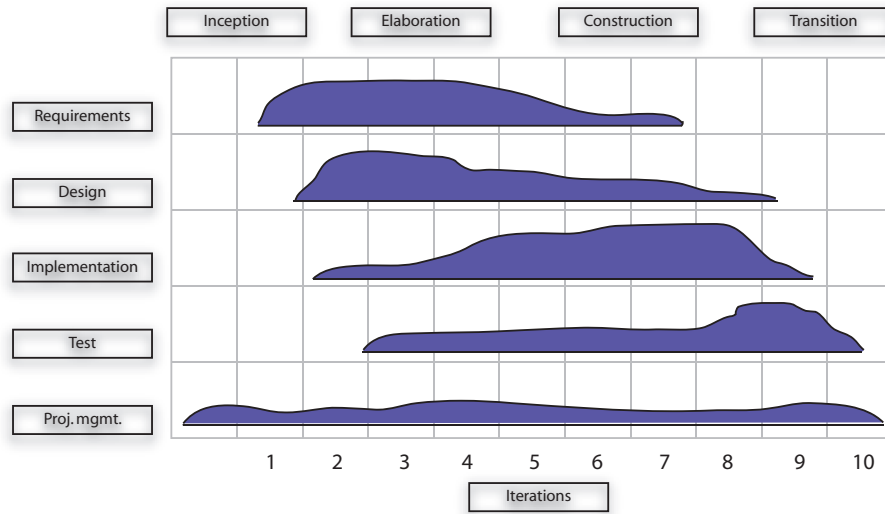


Figure 2.1: The Unified Process life cycle. Image adapted from [2].

phase, the product is developed enriching the baseline and putting much more effort on the testing activity. The final project phase is the transition phase. In this phase the system is deployed to the target users. Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several iterations within the transition phase. Furthermore UML, for its characteristics⁷, plays an important role within UP. It is used during almost the entire development process. Finally, based on the time available, the team has estimated that two iterations will be feasible. Even if the process is built up to include two iterations, the flow in the report will be as if there was only one. This is done to avoid the risk of introducing confusion to the reader.

2.3.1 Iterations and Phases

Some phases will be included in both iterations. These phases are requirements specification, design, implementation, and testing. The rest of the phases will only be done once. Some phases will start at the beginning of the project, and they will not end until the project is finished. See Table 2.3 for the planned structure. The Gantt diagram for the original estimated work is shown in Figure A.2, while the updated Gantt diagram after the first iteration is shown in Figure A.3. The detailed Gantt diagrams for some of the phases can be found in Appendix A.3.2.

⁷UML presents a widespread family of semi formal languages for specification, analysis, design and documentation of software artifacts.

Planning Preliminary studies	
Iteration one	Requirements specification Design Realization Testing Iteration evaluation
Iteration two	Requirements specification Design Realization Testing Iteration evaluation
User documentation and installation guide Project evaluation Presentation	

Table 2.3: Iterations and phases during the project.

Activity	Responsible
Phase responsible	Kris-Mikael
Project mandate	Kris-Mikael
Concrete project work plan	Kris-Mikael
Risk analysis	Eilev
Organization	Kris-Mikael
Templates and standards	Kris-Mikael
Version control	Kris-Mikael
Sourceforge account	Eilev
Documentation of project work	Kris-Mikael
Quality assurance	Kris-Mikael
Overall test plan	Eilev

Table 2.4: Activities during the planning phase.

2.3.2 Description of phases

Planning

The planning phase is the first part of the project. The written result of the planning phase is the project plan from this chapter. This document is updated regularly as the project develops. Table 2.4 shows which tasks the planning phase consists of, as well as who is responsible for them.

Preliminary studies

The process of gathering information takes place in this phase. Since the term “security modeling” was more or less vague for the entire team at the start of the project, this is a particularly vital phase of the project. Results from this phase are general knowledge about the problem that needs to be solved, as well as how the current solutions can be used. The term “security modeling” was discovered to be quite general with a large amount of material available. Each

Activity	Responsible
Phase responsible	Ketil
Threats and attacks, methods and modeling types	Ketil
Software vulnerability and cause modeling	Egil
Countermeasures	Eilev
Sources of information	Kris-Mikael

Table 2.5: Activities during the preliminary studies phase.

Activity	Responsible
Phase responsible	Daniele
Overall domain description	Daniele
Description of functional and non-functional requirements	Daniele
Software license research	Kris-Mikael

Table 2.6: Activities during the requirements specification phase.

member of the team got sub topics to learn, and a short presentation of these topics had to be given to the rest of the team. This was done in order to give everyone a decent understanding of all topics. See Table 2.5 for responsibilities during this phase.

Requirement specification

In the requirement specification phase, the goal is to clarify the customer's needs for the project. The requirements will mainly be split into non-functional and functional requirements, with the responsibilities distributed as shown in Table 2.6.

Design

The goal of this phase is to get a complete outline of the application to be implemented. Decisions, such as choice of frameworks, are made during this phase together with elucidation of them and their dependencies. The responsibilities for this phase are as shown in Table 2.7.

Realization

In the realization phase, the development of the actual application, based on the documents from the requirements and design phase, is implemented. It is important to notice that neither the requirements nor design needs to be frozen at this stage, since any problems discovered means stepping back and review (and perhaps make alterations of) the earlier work done. In this phase, a large

Activity	Responsible
Phase responsible	Daniele
Elucidations regarding the chosen frameworks	Eirik

Table 2.7: Activities during the design phase.

Activity	Responsible
Phase responsible	Eirik

Table 2.8: Phase responsible for the realization phase.

Activity	Responsible
Phase responsible	Eilev
Carry out system tests	Ketil
Fix errors detected by tests	Eirik
Test documentation	Eilev
Validation and verification of test results	Eilev

Table 2.9: Activities during the testing phase.

amount of documentation is done. Both code and the application in itself (i.e. User documentation) are documented. The responsibilities for this phase are shown in Table 2.8.

Testing

User and installation documentation is developed during this phase. There is a focus on continuous testing of the application, as the project evolves, in order to ensure quality of the product at all stages. The phase consists of two iterations which overlap the last part of the realization iterations. Table 2.9 shows the tasks and the responsibilities that exist during this phase.

User and installation guide

In this phase, the installation guide as well as the manuals for the implemented application are written. The responsibilities for this phase are shown in Table 2.10.

Project evaluation

During this phase, experiences from the project is documented. A brief evaluation of the course is also included in this phase. The responsibilities are distributed as shown in Table 2.11.

Presentation and demonstration

In addition to the delivery of the final report, this phase includes the final presentation and demonstration of the project. The phase is comprised of the project presentation, presentation of the reached goals and demonstration of the implemented application. In advance of the presentation, Chapter 3 and

Activity	Responsible
Phase responsible	Ketil

Table 2.10: Phase responsible for the user and installation guide phase.

Activity	Responsible
Phase responsible	Kris-Mikael
Gathering of opinions, views, thoughts and expressions from the team members	Kris-Mikael

Table 2.11: Activities during the project evaluation phase.

Activity	Responsible
Phase responsible	Ketil
Invitation	Kris-Mikael
Presentation	Egil
Hand outs	Kris-Mikael

Table 2.12: Activities during the presentation and demonstration phase.

Chapter 4, together with an invitation, will be sent to the external examiner, so he or she can get a chance to prepare properly. The responsibilities are distributed as shown in Table 2.12.

Conclusion

The conclusion phase contains a discussion of the results of the project. This includes an evaluation of the software developed during the project and important results regarding software security modeling. See Table 2.13 for the responsibility distribution.

2.4 Organization

In this section, the formal roles in the project are defined and given responsibilities. A description of the team organization is also included.

2.4.1 Organization chart

This section presents the division of responsibilities in the form of a chart. Only the highest responsibility areas are listed, so it is a high level structure of the team. The responsibilities are distributed equally among the members of the team, but a superior leader is chosen to take care of possible conflicts. The leader should also manage the rest of the team's work tasks. The organization chart is shown in Figure 2.2 with descriptions of the roles in Section 2.4.2. Note that the project leader is listed as the only person having contact with the customer. This, of course, does not apply to the weekly meetings. The following list summarizes the organization chart in plain text.

Activity	Responsible
Phase responsible	Egil

Table 2.13: Phase responsible for the conclusion phase.

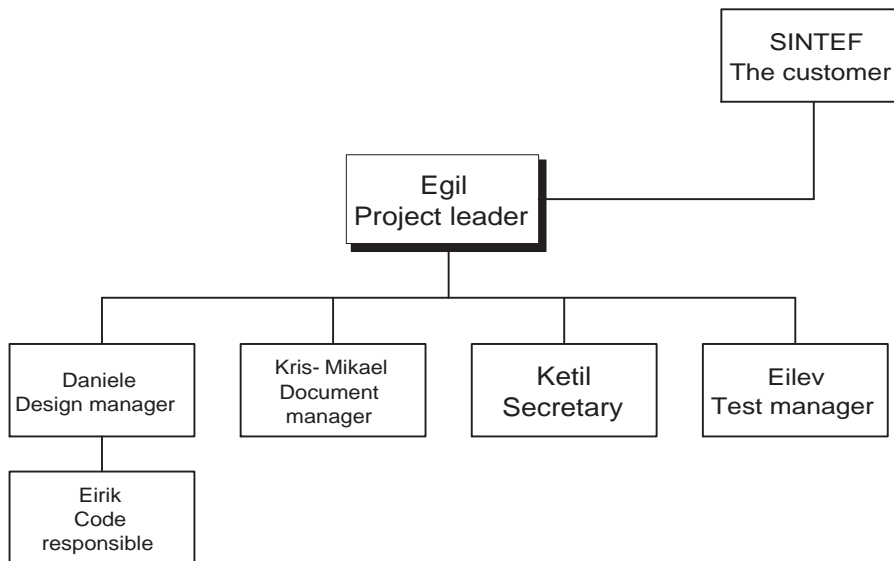


Figure 2.2: The organization chart for the project.

- Project leader: Egil Trygve Baadshaug
- Design manager: Daniele Spampinato
- Code responsible: Eirik Benum Reksten
- Document manager: Kris-Mikael Krister
- Secretary: Ketil Sandanger Velle
- Test manager: Eilev Hagen

2.4.2 Project roles

In this section, the different roles in the project are presented, in addition to their main tasks.

Project leader

The project leader has the final word in discussions where the team is not able to reach an agreement. He is the chairman of the meetings, and he has the responsibility to send out notices in advance of the meetings. The project leader is to manage the team's overall phase responsibilities and ensure it is done properly. If there is a need for customer contact besides sending notices and minutes of meetings, the project leader, and only him, should do this.

Design manager

The design manager has the main responsibility for the construction and planning of the implemented system. He will know at all times how the different components in the system work together and affect each other.

Code responsible

The person responsible for the code controls the algorithms implemented in the system, and divides the work of realization phase and distributes it to the rest of the team.

Document manager

The person responsible for the documents will structure the layout of the report and arrange consistency within the style of writing throughout the report. He will also have the responsibility to control spelling and grammar, and assure that the document has a clear overall structure. He has the responsibility to see to that all documents are updated with the team's latest work all time. In addition to this, he needs to instruct the team members to how the file structure is defined and how to use the `svn` report repository.

Secretary

The secretary writes and sends out minutes after each customer and supervisor meeting. Essential decisions done by the team should be written and distributed by email by the person having this role. He will also write the status reports every week. It is also the secretary's task to store these minutes digitally on the shared network location.

Phase responsible

Each phase has a responsible for the progress, structure and completion of the phase, as well as the corresponding chapter in the report. Before starting a phase, the responsible *must* understand which activities to include. He must delegate the activities and instruct the team properly. The phase responsible will also create the introduction and summarize the phase document.

Test manager

The test manager is responsible for setting up the test plan, and he must ensure that the application complies and satisfies the customer requirements.

The quality assurance team

In addition to individual quality assurance on each section, the persons having the roles project leader, document manager, and test manager are to make sure that the quality assurance is not neglected.

2.5 Templates and standards

This section describes the templates and standards used throughout the project.

2.5.1 Templates

Notice of meetings

Notices of meetings are sent before each meeting according to agreements made in Section 2.8.4.

Two different templates, depending on who is meant to receive the notice, are used. The templates for customer meetings and supervisor meetings are shown in Appendix A.4.1 and Appendix A.4.3, respectively.

Minutes

Minutes are written after each meeting and follows the templates shown in Appendix A.4.2 and Appendix A.4.4. They are sent to the customer and supervisors, respectively.

Status reporting

Status reports follows the template found in Appendix A.4.5. The status report is written during the internal meetings and is included on all the external meetings.

2.5.2 Standards

File organization

ITEA's⁸ file servers are used for the digital storage. All team members have access to a shared folder on this area, and the files can be changed and rewritten by the whole team as needed. At the root of the shared folder, literature (documents, papers and articles) in the field of security modeling can be found in a directory with the same name. The project documentation, as explained in Section 2.7, can also be found as directories at the root. In all the mentioned folders, there are rules for creating and using files as explained in Section 2.5.2. The teams central repository, as mentioned in Section 2.6, is located in the “svn” folder. Miscellaneous files used by the team should be stored in a suitable folder under the “CommonPlayground” directory. A tree structure of the directories and *checked out* report are shown in Listing 2.1 and Listing 2.2, respectively. Java code is not shown due to the enormous amount of files and directories, but the structure is explained in Chapter 6.

```
|-- CommonPlayground
|   |-- Daniele
|   |-- Egil
|   |-- Eilev
|   |   |-- SmartDraw files
|   |   '-- sendt to svn
|   |-- Eirik
|   |-- KM
|   |   |-- Icons
|   |   |-- PSD
|   |   '-- old_png_files
```

⁸The IT section at NTNU

```

|-- Ketil
|   |-- AttackTree+
|   |-- smartdraw
|       |-- eps
|       |-- pdf
|       |-- png
|-- Tien
|   |-- diagram
-- Literature
|   |-- Security Pattern documents from Per
|   |-- swsec-papers
-- Meetings
|   |-- customer
|   |-- internal
|   |-- supervisor
-- Phases
|   |-- 1_Project-directive
|   |   |-- old-sdrs
|   |-- 2_Prestudy
|   |   |-- SmartDraw files
|   |-- 5_Design
|   |   |-- Ikoner
|   |   |-- SVGs
|   |-- 6_testing
|   |-- 8_Evaluation
-- WorkDocumentation
|   |-- StatusReports
-- public_html
-- svn
|   |-- report
|   |   |-- conf
|   |   |-- dav
|   |   |-- db
|   |       |-- revprops
|   |       |-- revs
|   |       |-- transactions
|   |-- hooks
|   |-- locks
|-- scripts

```

Listing 2.1: File structure.

```

-- 0_bibliography
-- 0_images
-- 0_other-tex-files
-- 10_appendix
-- 1_planning
-- 2_prestudy
-- 3_requirements_specification
-- 4_construction
-- 5_programming
-- 6_testing
-- 7_userinstallationguide

```

```
|-- 8_evaluation
'-- 9_presentation
```

Listing 2.2: Report folder structure in checked out form.

Naming of files

Phase documents are split into two levels of classification, where “Chapter” is the top level followed by “Section”. In this way, it is easier to get an overview on the document structure, and it is possible to work on the same phase document if this is desired. There is no spacing in filenames. White spaces are replaced with an underscore. Each phase document follows the \LaTeX standard and is named using the following form. File tree structure for the report is shown in Listing A.1 in Appendix A.

```
<phase>_<section>_<section-name>.tex
```

The files will be placed in directories named symboling the phases on the following form:

```
<phase>_<phase-name>
```

Coding standards

As the application is to be built upon the eclipse framework, it will adhere to the same coding conventions as used by the Eclipse project. This means following the standard coding conventions given by Sun Microsystems [86]. More information on these standards are given in Chapter 6.

2.6 Version control

The team will need a way to manage source code and documents from a globally shared area. Members of the team should also be able to work on the same file at the same time, without losing any changes done to the file. It is also favorable to have a repository storing old revisions of files in case of requests for earlier versions. By using a file system without tweaking, this is not feasible. However, both Concurrent Version System (CVS) [52] and Subversion (SVN) [7] supports the desired functions. The two programs are similar, but there are several small, but still important, differences. As an example, the handling of file renaming, moving and deletion is better done by SVN than CVS. The revision handling is also more conveniently implemented in SVN, so for these simple reasons, SVN is chosen for the central repository used by the team. Since NTNU’s login servers will be used, the files will be backed up every night resulting in less resources from the team to manage copies of the written work. There are some guidelines on using the repository. They are all listed below.

- Always do an *svn update* before editing files.
- Do not run *svn commit* if the code cannot be compiled.
- Always include text explaining what kind of changes are done.

2.7 Documentation of project work

2.7.1 Project meetings

There will be three kinds of scheduled weekly meetings. Internal meetings, meetings with the supervisors, and meetings with the customer. Customer, supervisor, and one of the internal meetings will be held in R41.⁹

Internal meetings

Internal meetings will be held every Tuesdays 16:00 - 17:00 in R41 and Mondays from 09:00 - 12:00 at the P15 building. In addition to this, meetings will be held whenever the team finds it meaningful or useful. Internal meetings will only be noticed by email if the time varies from the fixed scheduled ones.

Meetings with the supervisors

Supervisor meetings will be held every Tuesday 15:15 - 16:00.

Meetings with the customer

Meetings with the customer will be held every Tuesday 14:15 - 15:00.

2.7.2 Internal reporting

Work amount, activities and project status will be discussed and documented during the internal meeting on Tuesday.

2.7.3 Status reporting

Together with the notice of the next meeting and minute from the previous meeting, the status report is sent to the supervisors. This will be done no later than 12:00 on Mondays.

2.7.4 Document working hours

All members of the team are responsible for updating their worked hours on the shared Google spreadsheet mentioned in Section 2.2.8.

2.8 Quality assurance

This section contains information about how the team will ensure the best possible quality of documentation and deliveries throughout the project.

2.8.1 Response times

The following list summarizes agreements made with the customer. However, response times will vary because the representatives from SINTEF are not available at fixed times.

⁹Booked group room in Realfagsbygget at NTNU, Gløshaugen.

- Response on minutes of customer meeting: Feedback will be sent within 24 hours.
- Approval of phase documents: Feedback within 48 - 72 hours.
- Answers to general questions: 24 - 48 hours, depending on the question.
- Feedback or response on pre determined, agreed deliveries: 24 - 48 hours.

2.8.2 Routines for producing good quality

Each paragraph in the report will be written as good as possible before uploaded to the repository. Implementation code will be documented at the same time as it is written, leading to better code comprehension by the other team members.

2.8.3 Routines for approval of phase documents

Each phase document will be internally approved by the whole team, with the project leader responsible for taking suitable actions if there are disagreements. The supervisors will then receive a copy by email, with a request for approval. If the document is approved, it will be sent to the customer and added to the agenda for the next customer meeting.

2.8.4 Calling for a meeting at Tuesdays

Notices for meetings will be written using a predefined template as listed in Appendix [A.4.1](#) and [A.4.3](#) for customer and supervisor meetings, respectively. A copy of each notice is sent to the team mailing list to keep team members up to date.

2.8.5 Minutes of meetings

The minutes will be sent to the interested parties after each meeting. Minutes from customer meetings are sent for approval with a copy to the supervisors. See Appendix [A.4.2](#) and [A.4.4](#) for templates regarding the minutes.

2.8.6 Ground rules for the team

See Appendix [A.5](#) for the common ground rules of the team.

2.8.7 Risk analysis

The risk analysis and the corresponding risk matrix can be found in Appendix [A.3.3](#). This will grow continuously during the project, due to unforeseen risks. Updates are done at least once every month.

2.9 Overall test plan

The overall test plan can be found in Chapter [7](#).

Chapter 3

Preliminary studies

3.1 Introduction

This chapter contains background information about security modeling; how it is done today, what the desired state is, and how this can be reached by the SeaMonster project. The chapter also lists alternative solutions to the stated problem, an evaluation of them, and finally it concludes with the chosen solution. This process is called preliminary studies, but will henceforth be referred to as “pre-studies”.

3.1.1 Purpose

The purpose of the pre-studies is to get an overview of existing solutions of security modeling. This includes modeling methods, vulnerability databases, modeling applications, and previous studies. The solutions are described and evaluated. Based on the evaluation, a conclusion is made. It describes how the pre-studies affect the further work of the project.

3.1.2 Background

Today, computers are used to do everything from storing information to controlling essential systems, and the amount is increasing year by year. This increased usage requires more software to be made now than ever before, introducing a large number of new vulnerabilities. According to The National Vulnerability Database¹, the amount of discovered vulnerabilities is over 30 times higher today compared to ten years ago, and the number is continuously rising. Since nearly all computers in some way are connected to the Internet, many of these vulnerabilities can easily be exploited from remote locations [4]. In order to assist developers in securing new software, a number of methods, applications and databases have been established. These are described in detail in the following sections.

¹An interface to other security related databases discussed later in this chapter. See [97] for more information about this instance.

3.2 Business requirements

Business requirements are general demands from the customer. They state the overall goals of the project, as opposed to the software requirements, which state how these goals are to be reached. The relationship between software and business requirements, makes it possible to verify whether or not the requirements have been fulfilled. The following is a list of the business requirements stated by the customer.

BR.01 Help move the security assessment of a system to an earlier phase than usually done today.

BR.02 Provide a visual understanding of several concepts in the field of software security.

BR.03 Standardize used notation for creating models.

BR.04 Allow application interoperability.

BR.05 Bridge the gap between security experts and programmers.

BR.06 Support the Open Source community, utilize its capabilities.

3.3 Current situation

Today, there are few software developers who follow the secure coding best practices [43], even though most vulnerabilities are caused by the same well known mistakes [20]. The continuously increasing number of vulnerabilities, especially concerning their consequences, are now forcing developers to pay more attention to this problem. Computers are to a great extent used to handle sensitive information and critical processes. This can make them highly dependent upon complex software. When the software is more complex, it is usually more open for vulnerabilities leading to possible attacks [8]. When crucial systems highly depend upon software, even small mistakes (both *bugs* and *flaws*) can have enormous consequences. Take for instance the clock drift error in Dhahran that caused the loss of 28 US Army soldiers [100], or the destroyed Mars Climate Orbiter [48] due to an entry of momentum data in imperial units instead of the metric system.

Even though the need for security in software increases, released software does not seem have any noticeable improvements [78]. In order to assist this, a few online vulnerability databases have been established. The National Vulnerability Database [97] and the Open Source Vulnerability Database [74] are examples of such databases containing information, countermeasures and possible threats². They show an enormous growth of reported security issues in software over the last few years. See Figure 3.1 for the yearly increase in discovered software flaws from 1997 until 2007-09-26 [93].

Designing secure software is difficult, and it is often not given enough consideration during development. This leads to software being released containing

² The resources mentioned are explained later on in this chapter.

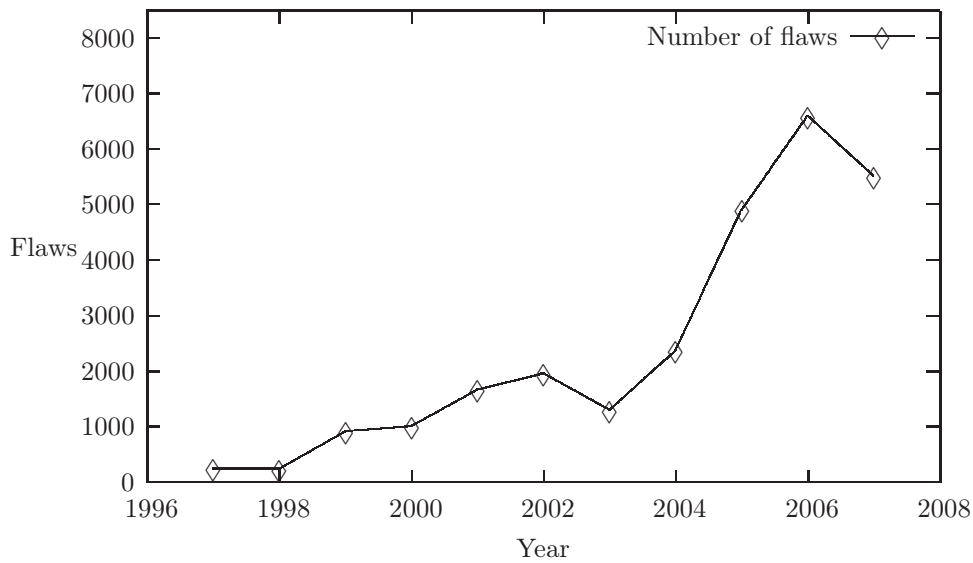


Figure 3.1: Increase in discovered software flaws.

vulnerabilities which could (easily) have been avoided. The normal way to handle this is by making security patches after the vulnerabilities are detected. This is not the optimal way of handling security.

As said before, the most common vulnerabilities come from well known mistakes, and they could have been prevented before the software release without too much effort. One problem is that there is no standardized way of doing this. At the moment, no decent solution for handling security in software development exists, and currently no purpose-built security application is available [8]. It is estimated that late correction of requirement error costs more than 200 times as much as early correction [55]. Software developers who are experts in their field, are not necessarily experts in security, and security experts are not necessary experts in software development [8]. High costs in employee training are claimed to be a important reason for lack of security knowledge among developers [55]. There is also a general lack of security experts [78], so the constantly increasing amount of software developers can not have a security expert to complement their security skills. Better modeling of security in early phases of software projects may be a solution for many of the problems in implementation. Security modeling is often done using general purpose drawing tools [8]. This leads to different standards, extra work, and models that can result in misinterpretations. Microsoft Visio [64] and SmartDraw [84] are examples of tools used for modeling, but these programs are quite complex. They are made for covering a wide spectre of models, and they include a large collection of functions. This might cause more confusion than good for a software developer who only needs a simple tool to model security issues.

Microsoft has released a freeware threat modeling tool called “Threat Model Tool” [63]. Microsoft’s tool includes only one way of modeling security, and it is only a textual representation. This makes the tool too simple for a complete se-

curity analysis. There also exists another free modeling tool named Coras [83]. The Coras project was an EU-funded project that was completed September 2003. It was made by a range of eleven institutions from four European countries, including SINTEF and Telenor [92]. As Coras is used for model-based risk analysis of security critical systems, and it has a slightly irregular syntax, the team did not see the tool fit to use as a basis for further development.

There also exist some commercial products. SecurITree by Amenaza [3], for instance, is a tool for modeling attack trees. It includes automatic scans through an attack tree to identify how possible attacks could occur. Isograph's AttackTree+ [49] is another example of a commercial product that models attack trees. See Appendix B for screenshots of the mentioned programs. A modeling tool based on UMLsec [55] also exists, but none of these tools are available for free testing.

With SeaMonster, the focus will be on helping developers and security experts to easily model security in any phase of software development, also the early ones. This will help them get a complete overview over the vulnerabilities they face, and also make it easier to eliminate or avoid these at the implementation stage. For this reason, software will (hopefully) be released with fewer vulnerabilities, and this again will result in saved money and less need for issuing patches.

3.4 Sources of information

There are several sites established on the web containing information about different digital security related issues. While some of them are more or less connected to each other sharing the same standards and methods, others use their own way of reporting and distributing the information. This section describes some of the vulnerability information sources that the SeaMonster project can benefit from.

3.4.1 The Common Vulnerabilities and Exposures (CVE) dictionary

The CVE [67] dictionary is a large list of publicly known information about security vulnerabilities and *exposures* distributed freely on the web. The list was created to gather and standardize common problems found in programs. The items in the list are often connected to software flaws in specific programs. The web site offers three ways to obtain the stored data; view the dictionary as an HTML-formatted page, download the dictionary in various formats, and search the dictionary using keywords and CVE-Identifiers. Such an identifier includes the unique identifier name, status of the identifier, short description, and possible references. See Figure 3.2 for an example regarding a general problem called "Algorithmic Complexity".

The information stored in the dictionary is designed to be as compact as possible, and it does not contain information such as risk management or fixes to the problems. Also, there are no technical information beyond brief descriptions or references to related vulnerability reports. Since the main target group is security experts, the dictionary assumes a certain level of knowledge from the reader.

CVE-ID	
CVE-2005-3595 (under review)	Learn more at National Vulnerability Database (NVD) • Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings
Description	
By default Microsoft Windows XP Home Edition installs with a blank password for the Administrator account, which allows remote attackers to gain control of the computer.	
References	
<p>Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.</p> <ul style="list-style-type: none"> • BUGTRAQ :20040915 Vulnerability in IBM Windows XP: default hidden Administrator account allows local Administrator access • URL:http://www.securityfocus.com/archive/1/375335 • BUGTRAQ :20051108 Re: Hidden accounts on sony vaio laptops • URL:http://m.ars-technica.com/?b=bugtraq&m=113147602208113&w=2 • CONFIRM :http://www.microsoft.com/windowsxp/using/setup/getstarted/installxp.mpx • BID :11199 • URL:http://www.securityfocus.com/bid/11199/ • XF:idm-w-xp-default-admin (17412) • URL:http://xforce.ibm.net/xforce/xftb/17412 	
Status	
Candidate	This CVE Identifier has "Candidate" status and must be reviewed and accepted by the CVE Editorial Board before it can be updated to official "Entry" status on the CVE List. It may be modified or even rejected in the future.
Phase	
Assigned (20051116)	
Votes	
Comments	
Candidate assigned on 20051116 and proposed on N/A	

Figure 3.2: Example of an item from the CVE list. Screenshot taken from [67].

Any developer can create an application and get it certified as CVE compatible. This means that the application guarantees functionality such as displaying CVE-identifiers, searching based on CVE-identifiers, and the possibility to map vulnerabilities and exposures produced in the application to related CVE-identifiers. This listing is somewhat shortened, but it includes the most important characteristics of the “CVE-compatible” title. CVE is designed to allow vulnerability databases to be linked together, and the content of the list are issues with a direct relation to the modeling functions of this project implemented application. For these reasons, the CVE database is highly relevant for the SeaMonster project. However, it is out of the project scope to apply for the CVE certification.

3.4.2 Common Weakness Enumeration (CWE) database

CWE is a community-developed formal list of common software weaknesses. It was created to address problems regarding obtaining information about weaknesses in software code. [68]. The main target of the CWE initiative is software developers and security experts. It uses a common language for describing weaknesses in software architecture, design, and code. As opposed to CVE, CWE covers more generic problem and has no focus on specific applications or programs. Each item in the list is uniquely defined with a “CWE ID”, a title, a brief description of the case, consequences, and observed examples of the vulnerability. See Figure 3.3 for an example. CWE includes most of the vulnerabilities and exposures from the CVE list, but it also includes detailed structure from industry and academic sources. An application can be certified as “CWE-compatible”, which is similar to the CVE-compatible title. The descriptions of the common software weaknesses are the most important part of CWE in regards to the SeaMonster project, since these descriptions can be mapped to situations modeled with the application.

Algorithmic Complexity	
CWE ID	407
Description	An algorithm in a product has an inefficient worst-case computational complexity that can be triggered by an attacker, typically using crafted manipulations that ensure that the worst case is being reached. Note: the typical consequence is CPU consumption, but memory consumption and consumption of other resources can also occur. Note: similar issues can occur in cryptography.
Common Consequences	The typical consequence is CPU consumption, but memory consumption and consumption of other resources can also occur.
Observed Examples	<p>CVE-2003-0244 - CPU consumption via inputs that cause many hash table collisions.</p> <p>CVE-2003-0364 - CPU consumption via inputs that cause many hash table collisions.</p> <p>CVE-2002-1203 - Product performs unnecessary processing before dropping an invalid packet.</p> <p>CVE-2001-1501 - CPU and memory consumption using many wildcards.</p> <p>CVE-2004-2527 - Product allows attackers to cause multiple copies of a program to be loaded more quickly than the program can detect that other copies are running, then exit. This type of error should probably have its own category, where teardown takes more time than initialization.</p> <p>CVE-2005-1792 - Memory leak by performing actions faster than the software can clear them.</p>
Context Notes	Similar issues can occur in cryptography.
References	Crosby and Wallach. "Algorithmic Complexity Attacks". < http://www.cs.rice.edu/~crosby/hash/CrosbyWallach_UseKSec2003/index.htm >.
Node Relationships	Child of - Asymmetric resource consumption (amplification) (405)
Source Taxonomies	PLOVER - Algorithmic Complexity
Applicable Platforms	<p>C</p> <p>C++</p> <p>Java</p> <p>.NET</p>

Figure 3.3: Example of an item from the CWE list. Screenshot taken from [68].

3.4.3 Common Configuration Enumeration (CCE)

The Common Configuration Enumeration list [66] is formatted as a spreadsheet (Excel [62]) document containing about 1000 entries. The list provides unique identifiers to security issues regarding system configuration. It can also be used to assist software based configuration of a system. If a “workaround” section of the countermeasure view is added or needed in a later implementation of the SeaMonster application, CCE could be used as a source of information. However, this is outside of the current scope. See Table 3.1 for a cropped CCE entry example.

CCE Id	CCE Definition	CCE Parameters	CCE Technical Mechanisms
CCE-100	The minimum password length policy should meet minimum requirements.	(1) number of days (1) defined by Local or Group Policy	All passwords are at least 8 characters long (minimum).

Table 3.1: Cropped CCE entry example of minimum password length in a system.

3.4.4 Common Platform Enumeration (CPE)

CPE is a structured naming scheme for software systems, platforms, and packages [69]. CPE uses a name format based on the Uniform Resource Identifier (URI)³ syntax, including standardized abbreviations on different type of software and hardware architecture. The syntax for a CPE-name is structured as

```
cpe:{/part}:{vendor}:{product}:{version}:{update}:{edition}:{language}
```

The different parts are more or less self explanatory, but see [12] for a detailed explanation. As an example, the “Red Hat Enterprise Linux 3 Advanced Server” and the “apache web server version 2.0.52” are named in the following form.

```
cpe:/o:redhat:enterprise_linux:3::as  
cpe:/a:apache:httpd:2.0.52
```

The CPE naming structure is used in the field of explaining software security, and since SeaMonster is a part of this topic, it will use the CPE standard to represent other pieces of software when needed.

3.4.5 Open Vulnerability and Assessment Language (OVAL)

OVAL is an information security standard used to standardize the transfer of its information across security tools and services [70]. OVAL is using a language to encode system details, and bases the content on information from the CVE list. OVAL has also published a free tool for interpreting the Extensible Markup Language (XML)⁴ data files available [71]. The application is able to scan the computer and present vulnerabilities and installed programs as shown in Table 3.2. Notice the usage of CPE URI’s and the CVE id in the third column. It is important to have the knowledge of the existence of OVAL when working with security databases, but OVAL, or the usage of OVAL, will not be included in the implemented version of SeaMonster.

3.4.6 Security Content Automation Protocol (SCAP)

SCAP is a method for using standards (CVE, CCE, CVSS⁵, CPE, XCCDF⁶ and OVAL) to do automated vulnerability management and measurement. [95] SCAP also defines how these standards are combined, and a goal is to standardize how computers communicate vulnerability information, and what kind of information will be exchanged. SCAP is based on open standards distributed as similar textual formats (XML), the mapping can be done without any type conversions. See Figure 3.4 for an example of a simple mapping between XCCDF and OVAL, and notice the “Platform” is written as a CPE URI and “Check 1” and “Check 2” can be linked to a CCE- and CVE-id, respectively. Table 3.1 and Figure 3.2 include the content of these references. SCAP is not directly

³A compact string of characters used to identify or name a resource.

⁴General purpose markup language used for sharing and encoding structured data across systems.

⁵CVSS is a standard for assessing the severity of vulnerabilities in software security [96].

⁶XCCDF is an XML formatted list specifying security checklists, benchmarks and configuration documentation [94].

OVAL ID	Class	Reference ID	Title
oval:def:521	i	<code>cpe:/o:microsoft:windows-nt:xp:sp2</code>	Microsoft Windows XP SP2 is installed
oval:def:105	i	<code>cpe:/o:microsoft:windows-nt:xp</code>	Microsoft Windows XP is installed
oval:def:1873	v	CVE-2007-0043	.NET JIT Compiler Vulnerability
oval:def:310	i	<code>cpe:/a:microsoft:.net_framework:2.0</code>	Microsoft .NET Framework 2.0 is installed
oval:def:157	v	CVE-2007-0025	MFC Memory Corruption Vulnerability
oval:def:2070	v	CVE-2007-0042	ASP.NET Null Byte Termination Vulnerability
oval:def:2093	v	CVE-2007-0041	.NET PE Loader Vulnerability
oval:def:3556	v	CVE-2004-0847	.NET Framework v1.1 Security Bypass

Table 3.2: Cropped result from the OVAL Interpreter application ran on a WinXP machine. The content of column two is either a vulnerability in the system (v), or marked as an installed application (i).

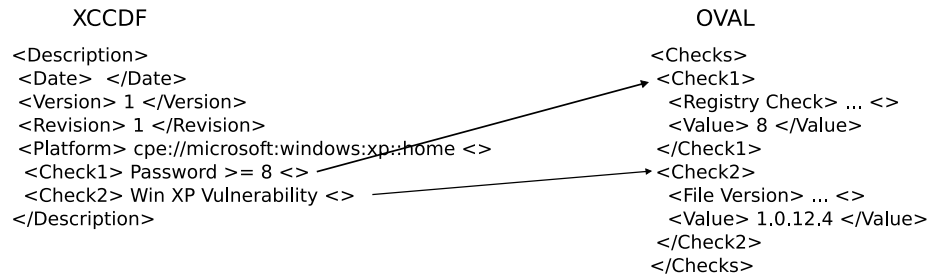


Figure 3.4: Mapping example between XCCDF and OVAL. Example adapted from [10].

related for the scope of this project. However, since SeaMonster include security aspects linkage as explained in Section 3.5, it is still important to have the possible mapping between application descriptions, vulnerabilities and counter-measures in mind when creating this kind of modeling software. A textual or graphical function regarding this mapping will definitively extend SeaMonsters usability. It is also possible this will lighten the way of expressing a vulnerability to an even greater extent than the current scope will, so a later release of SeaMonster can gain possible benefits for using a protocol like SCAP actively.

3.4.7 CERT Coordination Center (CERT/CC)

CERT/CC, the most widely known group within the CERT Program [14], is a major coordination center in dealing with internet security issues. Their main focus is on addressing existing and potential software threats, and notify the system administrators and software vendors in conjunction to the threat.

CERT/CC coordinates with the incident response teams around the globe, and notifies them when threats are discovered. CERT tries to reduce new vulnerabilities by secure coding effort, but also to reduce the threat posed by already existing ones by using vulnerability remediation.

Vulnerability remediation CERT states that there are often ways to reduce the risk without patching the software [17]. By changing the system's configuration or by using certain workarounds, the vulnerability might be closed and useless for possible attackers. Analyzes and tests on the vulnerability needs to be done to ensure the workaround is valid.

Secure coding Common programming errors are often the cause of insecure software [15]. Therefore, to prevent new vulnerabilities from happening, CERT has established some practices that can lead to more secure software deployed without having the need for later patches and fixes. Concrete rules and standards for secure coding in C and C++ are listed and discussed on the CERT Secure Coding Standards web page [16].

Artifact analysis The artifact analysis team from CERT tries to understand how *malicious code* works by examine and sometimes *reverse-engineer* the software. In this way, trends and patterns can be identified to reveal potential threats in other written code.

Ideas from the secure coding part of CERT will be included in the background for developing the SeaMonster application. In addition to the modeling part of the application, users should also apply standards to avoid common mistakes as a regular programming habit. Also, users of the application can benefit by knowing the vulnerability remediation since workarounds can be just as important as fixing the problem on an early stage of the correction process.

3.4.8 BugTraq

BugTraq is an open-for-all mailing list with discussions and detailed information about computer vulnerabilities and exploits, where and how they can be found, taken advantage of, and how to fix them [81]. It was created by Scott Chasin in 1993, but is now run by SecurityFocus [80], owned by one of the leading software security companies in the world, Symantec Corporation [91]. The mailing list is based on the security philosophy "Full Disclosure", which means that a system must handle examinations and reviews on all levels to be truly secure. The vulnerability details should also be available for the public crowd. This could lead to a great number of individuals going through potential vulnerabilities, forcing software vendors to issue fixes quickly. The goal of spreading information about the faults is to help other people to avoid them. It also causes an awareness of similar vulnerabilities in other systems or programs. The problem with such a philosophy is that attackers also get the information quickly, and can take advantage of the vulnerabilities.

BugTraq is moderated, but emails are not validated. Therefore, there is no guarantee on the correctness of the information. With 27.000 email addresses on the list, and 5 - 30 new emails every day, a lot of information is spread swift and brutal. The subjects discussed span most of the security related topics, but network related vulnerabilities on application and operating system levels

are the main focus for the list. This mailing list will not be integrated or used with SeaMonster, as it has no direct relevance to the specified requirements. However, BugTraq can be, and often is, the first source of information about discovered vulnerabilities. Therefore, SeaMonster could gain possible benefits by using this source, but it will not be included in the release of this project.

3.5 Security modeling techniques

There have been developed several techniques to support security modeling and these could be divided into three groups, or “views”, of security modeling namely “Vulnerability and cause modeling”, “Threats and attacks” and “Countermeasures”. The following three subsections address each of them in turn.

3.5.1 First view - Vulnerability and cause modeling

Vulnerability and cause modeling is a generic term used to describe formalisms and methods in security modeling, which focuses on the causes of vulnerabilities. The goal of these formalisms is to get an in depth understanding of what causes the vulnerabilities, so that they can be prevented. In this section, some commonly used formalisms and methods will be described.

Root Cause Analysis (RCA)⁷ is a collective term used to describe a wide range of approaches, tools, and techniques, which are used to uncover causes of a problem [6]. RCA is based on a belief that solving a problem is best done by correcting or eliminating its roots causes, as opposed to merely addressing the immediately obvious symptoms. It differs from troubleshooting and problem solving in that these disciplines typically seek solutions to specific problems, whereas RCA is directed at underlying issues. To get the best result when analyzing, several methods should be used. RCA has three general principles:

- Aiming corrective measures at root causes is more effective than merely treating the symptoms of a problem.
- To be effective, RCA must be performed systematically, and conclusions must be backed up by evidence.
- There is usually more than one root cause for any given problem.

The five whys

The five whys [6] is one of the methods used in root cause analysis. The idea behind it is that the root cause of a problem can be found by repeatedly asking the question “why”. For every answer, you get one step closer to the root cause. The number of why-iterations needed depends on the problem, but as a rule of thumb, the method requires five iterations. As an example, consider the problem statement: “You get up in the morning and try to turn on the light in the kitchen, but it does not work.”

- Why is there no light? Because there is no electricity.
- Why is there no electricity? Because I didn’t pay the electricity bill.

⁷Method used to uncover specific causes of a problem.

- Why didn't you pay the bill? Because I had no money.
- Why didn't you have any money? Because I lost it all playing blackjack in the casino.
- Why did you lose your money playing blackjack? Because I played too aggressively.

The advantage of the five whys is that it goes deep into the underlying causes of the problem. It is also easy to use. Disadvantages are that the entire analysis can get thrown off if a mistake is made when answering just one of the “why” questions, and that the method offers no visual representation of the results.

Fault tree analysis

Fault tree analysis (FTA) [56] is another method used in root cause analysis. It is a logical, structured process that can help identify potential causes of a failure before the failures actually occur. The process starts by selecting a top level event for analysis. The next step is to identify faults that could lead to this event. The third step is to identify all possible causes to these faults iteratively. A diagram should then be made, using the logic operators AND and OR to represent the sequence of faults and causes. In the end, it should be possible to identify one or more root causes to the event.

Vulnerability cause graphs (VCG)

VCGs [13] are visual representations of vulnerability modeling, allowing the developer to get an overview of the relations between vulnerabilities and their causes. In this representation, the causes and the vulnerabilities are presented in a directed acyclic graph with four kinds of nodes: simple, compound, conjunction and exit nodes. See Figure 3.5 for an example. All nodes but one, the exit node, represent causes. In the figure, the exit node is called “CVE -2005-2558”. The exit node represents the vulnerability in which is analyzed, and it is the only node in the graph without any successors. The simple node represents a cause that may lead to the vulnerability. In the figure, “Use of C -like strings” is a simple node. Compound nodes represent other VCGs. In the figure, the compound node is called “Wrong source size is used”. The last type of nodes is the conjunction node. The conjunction node represents the conjunction of two or more nodes. In the figure, the conjunction node is represented by a dotted line.

VCG's give a good overview of the relations between a vulnerability and its causes. A reason for this is that they support generalization and specialization. If the goal of modeling the vulnerabilities is that the VCGs replace the way the vulnerabilities are actually described, it could result in a loss of information about the vulnerabilities. Compared to the reports that are posted when vulnerabilities are encountered, the VCG method does not provide any elaborate descriptions of the vulnerabilities or its causes, only the relations between them.

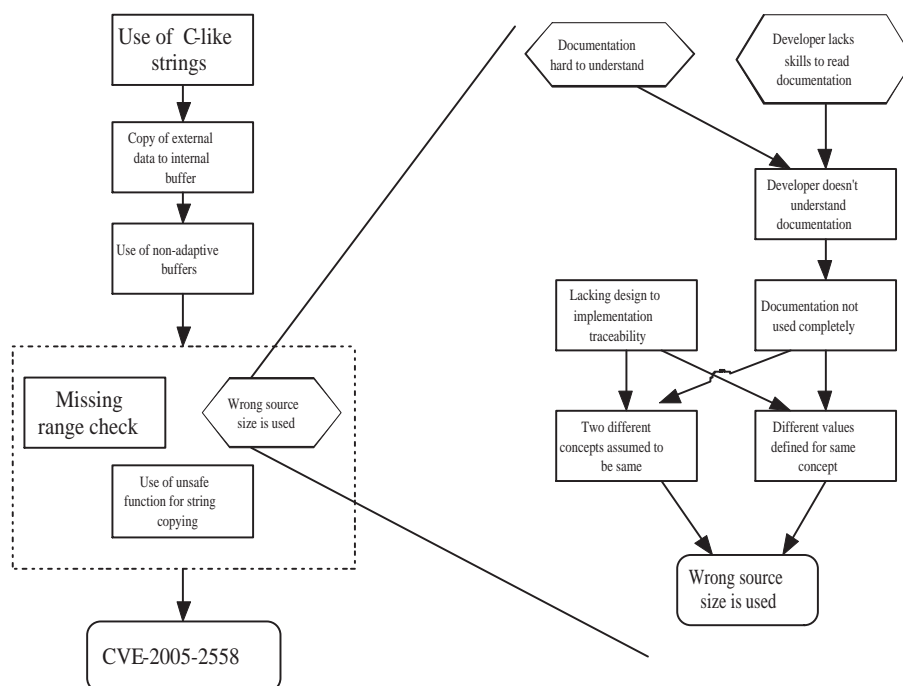


Figure 3.5: Vulnerability cause graph example.

3.5.2 Second view - Threats and attacks

This section describes the second view of security modeling, namely threats and attacks. Three modeling methods from this view are described in detail. These are “attack trees”, “threat modeling” and “exploit graphs”.

Attack trees

The attack tree is a formal and methodical way of representing the security of a system, based on attacks [21]. This is done by representing an attack in a tree structure. The root node represents the main goal, while the successor nodes represent ways to reach the main goal. In this way, all child nodes become subgoals. See Figure 3.6 and Figure 3.7 for a graphical and textual form, respectively.

When modeling an attack tree you may use AND nodes or OR nodes. OR nodes are alternative ways of achieving a goal. AND nodes are different steps that all have to be satisfied in order to reach the goal. AND nodes are also explicitly marked, and all other nodes are by default OR nodes. In addition to these logical operators, two other symbols may be used, I (impossible) and P (possible). This is done by starting at the leaf nodes, and then go step by step to the top. Dotted lines are used for possible attacks, solid ones for impossible attacks. Other boolean values may also be assigned, like the use of “special equipment” (SE) or “no special equipment” (NSE), see Figure 3.8.

Another way of using attack trees is to include a cost in each node. At an OR-gate choose the cheapest child, at AND nodes sum the children as shown in Figure

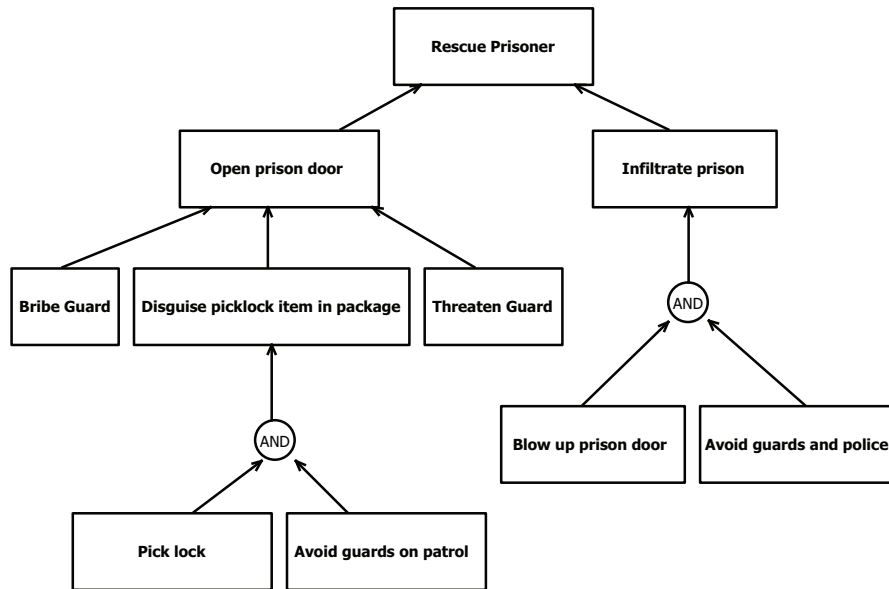


Figure 3.6: A graphical attack tree.

- Goal: Rescue prisoner
1. Open prison door (OR)
 - 1.1 Bribe guard (OR)
 - 1.2 Disguise picklock item in package (OR)
 - 1.2.1 Pick lock (AND)
 - 1.2.2 Avoid guards on patrol
 - 1.3 Threaten guard
 2. Infiltrate prison
 - 2.1 Blow up prison wall (AND)
 - 2.2 Avoid guards and police

Figure 3.7: A textual attack tree.

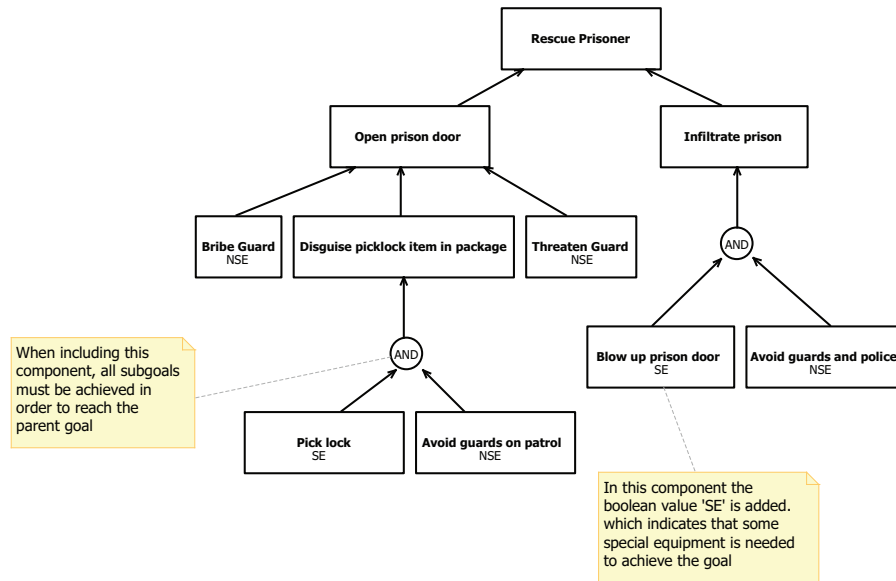


Figure 3.8: An attack tree using boolean values and logical ANDs.

3.9. This could further be used to calculate the total cost of the different attacks, and by doing this, figure out which attacks are the most likely. In this way, one can rule out certain attacks. This will make sense when, for example, the value of an object in a safe is less than the cost of some attacks. Combinations of boolean and continuous values could also be used to model attack trees, for instance to find the cheapest attack using no special equipment.

The nature of the attacker may determine which part of the attack tree should be taken into account. It is therefore important to consider who the possible attackers are and what their skill is. (Risk aversion, access to money, and so on.)

The advantage of using attack trees is that they are simple, informative, and relatively easy to understand. A disadvantage is that there is an issue regarding the notation described in the article when a goal has three or more subgoals, that are AND'ed together. It is not perfectly clear how to model this in a good way, but there is a suggestion in the design chapter.

Threat modeling

Microsoft [61] is an important actor in the field of threat modeling. They have published several books and articles on the subject and some are free for download (See Section 3.5.2 for more information). Their approach on threat modeling is for instance mentioned in the book [90]. It states that a threat model is a document that includes background information about a system, a threat profile and an analysis of this profile. This kind of modeling could prove very useful for development teams in order to identify both security strengths and weaknesses of a system, but it could also consume a lot of time due to its high level of detail. The basic approach of Microsoft's threat modeling is decomposed

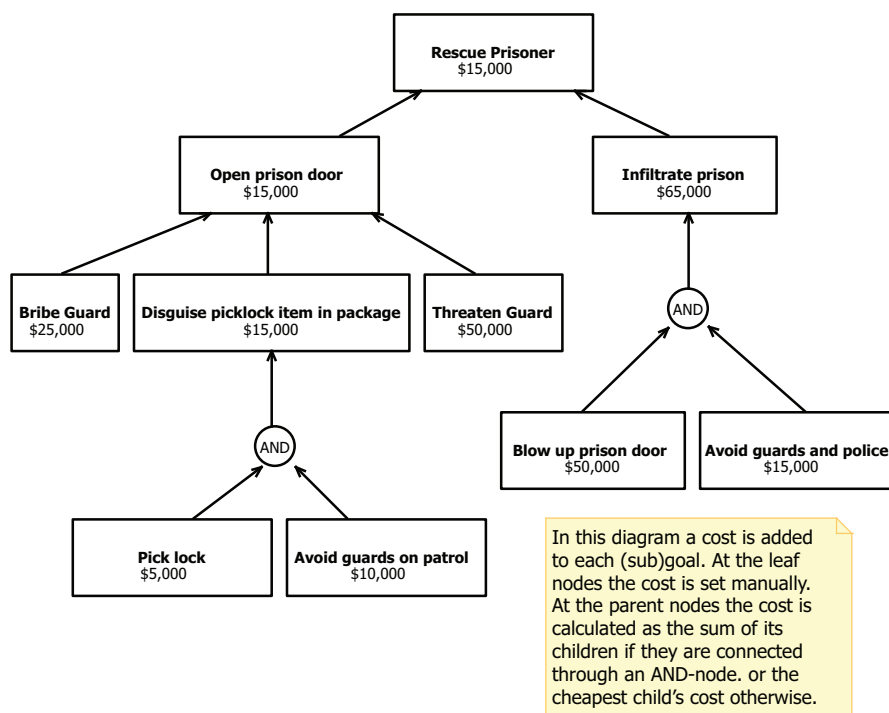


Figure 3.9: An attack tree with costs linked to the nodes.

in the following steps, which are described in detail in the following text:

- 1 Identify Assets and trust levels
- 2 Create architectural overview
- 3 Decompose application - identify entry/exit points
- 4 Identify threats - STRIDE classification
- 5 Document the threats
- 6 Rate threats - DREAD classification

First, one has to identify the assets. That means what is to be protected. In addition to these assets, it will be useful to do some research on the trust levels. Trust levels define the credentials and privileges of the different users. At the next stage, an architectural overview of the system with different diagrams and scenarios is created. This is the preliminary work for the next step where the application is decomposed.

The important part in the decomposition is to identify all possible entry and exit points of the system, in other words interfaces that the application has to the outside world. Data flow diagrams (DFD) [11] could prove to be useful at this point. After the entry/exit points have been identified, the next step is to identify the threats. This is done by using attack trees, knowledge vulnerability databases, and classifying the threats with STRIDE (*spoofing, tampering, repudiation, information disclosure, denial of service, elevation of privilege*).

The next step is to document the threats. Every finding should be documented and added to the threat model document. This helps noticing how thorough the analysis of the system is, and ensures that the threats found are investigated decently. It also helps planning the testing to ensure that the security issues found are tested in an adequate way.

Finally, the threats are rated using DREAD (*damage potential, reproducibility, exploitability, affected users, discoverability*). This is a method for characterizing risk with an numeric value, documented in [42]. It is recommended to use a value from 1 to 3, since such a limited range allows a simplified way to rate each threat. In the example of a value from 1 to 3, the numbers could correspond to the what kind of users will be affected. The number '1' could correspond to an Edge case - few users. The number '2' could correspond to the common case - most users. While the number '3' could correspond to the default case, that is all users who have not specifically disabled this feature.

Positive aspects of threat modeling in general is that it follows an attackers "perspective", which means that it starts with the threats and afterward start searching for weaknesses. It also forces developers to think about security issues during the design phase, which is a better phase to fix programming errors than later in the development process.

Negative aspects about threat modeling is that it may consume a lot of time, and it may not be suitable for small scale projects.

Microsoft threat modeling tool

Microsoft has published an application that is freely available for download. This tool can be used to make an extensive threat model with great detail for

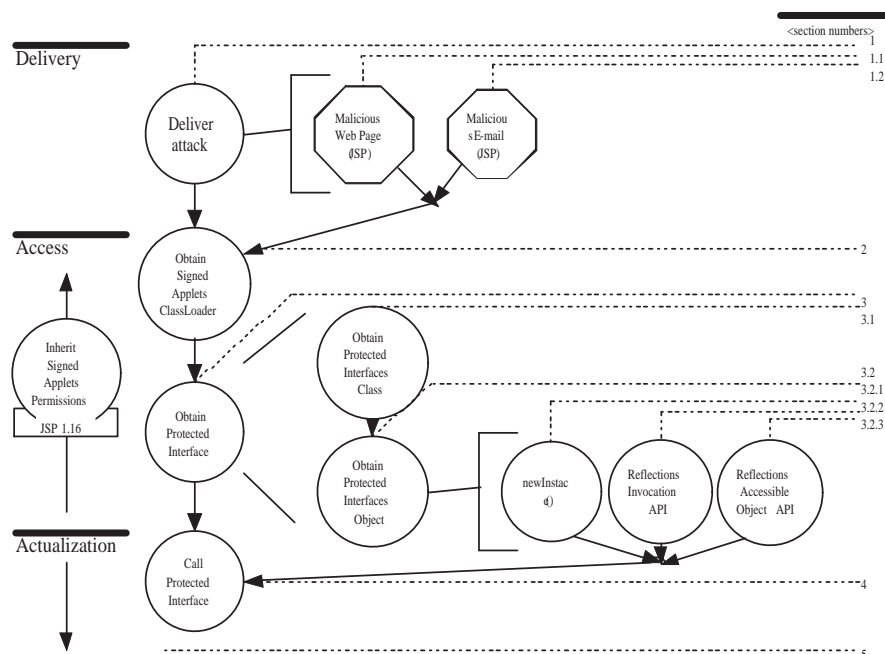


Figure 3.10: Exploit graph: a mobile code attack. Image adapted from [60].

an application that is being developed. The program is divided into three main categories: Background information, entry points and assets, and threats. In each category, it is possible to add relevant information, such as use case scenarios, data flow diagrams and STRIDE classification. In addition it is possible to link this information with relevant entry points and/or protected resources. This will help the software developers to get a clear picture of what an adversary wants and how he could try to get it.

The threat model information is stored in a tree-based view as you can see from the screenshots in Appendix B. The model may be stored as an XML file, or exported to HTML or MIME Hyper Text (MHT)⁸. The tool is primary based on textual input, and it is made to organize all the information that could be derived from Microsoft's way of threat modeling. The tool is not as focused on graphical models like the SeaMonster project is, and since it is closed source from Microsoft, any further work by the project could be difficult.

Exploit graphs

The exploit graph is a way of modeling how an attack can be carried out, given a software risk [60]. It describes what kind of access and/or pattern is required to do so. Flowcharts are used to model the exploit graphs, as shown in Figure 3.10. Exploit graphs also include an estimation of the level of effort to exploit a flaw in the application. Exploit graphs are not widely used [60].

The example shown in Figure 3.10 is a model of a mobile code attack. The example is fetched from [60]. The section numbers to the right in the figure refer

⁸Short form for MHTML

Step #	Detail: How/What	Conditions	Protection
Delivery 1	Deliver attack: get attach code onto machine.	Client must have Internet access.	
Delivery 1.1	Trick user to point browser to a JavaServer Page (JSP) [87].	Browser must have “run JSP” enabled.	Disable JSP in browser. NOTE: doing so prevents other sites from working.
Delivery 1.2	Send victim e-mail containing malicious JSP.	User’s mail reader must interpret JSP.	Disable JSP execution in mail reader.

Table 3.3: Exploit graph table regarding Figure 3.10.

to entries in Table 3.3. This table shows the details for the delivery segment in the figure. The table is only a part of the complete table for this example.

3.5.3 Third view - Countermeasures

It is not sufficient to discover software vulnerabilities and threats; countermeasures are needed in order to prevent them. In this section, four different ways of modeling countermeasures are described. UML misuse/abuse case diagrams, UMLsec, security patterns, and security activity graphs.

UML misuse/abuse case diagrams

The misuse case diagram is an extension of the use case diagram. Creating a use case diagram [18] is a way to model functionality in a system, from a user’s point of view. An example of a use case diagram is shown in Figure 3.11

Use case diagrams have four major elements. These are the system, actors using the system, the functionality of the system (use case), and relationships between the different elements. An actor could for instance be a customer, an administrator, or an employee. The functionality is the tasks the actors can do with the system (a use case), for instance logging into the system or ordering a book. The relationship is a connection between two elements, for instance a line between the employee and the log in function means that the employee has the opportunity to log into the system. In addition, there are three specified relations: “include”, “extend” and “generalization”.

The “include” relationship is used when a use case is dependent on another use case. A dotted, directed arrow points to the use case that has the needed functionality, with the title “include”. The “extend” relationship is used when a use case needs the functionality and the characteristics of another use case. The notation is similar to the “include” relationship, but with the title “extend”. The “generalization” relationship is quite similar to the “extend” relationship. When you connect two use cases with a “generalization” relationship, you can replace the parent use case (tip of the arrow) with the child use case (root of the arrow), without breaking the work flow. The notation is an arrow with a

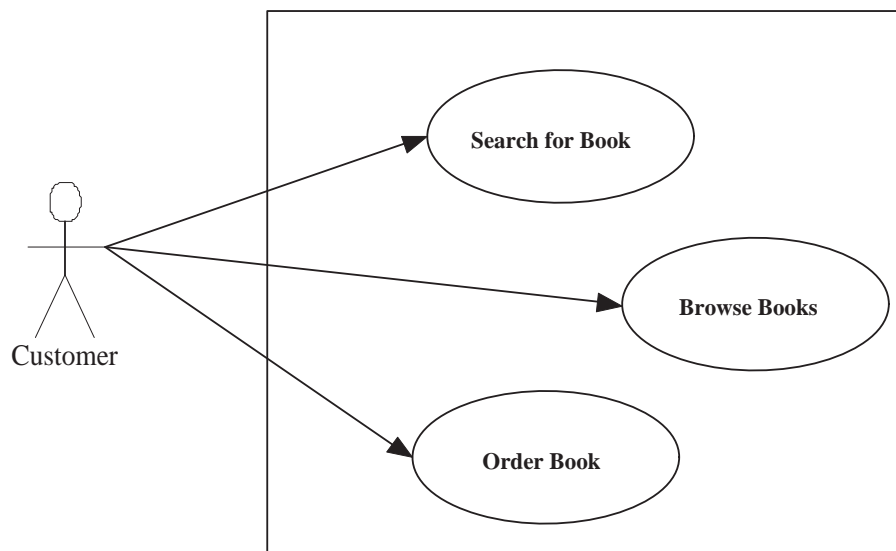


Figure 3.11: A simple use case diagram for an online bookstore.

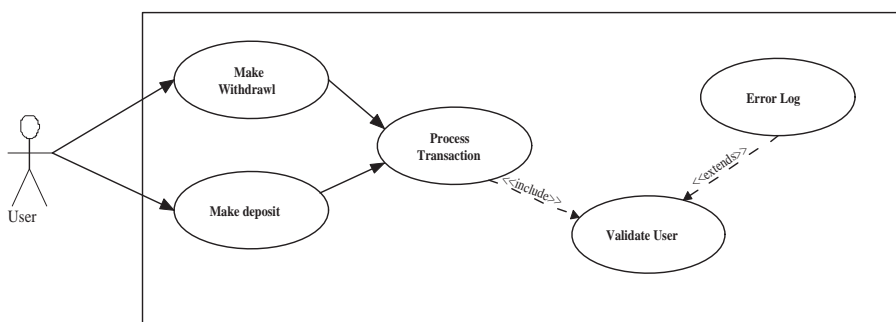


Figure 3.12: Use case diagram with the “extend” and “include” relationship.

triangular tip. See the Figure 3.12 for an example of using the “extend” and “include” relationship.

Use case diagrams are widely used in software design, but they are not good when it comes to security requirements. By adding functionality to the use case diagram, it is able to represent security issues. This extension of the use case diagram is called misuse case diagram [82, 77]. The purpose of the misuse case diagram is to model the desired functionality and relate it to unwanted functionality and harmful attacks. Figure 3.13 shows a simple example of a misuse case diagram.

The model is extended by adding an attacker [82] and an insider [77]. The attacker is a mis-actor that attacks the system from the outside. The insider is a mis-actor that is authorized within the system, but uses this advantage to exploit vulnerabilities. The model is further extended with three more relations: “detect”, “prevent” and “exploit”. The “exploit” relation is used between the insider and a vulnerability. The other two relations are used between the

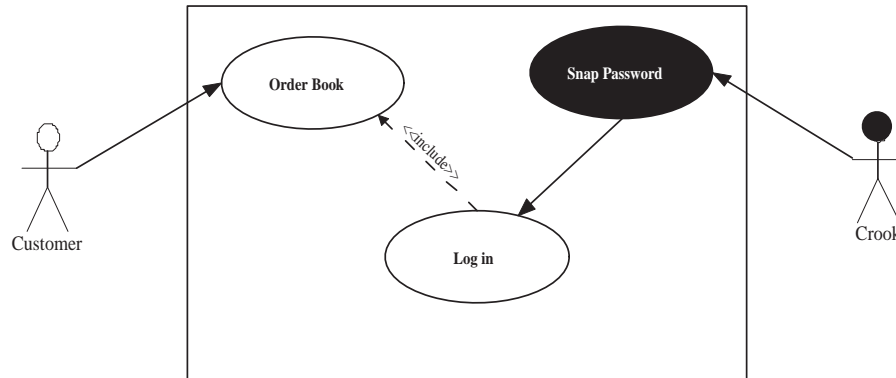


Figure 3.13: A misuse case diagram of a web shop.

attacker and the threat use cases. “Detect” and “prevent” are used in the example, Figure 3.14. The mis-actor and the use cases that the mis-actor makes use of (threatens) is colored in black or red. The insider is colored in grey, thereby the nickname “grey guy”. The use cases (vulnerabilities) are also colored in grey.

The misuse case diagram is not a complete method for documenting and analysing security requirements, but it provides support when working on these tasks. The misuse case diagram can get very extensive when covering all security details. Although it should only be used for doing a part of the job, it can be a good way of getting an overview of countermeasures. The misuse case diagram is also a good way of communicating security requirements between security experts, developers and customers.

UMLsec

UMLsec [55] is a UML security modeling method. The goal is security by design. The UML standard is used because it is common knowledge among a huge amount of developers, and it is accurately defined. UMLsec is an extension of the UML standard, and it includes, among others, use case, activity, deployment, sequence, and state charts. UMLsec allows one to express security related information in UML standard diagrams, so called stereotypes. Examples of stereotypes may be “secrecy”, “fair exchange” or “secure links”. To define the systems environment, stereotypes are used together with tags to formulate security requirements and assumptions. Constraints are criteria that has to be fulfilled in design to meet the requirements.

Figure 3.15 shows an example of a use case diagram with UMLsec notation. The stereotype “fair exchange” is represented, and the trade should be done in a way that prevents both parties from cheating.

A tool has been developed to model UMLsec, the “UMLsec tool” [54], but this is not freely available for testing, and it does not conform to the project’s requirements. (Mainly the open source requirement.)

There are more methods similar to UMLsec that is based on the UML notation. Work has been presented extending use cases and interaction diagrams to develop distributed system architecture requirements [41]. They propose to use role based access rights in use cases. Rights of a role can be derived from

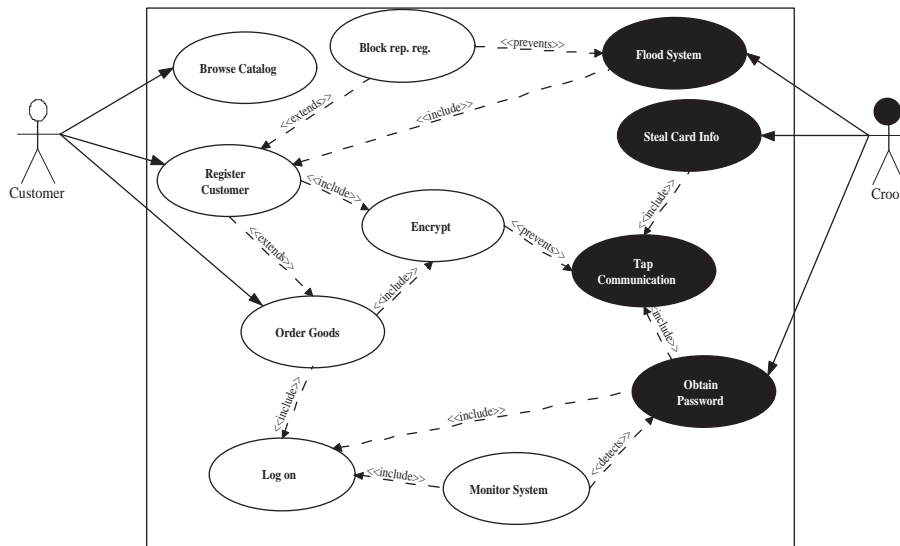


Figure 3.14: Misuse case diagram of a web shop in more detail, based on an example from [82].

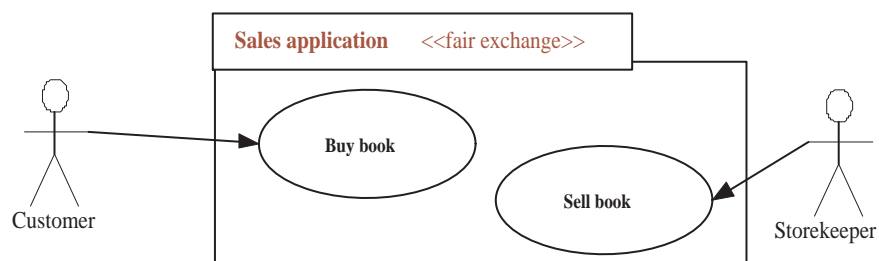


Figure 3.15: UMLsec use case diagram.

these. There has been a suggestion for a method designing secure databases with UML [29]. This method is used in the use case and class models of UML, where they classify different sensibility levels of information and access rights for different user roles.

Security patterns

Security patterns [78] describe security problems that frequently appear in certain contexts, and the solutions to prevent them. Security patterns help to minimize the gap between software developers and security experts by offering use of such best practice patterns. The advantage is that novice developers are able to create secure software by using the security patterns. This is helpful when a development team can not get hold of a security expert of various reasons. This also lets the security experts research more on general security issues, in stead of having to focus on one software development project.

Beneath is a template for representing security patterns [85]:

Problem Describes the security issues addressed by the pattern.

Forces Describes the motivations and constraints that affect the security problem. Highlights the reasons for choosing the pattern and provides justification.

Solution Describes the approach briefly and the associated mechanisms in detail.

Structure Describes the basic structure of the solution using UML sequence diagrams and details the participants.

Strategies Describes different ways a security pattern may be implemented and deployed.

Consequences Describes the results of using the security pattern as a safeguard and control measure. It also describes the trade-offs.

Security Factors and Risks Describes the factors and risks to be considered while applying the pattern.

Reality Checks Describes a set of review items to identify the feasibility and practicality of the pattern.

Related Patterns Lists other related patterns from the security patterns catalog or from other related sources.

“Intercepting validator” is a good example of a security pattern, and is described in [85]. The intercepting validator is said to be the pattern that has the most significant effect [38] in a specific security pattern test, where an application is tested first without the security patterns and then with the security patterns [40]. The subject of this pattern is validating input before it is used. Here is a summary of the most important parts of the pattern:

Problem Wrong use or lack of input validation may lead to *buffer overrun* , arbitrary command execution and *SQL injection attacks*.

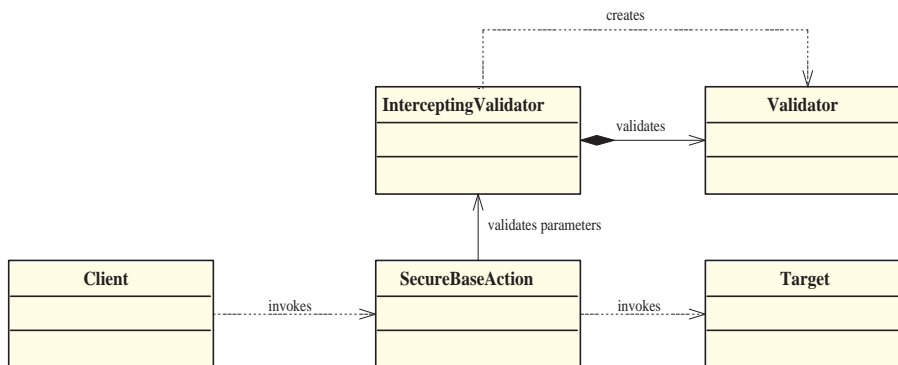


Figure 3.16: Security pattern class diagram for the “Intercepting validator” pattern, based on an example from [85].

Solution Use “Intercepting validator” to cleanse and validate data prior to its use within the application, using dynamically loadable validation logic.

Structure The structure of the pattern is shown using a class diagram in Figure 3.16. The participants and responsibilities is shown with a sequence diagram in Figure 3.17. The sequence diagram starts with the request from the client, and ends with a finished result, the Target, which is successfully validated.

There are several articles describing different kinds of security patterns. “The authenticator pattern” [53] is a pattern for identification and authentication of a server from a client. “Architectural patterns for enabling application Security” [102] has a collection of patterns to be used when dealing with application security. “Single access point” and “Limited view” are two of the patterns included in this article. “Single access point” provides a security module and a way to log into the system, and Limited View allows users to see only what they have been granted the rights to see..

Security activity graphs

The security activity graph [9] is a modeling method describing which activities are available to eliminate threats at various costs. Each vulnerability has a corresponding graph. The goal of the security activity graph is to be an activity during the development phase that helps preventing vulnerabilities in the final product.

The top node in the graph contains the vulnerability, and the leaf nodes contain the activity that can help prevent the vulnerability. The activities has boolean values, which are true if the activity is implemented perfectly. The activities can be connected together with AND, OR or SPLIT gates, illustrated with the notation from [9] in Figure 3.18. AND gates have a true output only if all inputs are true. OR gates have a true output if one or more of the inputs are true. SPLIT gates only have one input, and they send this input value to all the outputs.

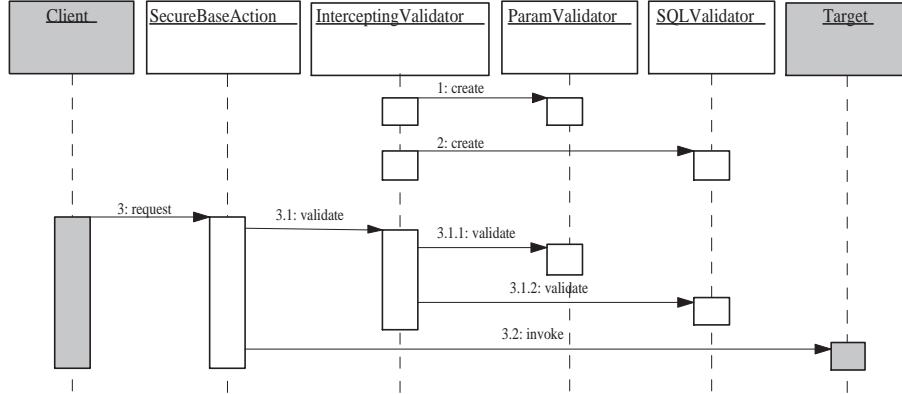


Figure 3.17: Security pattern sequence diagram for the “Intercepting validator” pattern, based on an example from [85].

Security activity graphs are closely related to vulnerability cause graphs, and they could partly be adapted from each other. This feature would be very helpful in a security modeling application, but very complex to implement. Security activity graphs are also complicated and difficult to understand, and the graphs can get quite extensive.

The security activity graphs are in a structured manner constructed from vulnerability cause graphs. The causes in the vulnerability cause graphs can be mitigated using different mitigation techniques, that corresponds to activities in the security activity graphs. The connection is illustrated in Figure 3.19. The vulnerability “Use of `strcat`⁹” is shown in Figure 3.20. This can be mitigated by “use preprocessor to prevent calls to `strcat`” and “Use `strlcat`¹⁰ insted of `strcat`”. It can also be mitigated by “use preprocessor to prevent calls to `strcat`” and “use safe string library”, and another two mitigation techniques.

Figure 3.21 shows an example of a security activity graph. The textual representation of the example is shown in formula (3.1).

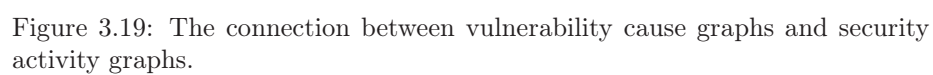
$$((M2 \wedge (M4 \vee (M3 \wedge M5))) \vee M1) \quad (3.1)$$

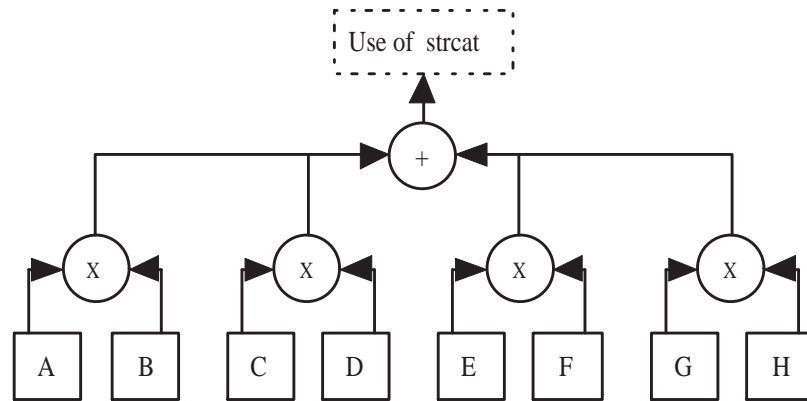
3.6 Desired solution

This section considers the ideal situation that the customer wants to reach, and the project team wants to achieve. The utopia for this project is to remove all vulnerabilities in software, but one of the more reachable goals is to make programmers more focused on software security. This is something that, in the end, could lead to a reduction of security issues in developed software. The following sections point out some ways to reach this situation.

⁹Abbreviation of “string concatenate” allows one memory block to be appended to another memory block.

¹⁰A more safer version of “string concatenate”, that prevents buffer overflows by checking the length of the destination memory block before appending.





- A Use preprocessor to prevent calls to strcat
 B Use strcat instead of strcat
 C Use preprocessor to prevent calls to strcat
 D Use safe string library
 E Use code inspection to find calls to strcat
 F Use strcat instead of strcat
 G Use code inspection to find calls to strcat
 H Use safe string library

Figure 3.20: Example of a security activity graph. Example adapted from [9].

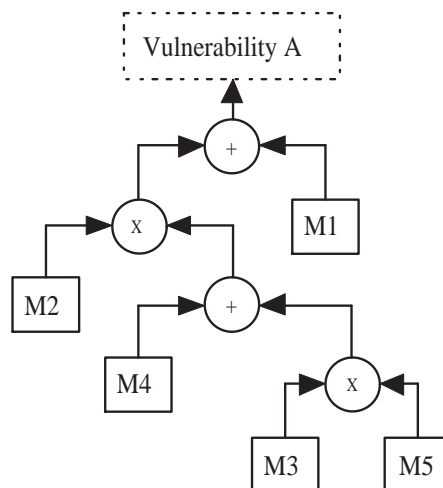


Figure 3.21: Example of a security activity graph. Example adapted from [9].

3.6.1 Standardized notation

The solution (implemented software) should use state of the art notation of each chosen modeling technique.

In addition, the customer requests a pre-defined set of symbols, objects, and relations used in the software to keep the syntax correct and make sure that the models are simple and informative.

3.6.2 Reduce the time used to make security models

The final product should be easy to use and made in such a way that security experts could reduce the total time spent making a security model, see Section 2.2.6.

3.6.3 Shrink the knowledge gap

As mentioned in Section 3.3, security experts are often not a part of development teams, and software developers have little security knowledge, so a gap between the two groups arise. The solution should address this issue as stated in BR.05 in Section 3.2, and it should shrink this gap, by letting the security expert use his knowledge to model the security issues in a good way. Then, after the model is saved, the software developer could open the relevant kinds of models, gain some of the security experts knowledge, and use this input further in the development process.

3.6.4 Link to vulnerability databases

The desired solution should link to known vulnerability databases, such as CVE and CWE, where it is appropriate to do so.

3.6.5 Improvements over existing applications

The implemented solution will exceed and improve functionalities partly covered by already existing applications, which are mentioned in Section 3.3. The main reasons for this are shown below.

1. The application has a standardized notation with a pre-defined set of symbols. This make the application free for unnecessary, possible confusing elements.
2. The application is open source, distributed public and freely available for changes, additions and improvements.
3. The application can model more than one type of security model and link the three views discussed in this chapter.

3.7 Market investigations

As mentioned in Section 3.3, security modeling today is often done using general purpose drawing tools like Visio [64] and SmartDraw [84] or tools like MagicDraw [72] and Poseidon [31] for creating UML diagrams. In addition there exist

Grade	Integer	Description
High	5	The solution covers most aspects of this criterion.
Medium	3	The solution covers some aspects of this criterion.
Low	1	The solution covers no aspects of this criterion.

Table 3.4: Evaluation criteria grades.

free modeling tools, like the Microsoft Threat Modeling tool, that is described in 3.5.2 and some commercial tools described in Section 3.3.

The customer clearly stated throughout the project that a usable tool for modeling security is definitively needed, not only by them, but also in general by security experts and software developers.

The research and pre-studies done in this project conclude with that there exists no free general security modeling tool, which includes all the views discussed in this chapter. This is also backed up by [8].

In the same article, there is a discussion about the need for an independent, free security modeling tool to support the development of secure software. This tool should be possible to use by security experts to create security models, and software developers should be able to use the models in the software development process.

3.8 Alternative solutions

In this section, some of the solutions used for security modeling are described. At the moment, no purpose-built tool exists [8], but several drawing tools, more general in nature, are used. These tools usually support the making of any type of chart, model or diagram. Because of this purpose, they do not have any restrictions when it comes to modeling syntaxes. When not following a specific syntax, the result is often hard to interpret by others, and the diagram can easily get messy and unclear.

3.9 Evaluation criteria

In this section the evaluation criteria of the different solutions of the task are discussed. The criteria listed are all high prioritized criteria, and there will be assigned a grade to each of them: low, medium or high, with integers 1, 3, and 5, respectively. This is shown in Table 3.4. The criteria are based on the business requirements, the team's skills, and the available time and resources.

There are four main options of solution in which the criteria will be evaluated:

- Terminate the project.
- Buy an existing solution.
- Further develop an existing solution
- Develop a new application.

The following subsections list the criteria that will be evaluated.

3.9.1 EC1: Resources needed

As the project has a pre-defined time frame, and an absolute deadline, this criterion is an important criterion. The grade “high” will be given if the solution could easily be finished within the time limit and within the resources available.

3.9.2 EC2: Intuitive graphical user interface

This criterion is listed in the initial requirement specification by the customer, with a high priority. It is important that the users of the program do not spend much time learning how to use the program, it should be intuitive. The grade high will be given if the solution lets the user start modeling immediately, without having to spend much time learning the program.

3.9.3 EC3: Modularity

The solution must be configurable and have the possibility to upgrade with new modeling techniques. The grade high will be given if the solution easily could be extended and configured.

3.9.4 EC4: Platform independent

As development today is done on a wide range of platforms, platform independence is an important criterion. A solution that is platform independent will be given the grade high.

3.9.5 EC5: Team qualifications

If the team is to develop a prototype, there is a need for a certain amount knowledge about the development tools, programming language, and plug-ins that will be used. If all the team members have a good amount of skills in the tools that will be used, the grade high will be given.

3.10 Evaluation

The following is an evaluation of four possible solutions, as well as the modeling tools described earlier in this section.

3.10.1 Terminate the project

Terminating the project could be based on lack of resources, man power, or funds. Since the team consists of six persons, and an estimated total of 1528 hours (which is available for usage), this option will not be discussed in more detail than in Table 3.5.

3.10.2 Buy an existing solution

Buying an existing solution will not be of interest since the customer requires an open source implementation not controlled by a third party. Besides that, another drawback with this solution is the funds needed, and the team have a

Evaluation Criteria	Grade	Comment
EC1	High	By terminating the project, the amount of resources needed is not particularly demanding.
EC2	Low	No prototype will be developed.
EC3	Low	No prototype will be developed.
EC4	Low	No prototype will be developed.
EC5	High	The solution do not need any extensive expertise from the team.

Table 3.5: Evaluation criteria for termination of the project.

Evaluation Criteria	Grade	Comment
EC1	Low	An extensive amount of funds is needed to buy a software with all the functionality the customer wants.
EC2	Medium	This depends on which software is chosen to buy.
EC3	Medium	Existing software may or may not be fully configurable and extensible.
EC4	Medium	Existing software may or may not be independent of the platform used.
EC5	High	If this solution is chosen, the team has adequate time to get to know the software well and modify it to a given extent.

Table 3.6: Evaluation criteria for buying an existing solution.

strictly limited budget. This option will not be evaluated in more detail than shown in Table 3.6.

3.10.3 Further development of existing software

During the span of the pre-studies, searches for already existing modules to build SeaMonster upon have been executed in parallel with other tasks. This has lead to results indicating no existence of such software neither in a complete form nor as code to use as a starting point for SeaMonster. The composition of requirements as usage of GMF, and the mapping between the three views are the main reason for the difficulties of finding relevant code. This makes the project exclusive in the field of security modeling with the mentioned functionalities. The option of using existing software will for these reasons not be discussed further than in Table 3.7.

3.10.4 Develop a new application

Developing a new application, based on the Eclipse GMF and EMF framework satisfies all the requirements given by the customer. For this reason, and because of the difficulties with the alternative solutions as discussed above, this is chosen as the method for reaching the desired solution. Table 3.8 lists the evaluation

Evaluation Criteria	Grade	Comment
EC1	High	The amount of resources needed are dependent on which programming language used and other technical issues such as amount of code.
EC2	Medium	The team is somewhat bound by what is already implemented of the GUI in the software that is to be further developed.
EC3	Medium	Dependent on the already implemented part of the software.
EC4	Medium	Dependent on which programming language that is used in the software.
EC5	Medium	Dependent on which programming language that is used in the software.

Table 3.7: Evaluation criteria for further development of an existing solution.

Evaluation Criteria	Grade	Comment
EC1	High	The amount of resources needed is dependent on which programming language and other technical issues.
EC2	High	The team are in full control of the GUI and should be able to develop a good and easy GUI.
EC3	High	Modularity will be achieved because of the application will be open source and based on GMF and EMF.
EC4	High	Java is platform independent and will be used as the programming language.
EC5	High	All team members have at least some experience with Java.

Table 3.8: Evaluation criteria for developing a new application.

criteria for developing a new application.

3.10.5 Evaluation of methods

As "Develop a new application" is chosen, the next step is to evaluate and conclude with which of the sources of security related information and security modeling techniques that should be included. See Table 3.9 and Table 3.10 for the advantages and disadvantages regarding each of the information sources and techniques discussed, respectively.

Source	Usability for this project	Usability for later phases beyond the scope of this project
The Common Vulnerability and Exposures dictionary	Easy mappings to relevant vulnerabilities related to specific software issued with an identifier.	Linkage between different vulnerability databases, aim SeaMonster for the “CVE-compatible” title.
Common Weakness Enumeration database	Easy mappings to relevant vulnerabilities related to common problems.	Aim SeaMonster for the “CWE-compatible” title.
Common Configuration Enumeration	None	Implement a function like “workarounds” within SeaMonster and collect information from this location.
Common Platform Enumeration	All software naming within SeaMonster are to use this standard.	
Open Vulnerability and Assessment Language	No direct usage.	
Security Content Automation Protocol	No direct usage.	Implement a function like textual or graphical expressions of mapping between vulnerability databases with help from the definitions stated in this protocol.
CERT/CC	Include theory researched by CERT in SeaMonster like secure coding practices.	Include theory in more detail, and force the users of SeaMonster to write more secure code and include vulnerability remediation practices.
BugTraq	No direct usage.	Include information from this mailing list to models requesting extensive details.

Table 3.9: Advantages and disadvantages regarding each source of security related information.

Technique	Advantages	Disadvantages
Root cause analysis - Five Whys	Deep understanding of the causes of a problem.	Does not have a graphical notation.
Vulnerability cause graphs	Supports generalization and specialization.	VCG's do not provide detailed descriptions of the vulnerabilities and their causes. They only show the relations between them.
Attack trees	Simple and informative.	Problem applying AND on three or more subgoals.
Threat modeling	Follows an attackers "perspective"(starts with the threats and afterward start searching for weaknesses).	May consume a lot of time and may not be suitable for small scale projects.
Exploit graphs	Detailed and does include an estimation of the level of effort to exploit a flaw.	Exploit graphs are not widely used.
UML misuse case diagrams	Easy, informative and it follows the UML syntax.	Diagrams can get very extensive.
Security patterns	Security-novice developers can use patterns to avoid vulnerabilities.	Not easy to implement in this software, due to a textual representation.
UMLsec	Follows the UML standard.	Complex with use in SeaMonster.
Security activity graphs	Related to the Vulnerability cause graphs.	Can get complex and hard to understand.

Table 3.10: Advantages and disadvantages regarding each security modeling technique.

3.11 Choice of solution

Development of a new application is favorable and chosen as the desired solution. This section concludes with which methods and sources of information, addressed earlier in this chapter, should be included in the implemented application. Table 3.9 and Table 3.10 are used as a starting point for making this decision. SeaMonster will include a limited, easy to understand notation that still is able to model a large variety of attacks, vulnerabilities and countermeasures. From each modeling view discussed in this chapter, only one of the mentioned methods is chosen to be implemented in the application, in order to keep the notation limited. The chosen method must also comply with the other notation specific requirements (easy to understand, possibility to model most attacks).

3.11.1 Chosen methods

From the first view, “Vulnerability and cause modeling”, the VCG method will be included. The main reason for this is that the method provides a good overview of the relations between a vulnerability and its causes. It also has more capabilities, notation wise, than the alternatives. The five whys on the other hand, lacks possibilities for representing the results visually, and cannot be used without large additions to the notation.

From the second view, “Threats and attacks”, the attack tree notation will be implemented because of its simplicity and intuitive nature. With this notation, it is possible to model a wide range of attacks in an easily understandable way, relatively spoken. Threat modeling will be excluded in SeaMonster due to its extensive amount of information. The information available for exploit graphs is skimpy compared to attack trees. The exploit graph notation could for this reason be unknown for most people, and it will not be included in SeaMonster.

From the “Countermeasures” view, misuse case diagrams will be included due to its intelligible and informative characteristics. The diagrams are based on the well known UML standard and is considered familiar to the users of SeaMonster. Another method based on the same standard is UMLsec. This notation will not be included due to the high number of UML models that need to be implemented, and the benefits of using misuse case diagrams eliminate the reasons for using UMLsec. UMLsec and Security activity graphs can be added at later stages, but this prototype will not prioritize these standards.

3.11.2 Sources of information included

In addition to the chosen methods, there will also be implemented a function to relate a vulnerability to the CVE and CWE databases. These databases are the most relevant, since they are directly linked to the scope of the project and also a requirement from the customer¹¹. When an application (actually any piece of software) is named in SeaMonster, CPE will be used as the syntax. The other databases and sources of information described in the pre-study chapter are outside of the project scope, and they will therefore not be implemented at this stage. All topics mentioned in Section 3.4, but excluded in the implementation are candidates for further work on SeaMonster.

¹¹See Chapter 4 for more information.

Chapter 4

Requirements specification

4.1 Introduction

This chapter contains the requirements specification of the SeaMonster project. It focuses on elicitation, analysis and specification of the requirements. A top-down approach to the requirements analysis is presented. The SeaMonster application is a novelty in the field of security modeling. For this reason the process of requirements engineering that is described, refers to a development starting from scratch. This kind of engineering process is often called “green-field” engineering and is characterized by the fact that it is impossible to base the requirements, the architecture design, and the code production on previous solutions. The chapter starts with a general introduction of the Requirements Analysis and Specification (RAS) activity. After that, it proceeds by giving a general definition of the system, in addition to detecting and classifying the users and their targets. Requirements related to the main external interfaces, such as software interfaces and user interfaces, are then taken into account. The application will mostly be used to present information to the user graphically. Requirements related to the models and their graphical appearance will therefore get a high priority.

The last part is focused on the different groups of requirements and their specification, which makes up the basis for the later phases of the project, in particular design, realization, and testing.

4.1.1 Purpose

Within the requirements specification phase it is quite important to describe the application’s qualities that has to be assured, rather than describing how such qualities will be designed and implemented (what versus how). Functionalities that the system has to supply are defined in this part of the documentation. This is done without indicating neither a specific architecture nor a particular algorithm to adopt in the implemented solution. Like in many other cases in software development, the application is part of a more general system. For this reason, it could be useful to elicit software requirements from the system requirements. Such software requirements underline the responsibilities that are assigned to this specific application within the global system solution. The main objective of the activities that are related to requirements, is to understand

as clearly as possible, the interface between the external environment and the software to be implemented (S2B). For this reason, the application domain is figured out along with the identification of stakeholders. Software requirements analysis is one of the most important activities within the development work flow. Other stages are based upon it, such as software design and testing. They express features and functionalities that the application must provide, so that is the reason because it is important to test the system from its requirements' point of view. As a matter of fact, The customer will base is usefulness assessment on their fulfillment.

4.1.2 Structure

The structure of this chapter is partly conformed to recommendations expressed in IEEE Standard 830 [44]. Some parts that are suggested in the standard, have been removed to fit the needs of the project. The following is a general overview of the chapter structure.

Overall description Description of the application domain and the objectives of the realization activity. This includes documentation of the knowledge of the application domain which is relevant for deriving the specifications. Relevant questions that are answered in this section, are listed below.

1. Who are the users, and what are their objectives and expectations?
2. What are the main entities that characterize the domain?
3. What are their main relationships (e.g. between users and use cases)?

Functional and non-functional requirements The analysis of the software requirements is described in these sections. The software requirements are divided into functional and non-functional requirements. The relationships between them and the business requirements, are investigated.

Summary Synthesis of the work done.

A complete collection of all the initial requirements that are given by the customer, divided by typology, can be found in Appendix C. For each requirement, an identifier, a name, an informal description, and a priority level are quoted. Later in this chapter, priorities are represented by a letter. The letter states the importance of each requirement to the customer, with respect to the other requirements. The following list summarizes the three levels of priority that are used.

L Low priority

M Medium priority

H High priority

Entity	Model
Cause of vulnerability	VCG
Threat/Attack	Attack trees
Countermeasures	UML misuse case diagrams

Table 4.1: Software security entities and their related models.

4.1.3 Intended audience

Another important feature of the requirements specification document is that it must represent a solid communication medium between the customer and the developer, allowing a more successful dialog focused on the project target. The main audience of the present document is thus the software security group at SINTEF ICT.

4.2 Overall domain description

This section gives a general description of the domain of the SeaMonster application. This is achieved by describing the product, based on general statements from the customer. Users are divided into classes, for easier detection of possible use cases. Finally, some preliminary decisions on requirements are taken.

4.2.1 Product perspective

The aim of the project, as indicated by the customer, is summarized in the following text.

Develop a security modeling tool helpful for security experts and developers of software. With such a tool it must be possible to model security related aspects like vulnerabilities, threats and countermeasures by use of a specialized graphical editor. The product must be based on state-of-the-art security techniques and possible improvements of these.

From this, it is indicated that the system has to be a graphical editor in the field of software security. It is also specified that the main functionality of the application, must be the ability to model three related aspects of software security. This should be done based on the leading modeling techniques. It is important to remark the entities managed by the application, which, at the end, are characterizing elements of its domain. Such entities could be elicited from the requirements, and they are essentially the targets of the modeling. They are threats/attacks, vulnerabilities, and countermeasures. The relationships between them are represented in Figure 4.1. Other desired functionalities are the possibility to link to CVE/CWE online databases and to convert the drawn models to a textual representation. Figure 4.2 gives a graphical perspective of the product.

Following up the pre-study chapter (Chapter 3), it is possible to state precisely which are the chosen modeling techniques. They are summarized in Table 4.1.

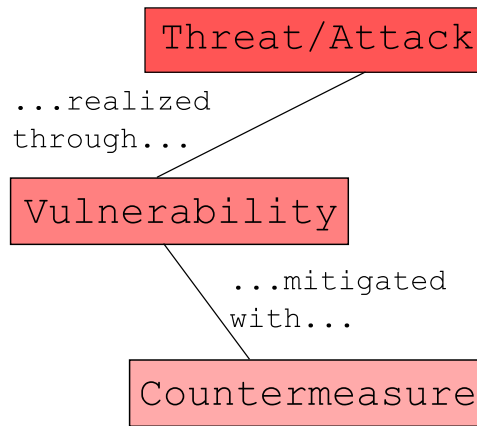


Figure 4.1: Relationships between the software security entities.

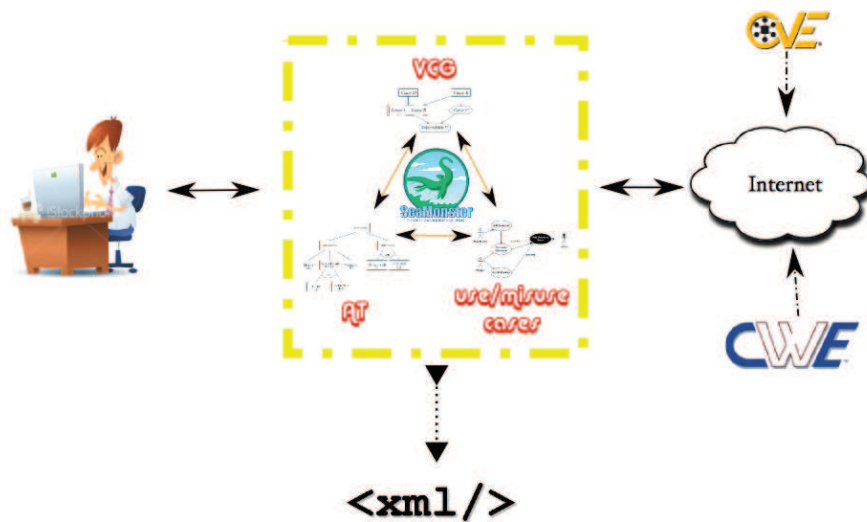


Figure 4.2: An overall perspective of the SeaMonster application.

Actor	Activities
Security expert	Consulting online CWE/CVE databases; Creating and modifying models (i.e. attack trees, use/misuse case diagrams and VCGs); Writing articles;
Software developer	Consulting online CWE/CVE databases; Creating and modifying models (i.e. attack trees, use/misuse case diagrams and VCGs) usually related to specific project; Coding, testing, debugging;

Table 4.2: Users and their activities.

4.2.2 Actor classes and characteristics

It is clear that the main actors, that fundamentally are also the most relevant stakeholders, may be collected into two main categories: security experts and software developers. Both of them should be supported by the implemented application in their activities. In particular, Table 4.2 lists the activities aided by SeaMonster.

The application is meant for both the mentioned classes of actors. More precisely, the first group - security experts - is composed of people that have a lot of knowledge about topics regarding security. During their activities, they are asked to analyze software security aspects and to produce results that may consist of new documentation, as well as definitions about such aspects. Thus they know all the models requested for the application and they are also able to create and manage them.

A software developer, on the other hand, has a role inside of the software development process. He is, in particular, involved in the realization phase. He has to understand directives from the experts in order to develop good and safe software.

4.2.3 Delayed Requirements

In the list of requirements that are specified by the customer, requirement SF.01 - data should be stored according to the UML 2.1 meta model - is not completely taken into account during the development process. Since it has been given a low priority, it does not represent a vital feature of the application. At least, this applies to the early phases of the application, so its analysis will be postponed to a later time. Such a choice has to be related fundamentally to the temporal constraints of the development. Defining an exportation format concordant with the UML 2.1 meta model could lead the team to easily exceed the strict established time boundaries defined within the planning phase. Nevertheless, the lack of some properties and functionalities within the application, does not mean that the related requirements cannot have any impact on the team's work. Also if they are not fully considered, they could lead to consequences when it comes to the workload. In the present case, for instance, producing a structure that could easily be exported according to the UML 2.1 meta model, could influence the way the application is going to be designed.

4.2.4 Specific requirements

The goal of the following sections is to analyse the customer's requirements, by looking for relations between them. This gives an impression of the overall system functioning. Software requirements are split up into two overall categories, functional and non-functional. A program must fulfill the specification of its functional requirements to be defined as correct [32]. However, there exist other kinds of requirements that are not related to the software functionalities. Usability and legal requirements are examples of this, and they are said to be non-functional requirements.

4.3 Functional requirements

Functional requirements make clear what a system should be able to do. They are initially described in an informal way, and they might therefore be incomplete at this point. In the SeaMonster project, several iterations of the requirements specification process are planned. This is done in order to make sure that the requirements are specified according to the customer's needs. The current section refines them by figuring out how to fulfill them and defining a set of techniques for the fulfillment measurements. Initially, the focus is on the possible use cases where the actors, described in Section 4.2.2, may be involved. Then, a set of possible interactions with the application is shown by means of use case scenarios. Finally, all the elicited requirements are collected together within a more complete view in Tables 4.11, 4.12, 4.13, 4.14, and 4.17.

4.3.1 Use cases analysis

The aim of this section is to figure out which are the most important use cases. They are extensively evaluated at the beginning under a textual form underlining the involved actors, the priority, pre-conditions, post-conditions, and basic flow of events. The latter includes possible alternative paths and exceptions that may raise during the normal course of events. Preconditions and postconditions are used for deciding when a use case has to start and to stop. They are specified only if the condition is not considered too trivial. Later in the section, Figure 4.11 shows such use cases as a whole by means of UML use case diagrams, permitting a more global and complete view of the SeaMonster's functionalities. Use case diagrams are helpful in at least the three areas listed below.

Determining requirements Creating and analyzing use cases can help determining new requirements.

Communicating with the customer The simple and effective notation can favor the communication between developers and the customer.

Generating test cases Defining scenarios based on use cases can help deriving new test cases for the application.

The following use cases are to be considered as the starting point for the functional requirements elicitation. Investigating them and their relationships makes it possible to better understand which are the most important services

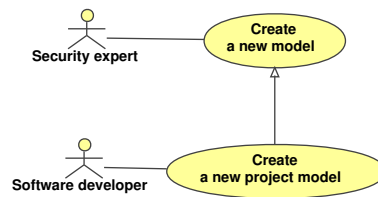


Figure 4.3: Use case diagram for “Create a new model”.

Name	Create a new (project) model
Actors	Security expert, software developer
Priority	High
Entry conditions	The application has been started.
Flow of events	<ol style="list-style-type: none"> 1. The actor selects creating a new model of the type he wishes to use first on his security model. The model may be related to a specific project. 2. The actor specifies name for the model. 3. The system presents an empty diagram corresponding to the model. 4. The actor models the security entity. See the use case in Table 4.7. 5. The actor saves the model. See the use case in Table 4.6.
Alternative flow	<ol style="list-style-type: none"> 5a. The actor does not save the model. 5b. The use case ends.

Table 4.3: Use case for “Create a new model”.

and functionalities that the application must supply. Implementing the described operations, moreover, is the best way to fulfill the expressed requirements.

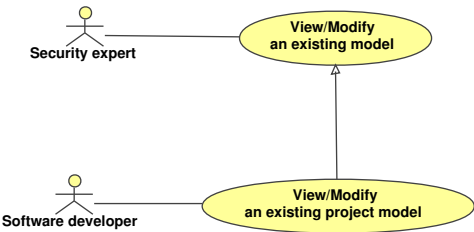


Figure 4.4: Use case diagram for “View/Modify an existing model”.

Name	View/Modify an existing (project) model
Actors	Security expert, software developer
Entry conditions	The application is running. The actor has permission to read (and write) a local copy of the model he wants to view (and modify). The model may be related to specific project.
Flow of events	1. The user opens a model diagram. See the use case in Figure 4.5. 2. The model is shown in the system window, and the user can consult the model.
Alternative flow	2a. The user wants to modify the model. See the use case in Table 4.7. 2b. The user chooses to save the modified model. See the use case in Table 4.6.
Alternative flow	2aa. The user wants to print the model. See the use case in Table 4.10.

Table 4.4: Use case for “View/Modify an existing model”.

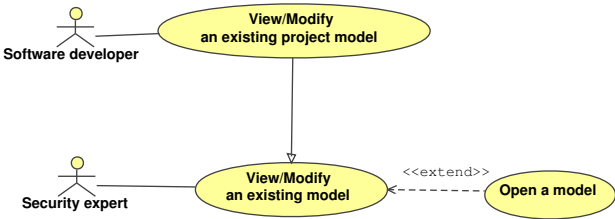


Figure 4.5: Use case diagram for “Open a model”.

Name	Open a model
Actors	Security expert, software developer
Priority	High
Entry conditions	A security model has been created.
Flow of events	<ol style="list-style-type: none"> 1. The actor chooses to open a model. 2. The system lets the user specify which file to open. 3. The actor chooses the wanted view of the model he wants to open. 4. The system opens the file in the selected view.
Alternative flow	<ol style="list-style-type: none"> 3a. The desired model view has not yet been created, so the user wants to initiate it. 3b. The actor chooses to initiate a diagram of the wanted type. 3c. The system lets the user specify on which model view he will base the diagram. 3d. The system creates a new diagram, already incorporating relevant information about chosen security model. <p>The use case ends.</p>
Exception	4x. The actor has specified an invalid file (either not existing or not containing a model). The use case ends.
Exit conditions	The diagram, containing the chosen view of the security model, specified by the actor, is opened and shown in the system window.

Table 4.5: Use case for “Open a model”.

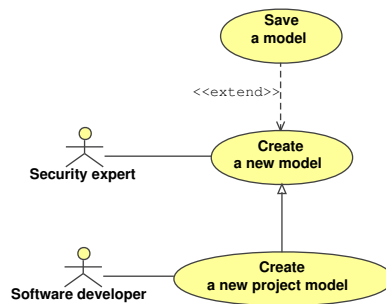


Figure 4.6: Use case diagram for “Save a model”.

Name	Save a model
Actors	Security expert, software developer
Priority	High
Entry conditions	A model, that could be related to a specific project, has been opened and is currently shown in the system window.
Flow of events	<ol style="list-style-type: none"> 1. The actor chooses to save the current model. 2. The actor provides a filename and location. 3. The system saves the current changes done to the model. If the model is opened again later, these are reflected.
Alternative flow	<ol style="list-style-type: none"> 1a. The actor wishes to export the model as an image. He selects this option. 1b. The system presents choices for what kind of image to export to. 1c. The actor selects image type and specifies a filename and location. 1d. The system creates an image of the model, stored at the path selected.
Exit conditions	The model is physically stored.

Table 4.6: Use case for “Save a model”.

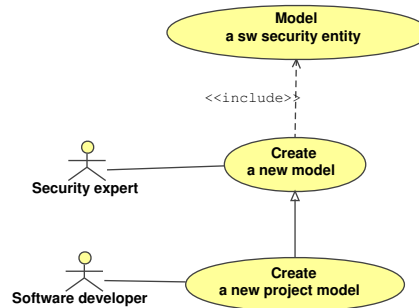


Figure 4.7: Use case diagram for “Model a security entity”.

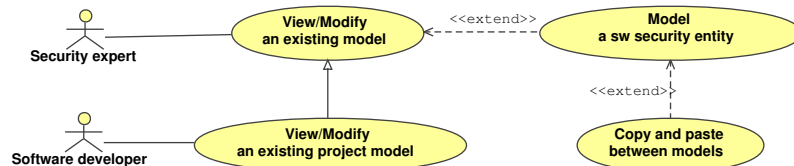


Figure 4.8: Use case diagram for “Copy and paste between models”.

Name	Model a security entity
Actors	Security expert, software developer
Priority	High
Entry conditions	A diagram representing the specified view of a security model, that could be related to a specific project, has already been opened in the program.
Flow of events	<ol style="list-style-type: none"> 1. The actor chooses the required model notation from a palette. 2. The actor draws the model on a modeling window in the system. 3. The systems shows an updated view containing the newly added component. It also presents a properties window for the component. 4. The actor specifies any required properties of the newly added component. 5. The system updates the component properties and effects that follows from this, and presents the changes in the model view.
Alternative flow	<ol style="list-style-type: none"> 1a. The actor wants to remove, rather than add, components. 1b. The actor selects the component he wishes to remove. He chooses to delete the component. 1c. The system removes the component, and any dependent components, from the model view. Go to 5.
Alternative flow	1aa. The actor wants to copy and paste one or more components from another model. See the use case in Table 4.8.
Exception	3x. The newly drawn model is inconsistent with the model. The system prevents the changes from taking effect.
Exit conditions	There are not particular conditions that must hold at the end of this use case. The actor may decide to stop the modeling activity and leaving the model in some unknown state.

Table 4.7: Use case for “Model a security entity”.

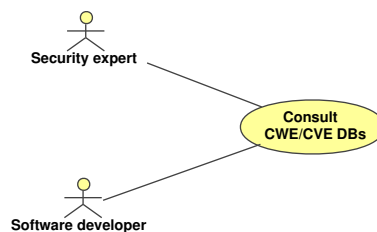


Figure 4.9: Use case diagram for “Consult CWE/CVE DB’s”.

Name	Copy and paste between models
Actors	Security expert, software developer
Priority	Medium
Entry conditions	At least a couple of models have already been opened in the program. Some of them may be related to a specific project.
Flow of events	<ol style="list-style-type: none"> 1. The actor selects the source model. 1. The actor chooses the components he wants to copy. 2. The actor copies the specified components. 3. The actor selects the target model. 4. The actor pastes the copied components. 5. The system updates the component properties and effects that follows from this, and presents the changes in the model view.
Exception	3x. Source and target model views are not compatible. The system prevents the changes from taking effect.

Table 4.8: Use case for “Copy and paste between models”.

Name	Consult CWE/CVE DB's
Actors	Software developer, security expert
Priority	Medium
Entry conditions	The application is running and an Internet connection is available. The actor should know what to look for.
Flow of events	<ol style="list-style-type: none"> 1. The actor starts a new search within the remote database. 2. The actor gives to the system some search key. 3. The system asks to the remote DB interface for some answer based upon the given search key. 4. The actor consults the obtained report.

Table 4.9: Use case for “Consult CWE/CVE DB's”.

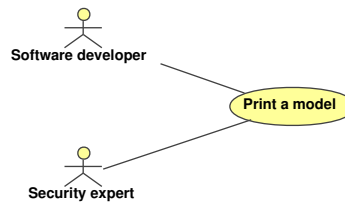


Figure 4.10: Use case diagram for “Print a model”.

Name	Print a model
Actors	Security expert, software developer
Priority	Medium
Entry conditions	A model is opened and shown in the system window.
Flow of events	<ol style="list-style-type: none">1. The actor selects the print option for the current model.2. The system presents the user with a print dialog, including option for print preview and page setup.3. The actor specifies his selections.4. The actor submits for printing.5. The system creates a print job, and sends this to the specified printer.
Alternative flow	<ol style="list-style-type: none">3a. The actor wishes to preview the print job, and selects the print preview option.3b. The system creates a visual representation of the print job and shows this on the screen.
Exception	<ol style="list-style-type: none">3x. The actor wants to abort the printing. He closes the print dialog. Use Case ended.

Table 4.10: Use case for “Print a model”.

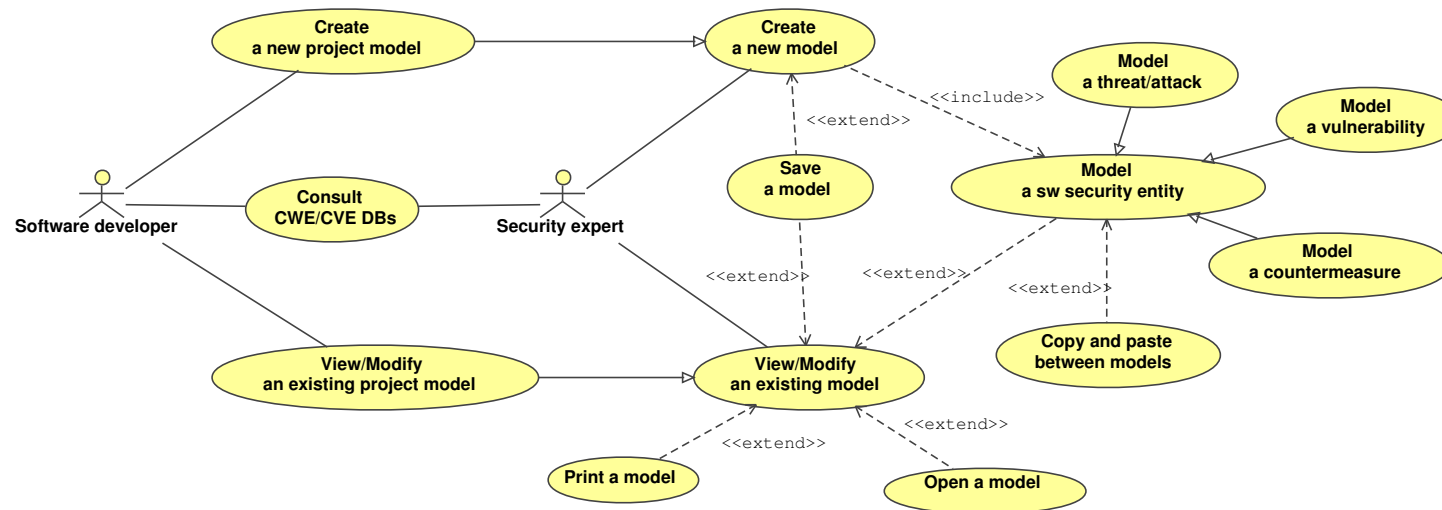


Figure 4.11: Global use cases diagram.

4.3.2 Some use case scenarios

A usage scenario describes a real-world example of how one or more people or organizations interact with a system. They describe the steps, events, and/or actions which occur during the interaction. Usage scenarios can be in high detail, indicating exactly how someone works with the user interface, or reasonably high-level describing the critical business actions but not indicating how they are performed. The detail level, in the present context, depends on the amount of information needed to well understand the specified requirements. The more an interaction is detailed, the more requirements are needed for that specific feature. Moreover, Figures from 4.12 to 4.16 are scenarios-based diagrams depicted using the UML sequence diagram notation. Sequence diagrams may be utilized at this stage for describing interactions between objects by means of messages [32]. They provide a dynamic vision of a system, showing graphically the scenarios that may exist at run-time when objects interact to achieve some goals. They display the progression of the time along the vertical axis, highlighting the temporal sequence of the exchanged messages among objects. The reason for having used such a notation is simply to favor a more pleasant reading to the customer during this delicate phase. Analysing such diagrams, he or she can confirm if the specification successfully capture the expected behaves. The scenarios have been thought within a story context and here below narrated.

Scenarios Description

WeLikeMoney Inc. is a company working with software development for bank systems. As a consequence, they are of course extremely thorough when it comes to security in their systems. WeLikeMoney has just started using SeaMonster, a new open source application for software security modeling. The scenarios below describe some of their first experiences using this program.

Scenario 1

Actor Fritz Cyver, security expert

Description Fritz Cyver is hired as a general security expert in WeLikeMoney. He has just discovered a new vulnerability that needs to be taken into account, and he decides to add information about it to SeaMonster. He starts the application and chooses to create a new security model, via a VCG. He is pleased to see that the regular VCG components are automatically shown in the palette, so he can start modeling right away. Cyver add the vulnerability and maps it to its causes. He also adds a link to its entry in the CVE database. As Fritz has created a compound node in the VCG, he also creates another diagram where he describes the model of the compound node. He saves, pleased with his work, and closes SeaMonster. (Figure 4.12)

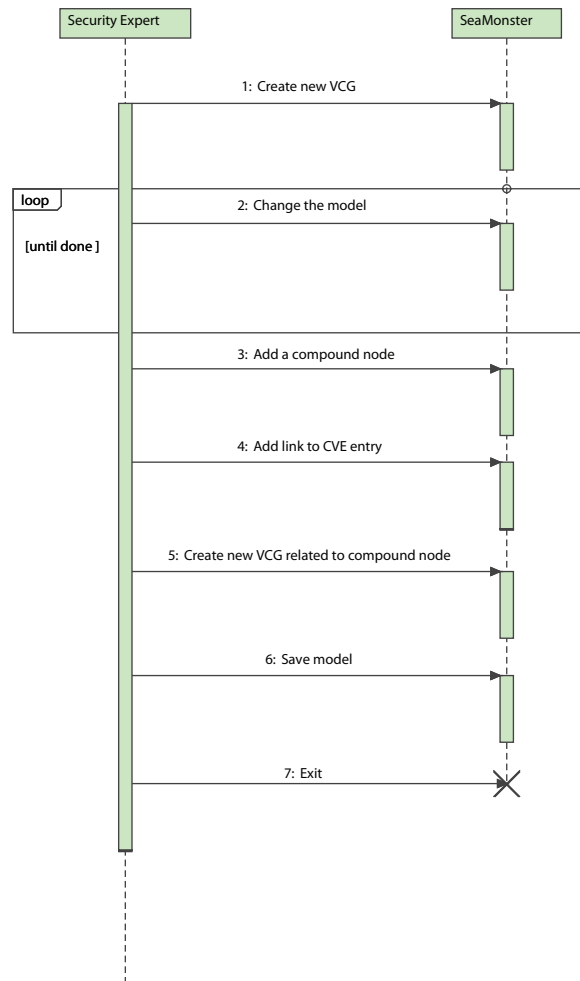


Figure 4.12: Sequence diagram for scenario 1.

Scenario 2

Actor Simon Ecure, Another security expert

Description Simon Ecure hears from Fritz that some new vulnerabilities have been discovered. He decides to explore these vulnerabilities by creating attack trees that show ways that an attacker could exploit the vulnerabilities. He starts up SeaMonster, and he chooses to create an attack tree diagram based on the model used in Cyver's VCG diagrams. He also opens those VCG's, and studies the information contained in them. From each vulnerability, he finds links to information in CVE. Giving these a read-through (from within SeaMonster), he gets a good knowledge base for his modeling. He creates the attack tree, and finds that SeaMonster is actually calculating costs and features in the nodes automatically. This lets him focus on the other aspects of the diagram, without having to think about this routinely calculations. Among other things, he adds a closer description of the main attack, so that people looking at the diagram will fully understand it. As the attack tree is a bit messy, Simon uses the arrange function on the diagram before he saves and leaves. (Figure 4.13)

Scenario 3

Actor Chris Rooks, Yet another security expert

Description Chris is in a team working on WeLikeMoney's new banking system. There are large demands what concerns security, and Chris is put on the task of identifying security risks and suggesting countermeasures for these. He goes through the VCG's and attack trees already present, and consider whether any of them applies to his case. When he comes across the diagrams from Cyver and Ecure, he decides that these threats should be taken into account. He therefore studies the diagrams a bit closer. This is made easy by the extra information contained in the diagrams. Having thought this through, he creates a misuse case diagram related to these. As the threats from the attack trees are already present, he only needs to focus on actors/crooks and countermeasures. After some work, he decides this needs to be discussed on the next meeting. After saving the diagram he e-mails this to the rest of his team. He also prints a copy for himself and exports the diagram to an image. (Figure 4.14)

Scenario 4

Actor Dave Elover, Software developer

Description Dave Elover, a software developer working in the Rooks' team, received a mail from Rooks with an attached file containing the misuse case diagram subject of the next meeting. He decides to take a look at it, so he opens the file with SeaMonster and selects the misuse case diagram. Then, he selects the VCG diagram to get an idea about the possible causes that could lead to that vulnerability. Moreover, he decides to consult the remote CVE contents related to a particular vulnerability. At the end, he prints the diagrams and quits the application. (Figure 4.15)

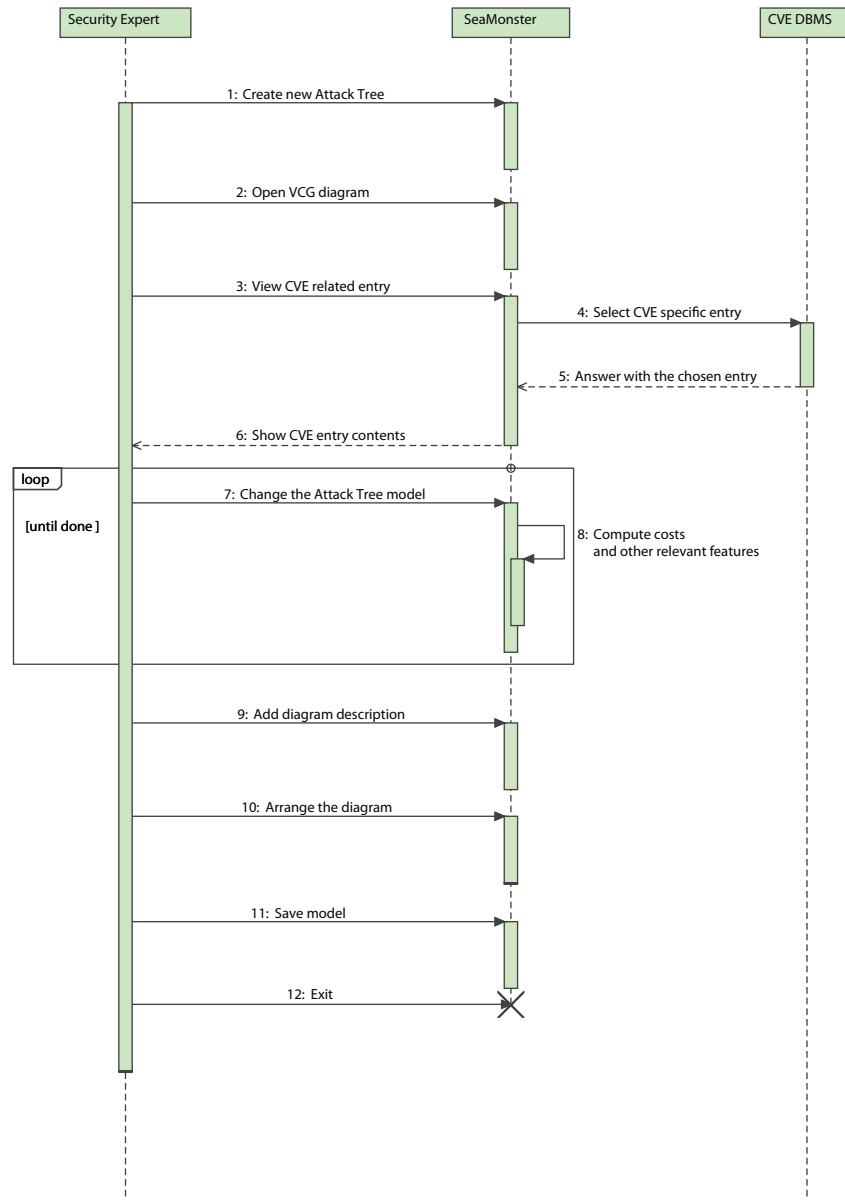


Figure 4.13: Sequence diagram for scenario 2.

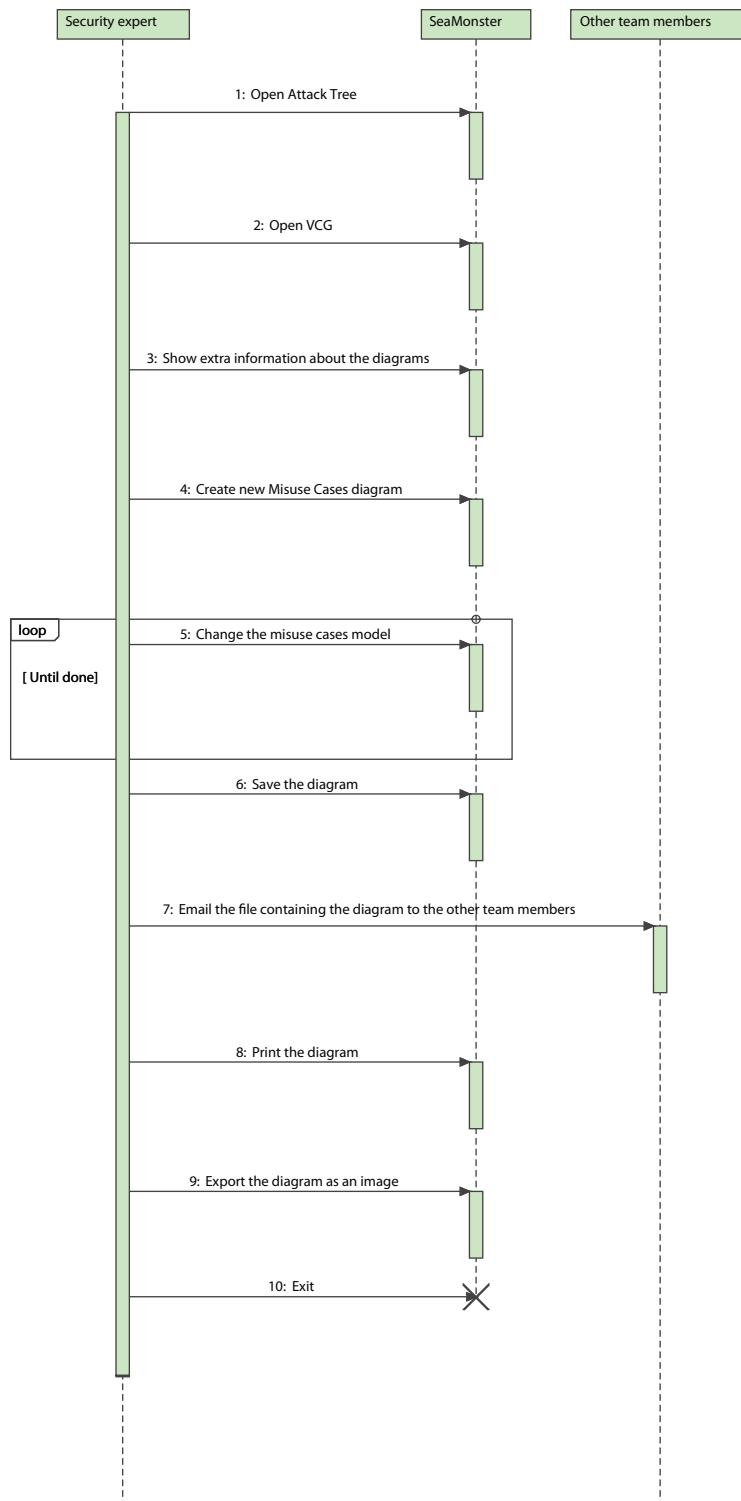


Figure 4.14: Sequence diagram for scenario 3.

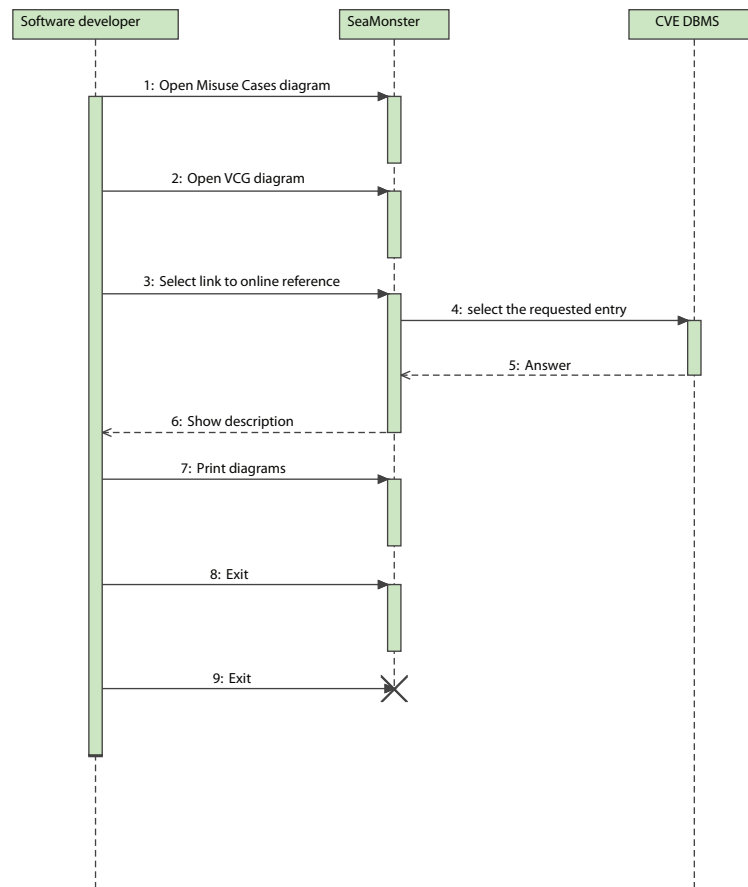


Figure 4.15: Sequence diagram for scenario 4.

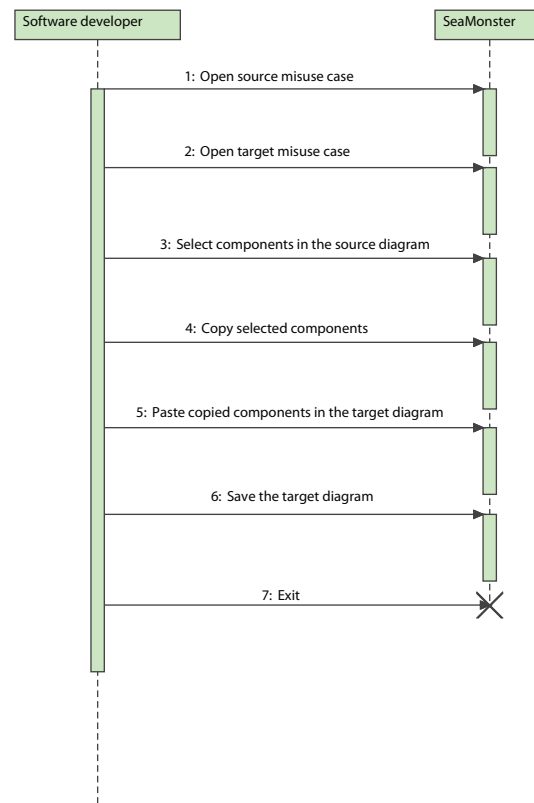


Figure 4.16: Sequence diagram for scenario 5.

Scenario 5

Actor Dennis Ploy, Another software developer

Description Dennis Ploy, a software developer working in the Rooks' team, received a mail from Rooks with attached a file containing the misuse case diagram subject of the next meeting. He decides to take a look at it, because is working on some system's functionalities, subject to some misuse described there. So he opens the file with SeaMonster and selects the misuse case diagram. Moreover, he is studying some misuse case diagrams too, and the Rooks' diagram contains some preventions useful to his work. So he selects such prevention use cases and, after having open his diagram, he copies and pastes them into his work. At the end, put every needed relation, he saves his diagram and quits the application. (Figure 4.16)

4.3.3 Functional requirements and their measurements

The following is the list of functional requirements elicited from the given directives and from the use case analysis of the previous section. They have all been refined and added to this section for giving a better understanding of

ID	Name	Refined description	Measurement	Priority
EF.01	Import external definitions	The CWE/CVE databases have an export functionality. Content of these databases shall be possible to import into the application for identification purposes.	Trying to import some existing definitions within the application.	L
EF.02	Link to external resources	The application must provide functionalities for linking security model components to external resources.	Trying to create a diagram and to link some external resource to some of its components.	M
EF.03	Import external information	The application must provide functionalities for displaying information related to external resources.	Trying to display information related to some component within the application.	M

Table 4.11: Functional requirements group EF.

the whole application. Every requirement has its own priority represented by a letter. Such priority is expression of the importance of the requirement to the customer's eyes with respect to the others, and it is quite important to define a list of prioritised goals to be achieved by the implemented application.

A first set of functional requirements, summarized in Table 4.11, are aimed at defining functionalities of the application related to external resources and systems (EF). The second list of requirements in Table 4.12 are aimed at refining the concept of "intuitive GUI" (IGF) expressed by the customer. Functional requirements concerning the diagrams follow in Table 4.13. They are classified in the following groups.

- ATF group, for attack trees functional requirements
- VCGF group, for VCGs functional requirements
- MCF group, for use/misuse case diagrams functional requirements
- MF group, for more general requirements fitting in every kind of diagram

Finally, the last group of functional requirements are about storage features (SF) and they are listed below in Table 4.14.

ID	Name	Refined description	Measurement	Priority
IGF.01	Zooming functionalities	The application must be able to zoom in and out over the modeling board.	Trying to zoom in and out a diagram.	M
IGF.02	Outline window	The application must provide an outline window for better moving within the modelled diagram.	Taking a complete tour over a possible huge diagram.	M
IGF.03	Automatic arranging components	When the diagram is composed of a considerable number of components, managing an ordered diagram could become harder. The application must provide the possibility of automatically arranging the components within the board.	After having create a messy diagram, trying to arranging the components.	M
IGF.04	Several diagrams open simultaneously	The application must be able to have several models open contemporaneously.	Trying to open several diagrams.	H
IGF.05	Adding notes	The application must leave the user free to add textual notes to the diagram.	Try to comment a digram leaving some notes.	M
IGF.06	Copy and paste	The application must allow copy and paste of components between compatible diagrams.	Try to copy and paste some components between different diagrams.	M

Table 4.12: Functional requirements group IGF.

ID	Name	Refined description	Measurement	Priority
ATF.01	Models of threats/attacks	The application must be able to model threats/attacks with Attack Trees (See Section 3.5.2).	Try to create a new Attack Tree diagram.	H
ATF.02	Automatic cost calculation	The application must be able to automatically calculate the cost of an attack.	Comparing automatic computations with correct precalculated results.	L
ATF.03	Marking resolved attack paths	The application must allow the user to visibly mark resolved attack paths.	Trying to add marks on a attack tree diagram.	L
ATF.04	Attack Trees notation	The application must implement a specific Attack Trees' notation. Such a notation will be precisely specified in the next chapter.	Comparing the implemented notation with the one defined in literature.	M
ATF.05	Attack Trees components' properties	Goals must have the following properties: cost, boolean, name and description.	Check the presence of the above properties using the application.	M
ATF.06, VCGF.02	Preventing cycles	The application must be able to automatically detect and forbid cycles in the diagram.	Trying to force a cycle in the diagram.	M
VCGF.01	Models of vulnerabilities	The application must be able to model vulnerabilities with VCGs (See Section 3.5.1).	Try to create a new VCG diagram.	H
VCGF.03	VCG notation	The application must implement a specific VCG's notation. Such a notation will be precisely specified in the next chapter.	Comparing the implemented notation with the one defined in literature.	M

Table 4.13: (continued)

ID	Name	Refined description	Measurement	Priority
VCGF.04	VCG' properties	Every VCG component must have the following properties: name and description.	Check the presence of the above properties using the application.	M
MCF.01	Models of counter-measures	The application must be able to model counter-measures with Use/Misuse case diagrams (See Section 3.5.3).	Try to create a new Use/Misuse Case diagram.	M
MCF.02	Use/Misuse Case diagram notation	The application must implement a specific Use/Misuse case diagram's notation. Such a notation will be precisely specified in the next chapter.	Comparing the implemented notation with the one defined in literature.	M
MCF.03	Use/Misuse Case diagrams properties	Every Use/Misuse Case diagrams's component must have the following properties: name and description.	Check the presence of the above properties using the application.	M

Table 4.13: (continued)

ID	Name	Refined description	Measurement	Priority
MF.01	Specialisation and generalisation	It must be possible to specialize and generalize objects allowing the creation of objects containing several sub-objects. In particular, using use/misuse case diagrams the user must be allowed to add cases within the system area, while using VCG diagrams he must be allowed to create conjunction nodes containing simple and compound nodes.	Trying to create nested objects within all the models where it should be allowed.	M
MF.02	Pluralism of the diagrams	The application must allow the user to associate more then one diagram to a security model.	Trying to create several diagrams of different type within the same model.	M
MF.03	Visible properties	The application must visibly manage components' properties, if any.	Try to view and modify properties of components belonging to the several diagrams.	M

Table 4.13: Functional requirements groups ATF, VCGF, MCF and MF.

4.3.4 Security functional requirements

One of the project goals is to emphasize the necessity to consider security aspects in the early phases of the process development. This section discusses some security issues related to the SeaMonster application. SeaMonster is a modeling stand-alone application. This means that most of the usual security requirements (security audit, cryptographic support, user data protection, identification and authentication, etc...) are not taken into account, because they

ID	Name	Refined description	Measurement	Priority
SF.01	UML 2.1 meta model	Data should be stored according to the UML 2.1 metamodel.	Not defined.	L
SF.02	Textual representation	The graphical security models drawn in the application must be converted to an XML textual representation, describing the model completely, stored in a database (or similar functionality).	The output XML file of the application must be well-formed and valid. This has to be assessed by means of conforming and validating parsers.	M
SF.03	Save and open	Models drawn in the graphical editor must be possible to save, and open at a later stage.	Trying to repeat the (create-)save-open loop a good number of times with some changes applied.	H
SF.04	Provident creation	When creating new diagrams, if some relevant information about the model are stored yet, the application must allow the user to start the draw from them.	Not defined.	M
SF.05	Exporting as image	The application must allow the user to export diagrams as image. In particular, it must provide the possibility to save the image both in bitmap and in vectorial format.	Trying to export a diagram as an image using all the possible formats. Then trying to open the saved files with suited editors in order to evaluate the quality of the image.	M
SF.06	Printing with preview	The application must allow the user to print diagrams giving him the possibility to preview the job.	Trying to print a diagram.	M

Table 4.14: Functional requirements group SF.

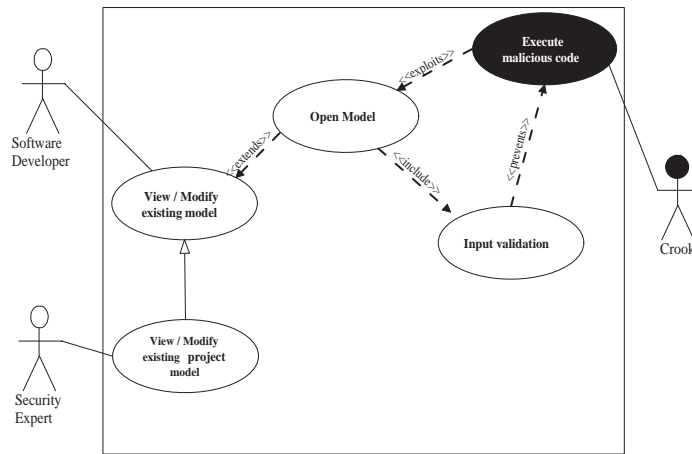


Figure 4.17: Misuse case of “Open a model”.

are not interesting in this kind of project. Anyway, there is possible to integrate some security requirements, in order to show how this activity can be embedded in the development work flow. Figure 4.17 shows a misuse case diagram of the “Open a model” case. As shown, the crook may corrupt the input file by injecting malicious code to be executed by the JVM. This kind of attack can be prevented by adding the use case “Input validation” in the opening activity flow. The latter use case is described in Table 4.16. Table 4.15 presents the modified “Open a model” use case. In the described example, the exploited use case is marked with a high priority. This helps understanding that to introduce reasoning about security issues at this stage is an effective way to avoid threats, that could affect the most important functionalities of a system. From the reported misuse case diagram, requirements listed in the SECF group in Table 4.17 may be inferred.

4.4 Non-functional requirements

In addition to functional requirements, which are used to define the external behavior of the S2B, there are non-functional requirements, also called quality attributes. These are not directly related to the software functionalities nor to the written code. However, this does not mean that design and realization phases are not sensitive about non-functional requirements, since these are able to establish some characteristics of the mentioned development phases. For instance, some platform and portability constraints could lead the team of developers to adopt a particular programming language rather than others. Such requirements are quite important for conveniently fulfilling the customer’s interests. Non-functional requirements express, usually in words, software properties that must hold. They may be defined with different priority levels, and the specification needs to describe any such attributes that the customer requires. Non-functional requirements are grouped in implementation (INF), usability (UNF), packaging (PKNF), platform (PNF), and legal requirements (LNF). For each mentioned requirement, a fulfillment measurement is proposed.

Name	Open a model
Actors	Security expert, software developer
Priority	High
Entry conditions	A security model has been created.
Flow of events	<ol style="list-style-type: none"> 1. The actor chooses to open a model. 2. The system lets the user specify which file to open. 3. The system tries to assess the validity of the input file selected. See the use case in Table 4.16. 4. The actor chooses the wanted view of the model he wants to open. 5. The system opens the file in the view contained in the model file.
Alternative flow	<ol style="list-style-type: none"> 4a. The desired model view has not yet been created, so the user wants to initiate it. 4b. The actor chooses to initiate a diagram of the wanted type. 4c. The system lets the user specify on which model view he will base the diagram. 4d. The system creates a new diagram, already incorporating relevant information about chosen security model. The use case ends.
Exception	3x. The actor has specified an invalid file (either not existing or not containing a model). The use case ends.
Exit conditions	The diagram, containing the chosen view of the security model, specified by the actor, is opened and shown in the system window.

Table 4.15: Use case for “Open a model”.

Name	Input validation
Actors	Software developer, security expert
Priority	Medium
Entry conditions	The application is running and the actor selects a file to be opened.
Flow of events	<ol style="list-style-type: none"> 1. The system validates the input file based on a predefined structure. 2. The system tries to continue the opening operation.
Exception	1x. The input file is not conformed to the specified structure. The system avoids to open the file.

Table 4.16: Use case for “Input validation”.

ID	Name	Refined description	Measurement	Priority
SECF.01	Validate XML input	The application must validate the XML input referring to a predefined XML schema.	Trying to open some model with modified content.	M

Table 4.17: Functional requirements group SECF

ID	Name	Refined description	Measurement	Priority
INF.01	Eclipse	The application should be based on the Eclipse framework.	Customer acceptance test.	H
INF.02	GMF	The Eclipse Graphical modeling Framework (GMF) should be used to develop the graphical editor.	Customer acceptance test.	H
INF.03	EMF	The Eclipse modeling Framework (EMF) should be used as the modeling framework.	Customer acceptance test.	M
INF.04	Java	Model logic should be programmed in Java.	Every application modules must be Java compatible.	H

Table 4.18: Implementation non-functional requirements.

4.4.1 Implementation requirements

The security modeling application must be platform independent. This means that it has to be developed in a language not strictly associated with a specific operating system. Nowadays, there are plenty of object-oriented programming languages that could reach this objective. The one required by the costumer is Java, which is thought of as one of the best known in the field of cross-platform development [22]. Because, for running a Java program, a virtual machine is needed. A virtual machine represents a software interface that the application has to use. Also because GMF and EMF are involved in the process development, the right version of the Java Virtual Machine (JVM) has to be chosen. GMF and EMF are able to produce Java source code automatically for the models and the editor environment, and that source code is Java 1.5 compatible [88]. For this reason, the team decided to go for the JVM 1.5 version, instead of the newer 1.6, which is not as well-established.

ID	Name	Refined description	Measurement	Priority
UNF.01	Intuitive GUI	The GUI should be based on the Eclipse pattern, allowing users to find a well-known environment.	Submitting a set of simple tasks to a predefined group of users with a usability test. Users' comments and obtained outcomes must be analysed.	H
UNF.02	Screen layout	The application should present an intuitive and well-structured GUI, with a palette, a model selection pane, and a drawing board.	Customer acceptance test and usability test. The measure task may be improved submitting the application to an additional group of testers.	H

Table 4.19: Usability non-functional requirements.

ID	Name	Refined description	Measurement	Priority
PNF.01	Platform independent	The application must be independent of specific operating systems.	Testing the system on the following operating systems: Microsoft Windows XP, Microsoft Windows Vista and Ubuntu Linux.	H

Table 4.20: Platform non-functional requirement.

4.4.2 Usability requirements

Non-functional requirements other than being tightly related to software requirements described in Table 4.12, are also undoubtedly connected between themselves. Together they define not just that a comfortable and well-known environment is required, but also what is supposed to be found inside such an environment; giving a definition, in terms of components, of intuitive and well-structured GUI.

4.4.3 Packaging requirements

The packaging requirements is shown in Table 4.21.

ID	Name	Refined description	Priority
PKNF.01	System re-sources	The application must not require more than 128 MB of RAM, and 100 MB of hard drive space.	M

Table 4.21: Packaging non-functional requirements.

4.4.4 Software license and legal requirements

There is a large variety of software licenses available [30], but in order to choose one, some characteristics from the license must be fulfilled. The following lists the two most important characteristics requested by the software license, coded to facilitate referencing them.

SL.1 - The license supports the users of the software to run, copy, distribute, study, change and improve it. (Adopted from GNUs definition of the “free software” philosophy [35]).

SL.2 The released program requires all modified and extended versions of the program to be free as well. (Adopted from GNUs definition of “copy-left” [36].)

The GMF and EMF framework (as well as dependencies) are released under the Eclipse Public License (EPL). This license allows further work on the released software to use any license, as long as the EPL licensed work keeps the EPL license [23]. In other words, the implemented code in SeaMonster can be released under any license as long as the EPL-based parts are released under EPL. This leads to another required characteristic, shown below.

SL.3 The license must allow dependencies within the current software to be released under any other license (at least under the EPL).

As long as these characteristics are held, the license can be chosen for use in release with SeaMonster. At a first glance, General Public License (GPL) seems relevant since it is intended to guarantee the freedom to share and change all versions of a program [33, 76]. However, to release software under the GPL license, any dependencies must also be based on the same license. Since this is in direct conflict with the EPL, it must be rejected.

Another license released to handle this issue is the Lesser General Public License (LGPL) [34]. Software released under LGPL can be built upon other applications or libraries, without being considered derivative work. This complies with SL.3. In addition, this license allows changes done to the released software as long as the new code is released under LGPL or GPL. Furthermore, LGPL requires the source code to be released with the application. Together, this implies compliance with SL.1 and SL.2, and the selection is complete. There might be other licenses available that satisfy the required characteristics, but that is an issue considered less important. For these reasons, the EPL based software keeps the EPL license, while LGPL is applied to all additions to the code. The software license requirements and the use of Sourceforge as project repository are covered by the LNF.01 requirement shown in Table 4.22.

ID	Name	Refined description	Measurement	Priority
LNF.01	Open source	The project will be considered an open source project and available to a wider community for further development and suggestions. Use of Sourceforge as the project repository.	Assessing if implemented modules, used frameworks, and the project repository are compatible with the adopted license.	H

Table 4.22: Legal non-functional requirement.

4.4.5 Relationships with business requirements

In order to verify the fulfillment of business requirements, relating them to software requirements is very important. As a matter of fact, when some software requirements related to a business requirement are fulfilled, then the fulfillment of the business requirement is more likely. For representing relationships between the two groups of requirements, a traceability matrix is used. The traceability matrix is a well-known technique in the world of requirements engineering. The two dimensions of the matrix represent the two groups of requirements. For the sake of readability, the groups of software requirements defined in the previous section are considered, instead of every single requirement. So, for instance, this means that within the traceability matrix, requirements ATF.01, ATF.02, and so on are replaced by “ATF”. If a relation between a couple of coordinates exists, then it is signed by a mark in the respective cell. For example, the mark between BR.02 and ATF, underlines that the relation exists between the business requirement BR.02 and some, in this case functional, software requirements within the ATF requirements group. More precisely, fulfilling requirements in ATF could lead to fulfilling BR.02 as a consequence. Business requirements are listed in Section 3.2, while software requirements are presented within Tables 4.11, 4.12, 4.13, 4.14, 4.18, 4.19, 4.20, 4.21 and 4.22 and summarised in Appendix C. Table 4.23 depicts the traceability matrix in question.

From the above matrix, it appears clearly that no business requirement is related to requirements in groups PNF and PANF. This is due to the nature of these groups of requirements. They are just related to technical properties of the software, which are not strictly and directly related to any business requirement. Another consideration could be made about the business requirement BR.01. It states that the application should help moving the security assessment of a system to an earlier phase than usually done today. Even if the application fulfills all of the software requirements, it is not enough to say anything about the fulfillment of this business requirement. More likely, a good estimator of such a business requirement may be statistical information about its employment, which would be even better after a good promotion of its use. The vertical profile reveals that requirements related to modeling capabilities (i.e. requirements

	BR.01	BR.02	BR.03	BR.04	BR.05	BR.06
EF	✓				✓	
IGF	✓	✓				
ATF	✓	✓	✓		✓	
VCGF	✓	✓	✓		✓	
MCF	✓	✓	✓		✓	
MF	✓	✓			✓	
SF	✓	✓		✓	✓	
INF						✓
UNF	✓	✓				
PNF						
PANF						
LNF						✓

Table 4.23: Business requirements traceability matrix.

in groups ATF, VCGF, MCF and MF) are strongly linked to the business requirements. This underlines once more the importance of pressing forward with them. A horizontal view of the table shows which business requirements would be more supported by an eventual good turn out of the application. To be more precise, they are BR.02 and BR.04. In particular, such requirements are oriented towards leading software security discussions on a common playground: a standardised notation for a better communication between software developers and security experts.

4.5 Summary

First, a general overview of the system has been given, starting from an evaluation of the most relevant actors and of the external interfaces requirements. This has raised up the central importance of the user interface that has to present the main modeling views, mentioned and described in Section 3.4, in a way that it is possible for the user to manage them easily and intuitively. Then, elicitation and analysis of the software requirements, along with their specification, have been considered as the heart of the actual documentation part. Functionalities have been analysed and described by means of use cases and scenarios. UML has played a fundamental role in making the communication with the customer easier and more effective, as well as simplifying the connection to the next design stage. On the other hand, non-functional requirements are studied looking at their impact on the team's work flow, in terms of activities that they are able to require in order to be properly fulfilled. Finally, software requirements have been related to business requirements with the purpose of establishing the main relationships between them. What has been noticed is that from the point of view of the business requirements' fulfillment, requirements associated with modeling techniques are much more interesting than other requirements. This suggests to consider them with great care during the whole development process.

Chapter 5

Design

5.1 Introduction

This chapter includes four main sections. The first one is about the system architectural design. Starting from the SeaMonster functional model, it describes at a high level the chosen architectural solution. The second section presents a more detailed description of the several components that constitute the application. The third section goes through the user interface design, illustrating the choices made and how the SeaMonster graphical front-end will appear. Finally, the last section lists which requirements are not taken into account in this design. As this is only a prototype, these will be taken into consideration at a later stage.

5.1.1 Purpose

During the design phase, the system will be structured at different detail levels. Starting from a high level definition of the system architecture, a more detailed design activity will be explained, dividing the application into modular components and defining their interfaces in an accurate fashion. Each component in turn could be decomposed into smaller sub-components. Other than describing the system in terms of components, their interfaces, and relations among them, this part of the documentation will register all the significant decisions about the design and the motivations that led the team to make such decisions. This is important for future changes to the software that may lead to modify the architecture. Also in this case UML will be used, that is the de facto standard for documenting software architectures, especially for systems designed according to an object oriented style.

5.2 System architectural design

5.2.1 Functional model

Before starting with describing the system architecture, the functional model of the application is summed up. The UML diagram in Figure 5.1 presents the whole set of functionalities described in the previous chapter.

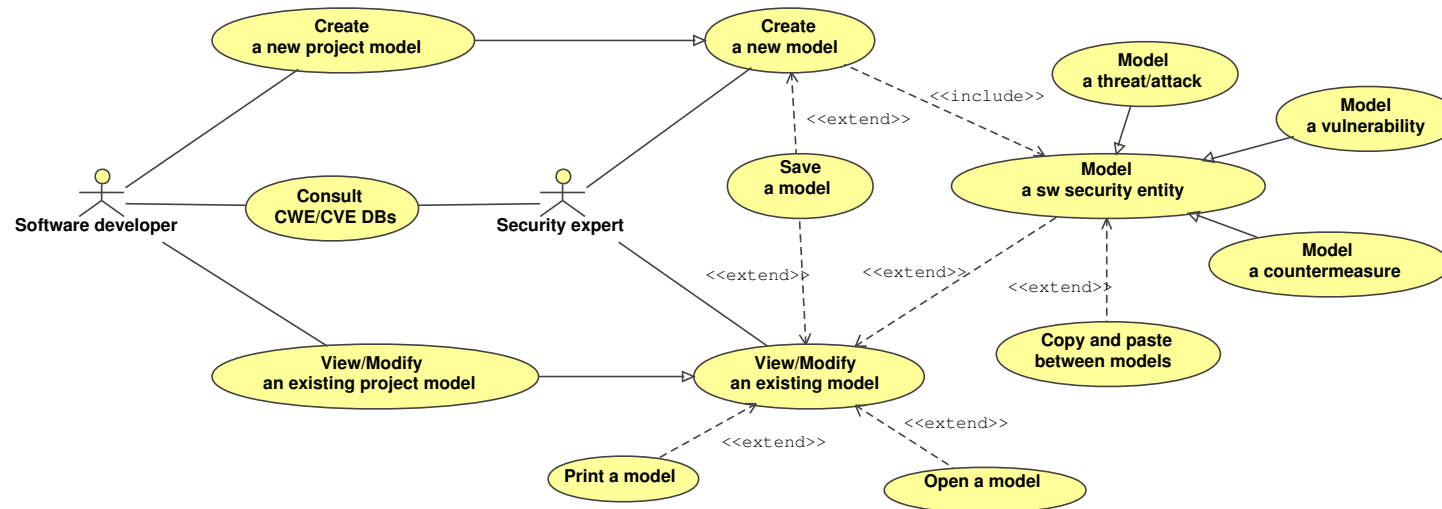


Figure 5.1: Global use cases diagram.

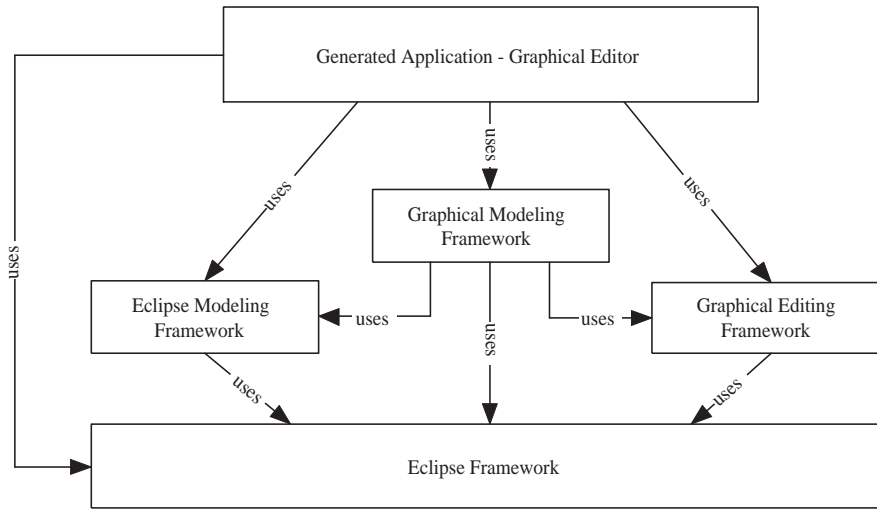


Figure 5.2: Top-level GMF architecture.

First a brief recall on the concept of security model as intended in this report. Security model, or simply model, is a term used to summarize a set of diagrams aimed at describing different views of a particular software security issue. The software developer and the security expert share almost every possible use case that involves a model. They can create, open, view, modify, save, and print a software security model. The only difference between them, is on the perspective that they have about the model. While the security expert is working on security models without a secondary goal in general, the software developer does that within the context of a project. So, from this point of view, the latter is considered a specialization of the first one. Furthermore, they also share the possibility to consult online resources, such as CVE/CWE databases. Every time the actor (i.e. either the software developer or the security expert) decides to model something, he may decide to model a threat/attack, a vulnerability, or a countermeasure. During the modeling activity he is allowed to copy and paste components from other compatible diagrams.

5.2.2 Chosen system architecture

In accordance with the requirements, this system will be built using the eclipse Graphical Modeling Framework (GMF). This means that a lot of the system architecture is defined by this framework. To get a better understanding of what this means, an overview of the architecture of the GMF framework is given.

GMF architecture

GMF is a framework that works as a bridge between the two eclipse frameworks Graphical Editing Framework (GEF) and Eclipse Modeling Framework (EMF). The relation between all these frameworks is shown in Figure 5.2. To better understand GMF, a brief description of both EMF and GEF are shown in the following paragraphs.

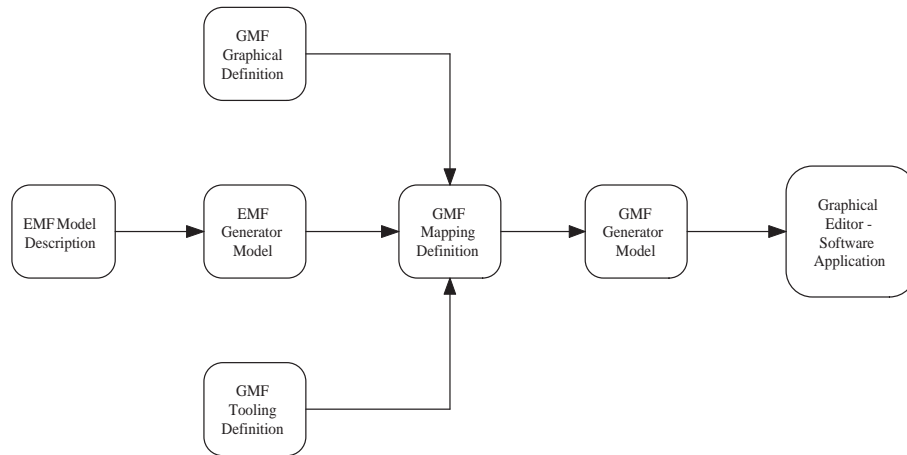


Figure 5.3: Basic GMF development process.

EMF is a modeling framework and code generation framework for building tools and other applications based on a structured data model. It helps develop a data model that can be used within the application, and it also generates the classes containing this model.

GEF is a framework that contains a lot of standardized tools/features to be used in graphical editors. It also provides a layout for rendering graphics. GEF applies a Model View Controller (MVC) architecture in the editor, which makes it easy to make changes to the data within the model from the view.

GMF, as already stated, is based upon and uses these frameworks. While using EMF and GEF through GMF, one can easily combine the two to create a feature-rich editor for modeling. Through GEF one then gets a graphical editor that can be used to work on the model made through EMF. An application generated through GMF is therefore used to create and modify diagrams that are related to the EMF-generated model.

All of these frameworks are based upon the eclipse platform, which then of course also will serve as a basis for the application developed in this project.

For a more in-depth description of the architecture of GMF-based applications, refer to the eclipse GMF documentation [24].

Process of generating a GMF Application

As the implementation process differs quite a lot from what is usual when creating a GMF-based application, it is appropriate to give an overview of this process. This section explains the basics of this, including a description of the file types involved in the process. These are model files that describe the data model, diagram components, diagram tools, mappings between the last three and of course how the editor is supposed to be. All these files and their relations are shown in Figure 5.3.

To explain the figure a bit further, here is how the standard process basically works. First you create a definition for your data model. This model should store all meta information to be contained in your diagram, e.g. data, and relations between the data. Having done that, you make a graphical definition

for the diagram. This is where you define how everything is supposed to look, such as rectangles, ellipses, polylines, etc. In other words, the graphical definition model contains all information about the figures that can appear in the diagram modeled. The main purpose of the tooling definition model is to define what kind of tools are to appear on the palette of the editor. All tools for diagram components are defined here.

The order of how you create the last three models does not really matter. You must have done these three before you can continue, however. The mapping definition will hold references to these. This model is where the relations between the data model, the graphical definition, and the tools are made. You define what parts of the data model (classes and relations) are to be included in the diagram. For each of these, you select a graphical representation from the graphical definition and a tool from the palette in the tooling definition that corresponds to this component.

Finally, you create a GMF generator model, which is based on your mapping definition. While the mapping definition and its dependencies define how the diagrams should work, the generator model is from where you generate the code for the application. In this file, there are a lot of settings that need to be considered, e.g. how the editor is supposed to work.

When the generator model is done, you can generate the editor code. This code will be dependent on the code from the data model, so you also need to generate model and edit code from the EMF generator model. When this is done, your editor can be run as a program. At this point, it is possible to apply any favorable changes to the code, in order to personalize the application. This is also a requirement for adding any features not supported by GMF.

5.3 Detailed description of components

5.3.1 Data model

This component is working as a basis for the editor. It contains all data to be stored by all diagram types. All concepts from the security model types to be used in the application needs to be incorporated, that is VCG, attack trees and misuse case diagrams.

EMF modeling

When developing a GMF-based modeling tool, one needs to model the data model using EMF, as mentioned above. This means that the data to be contained in the tools diagrams need to be described in some way. EMF supports doing this in three different ways. These are using annotated Java classes, using a UML diagram tool, or simply coding directly in the XML representation of the model. In this project the UML approach is used, and the model is created using eclipse's own editor for EMF models. This editor is in fact itself made from GMF, so this also gives a little more insight in how a GMF editor works.

After defining the data model and added it to an EMF generator model, one can generate code for it. This code includes the most essential methods needed for GMF editors, and will also work without modification. It is very basic, however, and if more advanced functionality is wanted, one needs to modify the code to incorporate this.

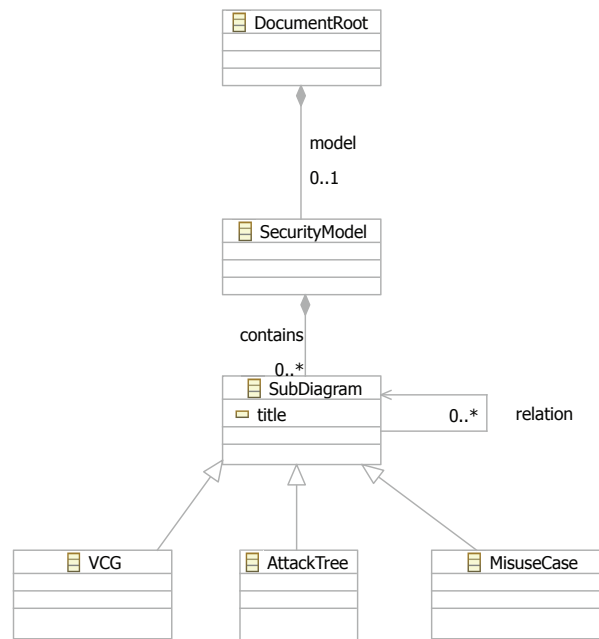


Figure 5.4: Top level class diagram of the applications data model.

The model

This is the component forming the basis for the diagrams. It stores all relevant data to the security models. In Figure 5.4 you can see a top-level view of how the model is organized. The main class is `SecurityModel`, which represents a set of diagrams. For each of these, you may have several diagrams of each type, represented by `AttackTree` (for attack trees), `VCG` (for VCGs) and `MisuseCase` (for misuse case diagrams). Because of readability issues, the data model has been split into four diagrams, one toplevel and one for each of the specific diagram types.

The toplevel diagram will be a tool for the user to organize related diagrams of the other types in a single data model. This allows the user to create diagrams showing several aspects of the same security model, without having to keep track of all this himself. The diagram contains components representing misuse case diagrams, vulnerability cause graphs and of course attack trees. Once the diagram has been created, the user will be able to create new diagrams describing each of these components further.

The design supports adding features for direct links between components, although this is not directly incorporated in the current design. This is a feature that has to be added at a later stage.

The attack tree part is perhaps the simplest of the parts. It is shown in Figure 5.5. The attack tree diagram can contain several components, represented by `AttackTreeComponent`. These components can either be a goal (represented by a `GoalNode`) or an AND node (`AndNode`). An `AndNode` can of course not be a successor of another `AndNode`. This last design choice makes sure that all

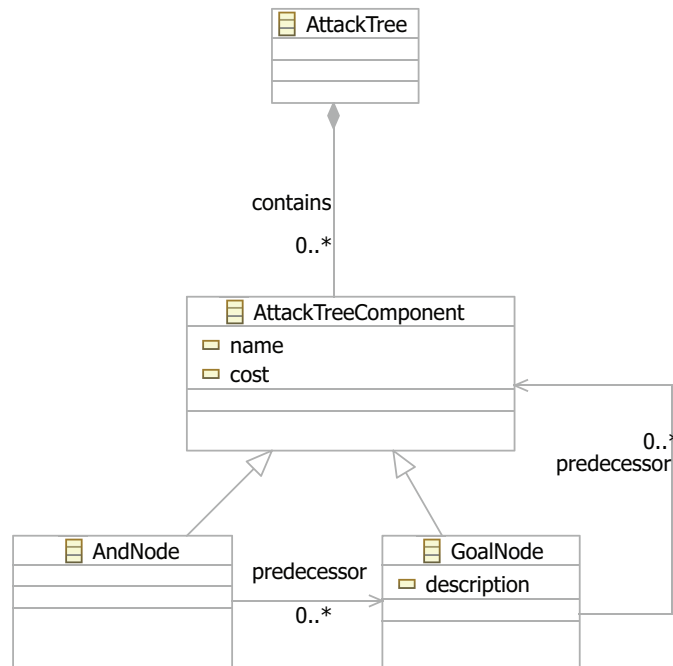


Figure 5.5: Class diagram for the AttackTree part of the applications data model.

AndNodes are connected only to GoalNodes in the diagram. This is an important feature because of GoalNodes both representing causes and consequences, while the AndNodes are representing collections of causes leading to a single consequence.

Another important features of attack trees is the components representing a tree structure through their relations, that is, no relations make up a cycle of components. It is then possible to calculate features of a component based on the features of all components relating to this component. The design of the Attack Tree in SeaMonster is setting some constraints on creating attack tree relations, to prevent the user from creating cycles in their trees. The intention is that it is then easier to extend the application later by including automatic calculation of features like component cost. This is a feature set aside in the current design, to keep the focus up on the other aspects of the application.

The cycle prevention algorithm will be called whenever the user is trying to create a relation in the diagram. It takes as parameters the two components in the relation, and returns true/false depending on whether the two components can be related in that manner or not. The user is then only allowed to make relations between components accepted by the algorithm, and attempts at other relations will be ignored. Pseudo-code for the cycle prevention algorithm is shown in Listing 5.1.

```

1 cyclePrevention(component 1, component 2)
2 Q <- {Component 2}
3 while Q not empty:
4     Z <- removed first element of Q
5     if Z = component 1:
6         a cycle exists - return false
7     add all predecessors of Z to Q
8 the components can be related - return true

```

Listing 5.1: A method called when the user wants to make “component 2” a predecessor of “component 1”.

It can be mentioned that this algorithm does not constrain the component graph to being a tree, but prevents cycles and thereby forces the constraint that it is a directed acyclic graph. This is sufficient to make automatic cost calculation based on predecessors.

The VCGs are the most complex of the diagram types, primarily because of the many relations between the different components. In addition, there are some constraints on how they can be related. The solution agreed upon is shown in Figure 5.6. The design is emphasizing simplicity while still staying consistent with all the rules of vulnerability cause graph notation and constraints.

VCG components are divided into two categories, ExitNodes and CauseNodes. ExitNodes represent the main vulnerability described by the VCG, and there can be only one ExitNode for a single VCG. All other nodes fall into the CauseNode category, and describe components in the VCG that are modeling causes of other components.

CauseNode is subclassed by three different entities. These are SimpleNode, CompoundNode and ConjunctionNode. In this design, the only difference between the two first is the visual representation, but the ConjunctionNode differs from the other two in that it is a container component and therefore can contain other CauseNodes.

The last part of the data model is the misuse case part. There are few advanced relations in a misuse case diagram, but the large amount of different relation types and components makes the model a lot larger than the other two. The chosen solution does not put too many constraints on the user. The intention is for the user to be able to model the misuse case diagrams in the way he/she feels is more appropriate. The general structure of the misuse case part is shown in Figure 5.7.

The misuse case diagram primarily contains three types of components. The first one is the Actor, which corresponds to an actor/user in misuse case terminology. The second one is the UseCase, which is representing all three types of use cases, the regular, vulnerability and threat/attack use case. In addition to these two, the system box is added as its own component, in the SystemBox class.

The SystemBox will work as a container in the diagram. Use cases can be placed inside this component, but they aren’t required to. They can also lie directly in the diagram.

The specific use case classes are all subclasses of UseCase. The difference between the three classes is the relations they may belong to, and also of course their visual representation, according to the notation.

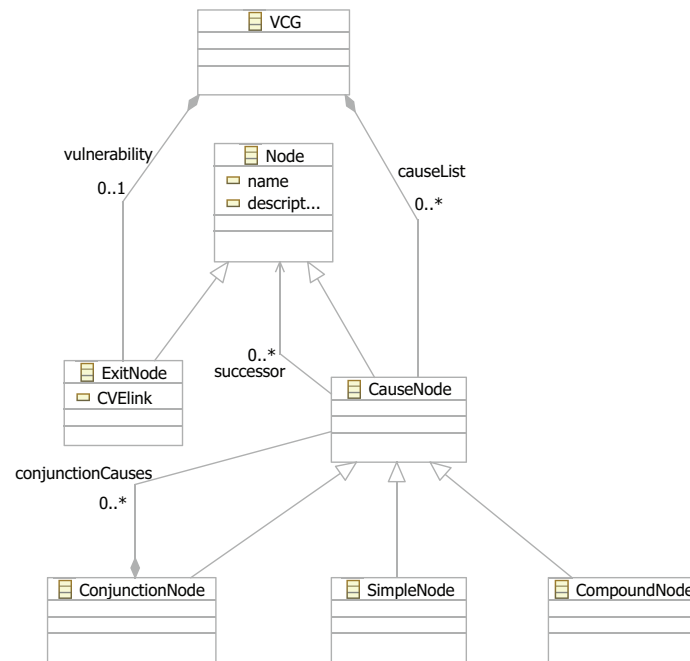


Figure 5.6: Class diagram for the VCG part of the applications data model.

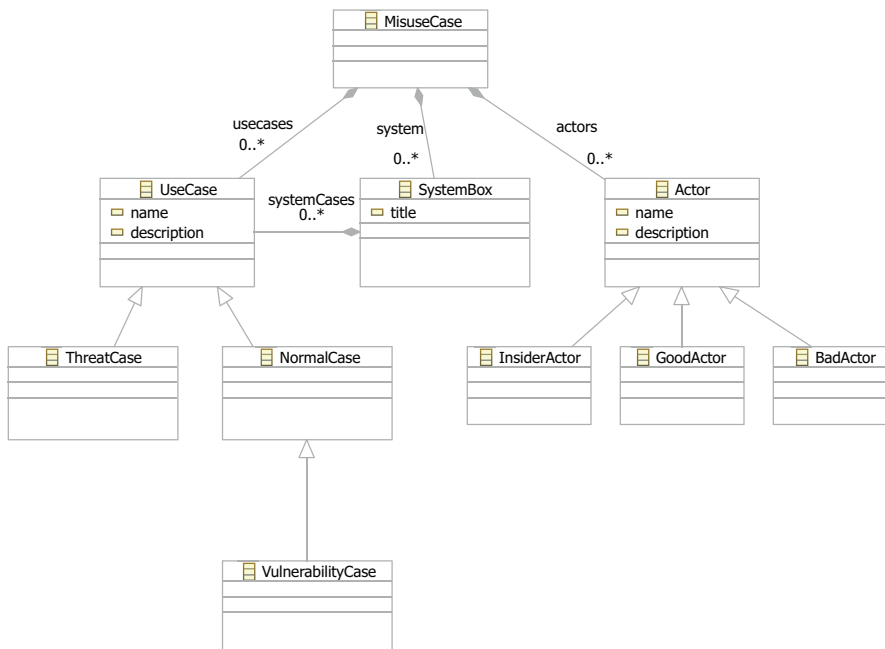


Figure 5.7: Class diagram for the Misuse Case part of the applications data model.

The Actors are designed in the same manner as the use cases, with one defining superclass and three different subclasses representing the specific types of actors. These need to be separated due to their visual representation and also because of them not being allowed to be related to other components (use cases) in the same manner. An Actor cannot be contained in a system box.

The misuse case model, including all of its relations, is shown in Figure D.1 in Appendix D.

5.3.2 Graphical definition

This component is part of the generation framework of GMF. It deals with the graphics of the diagrams in the application.

In the graphical definition model, all graphical components that can appear in the diagram are to be defined. This means adding both descriptions of all nodes and connection that could appear in the diagram, as well as linking these to figure descriptors defining how things are going to look. A component in this file could represent a class, relation or simply an attribute from the data model. Although this should be kept in mind, this relation is not in itself relevant for this file, and will be handled in the mapping definition.

5.3.3 Tooling definition

This component is part of the generation framework of GMF. It deals with the diagram tools that are present in the application.

In the tooling definition, one can describe palettes. A palette is a collection of tools related to the application. This could be specific to each diagram type, or merely a larger collection of tools more general to the whole application. A tool would be linked in the mapping definition to specific component(s) (nodes, connections) in the diagrams, and using this very tool would of course create a new instance of component in the diagram. It is also possible to link the same tool to different components. The generated code will then try to differentiate between the two by discovering what case is represented by the specific usage of the tool.

5.3.4 Mapping definition

This component is part of the generation framework of GMF. It deals with relations between the definitions of the data/domain model, the graphical definition model and the tooling definition model.

The relating is done by adding component descriptions to the mapping model. One may then relate it to a tool, and also select what diagram component it will be represented by. Which components that may contain others in the diagram is also to be defined here. When this model has been created, the basis for the diagram is done.

As for diagram components that represents something in the data/domain model, the mapping definition is where you define what these are. For each such diagram component, one can choose a domain model correspondence.

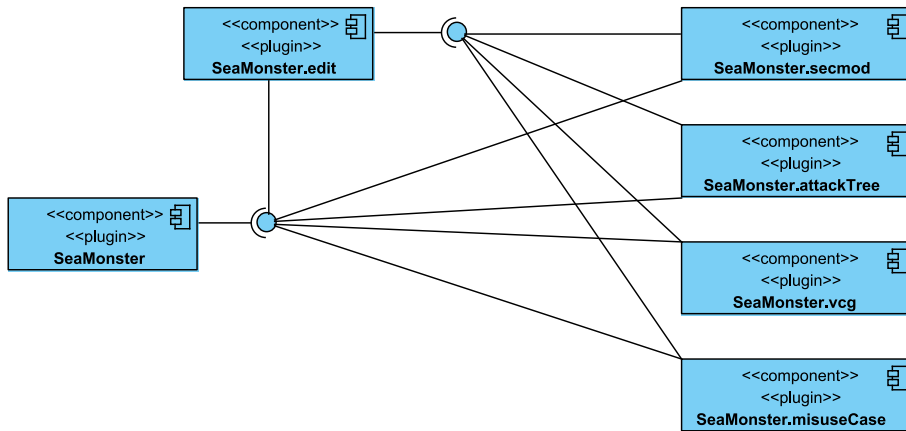


Figure 5.8: System plugins.

5.3.5 Generator model

This component is part of the generation framework of GMF. It is the last model that needs to be created for the framework, and is the basis from which the editor code will be generated.

Having defined the diagram through the other mapping files, the next step involves defining a set of parameters as to how the diagram editor will be. This is the purpose of the generator model. In addition, one gets some influence as to how to generate the editor code, i.e. copyright text, package name, etc. When all the settings are made, one can generate the editor code. This will generate code that is runnable through the eclipse platform, provided that the data model code has also been generated.

5.3.6 System decomposition

Next step is to provide a description of how the several modules generated by the framework must be grouped, and which are the relations between the groups. SeaMonster is released as a set of plugins. The application's core plugins are related between them in the way shown in Figure 5.8. From the diagram it is possible to observe that the plugins related to the security models are grounded on `seaMonster` and `seaMonster.edit` plugins. These last two provides classes and interfaces related to the objects to be modeled (e.g. causes, use cases, etc.). The decision to split the different types of diagrams in separated plugins improves the modularity of the application. This could allow to add other entities or modify the existing ones without affecting the entire system structure. Other plugins are delivered along with the latters. These are meant to assure standalone capabilities, and support the application with functionalities defined in the GMF framework (e.g. zoom functionality).

As GMF is used to structure the generated code into packages, descriptions of this and other programming entities will be described in the next chapter.

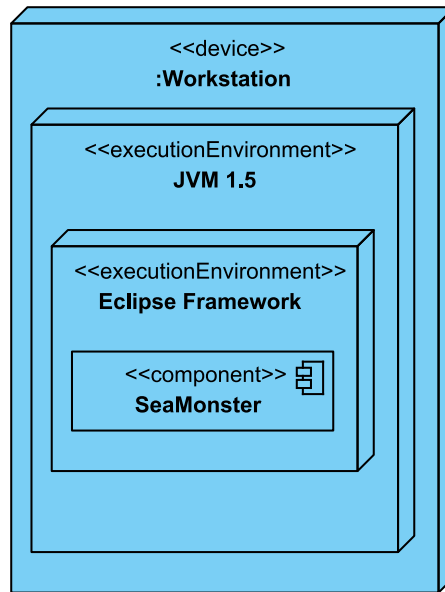


Figure 5.9: System deployment diagram.

5.3.7 System deployment

In this section the physical view is taken into account [65]. Until now all the other views, such as logical, and use case views, have been looked upon. The physical view is the only view still to be considered. Such a view shows the application within the environment where it has to be run. So, this means that the description of the physical view must go over the several entities that compose the environment, (e.g. software/hardware frameworks) and the way they communicate (e.g. involved communication protocols). The diagram in Figure 5.9 describes the system deployment by means of a UML deployment diagram. Being a stand-alone application, the depicted structure is quite simple and easy to understand. SeaMonster is released as an Eclipse application, and so it works based on the Eclipse framework. The framework is based in its turn on a JVM 1.5 compatible. Workstation indicates a machine running an operating system for which a JVM 1.5 exists.

5.4 User interface design

5.4.1 Description of the user interface

SeaMonster is based on the Eclipse framework, so its look and feel will be very similar to Eclipse (see Figure 5.11). Follows some decisions about the different areas presented by the application GUI.

Menu Bar The menu bar will contain several of the operations allowed by the application. Here are four of the most important menus that will be showed:

File groups all the input/output functionalities

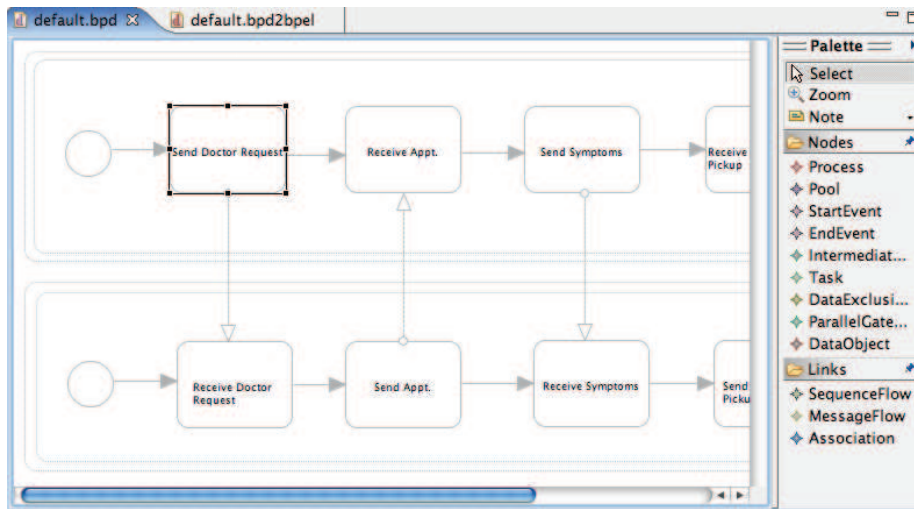


Figure 5.10: An example of how the main board will appear.

Edit groups all the editing functionalities, such as undo/redo, and cut&paste

Diagram groups all the functionalities related to the diagrams and their contents

Help set of user help functionality

Main Board The main board will contain two main sub-areas, that is the out-and-out board and the palette. The look and feel will be quite similar to the one in Figure 5.10.

Views The views that SeaMonster will present are essentially two, i.e. the Outline View, and the Properties View. The first one will show an overview of the diagram being modeled, and the latter a form for viewing/modifying properties related either to the diagram or to a selected element.

5.4.2 Adopted notation

The present section is meant to describe the notations that have been chosen for the three kinds of diagrams that SeaMonster deals with. The notation that will be used in SeaMonster for attack trees is mainly the notation presented in Section 3.5.2 derived from the article [21]. The only main change from this is the way to model AND. The notation presented in the article has a problem when three or more goals are AND'ed together. The solution chosen addresses this by slightly editing the original notation. Instead of having a small circle between the lines as described in the article a ring containing AND will be a connection point for the different subgoals as shown in Figure 5.12. This way it is both easier to understand and it looks better than the alternative solution. Another small change is that arrowheads, on the connection between the goals, is included in SeaMonster. This is just to emphasize what is the main goal,

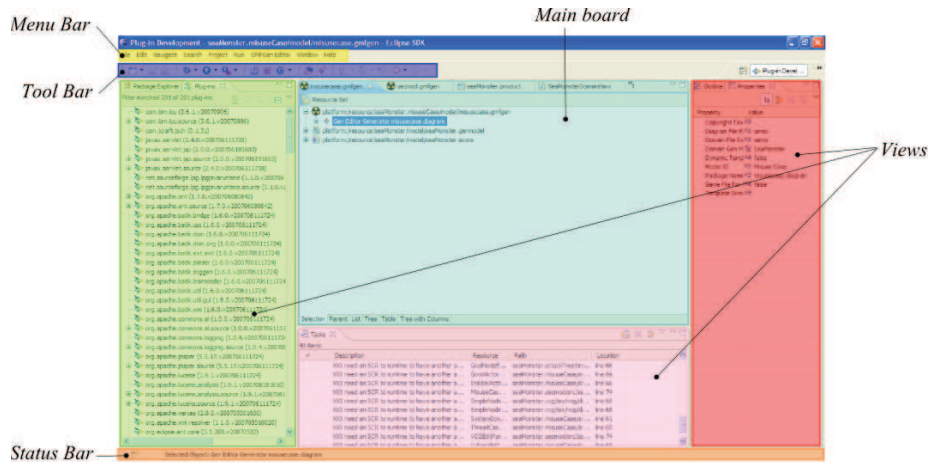


Figure 5.11: Typical Eclipse look and feel.

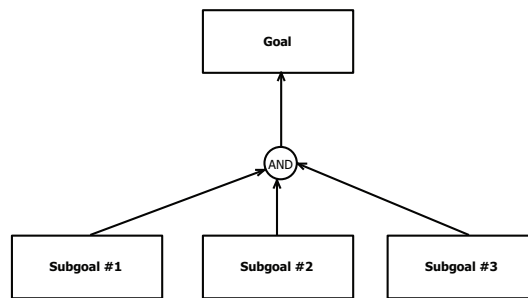


Figure 5.12: Notation for the AND nodes in attack trees.

and what is a subgoal. The complete notation for attack trees could be seen in Figure 5.15.

About VCG the notation is the one suggested in [13] and discussed in Section 3.5.1. Figure 5.13 summarize it showing the whole set of symbols.

Similarly, for misuse case diagrams (see Section 3.5.3) the adopted notation is depicted in Figure 5.14. The notation for the misuse case diagram is mainly based on [82], that introduces the threats/attacks and mis-users into use case diagrams. Further, the notation about vulnerabilities and insiders from [77] is included. The notation in [82] is updated in a newer version of the document, according to [77]. This will not be taken into consideration in this prototype, because the pre-studies on the misuse case diagrams was based on [82], and the information about the updated notation came in a late stage of the project. Updating the notation can be done in further development of the prototype.

5.5 Requirements not designed

In the previous chapter, the system requirements have been described and categorized in several groups depending on their nature. Unfortunately, due mainly

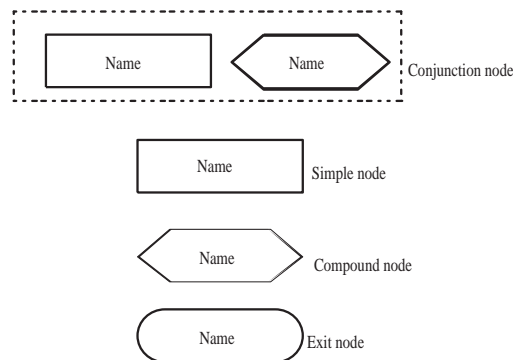


Figure 5.13: Set of symbols of the VCG notation.

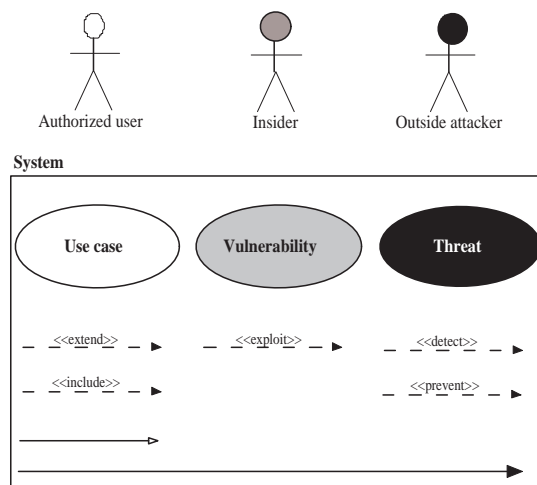


Figure 5.14: Set of symbols of the Misuse Case notation.

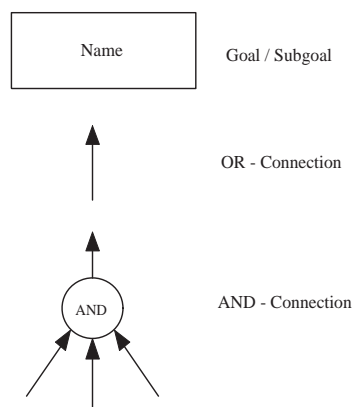


Figure 5.15: Set of symbols of the Attack Tree notation.

to time constraints, some of them have not been designed in the present chapter. The not design functional requirements have at most a medium priority level, so the application will not be seriously affected by this. Concretely, omitting the design of such requirements will permit to finish in time with a working and usable prototype. The following Table 5.1 summarizes the missing requirements.

5.6 Summary

This chapter described the design of the application. At the beginning, an introduction to the chosen way of development was given. The section described how the framework works, and how one should proceed in using this framework for development.

Due to much of the editor framework being generated by the eclipse GMF framework, this section focused on the domain model. This model describes the data structure containing all the diagram information, and is a basis for the rest of the generation process. An extensive description of the different parts of the domain model is described, and also contains one figure for each of the different parts of the domain. Later, the user interface design of the application was described.

At the end of the chapter, a listing of all the requirements not taken into account in this design is given. These are requirements that could be taken into consideration during further development of this application.

Group EF	
EF.01	The CWE/CVE databases have an export functionality. Content of these databases shall be possible to import into the application for identification purposes.
EF.02	The application must provide functionalities for linking security model components to external resources.
EF.03	The application must provide functionalities for displaying information related to external resources.
All the above requirements come with a medium priority level but EF.01 that is labeled as low. EF.02 and EF.03 may be fulfilled adding and customizing the browser plugin to the application. This would take more time than that at team's disposal.	

Group ATF	
ATF.02	The application must be able to automatically calculate the cost of an attack.
ATF.03	The application must allow the user to visibly mark resolved attack paths.
These two requirements are part of the set of requirements related to the attack tree security model. They are expressed as low level requirements. Design them would not have taken so much time, but the team decided to delay them in favor of other requirements with an higher priority.	

Group SF	
SF.01	Data should be stored accordingly to the UML 2.1 metamodel.
The low level requirement SF.01 has been already mentioned in Section 4.2.3 as a delayed requirement. Anyway, the XMI Schema in Appendix D may be considered a starting point for further works in this direction.	

Group SECF	
SECF.01	The application must validate the XML input referring to a pre-defined XML schema.
The security requirement is to be considered partially designed. The XML Schema file (see Appendix D.2) has been developed. What is not designed is the validation during the opening operations. Such a requirement is marked with a medium priority.	

Table 5.1: Table of requirements not taken into account during the design phase.

Chapter 6

Realization

6.1 Introduction

This chapter contains description of the realization of the application. It starts with a description of coding conventions used in the source code. Next follows descriptions on the proceedings in creating the application. This includes a summary of the model files used for generating the editor, and also a brief description of the changes that has been made to the generated source code. At the end of the chapter there is given an explanation of the package structure in the application.

The purpose of this chapter is to allow the reader to get an introduction to the application source, and how it is to be generated. This makes it easier to understand how the code will work, and thereby supporting updates to the application.

The realization of the application bases itself on a code generation framework to produce much of the code. Therefore most of this section is focused on the files used in the generation process.

6.2 Coding Conventions

As the SeaMonster application mainly is a prototype for further work in the area of software security modeling, it is important to adhere to some coding conventions. This makes it easier for those involved in further development to understand the code, and thereby make the changes necessary to include desired updates.

The application is developed in adherence to the Sun Microsystems coding conventions [86]. These conventions include rules for declarations, naming, indentation and white space among other things.

6.2.1 Code example

Listing 6.1 shows example code that is written according to the conventions.

```

1
2  /**
3   * This method scans a structure of AttackTreeComponents
4   * to see if adding a relation between self and
5   * oppositeEnd would result in a cycle. It returns true
6   * if the structure would still be free of cycles if the
7   * relation is added.
8   *
9   * @generated NOT
10  */
11
12  private static Boolean GoalNodeConstraint(
13      SeaMonster.AttackTreeComponent self,
14      SeaMonster.GoalNode oppositeEnd) {
15      if (self == null) return true;
16
17      LinkedList<AttackTreeComponent> queue = new
18          LinkedList<AttackTreeComponent>();
19      //Queue for breadth-first search
20      queue.add(self);
21
22      while (!queue.isEmpty()) {
23          AttackTreeComponent next = queue.poll();
24          // Reference to next object in breadth-first
25          // search
26
27          if (next == oppositeEnd)
28              return false;
29
30          /* Adding all children nodes to the queue
31          */
32          if (next instanceof GoalNode) {
33              EList<AttackTreeComponent> list1 =
34                  ((GoalNode) next)
35                      .getPredecessor();
36              for (AttackTreeComponent at : list1)
37                  queue.add(at);
38          } else if (next instanceof AndNode) {
39              EList<GoalNode> list2 = ((AndNode) next)
40                  .getPredecessor();
41              for (GoalNode go : list2)
42                  queue.add(go);
43          } else
44              throw new IllegalArgumentException(
45                  "Unknown_AttackTreeComponent!");
46      }
47      return true;
48  }

```

Listing 6.1: Code example: Algorithm for cycle detection in attack trees.

6.3 Process of generating the application

This section describes the development of the different plugins described in Section 6.3.7. Focus is directed at describing the process and how the generating of the code has been done. In addition, whenever there are changes compared with the originally generated code, this is documented in this chapter.

6.3.1 Domain model

The first two plugins, the `seaMonster` and `seaMonster.edit` plugins, were generated using the Eclipse Modeling Framework as described in Section 5.3.1. The generation model used for this purpose is located in the model folder of the `seaMonster` project.

As a basis for the domain model, an ecore file was developed, `seaMonster.ecore`, defining the data model described in Section 5.3.1. Using eclipse's built-in diagram editor for ecore files, it was possible to describe this model directly using UML class diagrams.

Having created the `seaMonster.ecore` file, the next step was to use it as a basis for an EMF genmodel file. The genmodel was then used in its original condition to generate the code for both the `seaMonster` and `seaMonster.edit` plugins. No other changes were made to the code of the domain model. The UML class diagram worked as a complete description of how this model was to be constructed.

6.3.2 Attack tree diagram

This diagram is defined in the `seaMonster.attackTree` plugin. It was the plugin that took the least time to implement, and also the one that was used in the earliest prototypes of the SeaMonster application. The plugin contains the code for the attack tree diagram, and is also runnable by its own as an Rich Client Platform (RCP)¹ application. Most parts of this plugin were created using the GMF code generation facility. This involves defining a set of model files (as described in Section 5.2.2).

This plugin bases itself on the AttackTree part of the domain model.

Graphical Definition

The first model is the GMF Graphical Definition Model, in which the visual representation of different diagram nodes and connections are defined. A diagram node is a component that can be represented in a diagram. Connections are links between nodes in the diagram. For the AttackTree diagram, two nodes and two connections exist. The two nodes represent `GoalNode` and `AndNode`, respectively, while the connections are corresponding to predecessor of `GoalNode` and predecessor of `AndNode`.

In addition to defining them as components in the diagram, both the nodes and connection need a visual representation description. This is done by creating figure descriptors and assigning them to the mentioned nodes and connections. For the connections, a standard polyline with the small arrowhead as decoration was chosen. Representing the same semantic connection in the attackTree, the

¹The minimal set of plugins needed to build a platform application with an user interface.

connections only needed one single Figure Descriptor. As for the nodes, a simple “rectangle” was used as descriptor for the GoalNodes, while an “ellipse” represent the AndNode. The AndNode is also given a preferred size, making it easier to make it a circle.

Tooling Definition

The tooling model for the contains three simple tools, one for creating goal nodes, one for and nodes, and one for creating the connections. The only thing changed from the basic settings in this model is the small image for the tools. This has been set to bundle images using more intuitive images when it comes to visual identification of the tools.

Mapping Definition

At this point, the relation between the tooling, graphical, and domain model needs to be described. The mapping definition model for the attackTree plugin can be seen in Figure 6.1.

As apparent, the mapping definition model for this plugin is quite simple. It contains Top Node References (nodes that can be directly placed in the diagram) for the nodes GoalNode and AndNode, with one label included for each of them. These labels are mapped to the name feature of the AttackTreeComponent. There also are two link mappings related to the two connections from the graphical definition.

For the two link mappings, link constraints are added in order to take into account the cycle prevention in the attack tree. The language feature of the constraints is set to Java, which means that the generation facility creates skeleton for the constraint methods, while the developer needs to fill in their bodies.

Both the link mappings are mapped to the same tool of the tooling definition. The application identifies which of the two connections it should use based on what kind of components are related.

When this was done, the mapping model worked as a basis for creating a GMF generator model. In this model, it is defined that the program should have printing enabled, and how the package structure of the plugin should be named. The default settings are used for the package structure, except for changing the base package name to `attackTree.diagram`.

Changes to the generated code

There are two main parts in which the code has been changed in the attackTree plugin. The first one is the link constraints, located in the `SeaMonsterBaseItemSemanticEditPolicy` class of the `attackTree.diagram.edit.policies` package. The second one is the EditParts of the diagram components, where the graphical representation of the components are created.

Regarding the first change, it was mentioned above that the generation framework creates skeleton code for the link constraints. This is what is implemented in this class. The two methods `GoalNodeConstraint` and `AndNodeConstraint` is implemented according to the pseudocode given in Listing 5.1. While implementing this method, entity tests were run at each change made to ensure a correct implementation of the method. The tests are described in Section 7.3.4

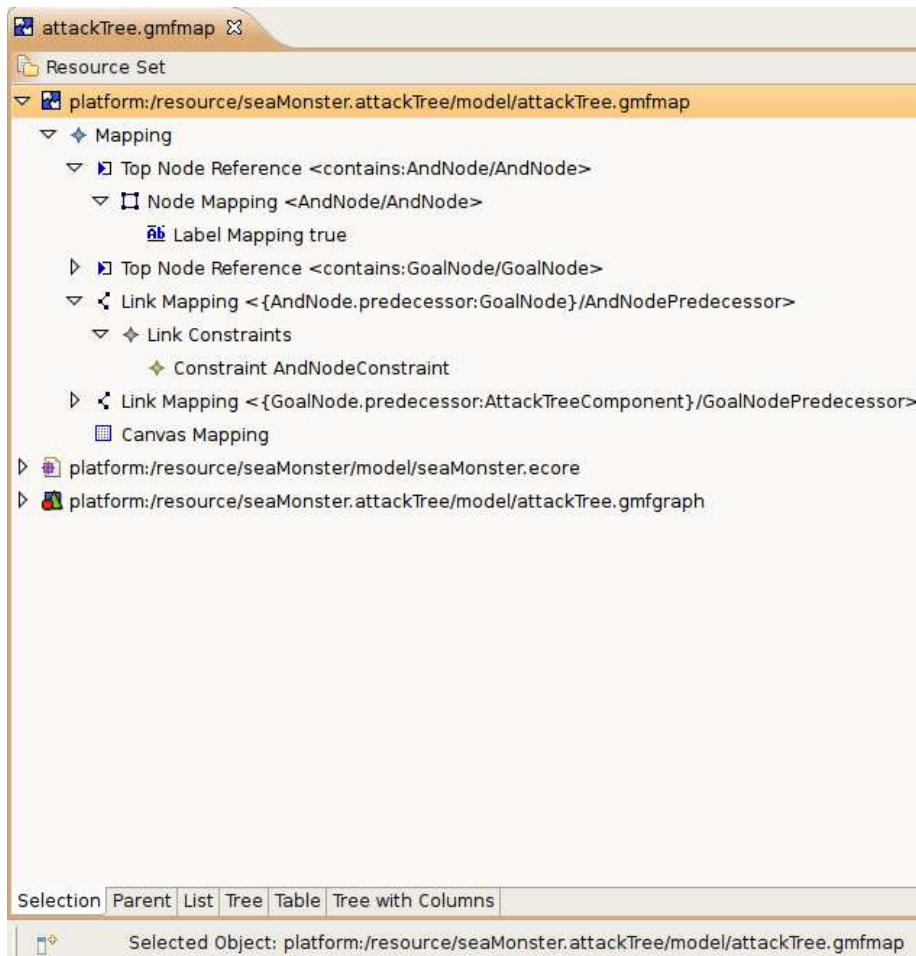


Figure 6.1: Mapping definition model for AttackTree plugin.

The second change was made in the constructor of the figures for `AndNode` and `GoalNode`. The layout of these components are explicitly set to a custom layout. This layout is called “`CenterStackLayout`”, and will locate all components added at the center. This makes for a more intuitive location of the labels in the `GoalNode` and the `AndNode`.

6.3.3 Misuse case diagram

This diagram is defined in the `seaMonster.misusecase` plugin. The plugin contains the code for the Misuse Case diagram, and is also runnable by its own as an RCP application. As for the Attack Tree plugin, most parts of this plugin was created using the GMF code generation facility.

Graphical definition

The graphical definition model for the misuse case diagram is quite extensive, due to the number of different components and links that can appear in the diagram. Most of the components are quite simple, though, and this section will focus on the components that differ in some way from a standard setting.

To begin with, there are seven components, one container and eight connections in the graphical definition. All these are connected to a figure descriptor, although several connections share descriptor. There basically is three descriptors for the connections, one between actors and use cases, one standard descriptor for connections between use cases, and finally one for the generalization linking. The latter uses an external graphical definition that contains the arrowhead used by the UML standard.

The components are basically of three different kinds. These are use cases, actors, and the system box. As the graphical definition of different use cases and different actors only differ in their colors, only one of each is described here.

The use cases are simply defined to being ellipses. For each of them, the color features are set to match the notation. In addition, they all contain a label to be used for identifying different use cases in the diagram.

The actors were a bit more complex figures, due to their non-standard shape and that their label should be located below the figure. A bounding rectangle is used for this purpose. It uses a “`BorderLayout`” to place the label at the bottom, and also includes a custom figure based on `org.eclipse.gmf.runtime.draw2d.ui.render.figures.ScalableImageFigure`. In this way, it was possible to define SVGs as the actors appearances. To do this, the generated code of the actor classes were changed in order to use a “`ScalableImageFigure`” based on custom images. Parts of the graphical definition model, including the custom figure, is shown in Figure 6.2.

The last part of the misuse case graphical definition is the System Box. This is basically a `Rectangle` that consists of two parts, a label at the top and a container rectangle at the center/bottom.

Tooling definition

The tooling model for the misuse case diagram is similar to the one for the attack tree diagram, with the exception that it contains a lot more tools. There are separate tools for all the different relations except for the uses relation between

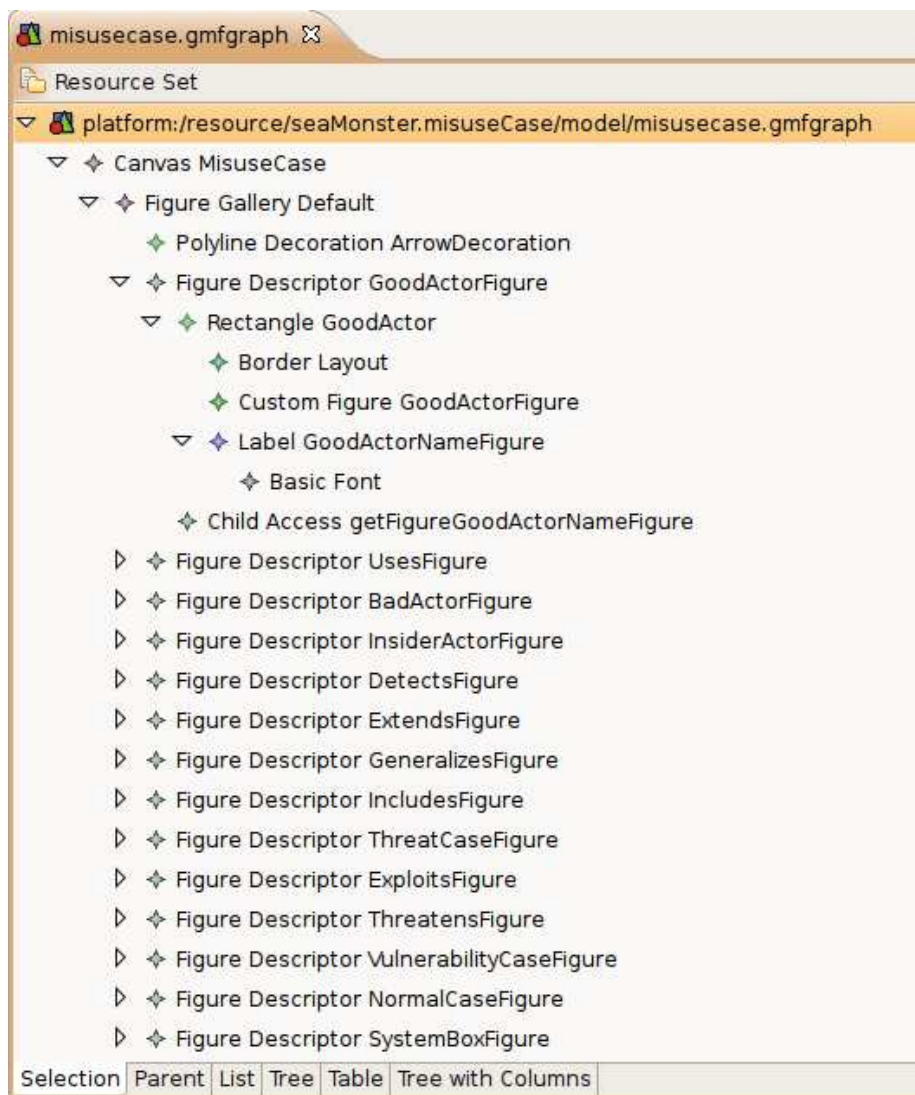


Figure 6.2: Graphical definition model for AttackTree plugin.

actors and use cases, for which there is only one combined tool. In addition, the small image icon of the tools has been set to custom bundle images.

Mapping definition

In the mapping definition model there are Top Node References for all the components of the misuse case diagram. All are given a feature label that corresponds to the name of the component. In addition, there are twelve link mappings that constitute all the different links in the diagram. All these are straight-forward, with the exception of the use case relations including a label which will describe what kind of relation it is representing.

The system box is perhaps the most complex component in the misuse case diagram. Due to the possibility of adding use cases to this box, the Top Node Reference for the system box contains child references to the Node Mappings of all the use case mappings.

Using this mapping definition, a generator model was created. The code was generated after a couple of small changes had been made. The most important changes were enabling printing and changing the name of the base package in which the plugin would be located.

Changes to the generated code

The most important code change in the misuse case diagram plugin is in relation with the custom figures of the Actor components. The constructor of the “XXXActorFigure” in the “XXXActorEditPart” classes was changed in order to make use of the SVG images used to define the figures.

Another change is that the CenterStackLayout described in Section 6.3.2 is used also for the use case components, in order to place their labels at the center of the components.

6.3.4 Vulnerability cause graph Diagram

This diagram is defined in the “seaMonster.vcg” plugin. The plugin contains the code for the Vulnerability Cause Graph diagram, and is also runnable by its own as an RCP application. Similar as the attack tree and misuse case plugins, most parts of this plugin was created using the GMF code generation facility.

Graphical definition

In the vulnerability cause graph, there is only one connection present, related to the “causes” relation. On the other hand, all the four different components have different visual representations. The “SimpleNode” and the “ExitNode” both use standard figures from GMF. The SimpleNode’s figure is a Rectangle, while for the ExitNode a Rounded Rectangle has been used. The rest are more complex, and will be described next.

The “Compound Node”’s figure is a hexagon, and might have been represented by a polygon. To be able to easily scale the figure, though, a solution using a custom figure was chosen. This means that it in the code had to be added a reference to the SVG defining the figure.

The Conjunction node is a container component, and this had to be taken into account in the graphical definition model. The figure descriptor of the

conjunction node contains a bounding rectangle that includes a label and a container rectangle inside.

Tooling definition

The tooling definition for the VCG is quite simple. It contains creation tools for all the four different diagram nodes and for the single connection type the VCG contains. All small tool icons has also been replaced by bundle images making it more intuitive related to the component it creates.

Mapping definition

Mapping definition for the vulnerability contains Top Node References for the four component types, and also a link mapping for the connection. The Top Node References all contain their own labels.

The Node Mapping of the Conjunction node contains child references to the SimpleNode and CompoundNode node mappings. These child references are mapped to the compartment of the conjunction node, to allow for placing compound and simple nodes in the compartment. By using the Top Node References node mappings as child references, it is ensured that it is possible to move the components in and out of the container. The mapping definition of the Conjunction node is shown in Figure 6.3.

Changes to the generated code

The “CenterStackLayout” is used also in the Vulnerability Cause Graph, to place the labels of components. This is done in the same manner as in the other plugins, by editing the constructors of the “XXXEditPart” in which the layout will be placed.

Another part that needed to be explicitly coded in this plugin, is the visual appearance of the CompoundNode. In the “createContents” method of the CompoundNodeFigure, a inner class of CompoundNodeEditPart, a ScalableImageFigure based on an SVG representation of the CompoundNode is added to the figure.

6.3.5 Security model diagram

The Security model plugin defines a diagram that works as an overview of the whole security diagram. It is a simple diagram that allows the user to create a security model containing several of the other types security diagrams. The plugin for the Security model diagram defines the SeaMonster application startup. It is the main plugin for running the application, and is therefore where one would want to specify how the application around the diagrams should appear.

Graphical definition

The Graphical definition of the Security model diagram defines three different types of Nodes, the AttackTree node, the vulnerability cause graph / VCG node, and finally the misuse case node. The only connection available between these

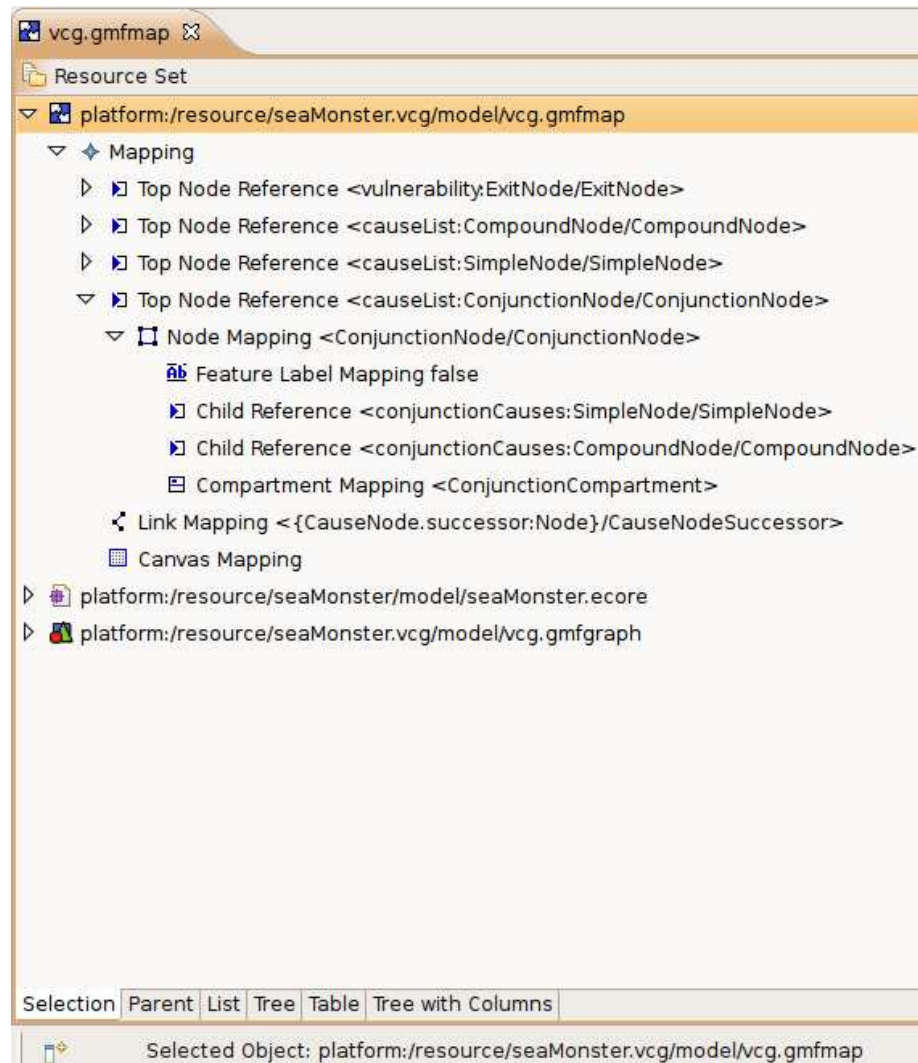


Figure 6.3: Graphical definition model for AttackTree plugin.

is the “relation” linking. Other than this, there are defined three different labels, corresponding to the title of the diagrams/components.

The figures of the nodes are all basic figures already present in GMF. The VCG Node is described by a rounded rectangle, to correlate with the Exit node from the vulnerability cause graph. The other two are also visually related to its diagram counterpart, by using the same kind of figure. The MisuseCase is described by the Ellipse corresponding to the UseCases, while the AttackTree is a Rectangle, which is the figure of GoalNodes from the attack tree diagram.

As for the connection, it is represented simply by a “polyline”.

Tooling definition

Tooling definition for this diagram is straightforward, with one tool corresponding to each component from the diagram. The images for each tool has been changed, to give a more intuitive correspondence to the actual item created.

Mapping definition

The mapping definition of the security model plugin is also pretty straightforward. There are Top node references to the three diagram nodes, VCG, MisuseCase and AttackTree. All these include a feature label corresponding to their titles.

There is also of course a link mapping relating the connection to its corresponding association in the domain model. The settings of this mapping is set to its default values.

Code generation model

The mapping described above is the basis for creating the GMF Generator model. After generating this model from the mapping definition, the other settings in the generator is to be set. These include how to generate the code, the ID of the model, some functionality to include, and also how the menu system is going to look.

To begin with, the model ID was set to security model, and the base package to `secmod.diagram`. Then the printing was enabled in the Gen Plugin section of the model. Having done this, some work was put into designing the menu system.

The most important change made is in the File menu. At this spot, some new menu tools were added to support printing and printing preview, among other things.

Changes to the generated code

The only change done to the code of the “secmod” plugin, is the addition of the `CenterStackLayout` and its application to the components. As in the other plugins, the `XXXEditPart` of the components contain construction code for the figure.

6.3.6 Product Configuration

When all the plugins are developed and tested, it is time to generate a complete build using all the different plugins developed. This build is able to run the whole program as a standalone application.

In the product configuration, all included plugins for the application are defined. For the SeaMonster application, there are a total of 108 required plugins, including the six ones developed specifically for this application. Together, these make up the functionality provided by the application.

In addition to the defining plugins, the launching configuration and application images are specified in the Product Configuration. This includes launcher icons, splash screen and application images. All these files are specifically created for the SeaMonster application.

6.3.7 Description of packages

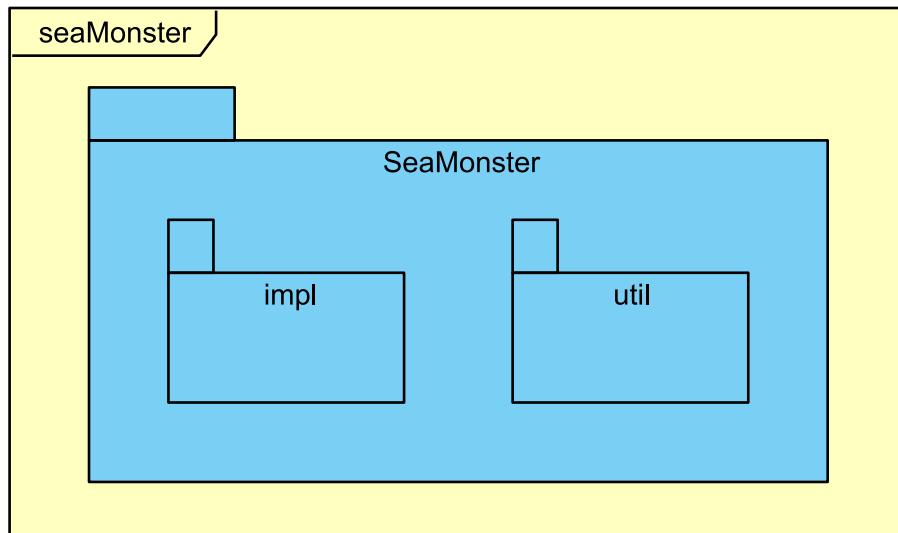
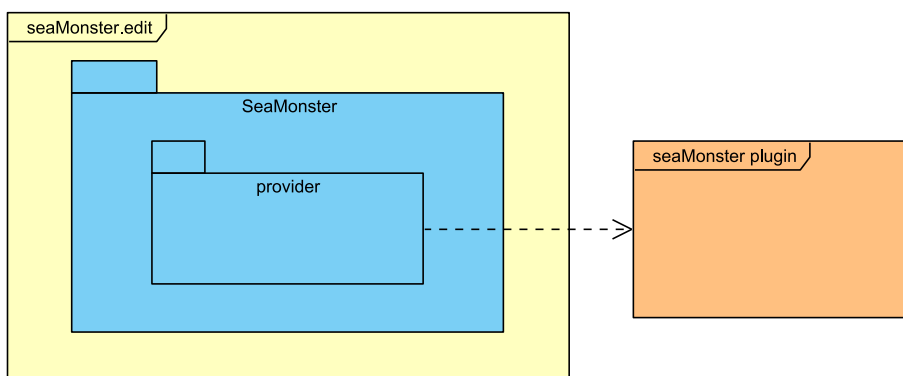
The Packages within the SeaMonster plugins have a structure that is imposed by the GMF framework when generating the code. This is the main reason why it was difficult to describe completely the system decomposition in the previous chapter. GMF supports generation of three kinds of plugins. Two of them are related to the objects defined in the EMF model and are basic for the application. The last type is strictly dependent on the different diagrams which can be modeled using the application. Every diagram has its own specific plugin. Following, there is a description of the content of every plugin supported by the package diagrams, which easily allow the comprehension of the hierarchical decomposition of the system.

SeaMonster plugin The root package `seamonster` contains the interfaces of all the objects defined in the EMF model, while an implementation of these is defined in the package `seamonster.impl`. Package `seamonster.util` contains some classes that are useful for properly managing the mentioned objects. All in all, it is possible to say that the plugin contains the model of the MVC pattern, which is exploited by GEF for generating the application. Figure 6.4 depicts this first basic plugin.

SeaMonster.edit plugin The content of the `seaMonster.edit` plugin is shown in Figure 6.5. This plugin contains the item provider adapters for every object in the EMF model.

seaMonster.xxx plugins Every diagram which is possible to model with SeaMonster has its own package. Moreover, every plugin has the same package decomposition as represented in Figure 6.6.

What is interesting to underline is that the `xxx.diagram.edit` package contains the View-Controller logic of the application. In particular, the set of classes `xxx.diagram.edit.parts.XXXEditParts.java` defines the way the diagram's elements will be displayed. This is done by mapping models of the diagram's elements with their figures. Then, by the means of policies, commands, and requests, they are able to properly manage the editing of the user. Commands are created for each operation allowed, e.g. creation of a node, or connection between two elements. They are collected within the

Figure 6.4: Content of the `seaMonster` plugin.Figure 6.5: Content of the `seaMonster.edit` plugin.

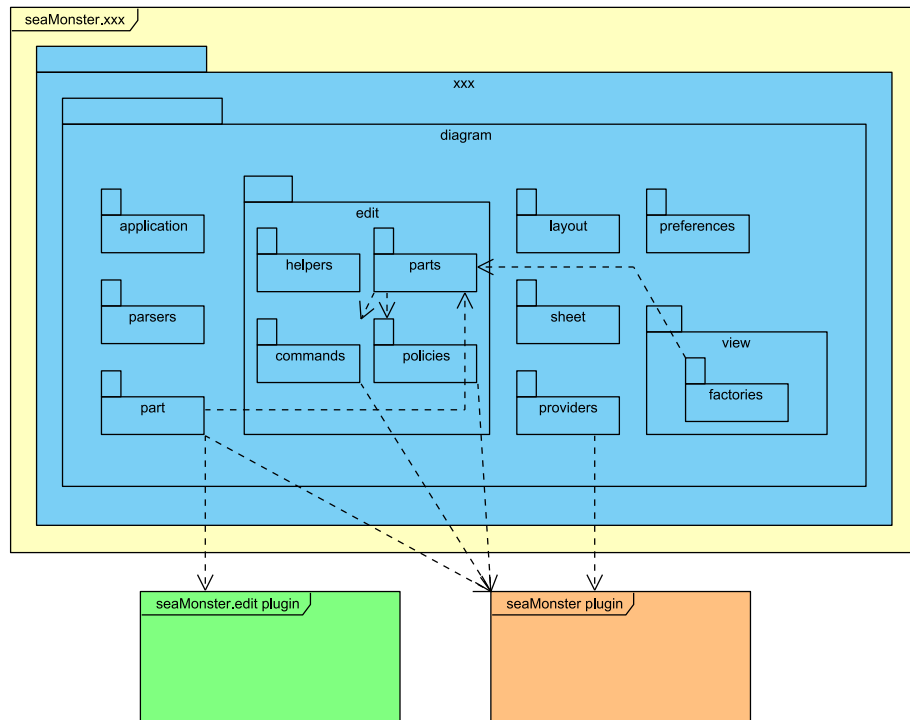


Figure 6.6: Content of the `seaMonster.xxx` plugins.

`xxx.diagram.edit.commands`. Policies determine the behaviour of the application in response to input from the user, or more in general, to events coming from the environment. Finally, requests are the communication mediums used for transporting information about the operations requested. The communication flow is sketched simply in Figure 6.7.

6.4 Summary

In this section, a description of the realization process of the application is described. As the realization is mainly based on a generation framework, this process is the part that primarily was documented.

Code conventions for the applications are described at the beginning of this chapter. These are as mentioned basically the Java Coding conventions given by Sun Microsystems. A code example is also given to show the structure of how the code is formatted.

Following this was a description of the model files used in the generation of the code, which showed the whole realization process. At the end, an overview of the inner structure of the application is described.

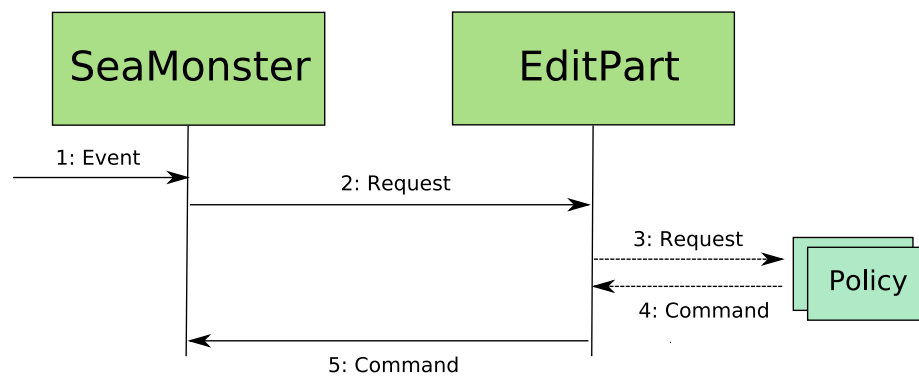


Figure 6.7: Communication flow for converting an event to a command.

Chapter 7

Testing

7.1 Introduction

This chapter describes in detail the testing phase of the SeaMonster project. The testing phase consists of defining and performing tests of the application. Testing is done in order to assure that the application is made according to the requirements in Chapter 4. The phase also includes correcting the detected errors. The tests are documented based on the IEEE standard 829 [45], as well as course guidelines, adapted to fit the project in the best possible way. The responsible for the testing phase will research and choose the best testing strategy for the project. The test plan, tracking of test, and all other testing details are documented in this chapter. One section is also dedicated to compare the tests with the requirements specification, to see if they have been fulfilled or not.

7.1.1 Purpose

The testing phase works as a quality assurance of the application. The approach, resources, and schedule for the testing activities are documented in this chapter. The items and features being tested, the tasks to be performed, and responsibilities are identified.

7.1.2 Scope

The scope of the testing phase is to assure that the application is of acceptable quality, and that it follows the demands stated in the requirements specification. Problems discovered during testing will be fixed, after that tests will be rerun. Problems with functions of high priority will be taken care of before low priority ones.

7.2 Overall test plan

This section gives a presentation of the test groups. It also describes how the testing will be done throughout the project. Testing is very important to ensure the quality of the final application. During early stages of the implementation,

Test group	Run by	Responsible	Time
Entity tests	Developers	Eilev, Eirik	Same time as realization phase
System tests	Developers	Eilev, Eirik	Same time as realization phase
Usability test	Students from another kpro-group	Eilev, Egil	End of realization phase
Acceptance test	Customer representative	Eilev, Egil	After usability tests

Table 7.1: Responsibilities for each test group.

the main focus will be on entity tests and system tests that concerns highly rated requirements.

7.2.1 Test methods

“Black box” and “white box” testing are two different ways of testing a system. Black box testing does not consider any internal behavior of the system. It only checks that the output of the system is as expected, when using a well defined set of input. This way of testing is well suited for the SeaMonster application, because it focuses on the functionality, as opposed to white box testing, which focuses on the internal structure of the code. Since the SeaMonster application is based on auto-generated code, which has been well tested by others, black box testing is considered the most important. Both the usability test and the acceptance test belong to this group of testing. Entity tests, which belong to white box testing, will also be performed, as well as system tests, which belong to both black box and white box testing.

7.2.2 Test groups

The prototype that is developed in this project will be tested with four main groups of tests. An overview of the test groups and a brief description of each group is found in this section.

- Entity tests.
- System tests.
- Usability test.
- Acceptance test.

The usability test and the acceptance test will be carried out by end users, while the system tests and the entity tests will be done only by the project team. The responsibilities are distributed as shown in Table 7.1.

Entity tests

Entity testing will be performed by the realization phase responsible, mainly during implementation. These tests deal with the smallest units in the prototype, such as classes, methods, and code. This project does not contain a huge amount of coding, so the entity tests will not be very extensive. Because it is not clear what methods has to be made or edited in advance, the tests will be created during implementation. The tests will be made before editing or creating any methods in the framework, and tested when the relevant code is ready.

System tests

System testing is a testing method that focuses on the complete systems internal coordination and integration, and the fulfillment of the requirement specifications. These tests will be performed by the project team, mainly while implementing. This is the most extensive testing, and hopefully all errors are discovered here, and not in the following usability test and acceptance test.

Usability test

This test will be performed with two persons outside the project team, that has no knowledge about the software to be tested. The main subject of this tests is the interaction between the system and the users. This will reveal whether the program is user friendly or not. The test will be made focusing on both the most important functional requirements and the non-functional requirements.

Acceptance test

The acceptance test will be a test of the finished prototype. The test includes the customer, and they will decide if the product meets their requirements. The test will be made focusing on the customers functional requirements. This test states the end of development phase for the prototype.

7.2.3 Approval and error handling

Errors will be handled as far as the time allows it. An overview over the most important tests will decide the order of which errors will be fixed first. The test phase responsible will be the one that decides if a test group is approved when it concerns the entity tests, system tests and usability test. The customer will decide if the acceptance test is approved.

7.3 Specific test plan

In this section, all the tests are listed, and their details are defined and explained. The test plan contains information about the approach, passing criteria, responsibilities, risks, and a schedule for performing the tests. The work flow is then explained, and a specification of each test group is presented. The actual tests are found in the test case specification, followed by information about the error handling.

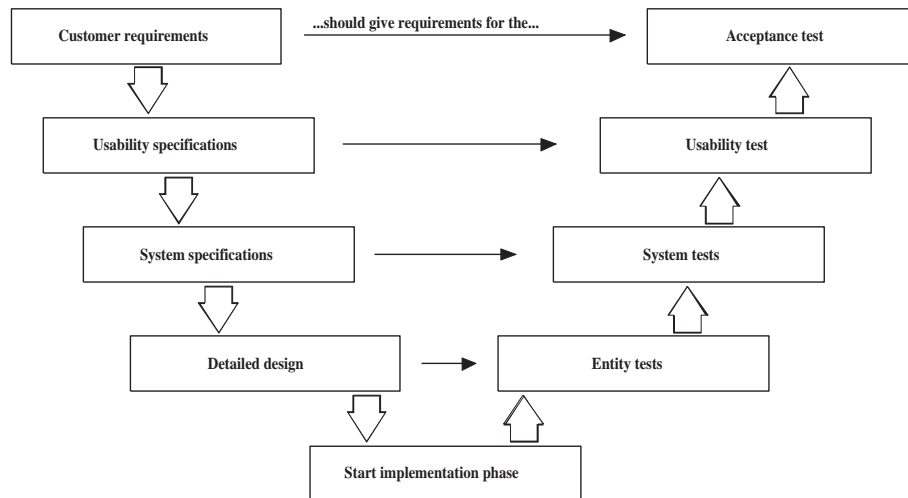


Figure 7.1: The V-model.

7.3.1 Test plan

Below follows a further specification of the test plan. The approach, passing criteria, responsibilities, risks, and schedule are described here.

Approach

While developing the tests, functional requirements and use cases are considered. The priority of a requirement tells how important it is to test that the requirement has been fulfilled. Although the system is a prototype, it is not necessary to meet all the functional requirements, and the efficiency and scalability may be downgraded. Figure 7.1 shows a modified V-model [59] for the approach of testing in this project. Tests are made while moving downwards in the model, so the acceptance test is made first and entity tests last. The tests are so performed in the opposite way, moving upwards in the model, so the acceptance test is the last to be performed.

Passing criteria

Each testing group has its own passing criteria, that should be met before moving on to the next group. Because the test object is a prototype, testing can continue to the next testing group without completing all tests in the former group, in order to meet time schedules. Passing criteria for each testing group can be found in Section 7.3.3.

Responsibilities

The overall responsible for the execution of all tests is the testing phase responsible. The tests are further divided among the team members, so that each test has one responsible. When a test has been performed, the overall responsible should be notified about the result, and this person also updates the test result section.

Risks and contingencies

The testing can be finished quickly, as long as only minor errors are detected, but it can also be very time consuming if critical errors are revealed. This makes it hard to estimate how much time is used on testing. Too little time to correct errors may affect the quality of the SeaMonster application. To avoid ending up with a huge error, which can be time consuming to correct, system tests are performed during the realization phase.

Schedule

The entity tests and the system tests will be performed while implementing. Therefore, they have a deadline for when they should be completed. The usability test will be performed a couple of days before the acceptance test, so there is time to improve usability before the acceptance test, if some problems are discovered. The acceptance test is the most important date to meet, because it concludes the testing phase and is close to the project delivery.

- 2007-11-09: Deadline entity tests and system tests.
- 2007-11-09: Usability test.
- 2007-11-13: Acceptance test.

7.3.2 Work flow

The procedure for performing the tests is illustrated in Figure 7.2. The smallest pieces of the prototype is tested first by entity tests. When the main functionality of the prototype has been implemented, the system tests are performed. When the entity tests and the system tests have met their passing criteria, a usability test is performed. All conclude with a final test, the acceptance test.

7.3.3 Test design specification

Here the structure is specified, along with passing criteria, for each test group.

Entity tests

The entity tests are annotated with the identifier ET-XX, where XX is a unique number. The passing criteria for this test group is when all tests are approved.

Structure of the tests:

- Id.
- Testing done by.
- Test case.
- Expected result.
- Result.

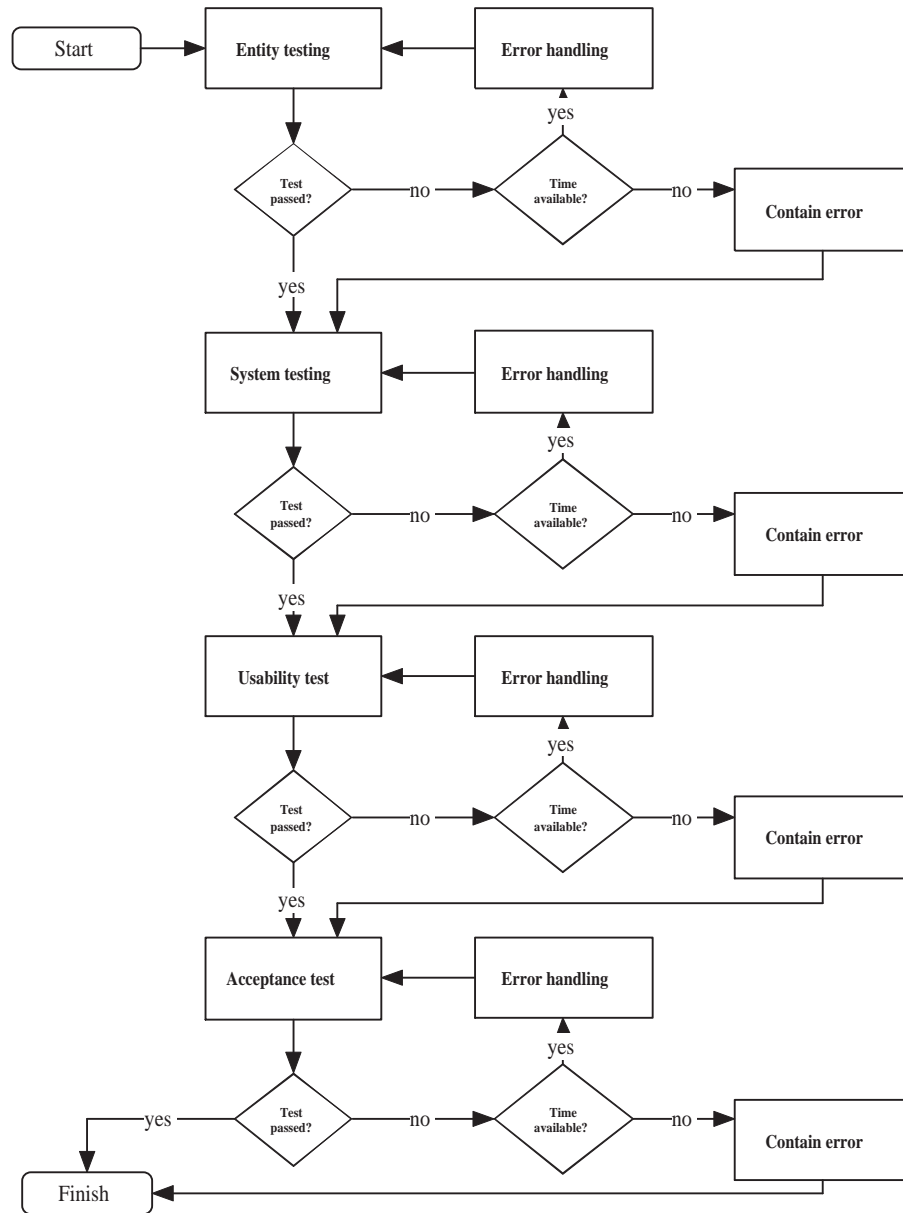


Figure 7.2: The work flow of the testing.

System tests

All system tests have ST-XX as the identifier base. Passing criteria for this test group is when all the tests relevant to the customers requirements are approved.

Structure of the tests:

- Id.
- Testing done by.
- Date.
- Responsible.
- Test name.
- Requirements tested.
- Test case.
- Expected result.
- Result.

When specifying the requirements tested, only the main requirements for the tests are listed. Other requirements that may be tested in the test, are left out from the requirement tested list.

Usability test

Usability tests have the identifier UT-XX. The person performing the usability test should finish his testing tasks without any help or guidance from the project team.

Structure of the test:

- Id.
- Testing done by.
- Date.
- Responsible.
- Test name.
- Requirements tested.
- Test case.
- Result.

Acceptance test

Acceptance tests have the identifier AT-XX. The acceptance test is passed only if the customer is satisfied with the product, and says that it is approved.

Structure of the test:

- Id.
- Task.

Testing done by		<i>Eirik</i>
Test case	Expected result	Result
ET-01: Testing the method <code>canCreateAndNodePredecessor</code> .		
The test starts with building an attack tree, consisting of GoalNodes and AndNodes. For each pair of AndNode and GoalNode, it then asserts the return value of the method, according to the description above.	The method accepts as argument the AndNode the predecessor is to be created at, and the GoalNode being the new predecessor. The return value should be false if adding this relation creates a cycle in the graph or if the GoalNode already is a predecessor of the AndNode, true otherwise.	<i>PASSED</i>
ET-02: Testing the method <code>canCreateGoalNodePredecessor</code> .		
The test starts with building an attack tree, consisting of GoalNodes and AndNodes. For each pair of GoalNode and Component (including the GoalNode itself), it then asserts the return value of the method, according to the description above.	The method accepts as argument the GoalNode the predecessor is to be created at, and the AttackTree being the new predecessor. The return value should be false if adding this relation creates a cycle in the graph or if the AttackTreeComponent already is a predecessor of the GoalNode, true otherwise.	<i>PASSED</i>

Table 7.2: Entity tests.

7.3.4 Test case specification

Here the actual tests are stated. The results from the tests is found in Section 7.4.

Entity tests

There are only made two entity tests, and they cover the two methods made for preventing cycles in attack trees. A summary of the tests are found in Table 7.2. In the code, they are located in the source folder (src) of the `seaMonster.attackTreeplugin`, in the `attackTree.diagram.edit.policies.testpackage`. The file containing the test is named `SeaMonsterBaseItemSemanticEditPolicyTest.java`.

System tests

The system tests are divided into several categories, that each includes a collection of tests. A test specified for modeling attack tree is found in Table 7.3, VCGs in Table 7.4 and misuse case diagrams in Table 7.5. Further there is a more general test in Table 7.6, a platform independence test in Table 7.7 and a

security test in Table 7.8. The platform independence test is limited to regard the platforms the project team have available. This is the Microsoft Windows Vista, Microsoft Windows XP and Ubuntu Linux platforms. Finally, there is a test regarding the linking between the different views, located in Table 7.9.

Testing done by	<i>Ketil</i>	
Date	<i>2007-11-14</i>	
Responsible	Eilev	
Test name	Attack tree modeling	
Requirements tested	ATF.01, ATF.02, ATF.03, ATF.04, ATF.05, ATF.06, MF.01, SF.03	
Test case	Expected result	Result
ST-01		
Choose to model a new attack tree diagram.	All model components for the attack tree should show up.	<i>PASSED</i>
ST-02		
Create the attack tree shown in the example in Figure 3.6.	All model components should be available. The components should be identical to the notation defined in Section 5.4.2. Components should be placed, and be connected together without any errors. All goals should have the following properties: cost, boolean, name and description.	<i>PASSED</i>
ST-03		
Click on different component boxes and move them around. Select several boxes at once and move them around.	All connectors to the selected box should follow the movement.	<i>PASSED</i>
ST-04		
Drag one end of a connector from one component to another.	The connector should connect to the new component without any errors.	<i>(PASSED)</i> <i>ER-12</i>
ST-05		
Delete one of each component.	The component should disappear.	<i>PASSED</i>
ST-06		
Add costs to the leaf nodes.	Costs should be summed up correct in all parent nodes automatically.	<i>FAILED,</i> <i>ER-13</i>
ST-07		
Mark a node as resolved.	Marking a parent node as resolved causes its sub-nodes to be marked as resolved. The cost on the resolved nodes are not summed up in parent nodes.	<i>FAILED,</i> <i>ER-14</i>

Table 7.3: (continued)

ST-08		
Try to make a cycle in the tree.	This is not possible, and the application should notice the user about this.	<i>PASSED</i>
ST-09		
Save the model, then close the application, start the application again and open the saved model.	The model should be saved at the specified location. The model should be identical before the saving and after opening it again.	<i>PASSED</i>

Table 7.3: System test: Attack tree modeling.

Testing done by	<i>Ketil</i>	
Date	<i>2007-11-14</i>	
Responsible	Eilev	
Test name	Vulnerability cause modeling	
Requirements tested	VCGF.01, VCGF.02, VCGF.03, VCGF.04, MF.01, SF.03, SF.05, IGF.05	
Test case	Expected result	Result
ST-10		
Choose to model a new vulnerability cause graph.	All model components for the vulnerability cause graph should show up.	<i>PASSED</i>
ST-11		
Create the VCG shown in the example in Figure 3.5.	All model components should be available. The components should be identical to the notation defined in Section 5.4.2. Components should be placed, and be connected together without any errors. The components should have the name and description properties.	<i>PASSED</i>
ST-12		
Click on different component boxes and move them around. Select several boxes at once and move them around.	All connectors to the selected box should follow the movement.	<i>PASSED</i>
ST-13		
Drag one end of a connector from one component to another.	The connector should connect to the new component without any errors.	<i>PASSED</i>

Table 7.4: (continued)

ST-14		
Delete one of each component.	The component should disappear.	<i>PASSED</i>
ST-15		
Export the model as an image.	A working image should be located where the image was saved.	<i>PASSED</i>
ST-16		
Try to make a cycle in the diagram.	This is not possible, and the application should not allow this.	<i>FAILED, ER-11</i>
ST-17		
Add notes to several of the components.	The notes should be stored and possible to be viewed until they are deleted.	<i>PASSED</i>
ST-18		
Save the model, then close the application, start the application again and open the saved model.	The model should be saved at the specified location. The model should be identical before the saving and after opening it again.	<i>PASSED</i>

Table 7.4: System test: VCG modeling.

Testing done by	<i>Ketil</i>	
Date	<i>2007-11-09</i>	
Responsible	Eilev	
Test name	Misuse case modeling	
Requirements tested	MCF.01, MCF.02, MCF.03, MF.01, SF.01, SF.03, SF.06, IGF.04, IGF.05	
Test case	Expected result	Result
ST-19		
Choose to model a new misuse case diagram.	All model components for the misuse case diagram should show up.	<i>(PASSED)</i> <i>ER-01</i>
ST-20		
Create the misuse case diagram shown in the example in Figure 3.14.	All model components should be available. The components should be identical to the notation defined in Section 5.4.2. Components should be placed, and be connected together without any errors. The components should have the name and description properties.	<i>(PASSED)</i> <i>ER-02,</i> <i>ER-03</i>

Table 7.5: (continued)

ST-21		
Click on different component boxes and move them around. Select several boxes at once and move them around.	All connectors to the selected box should follow the movement.	<i>PASSED</i>
ST-22		
Drag one end of a connector from one component to another.	The connector should connect to the new component without any errors.	<i>PASSED</i>
ST-23		
Delete one of each component.	The component should disappear.	<i>PASSED</i>
ST-24		
Print the model.	A preview should be displayed, and the model should be sent to the printer.	<i>(PASSED)</i> <i>ER-04</i>
ST-25		
Add notes to several of the components.	The notes should be stored and possible to view until they are deleted.	<i>PASSED</i>
ST-26		
Save the model, then close the application, start the application again and open the saved model.	The model should be saved at the specified location. The model should be identical before the saving and after opening it again. Locate the saved file on the file system, open it in a plain text-reader and check if it contains a plain text XML structure.	<i>(PASSED)</i> <i>ER-05,</i> <i>ER-06</i>
ST-27		
Open the other diagrams you already have saved.	The application should be able to have several diagrams open at once.	<i>PASSED</i>

Table 7.5: System test: Misuse case diagram modeling.

Testing done by	<i>Ketil</i>	
Date	<i>2007-11-09</i>	
Responsible	Eilev	
Test name	General application test	
Requirements tested	IGF.01, IGF.02, IGF.03, IGF.06, MF.02, MF.03, SF.04	
Test case	Expected result	Result
ST-28		

Table 7.6: (continued)

Create a new model of free choice, or open an already saved one. Zoom in on a component. Use the outline window to move around in the model.	The application zooms in on the component. In the outline window the whole model should be visible, and a marking that tells which part is currently zoomed.	<i>PASSED</i>
ST-29		
Zoom out again. Drag the components around so none are parallel or structured. Use the “arrange components” tool.	The components should automatically be arranged in a logical way.	<i>(PASSED)</i> <i>ER-09</i>
ST-30		
Click on a component and view its properties.	The properties of a component should show up when a component is clicked.	<i>PASSED</i>
ST-31		
Edit all the properties. Mark another component, and go back to the one you edited.	The properties should be the same as they were edited to be.	<i>PASSED</i>
ST-32		
Add another diagram to the security model.	The application should support having more than one diagram to each model.	<i>PASSED</i>
ST-33		
Copy components between the different models.	An exact similar component should show up were you paste the copied one.	<i>FAILED,</i> <i>ER-10</i>

Table 7.6: System test: General functionality.

Testing done by	<i>Ketil, Eilev and Kris-Mikael</i>	
Date	<i>2007-11-09</i>	
Responsible	Eilev	
Test name	Platform independence test	
Requirements tested	PNF.01, SF.03	
Test case	Expected result	Result
ST-34		
Install and open the application on the Microsoft Windows XP operating system. Make a very simple model. Save the model, close the model, reopen the model.	No errors should occur, and all functionality should be present and working.	<i>PASSED</i>
ST-35		
Install and open the application on the Microsoft Windows Vista operating system. Make a very simple model. Save the model, close the model, reopen the model.	No errors should occur, and all functionality should be present and working.	<i>PASSED</i>
ST-36		
Install and open the application on the Ubuntu Linux operating system. Make a very simple model. Save the model, close the model, reopen the model.	No errors should occur, and all functionality should be present and working.	<i>PASSED</i>

Table 7.7: System test: Platform independence test.

Testing done by	<i>Ketil</i>	
Date	<i>2007-11-14</i>	
Responsible	Eilev	
Test name	Security test	
Requirements tested	SECF.01	
Test case	Expected result	Result
ST-37		
Open a saved file from the application, modify some of the content to be invalid XML. Save the file. Open SeaMonster, and try opening the edited file.	It should not be possible to open the modified file.	<i>FAILED, ER-15</i>

Table 7.8: System test: Security.

Testing done by	<i>Ketil</i>	
Date	<i>2007-11-14</i>	
Responsible	Eilev	
Test name	Linking of views	
Requirements tested	MF.02	
Test case	Expected result	Result
ST-38		
Choose to create a new security model diagram.	An empty workspace and a palette with the misuse case diagram, vulnerability cause graph, attack tree and a relation component should show up.	<i>PASSED</i>
ST-39		
Create one of each component, and make some relations between them. Save the diagram.	All components are placed without any errors.	<i>PASSED</i>
ST-40		
Choose to initialize a new misuse case diagram. Locate the .semo file created in ST-38, choose misuse case diagram from the list. Model an easy misuse case diagram. Save and close the current diagram.	The .semo file should be located where it was saved, the list should contain all three components that was placed in ST-39. The misuse case diagram model window should appear.	<i>PASSED</i>
ST-41		
Choose to initialize a new attack tree. Locate the .semo file created in ST-38, choose attack tree from the list. Model an easy attack tree. Save and close the current diagram.	The .semo file should be located where it was saved, the list should contain all three components that was placed in ST-39. The attack tree model window should appear.	<i>PASSED</i>
ST-42		
Choose to initialize a new vulnerability cause graph. Locate the .semo file created in ST-38, choose vulnerability cause graph from the list. Model an easy vulnerability cause graph. Save and close the current diagram.	The .semo file should be located where it was saved, the list should contain all three components that was placed in ST-39. The vulnerability cause graph model window should appear.	<i>PASSED</i>
ST-43		
Close all windows, and try to open the diagram again.	All diagrams should appear without any errors.	<i>PASSED</i>

Table 7.9: System test: Linking of views.

Testing done by	-
Date	-
Responsible	Eilev
Test name	Usability test
Requirements tested	UNF.01, UNF.02, IGF.05, MCF.01, MF.01, SF.03, SF.06
Test case	Result
UT-1	
Open the application. Choose to model a new misuse case diagram.	
UT-2	
Model the misuse case diagram given in advance by the project team.	
UT-3	
Add notes to some of the components.	
UT-4	
Save the model. Close and reopen the application. Open your saved model.	
UT-5	
Print the model.	

Table 7.10: Usability test.

Usability test

The usability test is performed with two students from another project. Two persons are used so the outcome gives a clear idea of usability issues. There is not used a lot of time on making a very formal test, as usability tests often are. The project team does not prioritize this, because the final product to be delivered is a prototype and the time available is more useful in other phases. The usability test is shown in Table 7.10, and this will be handed out to the testers together with a misuse case diagram that is supposed to be modeled. The testers are informed about how the testing will happen before the testing starts. When the test starts, no help from the project team is given. This should show how intuitive the application is, and give the team an idea of what needs to be fixed until the acceptance test.

Acceptance test

The acceptance test is in Table 7.11, in addition the customer is recommended to test the application further than suggested in the test plan. The diagrams to be used in this test are given in Figure 7.3, 7.4 and 7.5. The test is scheduled to be performed in one of the customer meetings.

ID	Task
AT-1	Open the application.
AT-2	Create a new misuse case diagram. Model the diagram given by the project team (Figure 7.5), or one of free choice.
AT-3	Create a new attack tree. Model the tree by the project team (Figure 7.3), or one of free choice.
AT-4	Create a new VCG. Model the VCG given by the project team (Figure 7.4), or one of free choice.
AT-5	Save and close all open windows in the application.
AT-6	Close and reopen the application, open your diagrams and make sure the diagrams are exactly as they were when you saved them.

Table 7.11: Acceptance test.

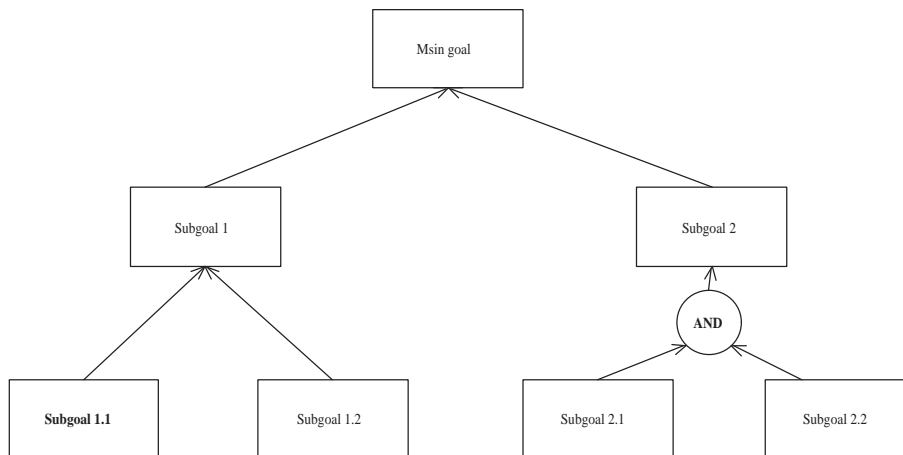


Figure 7.3: Acceptance test: attack tree.

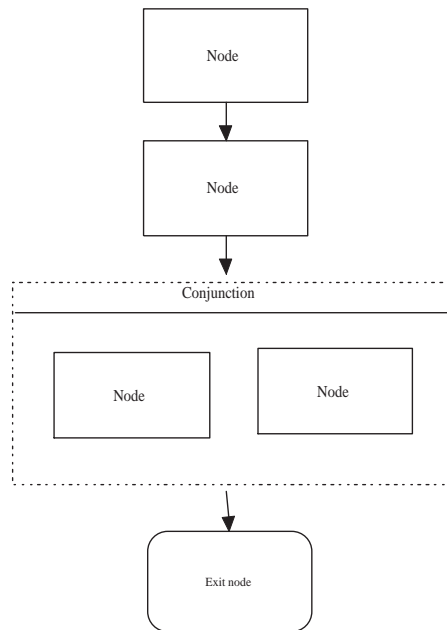


Figure 7.4: Acceptance test: vulnerability cause graph.

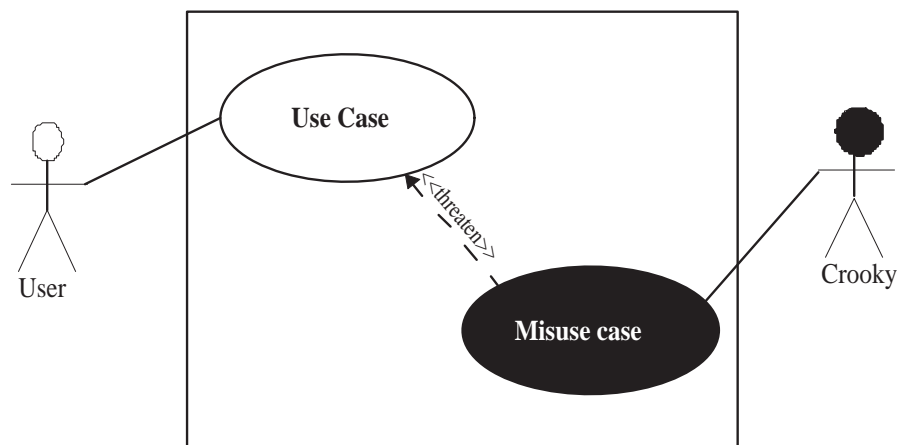


Figure 7.5: Acceptance test: misuse case diagram.

7.3.5 Error handling

When a test reveals an error, the error should be handled. Errors can be very time consuming to correct, there may not be enough time to correct them all. The tests are prioritized to show how important it is to fix eventual errors. Using this Table when testing helps developers decide which order the errors are to be corrected. The priorities are divided into critical, high, medium and low, shown in Table 7.12.

Errors in a critical test means that the software is not working at all, errors in high rated tests means that parts of the software is not working. Errors in medium and low rated tests causes reduced functionality in some parts of the software. The medium and low rated tests can be left uncorrected, due to the software just being a prototype. The main focus is on correcting the most serious errors.

7.4 Test results

This section contains the results for all the testing. All errors and comments in entity tests and system tests are given an unique identifier ER-XX, and a closer explanation. Comments from the usability testers are quoted, with an evaluation of the test from the project team. Comments and results from the acceptance test are found in the end of the section. In addition, the systems that were used for the testing are presented in the end of this section.

7.4.1 Entity tests

The tests are given in Table 7.2, and the results are in italic text. All test cases has been manually checked to see what the correct return value should be. The method returned the correct value on all test cases.

7.4.2 System tests

Attack tree modeling

The tests are given in Table 7.3, and the results are in italic text. One error was detected, ER-12, and a couple of functionalities was not implemented. Overall the attack tree modeling works satisfactorily, none of the errors or unimplemented method prevents an attack tree from being created.

ER-12 There is not possible to drag the arrow-head connected to an AND node to other components. Solution: This is a minor error, and the project team will not use time to correct this.

ER-13 Costs can be added on components, but there is not implemented any methods for summing these up automatically. Solution: This will not be implemented in this project.

ER-14 There is not implemented any method for marking a node as resolved. It can be done by adding a note that says its resolved, but that will not remove the cost from the automated cost addition (but this is not implemented either). Solution: This will not be implemented in this project.

Test ID	Critical	High	Medium	Low
ET-01				✓
ET-02				✓
ST-01	✓			
ST-02		✓		
ST-03		✓		
ST-04		✓		
ST-05			✓	
ST-06				✓
ST-07				✓
ST-08				✓
ST-09		✓		
ST-10	✓			
ST-11		✓		
ST-12		✓		
ST-13		✓		
ST-14			✓	
ST-15				✓
ST-16				✓
ST-17				✓
ST-18		✓		
ST-19	✓			
ST-20		✓		
ST-21		✓		
ST-22		✓		
ST-23			✓	
ST-24				✓
ST-25				✓
ST-26		✓		
ST-27			✓	
ST-28			✓	
ST-29				✓
ST-30				✓
ST-31				✓
ST-32			✓	
ST-33			✓	
ST-34		✓		
ST-35		✓		
ST-36		✓		
ST-37			✓	
ST-38			✓	
ST-39			✓	
ST-40			✓	
ST-41			✓	
ST-42			✓	
ST-43			✓	
UT			✓	
AT	✓			

Table 7.12: Consequences of test failure.

Vulnerability cause modeling

The tests are given in Table 7.4, and the results are in italic text. The test did just give one error, so the project team is satisfied with the implementation of the VCG.

ER-11 It is possible to make cycles in the diagram, any prevention of this is not implemented. Solution: A method for preventing this will not be included, because it causes no errors in the application if a cycle exists - even though it makes no sense.

Misuse case diagram modeling

The tests are given in Table 7.5, and the results are in italic text. The tests resulted in six errors or comments (ER-01 - ER-06), in addition to two other comments (ER-07 and ER-08) that was not related to any of these tests. Most of the tests were approved, and it was only minor details not working or missing in the rest of the tests.

ER-01 The system box is missing in the notation. Solution: Fixed, the system box is now present.

ER-02 The threat/attack case is white, should be black. Solution: The project team used a lot of time on this error. It seems like the framework not supports setting another color as default on the text on a component. The text color is set to black as default. If the component is black, as the threat/attack case is supposed to be, then the black text is not visible. A decision was made to not use more time on this problem, and the solution was to set the background color of the threat/attack to be dark grey. The black text inside this component is now possible to read.

ER-03 The relations are missing text, like “include” and “extend”. Solution: Fixed, the text is now present.

ER-04 There were no preview when printing, but the outline window in the application may give a good enough preview. Solution: The outline window will be a good enough preview, something else is considered further work.

ER-05 The location for saving the diagram can be chosen when creating a new diagram, but not afterwards. There should be possible to select a saving location at any time. Solution: This will not be implemented in this project.

ER-06 When opening a saved diagram, it is hard to know which of the files to open of the files the program has generated. Solution: This is explained in the user manual.

ER-07 During the test, there was tried to duplicate (using copy, paste) an use-case. When changing the text in one of them, the text in the other was automatically changed also. This is further commented and handled in ER-10.

ER-08 During the test, the “arrange all” tool was tried, and it really messed up the model. This should be removed from the misuse case diagram. This is further commented and handled in ER-09.

General application test

The tests are given in Table 7.6, and the results are in italic text. This test only revealed issues already stated in the misuse case diagram modeling test.

ER-09 The “arrange all” does not work in a good way regarding the misuse case diagrams, should be removed for this model. Solution: This will not be fixed in this project.

ER-10 When copying a component, there is some kind of relation between the copied component and the new one. When changing something on one of them, the other is updated. This relation does not only exists within a diagram, but also when copying between several diagrams. This also raised issues with saving. Solution: Using the “duplicate” tool solves the problem, but changing the functionality of copy/paste is considered as further work. The user manual explains how to duplicate elements.

Platform independence test

The platform independence test is given in Table 7.7. No errors or comments came up during this test. Java is supposed to work in the same way on any platform, and this seems to be true in this case.

Security test

The security test is given in Table 7.8. This test focuses on the security in the application, and tests if the XML files used to save diagrams in the application is checked against a predefined XML schema.

ER-15 The application does not prevent opening files with invalid XML. Solution: A XML schema is created, but a method for validating a file during opening is not created. This is considered further work.

Linking test

The linking test is given in Table 7.9. These tests did not uncover any errors, but the general impression of this functionality is a lack in usability. This is considered further work.

Summary

The most important requirements are fulfilled, so the the system testing is approved.

7.4.3 Usability test

The exact same usability test was done with two different persons, and the result is given in Table 7.13 and 7.14. Both persons have the customer driven course, but is on different projects. They have both experience with eclipse, which the framework is built upon, but neither of them has experience with GMF. They have knowledge about use case diagrams, and some introductory knowledge about misuse case diagrams. Both has the software security course this semester, so they have the wanted knowledge about security. The usability testers was asked for comments after the test was done, and this is quoted in this section.

Both tests showed similar problems with the application. In the beginning, both tried to drag a component from the palette into the diagram. When this did not give expected results, the click and place method was used successfully. Also the connectors seemed to be a bit confusing, the small arrows beneath the components did not seem intelligible. Opening of a saved diagram was confusing for both persons, the application creates to files, and it is not clear which of them that should be opened. In both tests, they chose the wrong file on their first try. Both persons commented that they missed several tools in the toolbar, like a button for creating a new model.

Results from usability tester one

Comments from the tester:

“There should be some more shortcuts on the toolbar, like create new model and zooming.”

“I had to create two files in the beginning, but I did not know what they were good for.”

“Drawing worked fine.”

“There should be possible to drag components from the palette.”

“The changing of names on the components was difficult.”

Results from usability tester two

Comments from the tester:

“When you insert a figure in the diagram, there should be possible to drag it from the palette.”

“I thought I had to click once on both of the components to place the a connector, this should be possible instead of dragging.”

“When opening files, only the correct filetypes should show up.”

“There should be more tools on the toolbar, like create new diagram and open. On the menu there should be possible to open the last used diagrams quickly.”

Testing done by	Usability tester one
Date	2007-11-09
Responsible	Eilev
Test name	Usability test
Requirements tested	UNF.01, UNF.02, IGF.05, MCF.01, MF.01, SF.03, SF.06
Test case	Result
UT-1	
Open the application. Choose to model a new misuse case diagram.	<i>The person did this without any problems, he located the misuse case diagram in the "File - New" menu on his first try.</i>
UT-2	
Model the misuse case diagram given in advance by the project team.	<i>The person first tried to drag the user component from the palette. When he discovered that this did not work, he clicked the user component one time, and clicked in the diagram where he wanted to place it - like it is supposed to be done in the application.</i>
UT-3	
Add notes to some of the components.	<i>The creating of the notes went smoothly, but he said he did not understand what the arrows beneath the note box was (that is the connectors used to connect a node to a component). He clicks one of them, and the note made a connection to it self. The person did not understand what this was.</i>
UT-4	
Save the model. Close and reopen the application. Open your saved model.	<i>All of this was without any problems, except the opening of the saved diagram. The person did not know if it was the .semo or .smmc file that was his diagram.</i>
UT-5	
Print the model.	<i>The person looked for a print option in the File menu. He did not find it there, so he checked the toolbar, and clicked on the printer icon. This resulted in a NullPointerException at every try.</i>

Table 7.13: Usability test comments from usability tester one.

Testing done by	<i>Usability tester two</i>
Date	<i>2007-11-09</i>
Responsible	<i>Eilev</i>
Test name	<i>Usability test</i>
Requirements tested	<i>UNF.01, UNF.02, IGF.05, MCF.01, MF.01, SF.03, SF.06</i>
Test case	Result
UT-1	
Open the application. Choose to model a new misuse case diagram.	<i>The opening was no problem. The person immediately found the misuse case diagram in the menu.</i>
UT-2	
Model the misuse case diagram given in advance by the project team.	<i>The person tried first to drag the actor component from the palette, this did not work, so he tried to click once on the tool and once in the diagram. When he was creating the “order book” use case, he was unsure which of the cases in the palette to use. When he made the first connector between two components, he had some problems. First he started with a click once in the first component. This made a relation from the component to itself. Then he tried again with dragging a connector from one component to another, this worked fine. In the diagram, the connector-arrow does not end completely on the component, but some pixels away. This confused the tester, and he thought that the connector was not connected to the component.</i>
UT-3	
Add notes to some of the components.	<i>This was done without any problems, but the person did not add any relation from the note to any components.</i>
UT-4	
Save the model. Close and reopen the application. Open your saved model.	<i>The person did not know which of the .semo and .smmc file to open, so this needed a couple of tries.</i>
UT-5	
Print the model.	<i>The printer button was located on the toolbar immediately, and the printing worked.</i>

Table 7.14: Usability test comments from usability tester two.

Solution

Solving issues uncovered in these tests all depends on the time available. Issues concerning the usability is then downgraded compared to most of the other tests. The team came up with different ways of solving the open file problem. One was is to have a filter in the open dialog, where the user could specify if it is a misuse case diagram, attack tree or a VCG that is to be opened. Another option is to add “open misuse case diagram”, “open attack tree” and “open VCG” in the File menu. The project team has when trying to print, not been able to provoke the `nullPointerException` to show again, as it did in the first usability test. This is probably just a minor bug in the framework, so this is not fixed. Hopefully this was just some java problems that will not occur again.

The project team is very satisfied with the outcome of the usability test. Many usability issues were uncovered, and several ideas for improvement was made. Unfortunately, time is preventing the team from improving all the weaknesses in the usability, so this will hopefully be improved in further development of the application. The conclusion of the usability test is that the application is overall very intuitive, but some things might need a couple of tries the first time.

7.4.4 Acceptance test

The acceptance test was done 2007-11-13 with customer representative Per H. Meland. The customer had already seen an early prototype of the application 2007-10-15, so the team had some idea of how the customer wanted the application to work in advance. Internal testing of this early prototype is found in Appendix E. The early prototype included only a simple attack tree notation, but during the acceptance test the customer got to test all three security modeling techniques. Comments during the acceptance test are given right below.

AT-1 No comments on this task.

Instead of moving on to AT-2, the customer wanted to create a new security model diagram to see what this was. He created a misuse case diagram component, and thought that he got a new misuse case diagram when double clicking on this. This did not give the wanted result, so he continued to AT-2 instead.

AT-2 The creating of the given misuse case diagram was done without any problems. He commented that the use case component did not automatically change its size when the text exceeded the available space inside the component. Other than this the misuse case seemed OK.

AT-3 The attack tree was created without any problems. The auto arranging did some wierd arranging. Other than this, the attack tree worked nicely.

AT-4 When creating a new VCG, he mentioned that it should say “Vulnerability cause graph” instead of just VCG in the menu. The default size on the boxes should be a bit larger. There should be possible to make a linebreak when adding text to the components. It was unclear how the arrows/connectors besides each component works.

AT-5 No comments on this task.

AT-6 This seemed OK.

Some overall comments from the customer in the end: When opening a model, there is difficult to know the location. The export to image tool is difficult to locate. The linking to the external resources, like CVE, is lacking. It would be nice to have the opportunity to add italic text, but this should be done in further work. The menus should be cleaned up. The application worked fast, and the diagrams was easy to create.

The customer clearly had much experience with GMF, and also showed the project team some tricks. The conclusion of the acceptance test is that it is accepted, under the conditions that the project team look into the issues mentioned.

7.4.5 Test systems

This section contains information about the systems used for testing of the application. The entity tests was tested with test system #6. Test system #2 was used for almost all system tests, except system #5 and #1 on platform tests. Test system #1 was used for the usability tests. Test system #4 was used for the acceptance test. The application was also generally tested with test system #3 and #6.

Test system #1 IBM ThinkPad Z61t, 32-bits Intel Centrino Duo T2400 1,83 GHz, 1024MB RAM, Windows Vista

Test system #2 Acer TravelMate 290E, 32-bits Intel Celeron M 1,3 GHz, 512MB RAM, Windows XP

Test system #3 HP Pavilion DV4000, 32-bits Intel Centrino 1,73 GHz, 2048 MB RAM, Windows XP

Test system #4 Dell D420, 32-bits Intel Centrino Duo 1,2 GHz, 1536 MB RAM, Windows XP

Test system #5 Dell D420, 32-bits Intel Centrino Duo 1,2 GHz, 1536 MB RAM, Ubuntu Linux

Test system #6 Toshiba Satellite, 32-bits Intel Pentium M 1,5 GHz, 496 MB RAM, Windows XP

7.5 Tracking of tests

This section shows how the requirements in Chapter 4.1 are tested, and during which tests. Table 7.15 shows a tracking matrix that links requirements together with tests. For each requirement, the matrix shows which tests that are relevant, and vice versa. For getting a better overview, the priority of each of the requirements (noted as “Prio.”) and the consequence of test failure (noted as “Cons.”) from Table 7.12 are listed in the matrix. The consequence of test failure is shortened C for critical, H for high, M for medium and L for low. Tests that failed and requirements which are not fulfilled are underlined in the matrix.

		Cons.	IGF.01	IGF.02	IGF.03	IGF.04	IGF.05	IGF.06	ATF.01	ATE.02	ATF.03	ATF.04	ATF.05	ATF.06	VCGF.01	VCGF.02	VCGF.03	VCGF.04	MCF.01	MCF.02	MCF.03	MF.01	MF.02	MF.03	SF.01	SF.02	SF.03	SF.04	SF.05	SF.06	UNF.01	UNF.02	PNF.01	SUCF.01
Prio.		M	M	M	H	M	M	M	H	L	L	M	M	M	H	M	M	M	M	M	M	M	M	M	L	M	H	M	M	M	H	H	H	M
ST-01	C								✓																									
ST-02	H								✓				✓																					
ST-03	H											✓																						
ST-04	H											✓																						
ST-05	M																																	
ST-06	L									✓			✓																					
ST-07	L										✓																							
ST-08	L													✓																				
ST-09	H														✓													✓						
ST-10	C															✓		✓																
ST-11	H															✓		✓																
ST-12	H																✓																	
ST-13	H																																	
ST-14	M																																	
ST-15	L																																	
ST-16	L																✓																	
ST-17	L																																	
ST-18	H																																	
ST-19	C																			✓														
ST-20	H																			✓	✓		✓	✓										
ST-21	H																			✓														
ST-22	H																																	
ST-23	M																																	
ST-24	L																																	
ST-25	L																																	
ST-26	H																																	
ST-27	M																																	
ST-28	M	✓	✓	✓																														
ST-29	L	✓	✓		✓																													
ST-30	L																																	
ST-31	L																																	
ST-32	M																																	
ST-33	M																																	
ST-34	H																																	
ST-35	H																																	✓
ST-36	H																																	✓
ST-37	M																																	✓
ST-38	M																																	
ST-39	M																																	
ST-40	M																																	
ST-41	M																																	
ST-42	M																																	
ST-43	M																																	
UT	M						✓																											
AT	C								✓	✓						✓	✓			✓	✓									✓	✓	✓	✓	

Table 7.15: Tracking of tests.

The matrix gives a good overview and is useful in several ways. When making the tests, this matrix helps determining which requirements are the most important to test, and if they are tested at all. Checking that the high rated requirements are tested is the most important task of this matrix.

All the requirements stated in Chapter 4.1 are listed in the matrix, except for the implementation non-functional requirements (INF, stated in Table 4.18), the packaging non-functional requirement (PKNF stated in Table 4.21) and the legal non-functional requirement (LNF.01, stated in Table 4.22). These requirements are not suitable for testing, because they are non-functional requirements that are not directly reflected in the application. In addition, the EF requirements are left out, because none of them are implemented. SF.01 is not covered by any test because this requirement was also not implemented. The requirements that are left out, are explained in Section 5.5.

The entity tests are left out of the tracking matrix because they do not relate directly to any requirements.

7.6 Summary

In this section, the testing of the prototype has been documented, and a test plan was stated and complemented with results. The goal was to cover as many of the requirements as possible and to fulfil these. The priority rating of the requirements and consequence of test failure have been used as guidance for

error correction, because of the tight time schedule. An overview of this was made in a tracking matrix (Table 7.15), and this was quite useful throughout the testing. Four different testing methods were used during the test phase; entity tests, system tests, usability test, and acceptance test. This section gives a summary of these tests.

The entity tests were not extensive, and they were pretty easy to handle due to a low amount of self made code. The two methods made for internal functionality in the application was tested and no errors were found.

The system tests uncovered some lacks in the implementation according to the model, and also some weaknesses in the model that had to be improved. These were the most important tests regarding the functionality of the application, and the application would probably not have been of good quality without them.

The usability test was done two times with two different persons, in order to get a good impression of the usability. This resulted in several issues regarding the usability that could be improved. Some of these issues were corrected, but the most time consuming corrections had to be left out. Usability is very important if this application is going to be used by a huge amount of persons, so the issues stated in these usability tests will hopefully be considered in the further work.

The acceptance test was done with customer representative Per H. Meland. The application was accepted under the conditions that issues emerged during the test were fixed.

The testing of the application did not reveal any critical errors, so the project team is overall quite satisfied. Most of the tests that were not approved were of low consequence, and the rest were of medium consequence. While implementing there were discovered several bugs in the GMF framework that could not be fixed without using enormous amount of time, but this will probably be fixed in a newer version of the framework.

Chapter 8

Installation and user documentation

8.1 Introduction

In this chapter an installation guide and a user manual for the SeaMonster application is provided. In addition, a small subsection regarding the contents of the enclosed CD is included.

8.2 Installation guide

SeaMonster is a Java application and it requires that Java Runtime Environment 5.0 is installed in order to start the application. Java can be downloaded from the Java website <http://www.sun.com/java>.

8.2.1 Windows

A list on how to install the SeaMonster application on a Windows platform is shown below.

1. Get `SeaMonster.zip` from the attached CD or download it from the project page (<http://sourceforge.net/projects/SeaMonster>)
2. Extract the archive file using a standard program like WinZip (<http://www.winzip.com>) or WinRAR (<http://www.rarlabs.com>).
3. After the files are extracted start the application by running the `SeaMonster.exe` file.

8.2.2 Linux

A list on how to install the SeaMonster application on a Linux platform is shown below.

1. Get the `SeaMonster.zip` from the attached CD or download it from the project page (<http://sourceforge.net/projects/SeaMonster>)

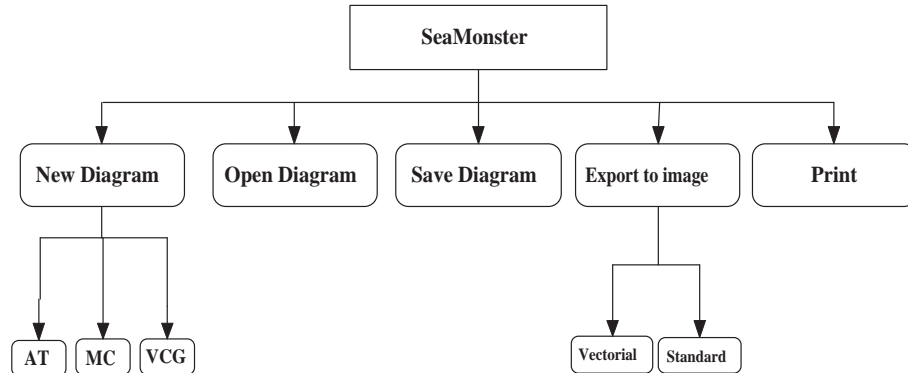


Figure 8.1: Main functionality in SeaMonster.

2. Open a Linux shell or terminal, change the directory to the directory where the .zip file is located, and extract the file with the command:
`unzip SeaMonster.zip`
3. After the files are extracted, enter the directory and start the application by typing `./SeaMonster`

8.3 The SeaMonster user manual

The main functionality of the SeaMonster application is summarized in Figure 8.1.

8.3.1 Menus and toolbars

The SeaMonster application has a standard menu structure located in the top of the screen and a toolbar with the most important functionality buttons underneath, see point one and two in Figure 8.2.

8.3.2 File-extensions in SeaMonster

There are five different file extensions in SeaMonster. The `.ssec` file is the diagram model and the `.semo` file contain domain model. These two files together makes it possible to link the different diagram types together. In addition, the extensions `.smat`, `.smmc`, `.svcg` refer to an attack tree diagram, misuse case diagram and a vulnerability cause graph respectively.

8.3.3 New model

The SeaMonster application includes following three main views.

1. Vulnerability and cause modeling, represented by vulnerability cause graphs. See Figure 8.3 for an example.
2. Threats and attacks, represented by attack trees. See Figure 8.4 for an example.

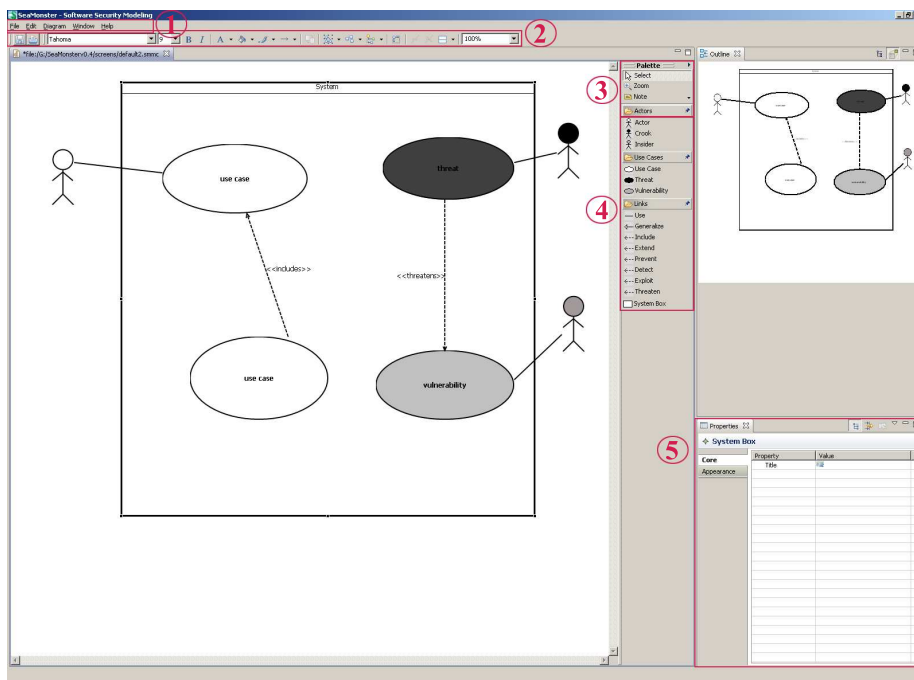


Figure 8.2: Structure overview of SeaMonster. Point one shows the menu, point two the top toolbar, point three the general palette, point four the model specific palette, and finally point five the properties view.

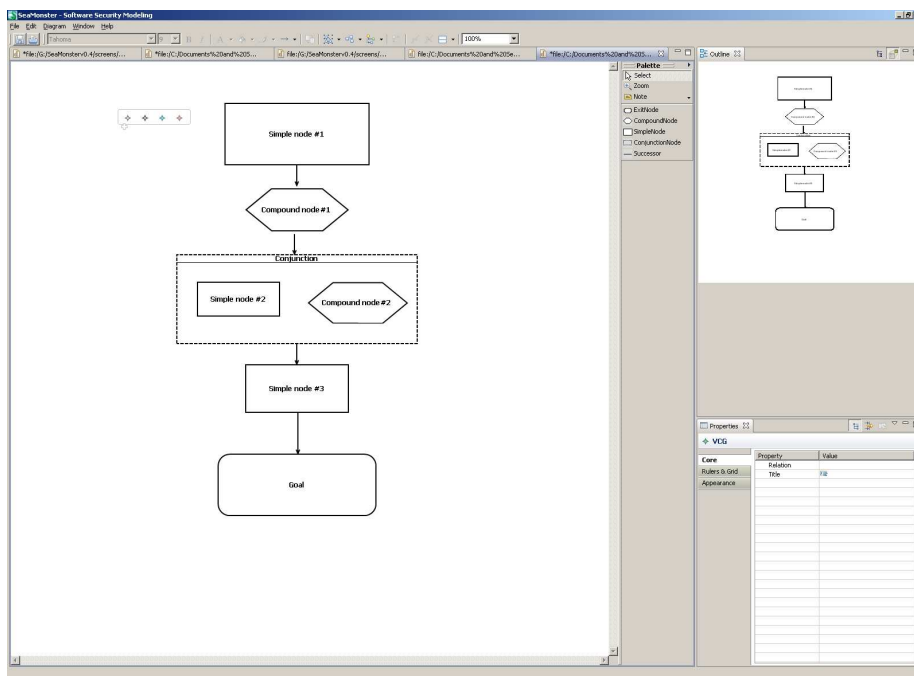


Figure 8.3: Example of a vulnerability cause graph.

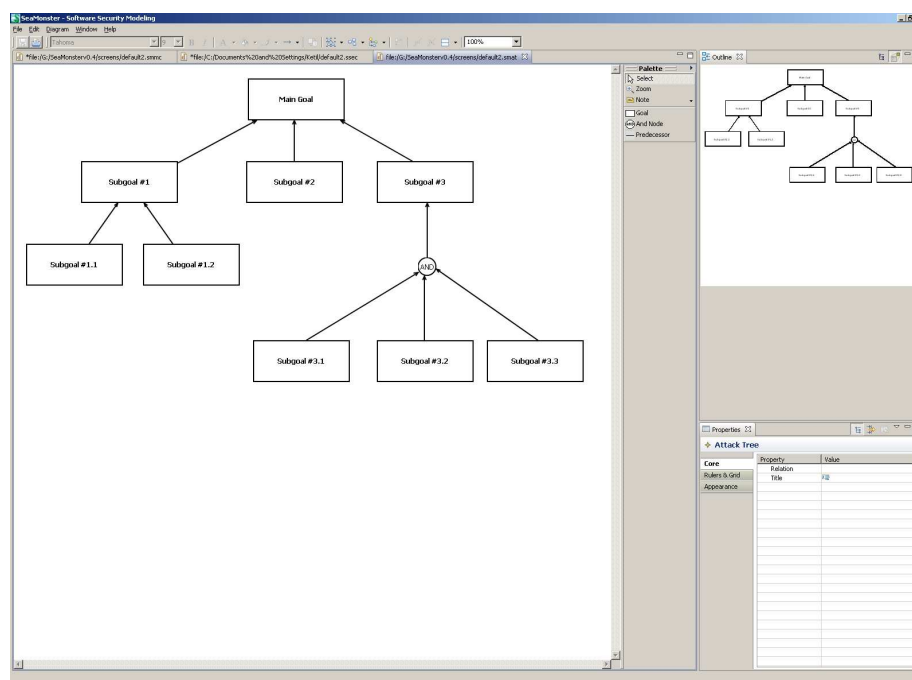
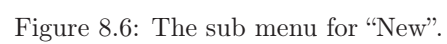


Figure 8.4: Example of an attack tree.

3. Countermeasures, represented by misuse case diagrams. See Figure 8.5 for an example.

These diagram types are located in the top of the file menu see Figure 8.6 for an example. In addition it is possible to create a new security modeling diagram. In this diagram you could add components representing an attack tree, a misuse case or a vulnerability cause graph and add appropriate relation between these different diagram types. After adding the component, the user could, after saving, initialize a diagram of the component by choosing the appropriate initialize command in the file menu. Then the user has to locate the .semo file, and find the wanted component to initialize. As mentioned in Section 8.3.2, .smat is the attack tree file, .smmc is the misuse case diagram file, and .svcg is the vulnerability cause graphs file. See Figure 8.7 to see an example of how to initialize an attack tree diagram.

After choosing a desired diagram and giving it a appropriate name, the user will see a palette with a set of objects corresponding to the syntax of the given diagram, see point four of Figure 8.2 for an example of the misuse case palette. In addition to this diagram specific palette, there is a general palette right above, see point three of the same Figure. The user then can choose an element from the palette by clicking it and then clicking the diagram area and drag the object to a wanted size. To add connections between objects in a diagram, choose the appropriate connection from the palette and drag from one object to the other object. To add a description or change the name of a component, the user may use the properties view located at point five of the same figure.



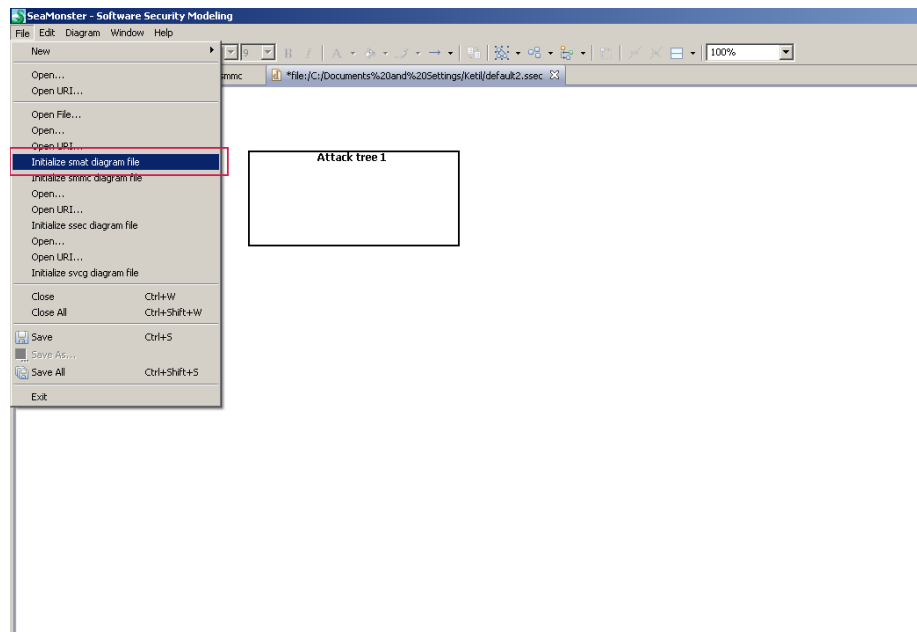


Figure 8.7: Initializing a new attack tree diagram.

Vulnerability cause graphs

The user is free to add the different components to his diagram, He may also drag and drop elements in and out of any conjunction node. When linking to or from the conjunction node, the user must click the top part of the conjunction node, as that is the only connection point to this node. At the exit node it is possible to add a number corresponding to the CVE id of the vulnerability. This is done in the properties section located at point five in Figure 8.2. An example of an vulnerability cause graph could be seen in Figure 8.3.

Attack trees

Here the user can add goals, subgoals AND-nodes and the predecessor-connection using the already described palette. Note that when you use the predecessor, you start by clicking where the arrow head should appear, and releasing where the start of the arrow should appear. So if you want to connect a maingoal to a subgoal you start by clicking the main goal, drag down to and release the mouse button on the subgoal.

An example of an attack tree could be seen in Figure 8.4.

Misuse case

The standard use case components could be located in the palette to the right. In addition it contains both the black “crook” and the gray “insider” with their corresponding activities and connections. If the user wants to use the system box, he must make sure to make this first and add or move components into the box.

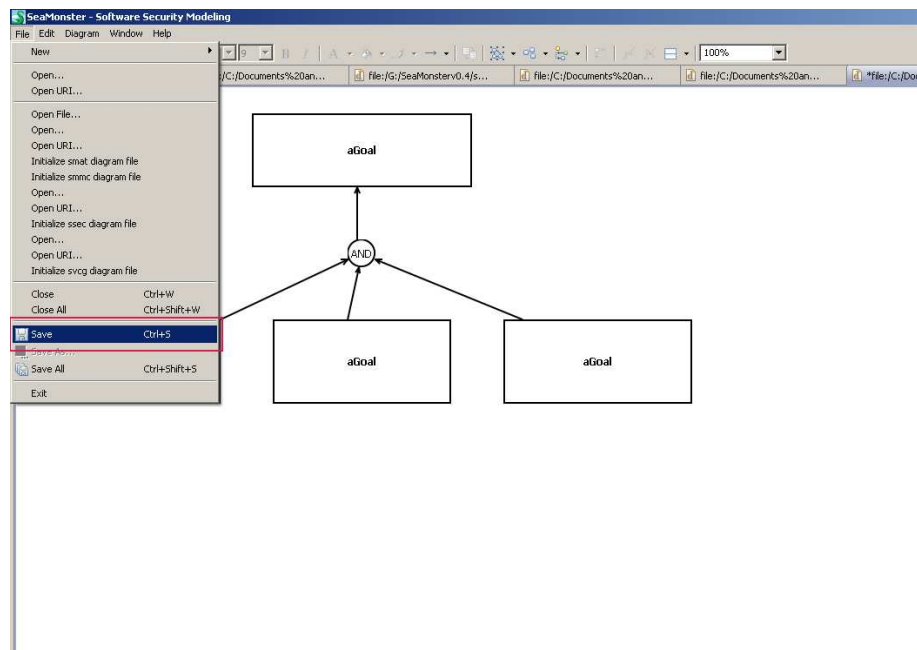


Figure 8.8: How to save a diagram.

An example of a misuse case diagram could be seen in Figure 8.5.

8.3.4 Save a diagram

To save a model, the user could either press the save button in the top toolbar, see point two in Figure 8.2 or use the save option in the file menu (point one). The save-as option is not enabled due to an unidentified bug in the frameworks used to implement the SeaMonster application. See Figure 8.8 for a save example.

8.3.5 Open a diagram

To open an already stored model, go to the file menu and locate open near the top of the menu. See Figure 8.9 for an open example.

8.3.6 Print a diagram

To print the diagram the user must click the print icon in the top-toolbar or use the appropriate keyboard shortcut (CTRL+p). See Figure 8.10 for a print example.

8.3.7 Export a diagram

To export the diagram to an image file, the user must right click on the diagram and choose File - Save As Image File. A window with different image formats will then appear. It is also possible to export just parts of a diagram. To do

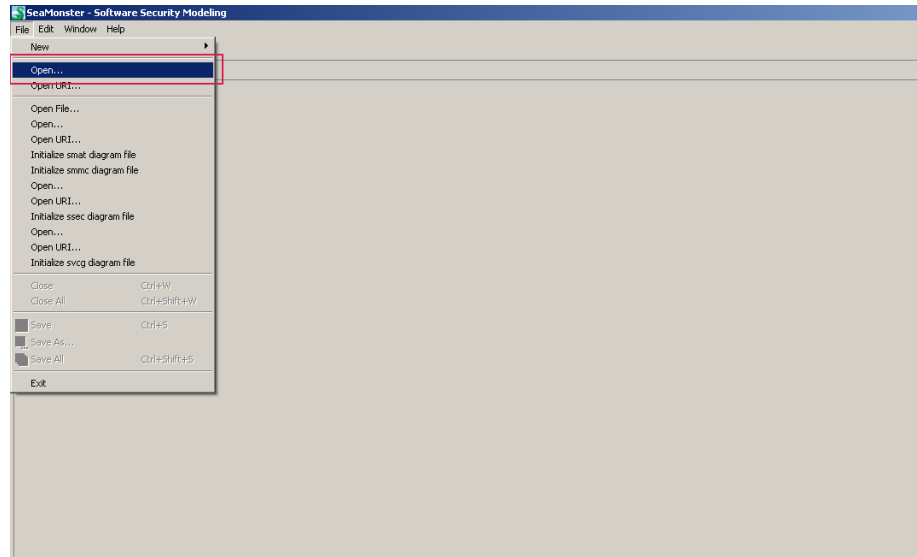


Figure 8.9: How to open a diagram.

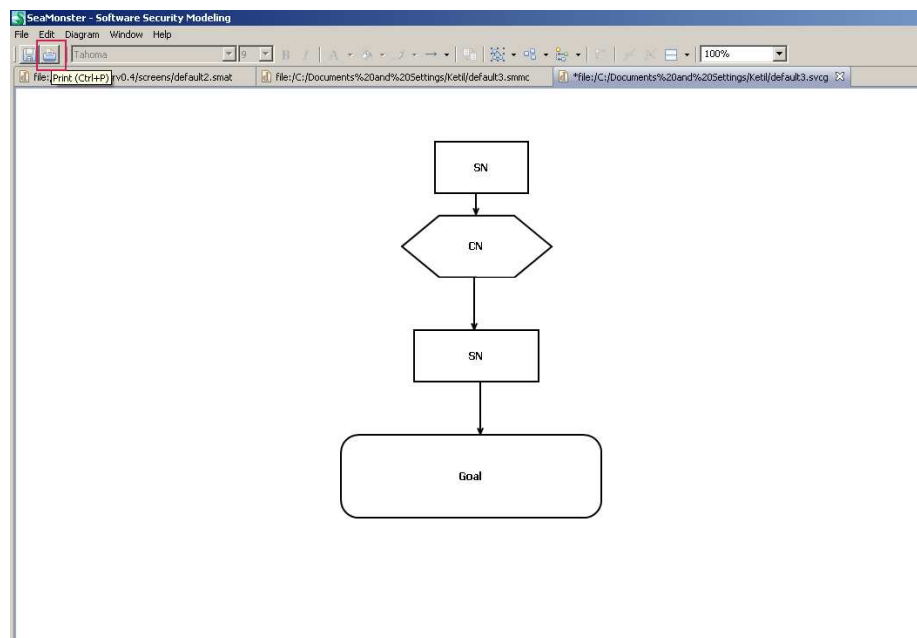


Figure 8.10: How to print a diagram.

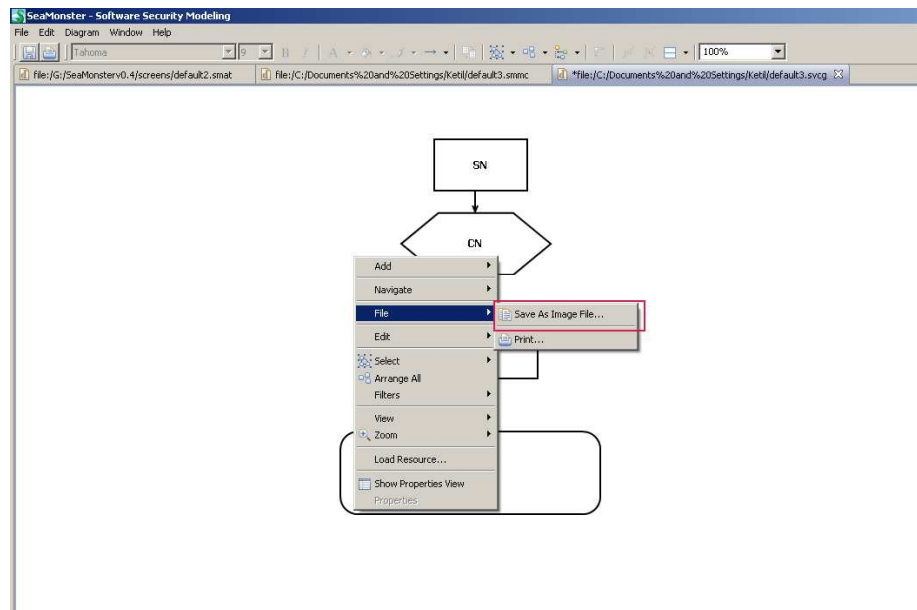


Figure 8.11: How to export a diagram.

this, simply mark the wanted objects and symbols before right clicking. See Figure 8.11 for an example.

8.3.8 Add notes to a diagram

If the user wants to add a note to a diagram, simply press the note button in the palette. There are two different kinds of notes that could be found if the user clicks the small down arrow by the note button. In addition there is a note-attachment that serves as a connection point from a note to a symbol in the diagram. Another way of doing this is to right click on the object in the diagram and click the Add Note button. See Figure 8.12 for a note example.

8.3.9 Copy/Paste

To copy a symbol in a diagram, right click the symbol and choose Edit → Duplicate. If the Copy command, located in the same menu, is used, the symbol will be an instance of the same object with the effect that if you delete one of the objects, both will disappear. Therefore the duplicate command is recommended. See Figure 8.13 for a duplication example.

8.4 Contents of the enclosed CD

The CD consists of four folders; Documentation, Report, Software and Source code.

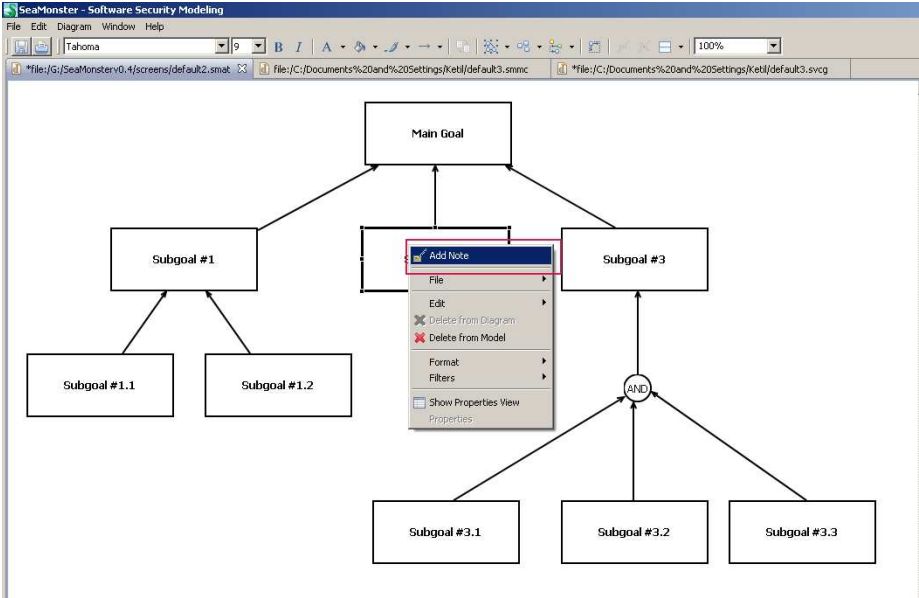


Figure 8.12: How to add notes to a diagram.

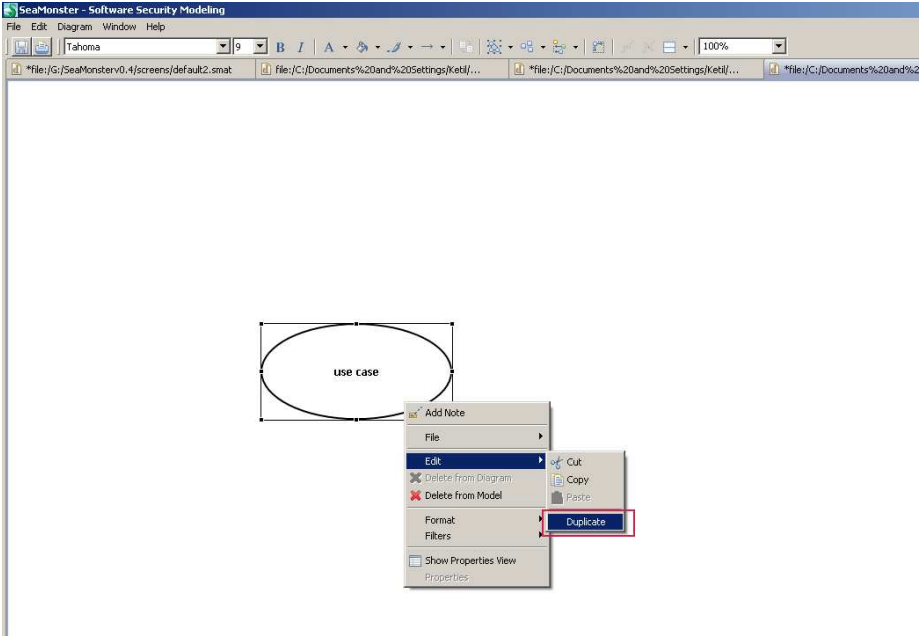


Figure 8.13: How to copy an element in a diagram.

8.4.1 Documentation

The documentation folder contains the documentation on how to install and how to use the software.

- The `installation_guide.pdf` includes the installation guide as described in 8.2.
- The `user_manual.pdf` file includes the user manual as described in 8.3.

8.4.2 Report

This folder consists merely of one file, that is the `report.pdf` which is the final report including appendices, glossary and bibliography.

8.4.3 Software

The Software folder also consists of only one file, that is the archive file, `SeaMonster.zip`, which has to be extracted to use the program.

8.4.4 Source code

Here, all the source code of the project is included. Note that most of the code is auto generated by the eclipse frameworks. Some of the auto generated code had to be edited in order to get the functionality needed. This is documented in Section 6. In addition, the different EMF and GMF specific files are included in the `model_definitions` subfolder.

Chapter 9

Evaluation

9.1 Introduction

This chapter contains the evaluation of five main parts of the SeaMonster project. These are the process, all involved parties, the course in general, the project task, and how to use the research and work done in this project in further projects or tasks.

9.1.1 Purpose

The purpose of the evaluation is to reflect upon different aspects of the project. The main motivation for doing this is to acquire an overview of what went wrong in the process, how it could be done better, and also mention the positive experiences.

9.2 The process

The first main part of the evaluation of is how the work process has been executed, if the chosen roles have worked out good, and how the available resources have been used. The most significant problems and difficulties that have emerged during the project's lifetime, are also listed in this section. The most important events that have happened are marked in the timeline shown in Figure [9.1](#).

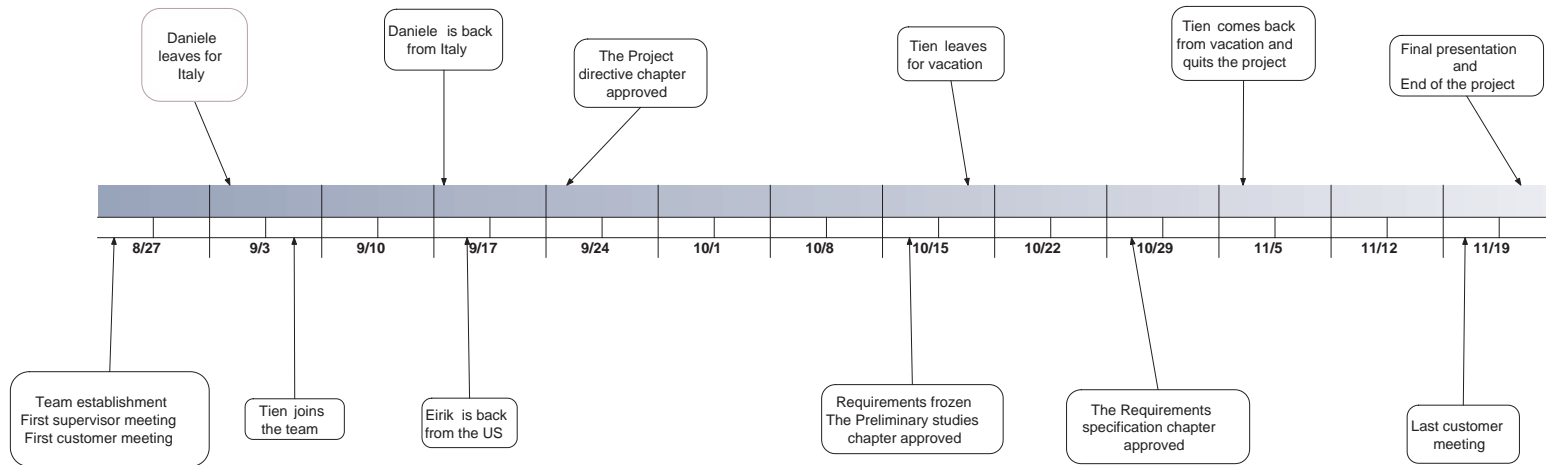


Figure 9.1: Timeline with important events marked.

9.2.1 The roles

In order to get a good structure of the team, roles were created and assigned to members of the team. This was also a requirement from the course staff. The team decided that the necessary roles were project leader, secretary, test manager, document manager, design manager, and code responsible. In general, the role system worked quite well, and there were no regrets of having this structure. Some problems have emerged, especially too broad responsibility areas per role, but the general impression of the role system is good. The original responsibility areas are closely followed with only few insignificant exceptions. The following paragraphs evaluate each of the chosen roles from Section 2.4.2.

Project leader

The first formal decision made by the team, was to have one superior leader, taking care of possible conflicts and with a final word when a common agreement would be hard (or impossible) to make. All team members felt this was necessary for the process to work, and the election was done at the first meeting. Since the team trusted the elected leader, there was no fear of unleashing a powerful and treacherous dictatorship. There has not been any regrets making this decision. The whole team feels that the project leader role has been working quite good, and that it was necessary to have this role in order to get a steady and controlled work process.

Document manager

Luckily, the team had one person with several years of L^AT_EX-experience, so he was elected to be the document manager. This may have been the most important role for quality assurance of the overall report. The document manager used considerable time formatting and structuring the report, but this has given great results. The team is quite pleased with the final report and does not think it could be done in a better way. To have one main responsible for the document managing resulted in a consistent report.

Design manager

Due to the framework dependency, the control of the design process was limited. Most of the design was already arranged by GMF and its underlying layers. These parts were organized by the code responsible, so he got the task of writing the content of the introduction to the design chapter. Still, it was good having a separate design manager that could write more general about the phase, and that could conclude and summarize the chapter. This made it possible to pull the code responsible out from the design process so he could focus on the implementation and its corresponding chapter instead.

Code responsible

The team found it hard to distribute responsibilities of the GMF study, since most of this process is about reading documentation and trying the functions. Naturally, this could lead to duplication of used hours. The huge amount of tasks, most of them with strict absolute deadlines, consumed much time, so the

team decided to let one person be the main responsible for both GMF and code. Due to the complexity of the GMF framework, this responsibility should have been shared between two persons. This might be one of the most unfortunate choices done by the team, since no one actually had any control over the code and implementation, and when the code responsible went away the last weekend before delivery, the team was stuck, confused, and tried to learn GMF, so that relatively important functions could be added to the application.

Secretary

The secretary has written good minutes from every meeting, with notes and comments from the parties involved. The minutes have been emailed to the team and stored on the common area for later references. This has been proven useful when looking back at the project process. One of the reasons for this is that it kept temporarily unavailable team members more updated.

Test manager

Having a main responsible for the testing and the tracking of tests was considered highly valuable and this lightened the other tasks. Even if the chapter was dependent on other processes, the responsible managed to do most of the work by him self. This ended with a good and satisfactory result.

9.2.2 Challenges and difficulties

In a project like this, there will be challenges. These challenges have to be handled, and the team did this in best possible way. The biggest challenges resulted in some delays of the phase deliveries (milestones). Even though some of the delays were more than noticeable, they did not pose an immediate threat to the project itself. The milestones not reached were internal, and they were created to help the team progressing. The result of not managing to hold these dates made the time distribution noticeable different from the one that was originally planned, as shown in the Gantt diagrams in Appendix A.3.

The next sections explain difficulties that arose during the process, and Figure 9.2 briefly shows the problems and consequence as a cause and effect diagram [75]. The most notable result is the fact that five of the phase documents were never delivered for approval, as planned. Since these were internal milestones, and the phases were completed before print, this was not considered necessary to evaluate further. Where a risk from the Table A.4 has struck, the current paragraph refers to the risk number.

Team members arrive, team members depart

Right from the beginning of the project, the team was split over the globe, literally. Two persons were missing, one of them joined the team the next day, but the other one was located in the US for two more weeks. This caused communication problems, especially since this person was responsible for the implemented application, due to his existing coding skills. To minimize the consequences of this, the team communicated by email and instant messaging to spread information, and to lighten the loss of time, the realization phase was delayed.

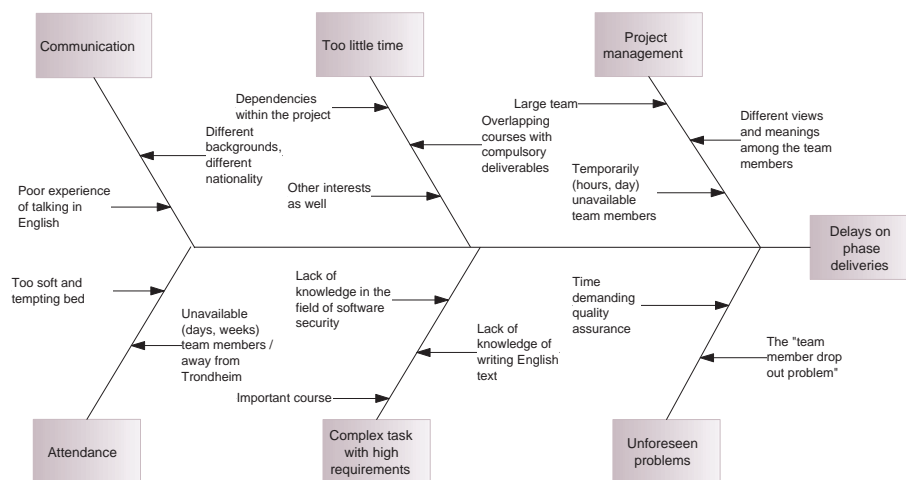


Figure 9.2: Cause and effect diagram that shows the most notable result of the problems during the project.

After a week, one of the team members left for Italy, and this introduced more trouble since the requirements specification phase was coming up. The person leaving was the phase responsible due to his earlier experience in similar tasks. Bad planning by the team maybe, but the requirements specification was temporarily put on hold. The phase before it, the preliminary studies, was not complete anyway, and the team used all available work force on that chapter while waiting.

Some days later, another person from Vietnam enrolled the course and got assigned to this team. Most of the knowledge obtained during the beginning of the project had to be retold to this person, and all emails already sent were bounced to his email address in order to help him familiarize with the project. Context dependent chats and instant messaging logs with possible valuable information were not forwarded, but still, a verbal summary was given. When the team member in Italy came back, summaries were again created. Some days later, the last member from the US arrived. Summaries were created and information was retold, for the last time. Finally, one month after the startup of the project, the first meeting with all team members present was held. The mentioned events caused delays and extra work, but the result was a large group with a potentially strong work force.

Affected entries in the risk table: 7, 8 and 13

Lack of time

The high requirements in this project, and the requested academic level on the produced material, forced a considerably large amount of time to be used on the project. While all persons in the team had other courses, some with compulsory and graded deliveries, spare time became rare, and late nights were common. Several phases depended on each other, and if one of them was late, it usually caused delays to the next ones as well. For about a week, the team suffered

from the missing content of the design chapter, and there was a lack of specified tasks to do. This time was used to quality assure already written parts of the report.

Affected entries in the risk table: 1, 4, 8 and 13

Communication and project management

Due to the fact that the team consisted of seven persons, management, coordination, and discussions within the project consumed considerably more time and effort than intended at the beginning of the project. Structuring the work force, distributing responsibilities, and keeping the team updated on all essential topics required a large amount of the available hours each week. Discussions did often emerge, and naturally, some disagreements turned up as well. Luckily, they were mostly easy to settle.

Affected entries in the risk table: 11

Complexity of frameworks

Familiarizing the team with the GMF framework together with trying out its available functions resulted in a large time consumption researching the topic. This was assisted primarily with outdated documentation, that made the process even harder. Luckily, the usage of a projector and whiteboards made it feasible without using too many hours.

Affected entries in the risk table: 14

Attendance and availability

All persons in the team had other courses, duties and doings. Some persons worked in parallel with this course, but everybody had compulsory deliveries in other courses. Some persons preferred working from 08:15 to 16:00, while other persons were more nocturnal and could not get up in the mornings. Therefore, it was difficult to gather the whole team. Some weeks it could be almost impossible, and this was a problem the team had to accept. The team ended up selecting entire working days, and people had to show up whenever they could.

Affected entries in the risk table: 11

Quality assurance

The lack of experience in using English as the main language among the team members, resulted in an overall poor quality of the written text, and it complicated the way of understanding each other. The quality assurance of the text required a vast amount of time, much larger than expected, and ambiguities easily arose. Still, the team managed to produce a decently written report and handled the problems of communication by improvising, and by trying and failing.

Affected entries in the risk table: 14

The team member drop out problem

There were some difficulties regarding the inclusion of one of the team members right from his arrival. He used a large amount of time on the tasks he was given, and the produced work was of so poor quality, that it could not be included in the report without making extensive changes to it. There were several attempts to give him different kinds of tasks, but since he had never used Java, and had problems understanding English, he did not seem qualified for neither programming nor text writing. As the team did not feel that this alone was a good enough reason to exclude him from the project, the process continued with several different attempts. Pair work and simplifying the degree of difficulty on the tasks was tried without any positive results. The team found it hard to work in this way and used more time trying to include and update the person, than what was gained by having him as team member.

Detailed and throughout explanations of the requested results were attempted, but it did not seem to help, even if the task was considered to be trivial. The rest of the team agreed upon that if he at least joined the meetings and work sessions, and tried to contribute as much as he could, the problem had to be accepted. When the member suddenly left for two weeks on vacation with only one day notice, the team decided to take action. The supervisors and the course manager were contacted, and they took more or less charge for what should be done. Two weeks later, when the current person came back, he left the project voluntarily.

Affected entries in the risk table: 10, 14

9.2.3 Phase completions and usage of time

See Figure 9.3 for the estimated phase distribution of work, and Figure 9.4 for the real distribution. The pie charts are measured in “used hours”, and are based on Table 2.1 from Section 2.2.10. Since the table excludes the last week, the charts are not describing the exact real usage of hours. Since especially the two phases evaluation and presentation was ongoing this week, the numbers are smaller than the actual amount. As shown in the figures, the planning done the first week was surprisingly consistent with the actual amount of hours registered. Even if the last week is not accounted for, the difference of only 13 hours from the planned amount is small and insignificant. Due to the high intensity in the last week, an estimation of 200 hours were used, but this is only speculation. If however it is a correct estimate, the value of used hours exceeds the estimated amount with about 190 hours, but due to the unforeseen problems explained in the last section, this was considered unavoidable.

The largest mismatch of the workload distribution is the requirement specification. This chapter became much larger than expected, and consumed 71% more hours than originally planned. A large amount of hours was used to re-structure it after comments from the supervisors and also to plan the overall content of the chapter. The same reason applies to the testing chapter, with a 46% increase of used hours. The reason for the small amount of used hours on lectures and self study, is that most of the lectures were considered unnecessary, and the used hours on self study were hard to keep track of and was instead registered to their most suitable phase. The rest of the phases, except for the

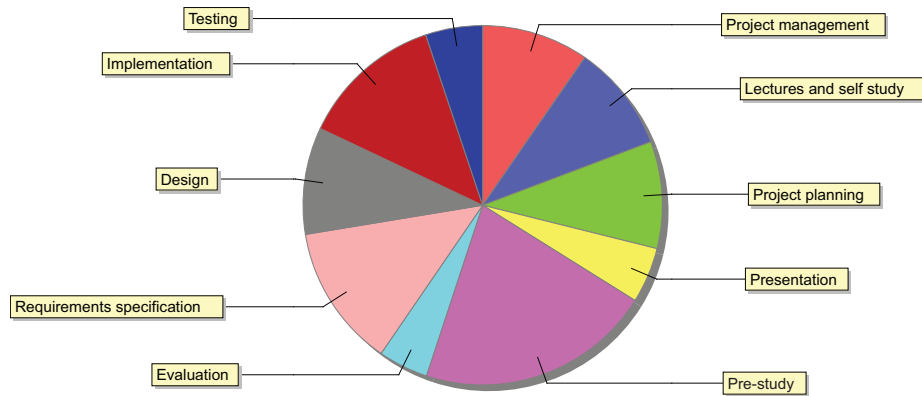


Figure 9.3: Estimated hourly distribution of work in the different phases.

phases worked on the last week, has only small discrepancies, consistent with the planned amount. The phases mentioned, which stand out, are shown as “exploded” pies in the latter figure.

The team was behind schedule already at the pre-study phase, since it was larger than expected. The “security modeling” term is broad, and there is a substantial amount of information about the topic. The content of the pre-study chapter is only what was considered vital for this project, with suggestions for the potential next further additions to the implemented software. Several sections and pages were written in addition to this, but removed later on due to no actual relevance to the given task.

The requirements specification was started at the end of the pre-study phase, and it was done in parallel with other tasks such as beginning design phase, and structuring the later chapters, testing and evaluation. When the team considered the requirements specification complete, it was not approved by the supervisors due to bad flow in the chapter. The entire chapter had to be restructured, but this was done in about one week.

After the completion of pre-study and soon requirements specification, the

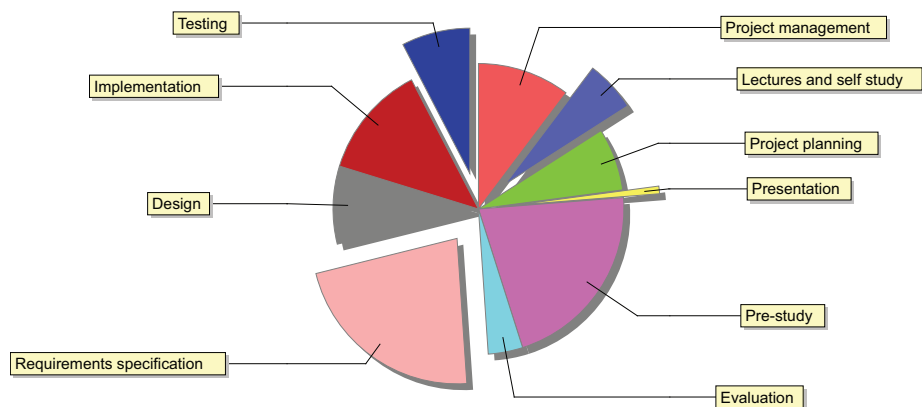


Figure 9.4: Real hourly distribution of work in the different phases.

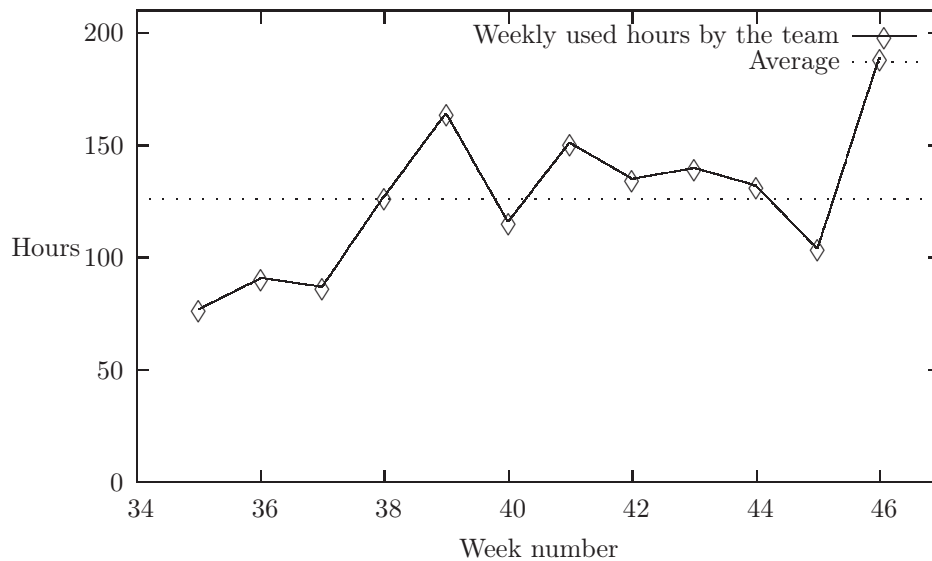


Figure 9.5: Graph showing total weekly used hours on the project by the team.

design was delayed, and it was hard doing anything with it. The frameworks used by SeaMonster had to be documented before any real progress could happen. Both testing and the installation manual was dependent on this phase, and only one person really understood the frameworks. As the phase was added to the report, the rest of the team started to write the testing and evaluation chapter. Quality assurance was done constantly, and more and more sections were corrected, and in some cases totally rewritten.

There were difficulties finding programs that could create the diagrams requested. Several was tried, but they all failed the wanted appearance and look. The Gantt diagrams were actually drawn five times in four different programs before accepted for inclusion in the report.

To compensate for all the delays, the phases were ran more in parallel than expected, and this is clearly shown in the Gantt diagrams in Appendix A.3. In this way, the phases with corresponding chapters continuously evolved. Since most of the phases were highly depended on each other, and the team consisted of too many people to focus on only one chapter at the time, this was considered a good solution.

As Figure 9.5 shows, the project started out with few hours used, but later on the used work hours increased and became 128 in average. The total amount of used hours per person per week was 21 hours, which is a little less than the expected 24 hours planned. Some people have worked more than others in general, and since the weeks when the team was missing persons are a part of the estimate, it was hard to reach this goal.

9.2.4 Tools used

During the span of the project, the amount of different applications and tools used have been quite large. There have been mostly positive experiences from these, and most of them were already familiar to the team members. The use of vim as the main \TeX editor, and the document preparing system, \LaTeX , has proven wise as predicted. Some of the tools used for minor tasks have proven to be problematic and quarrelsome. The reason for this is mostly the lack of experience in using them, but nevertheless, they have consumed time that could have been used in a better way. There were problems regarding image exporting from Adobe Illustrator [1], a vector based image editor. The same with Inkscape [46], and the file type conversion between the different vector graphics formats was in general not friendly at all.

9.3 Involved parties

This section consists of general thoughts and opinions about the most important involved parties throughout the span of the project.

9.3.1 The customer

The communication with the representatives from SINTEF was performed smoothly the entire span of the project, and the team received both assistance and guidelines when needed without any complaints or long delays. Both representatives have shown belief and good faith, and they encouraged the team with not just positive response, but also helpful constructive criticism. Face to face contact was favourable, so the team benefited from their location in Trondheim. Even though SINTEF informed the team about their busy schedule, they always showed up at the meetings at the scheduled times with high spirits.

9.3.2 The team

There have been some communication issues in the team during the entire project, but some cases have been worse than others. The large number of team members, all with different time schedules, made it hard finding time for meetings including all team members. Together with persons arriving and leaving, the team members' different backgrounds and spoken language, made transmitting a message surprisingly difficult if the intention of the message was decisive.

9.3.3 The supervisors

In general, the impression of the supervisors have been good, but they have not been consulted much. They have willingly helped the team, and they provided guidelines and tips regarding the process and report. They have also served as a middle link between the course staff and the team, but this has only been needed a couple of times. The fact that they know, or at least should know, what the report must contain has been useful and helpful for the team.

9.3.4 The course staff

Except from the supervisors, there were not many encounters with the course staff. After the “kick-off start”, they hibernated back to their offices, and did not contact the team. Messages went through the supervisors, but when the drop-out problem came up, the first contact was established by the team. This is more explained in Section 9.2. All the help needed was received, and the process proceeded as hoped without using too much time.

9.4 The course

There have been mixed feelings about the Customer driven project-course during the passed semester. The idea of the course is good, giving the students a real task to work on, with a real customer instead of the regular student assistants from other courses.

During the “kick-off start” at the end of August, ten teams were randomly created from about 50 people, and the tasks were randomly distributed. Short descriptions of the tasks were given in textual form, and about forty-five minutes were used to describe the course. After that, the teams were more or less on their own until the end. Only ten minutes were given to get to know each other, before a meeting with the customer was scheduled. All members of the team were confused, and the customer described the team as question marks after the first meeting. This could clearly be done differently and better. By preparing the teams more thoroughly in advance of the “kick-off” and first customer meeting, most of the confusion could have been avoided.

The only guidelines given of writing this report was an unclear, vague and sometimes misleading and inconsistent information document [5] given by the course staff. It was told that the document was an extraction from the most important content from the hand outs given earlier years, and it could easily be seen on the quality of the paper that it had not been inspected well enough. The document explains how to structure the report, and what kind of content it has to include. The specification presumes the process is waterfall based and that the task includes a huge amount of programming. The text is probably left overs from earlier years, but several of the tasks given this year, including the one connected to this project, are not suitable for a process like the information document explains. This project consists of an extensive pre-study and learning frameworks, not producing an extensive amount of code that the information document calls for.

It is a good thing to mix the teams with both local students and exchange students, and this certainly improves the skills in English by using it as the main language. However, as this team has experienced, exchange students could have a different background regarding earlier courses they have attended, compared to the local ones, and this must be compensated by the rest of team. A better background check should be done on the students taking this course, to see if they are qualified for one of the tasks. The last chance of enrolling the course should be no later than the kick-off date, to avoid the difficulties of including another person two weeks after the start of the project.

Due to the excessive amount of hours spent on this project compared to other courses, the complexity of the tasks and the requirements on the deliveries are

too high to be a 15 credit points course. This could either be raised, or the requirements lowered a bit.

Even if there are some negative aspects of the process, there are several positive ones as well. The course has supplied the opportunity to use a large spectre of methods learned in other courses regarding computer science, and there has been plenty of fun and challenging work, large responsibility in the team, and all team members have learned a lot during this course.

9.4.1 Lectures and seminars

Seminars and lectures were given by the course staff with strong recommendations of participation by the entire team in some of them. In general, the team did not feel the advantage of any of the seminars compared to how many hours they spanned (eight hours each). Still, they could be thought of as getting to know the other team member better, but should be more combined with actual relevant work for the project.

Most of the lectures given in this course were in Norwegian, and this should definitely be avoided since foreign students are taking the course. Since this is common knowledge, at least in the course staff, it is amazing that it was done in this way. Several of the team members had already attended similar lectures to the ones that were given, and none of them were considered very useful. The lecture about technical report writing should not be too focused on bibliography and reference lists, but instead use more of the time on general issues.

9.5 The project task

The task given by the customer was considered somewhat vague at the beginning, and none of the team members did understand what the concept and ideas actually were. The text could be more specific, and the focus on graphical modeling should be more explicitly stated in the text. The requirements given were good, and they guided the team in a positive way towards a complete requirements specification. There were some problems about the recommendation of taking the course TDT4237 - Software security, and this should be noted earlier. Only one person had this course at the start of the project, but two other members skipped their originally planned courses to take this. The task for this project and the current course were related, but it was still considered more than sufficient to have three of the six members in the team taking the course.

The task given was considered highly interesting by the team members, and if the tasks could be chosen at will, most of the team would still choose this task. Some of the team members will hopefully join projects improving the application, and also use the great opportunity to write a scientific report with help and guidelines from the customer. Other than the few areas mentioned, there are only good things to say about the given task.

9.6 Fulfillment of project goals

Each of the goals made in the project are evaluated in this section. This includes the objectives for the project, called “the result goals”, the closely linked business

requirements, and the measurable goals for the project, called “measurements of project effects”. They are listed in Section 2.2.5, Section 2.2.6, and Section 3.2 respectively.

The task given is considered to be fulfilled by the implemented application and report delivered to the customer. This assumption is primarily based on positive feedback from the customer, but also that all the result goals are reached, and confirmed reached by the customer, except for RG.06.

Result goals RG.01, RG.04, and RG.05 are covered by report itself (especially Chapter 3, while the SeaMonster application complies with and completes RG.02. Since the implementation is documented, and the programming language and the used frameworks are well known in the community of software development, the prototype can easily be extended where this is needed. This results in the accomplishment of RG.03.

M.01, M.04 and M.05 have been fulfilled, and this is confirmed by the customer. Prior to the public release of the application, it is hard to estimate the value of the work regarding the goal RG.06, together with the rest of the last two measurable goals (M.02 and M.03). Uncertainties of the marketing and acceptance of the tool is complicating this even more. However, the customer will help the team to market the produced results, so the goals and effects are feasible to reach. Still, there are only positive feelings about the accomplishment of these, and there is no reason to believe that they are not fulfilled, with exception of M.02. This particular measurement has been hard to reach with an application like SeaMonster, and only partially covered with the recommendation of adding a CVE/CWE identifier¹ to the exit node in the modeled VCG². This identifier can be used while searching the current online database to quickly gain information about the vulnerability.

After the testing phase, it is possible to analyze the fulfillment of the business requirements in the light of the fulfillment of the system requirements. The functional requirement groups, presented in Section 4.3.3, are almost all accomplished, as confirmed by tests. The requirements concerning the diagrams - i.e. ATF, VCGF, MCF, and MF - have been positively tested. In the same manner, requirements related to the intuitive GUI and storage features - i.e. IGF and SF - have been fulfilled by the implemented application.

As stated in Section 4.3.3, Some requirements in groups EF, ATF (attack tree), SF, and SECF (security requirement), have not been fulfilled. However, it is important to remark again that they do not represent a real problem since the most of them are considered low level requirements. Future developments may focus more on them. About non-functional requirements groups, exposed in Section 4.4, all of them are satisfied with the application. At this point, referring to the relations between system and business requirements traced in Section 4.4.5, it is possible to say more about the last ones.

The first business requirement, BR.01, is fulfilled if the implemented application is to be used during the development phases. For this to apply, the tool should be spread and widely used, and also be accepted as user friendly, intuitive and easy to use by the targeted areas. The usage of GMF and the underlying frameworks take care of the latter, but the first one is, as the measurements of project effects, affected by the software release and the need for an evolving

¹See Section 3.4 for more information.

²See Section 3.5.1 for more information.

community.

BR.02 and BR.03 are fulfilled by the implemented application, while BR.04 is considered mostly accomplished, as the application is already able to store diagrams using an XML format as required. At the same time, the diagrams' models are defined by using an XMI description. What is still missing is the adherence of such XMI definition to a common standard like UML.

BR.05 is also affected by the marketing and tool-acceptance by the community, but the work done can definitively be used to reduce, or maybe eliminate, the gap between security experts and software developers as discussed in Section 3.3.

By releasing the software under open source licenses and distributing the code on the SourceForge repository, the goal BR.06 is also reached.

9.7 Further work

Most of the requirements and goals are achieved by the work done in this project. The application is still a prototype and could gain benefits by improvements in certain areas. Adding usage of the UML metamodel, strengthen the linking between the views and making this more user friendly are possible additions. Minor issues like automatic cost calculation in the attack trees, enhancing the VCG modeling by adding content to the compound nodes, support for italic sub-text in components and improving the “open model” methods are just as well subjects to variation.

Other subjects of further work could be adding support for more modeling methods so that the application is able to cover a larger specter of security modeling. In addition external databases could be integrated in SeaMonster, for instance the CVE or the CWE database. All these improvements could be seen as positive additions to the application.

9.8 Summary

The process has taken time, making it hard to get good results in other courses. But in general, there have been a reasonable amount of positive experiences from the course. There have been great responsibilities, some fun, plenty of learning and much work. Sometimes, too much work, but it could be good having a lot to do. Team members never experienced boredom, at least.

The emerged problems have been dealt with, and all team members have gained valuable experiences in several fields, including technical report writing, some programming, and working in a random team. The cooperation in the team was good, and the team found the subject fascinating. The produced report and the implemented prototype are of high quality, and the latter is made in such a way that it is open for improvements and additions where this is needed. Most of the requirements and goals are reached and fulfilled, and the team are proud of the accomplished results and consider the project successful and well done. Figure 9.6 shows this in a cause and effect diagram.

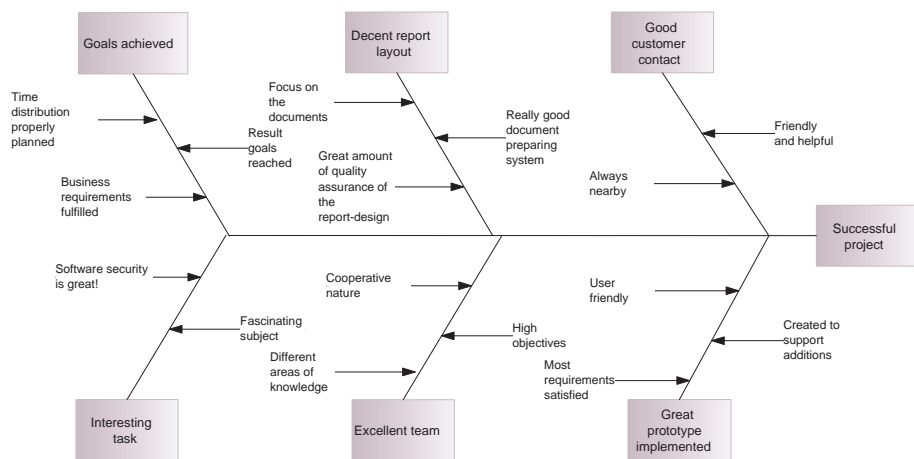


Figure 9.6: Cause and effect diagram for a successful project.

Chapter 10

Project conclusion

Today, security engineers are lacking a common way to write down their knowledge. Among other things, this is caused by the usage of different general purpose modeling applications that do not contain any restrictions concerning notations. When the security engineers are not able to impart their knowledge to other security engineers and software developers, a worse general understanding of the vulnerabilities may emerge.

The customer wanted to change this trend, by making a common platform for modeling security. This task was given to the SeaMonster project. More precisely, the SeaMonster project was supposed to make a prototype solely for modeling relations between security vulnerabilities. The goal was to increase the general understanding of vulnerabilities in order to reduce the amount of time it takes to model security. In addition, a long term consequence could be a reduction in the amount of vulnerabilities in released software.

The modeling of security can be summarized in three views; vulnerability cause modeling, threats and attacks, and countermeasures. Respectively, the modeling techniques; vulnerability cause graphs, attack trees, and misuse case diagrams were chosen to represent these views in SeaMonster.

The project had to realize an application which could model security in an easy and professional way. For this reason, the graphical modeling framework GMF was used. This resulted in an application with features surpassing any application the project could have developed from scratch, using the same amount of time.

Testing showed that the application works in a satisfying way, and it can be used for making diagrams in the chosen modeling techniques. Some framework bugs were discovered, but none of them were critical. Most of the requirements and result goals were fulfilled, which is sufficient as this is a prototype.

If the measurement of project effects are fulfilled, then the goals of the SeaMonster project are reached. Hopefully, the SeaMonster application will be able to satisfy the result goal concerning a notable reduction in common security vulnerability occurrences. If this is achieved, the project could be characterized as a great success. Either way, both the team and the customer are very satisfied with the result of the project.

Bibliography

- [1] Adobe Systems Incorporated. *Adobe - Illustrator. Digital Graphic Design Software - Computer Graphic Design Solutions*. <http://www.adobe.com/products/illustrator/> Last date accessed 2007-10-31.
- [2] Scott W. Ambler. *Evolving the AUP lifecycle*. <http://www.ambysoft.com/unifiedprocess/aupLifecycle.html> Last date accessed 2007-10-05.
- [3] Amenaza. *Amenaza Technologies Limited*. <http://www.amenaza.com/> Last date accessed 2007-10-26.
- [4] K.G. Coffman A.M. Odlyzko. *The size and growth rate of the Internet*. AT & T Labs Research. <http://www.dtc.umn.edu/~odlyzko/doc/internet.size.pdf> Last date accessed 2007-10-31.
- [5] Jon Atle Gulla Reidar Conradi Jon Espen Ingvaldsen. *Customer driven project information document*. <http://www.idi.ntnu.no/emner/tdt4290/docs/TDT4290InfoBook2007-doc> Last date accessed 2007-11-14.
- [6] Bjørn Andersen and Tom Fagerhaug. *Root Cause Analysis: Simplified Tools and Techniques*. ASQ Quality Press, 2006.
- [7] Apache. *Subversion version control system*. <http://subversion.tigris.org/> Last date accessed 2007-09-10.
- [8] Shanai Ardi, David Byers, Per Håkon Meland, Inger Anne Tøndel, and Nahid Shahmehri. How can the developer benefit from security modeling? Technical report, Linköpings universitet, Department of computer and information science and SINTEF, 2007.
- [9] Shanai Ardi, David Byers, and Nahid Shahmehri. Towards a structured unified process for software security. Technical report, Linköpings universitet, Department of computer and information science, 2006. <http://portal.acm.org/citation.cfm?id=1137630> Last date accessed: 2007-09-19.
- [10] Matt Barret. *FDCC Tech exchange*. <http://nvd.nist.gov/scap/docs/FDCC-Tech-Exchange-SCAP-2007-801.pdf> Last date accessed: 2007-10-06.
- [11] Eric Braude. *Software Engineering: An Object-Oriented Perspective*. Wiley, 2000.

- [12] Andrew Buttner and Neal Ziring. *Common Platform Enumeration Specification*. http://cpe.mitre.org/files/cpe-specification_2.0.pdf Last date accessed 2007-10-05.
- [13] David Byers, Shanai Ardi, Nahid Shahmehri, and Claudiu Duma. Modeling software vulnerabilities with vulnerability cause graphs. Technical report, Linköpings universitet, Department of computer and information science, 2006.
- [14] Carnegie Mellon University. *The CERT Coordination Center web page*. <http://www.cert.org/certcc.html> Last date accessed 2007-09-24.
- [15] Carnegie Mellon University. *The CERT Secure Coding project page*. <http://www.cert.org/secure-coding/> Last date accessed 2007-09-24.
- [16] Carnegie Mellon University. *The CERT Secure Coding Standards page*. <http://www.securecoding.cert.org/> Last date accessed 2007-09-24.
- [17] The CERT web page. *Vulnerability remediation*. <http://www.cert.org/vuls/> Last date accessed 2007-09-24.
- [18] M. Chitnis, P. Tiwari, and L. Ananthamurthy. *Creating Use Case Diagrams*. <http://www.developer.com/design/article.php/2109801> Last date accessed 2007-09-21.
- [19] Corel Corporation. *CorelDRAW Graphics Suite X3*. <http://www.corel.com/servlet/Satellite/us/en/Product/1150981051301> Last date accessed 2007-10-31.
- [20] N. Davis. Developing secure software. *The DoD Software Tech News*, 8, 2005.
- [21] Dr Dobbs. *Attack trees*. <http://www.ddj.com/architect/184411129> Last date accessed 2007-09-22.
- [22] Bruce Eckel. *Thinking in Java - 3rd Edition*. bruceeckel.com, 2000.
- [23] The Eclipse Foundation. *The Eclipse Public Licence*. <http://www.eclipse.org/legal/epl-v10.html> Last date accessed 2007-10-18.
- [24] Eclipse Foundation. *Graphical Modeling Framework Documentation*. <http://www.eclipse.org/gmf/> Last date accessed 2007-11-20.
- [25] Eclipse open source community. *The Eclipse Graphical Modeling Framework*. <http://www.eclipse.org/gmf/> Last date accessed 2007-09-21.
- [26] Eclipse open source community. *The Eclipse Modeling Framework*. <http://www.eclipse.org/> Last date accessed 2007-09-21.
- [27] Eclipse open source community. *The Eclipse Modeling Framework Project*. <http://www.eclipse.org/modeling/emf/> Last date accessed 2007-11-12.
- [28] Alexander Feder. *Bib_T_EX*. <http://www.bibtex.org/> Last date accessed 2007-09-09.

- [29] E. Fernandez-Medina, A. Martinez, C. Medina, and M. Piattini. Uml for the design of secure databases: Integrating security levels, user roles, and constraints in the database design process. Technical report, University of Castilla-La Mancha, 2004.
- [30] Free Software Foundation. *FSF Licences*. <http://www.fsf.org/licensing/licenses/> Last date accessed 2007-10-18.
- [31] Gentleware. *Poseidon to UML*. <http://www.gentleware.com/> Last date accessed 2007-10-08.
- [32] C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, 2004.
- [33] GNU Project. *The General Public Licence*. <http://www.gnu.org/licenses/gpl-3.0.txt> Last date accessed 2007-10-18.
- [34] GNU Project. *The Lesser General Public Licence*. <http://www.gnu.org/licenses/lgpl-3.0.txt> Last date accessed 2007-10-18.
- [35] GNU Project - Free Software Foundation. *The Free Software Definition*. <http://www.gnu.org/philosophy/free-sw.html> Last date accessed 2007-10-18.
- [36] GNU Project - Free Software Foundation. *What is Copyleft?* <http://www.gnu.org/copyleft/copyleft.html> Last date accessed 2007-10-18.
- [37] Gnuplot contributors. *Gnuplot homepage*. <http://www.gnuplot.info> Last date accessed 2007-11-07.
- [38] Karen Mercedes Goertzel, Theodore Winograd, Holly LYnne McKinley, Lyndon Oh, Michael Colon, Thomas McGibbon, Elaine Fedchak, and Robert Vienneau. Software security assurance: A state-of-the-art report. Technical report, IATAC, 2007.
- [39] Google. *Google accounts*. <https://www.google.com/accounts/> Last date accessed 2007-09-24.
- [40] Spyros T. Halkidis, Alexander Chatzigeorgiou, and George Stephanides. A practical evaluation of security patterns. Technical report, Department of Applied Informatics University of Macedonia, 2005.
- [41] John C. Hawkins and E. B. Fernandez. Extending use cases and interaction diagrams to develop distributed system architecture requirements. Technical report, Department of Computer Science & Engineering, Florida Atlantic University, 1997.
- [42] David C. Howard, Michael Leblanc. *Writing Secure Code*. Microsoft Press, 2002.
- [43] M. Howard. Building more secure software with improved development processes. *IEEE Security & Privacy*, 2, 2004.

- [44] IEEE. *IEEE recommended practice for software requirements specifications*, 1998. <http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=15571> Last date accessed 2007-10-22.
- [45] IEEE. *IEEE standard for software test documentation*, 1998. <http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=16010> Last date accessed 2007-10-22.
- [46] inkscape.org. *Inkscape - Draw Freely*. <http://www.inkscape.org> Last date accessed 2007-10-08.
- [47] International Organization for Standardization. *ISO 8601:1988. Data elements and interchange formats — Information interchange — Representation of dates and times*. International Organization for Standardization, pub-ISO:adr, 1988. See also 1-page correction, ISO 8601:1988/Cor 1:1991.
- [48] Douglas Isbell. *Mars Climate Orbiter Failure Board Releases Report*. NASA. <http://mars.jpl.nasa.gov/msp98/news/mco991110.html> Last date accessed 2007-10-06.
- [49] Isograph. *Isograph The World's Leading Reliability Analysis Software*. <http://www.isograph-software.com/atpover.htm> Last date accessed 2007-10-26.
- [50] Isograph. *Isograph The World's Leading Reliability Analysis Software, SecuriTree tutorial*. <http://www.amenaza.com/downloads/docs/Tutorial.pdf> Last date accessed 2007-10-31.
- [51] Andrew Jaquith. *The security of applications: Not all are created equal*. Technical report, @stake, 2002.
- [52] Jones, Larry Baushke, Mark D. Price DerekR. *CVS - Concurrent Versions System*. <http://www.nongnu.org/cvs/> Last date accessed 2007-09-22.
- [53] F. Lee Brown Jr., James DiVietri, Graziella Diaz de Villegas, and Eduardo B. Fernandez. *The authenticator pattern*, 1999.
- [54] Jan Jürjens. *UMLsec*. <http://www4.in.tum.de/~umlsec/> Last date accessed 2007-10-05.
- [55] Jan Jürjens. *Secure Systems Development With UML*. Springer, 2005.
- [56] Peter Kemper and William Huston Sanders. *Computer Performance Evaluation: Modelling Techniques and Tools*. Springer, 2003.
- [57] The L^AT_EX-project. *L^AT_EX-documentation*. <http://www.latex-project.org/guides/> Last date accessed 2007-09-18.
- [58] latexeditor.com. *L^AT_EXEditor*. <http://www.latexeditor.com> Last date accessed 2007-10-08.
- [59] Brian Marick. *New models for test development*. Technical report, Reliable Software Technologies, 1999.
- [60] Gary McGraw. *Software Security: Building Security In*. Addison-Wesley Professional, 2006.

-
- [61] Microsoft Corporation. *About Microsoft*. <http://www.microsoft.com/about/default.msp> Last date accessed 2007-10-15.
- [62] Microsoft Corporation. *Excel Home Page - Microsoft Office Online*. <http://office.microsoft.com/en-us/excel/FX100487621033.aspx> Last date accessed 2007-10-01.
- [63] Microsoft Corporation. *Microsoft Threat Modeling Tool for building security threat models*. <http://www.microsoft.com/downloads/details.aspx?FamilyID=62830f95-0e61-4f87-88a6-e7c663444ac1&displaylang=en> Last date accessed 2007-09-24.
- [64] Microsoft Corporation. *Visio homepage*. <http://office.microsoft.com/nb-no/visio/default.aspx> Last date accessed 2007-09-24.
- [65] Russ Miles and Kim Hamilton. *Learning UML 2.0*. O'Reilly, 2006.
- [66] The MITRE Corporation. *Common Configuration Enumeration (CCE): Unique Identifiers for Common System Configuration Issues*. <http://cce.mitre.org> Last date accessed 2007-09-26.
- [67] The MITRE Corporation. *Common Vulnerabilities and Exposures*. <http://cve.mitre.org/> Last date accessed 2007-09-22.
- [68] The MITRE Corporation. *Common Weakness Enumeration*. <http://cwe.mitre.org/> Last date accessed 2007-09-25.
- [69] The MITRE Corporation. *CPE - Common Platform Enumeration*. <http://cpe.mitre.org/> Last date accessed 2007-10-05.
- [70] The MITRE Corporation. *oval - Open Vulnerability and Assessment Language*. <http://oval.mitre.org/> Last date accessed: 2007-10-06.
- [71] The MITRE Corporation. *oval Interpreter download page*. <http://oval.mitre.org/language/download/interpreter/index.html> Last date accessed: 2007-10-06.
- [72] No Magic, Inc. *MagicDraw UML*. <http://www.magicdraw.com> Last date accessed 2007-10-08.
- [73] Open Source Development Network. *Sourceforge*. <http://sourceforge.net> Last date accessed 2007-11-20.
- [74] The Open Source Vulnerability Database. *OSVDB: The Open Source Vulnerability Database*. <http://osvdb.org/> Last date accessed 2007-10-01.
- [75] The quality library. *Ishikawa diagram*. <http://mot.vuse.vanderbilt.edu/mt322/Ishikawa.htm> Last date accessed 2007-11-14.
- [76] Zack Rusin. *Open Source Licences Comparison*. http://developer.kde.org/documentation/licensing/licenses_summary.html Last date accessed 2007-10-18.

- [77] Lillian Røstad. An extended misuse case notation: Including vulnerabilities and the insider threat. Technical report, Norwegian University of Science and Technology, Department of Computer and Information Science, 2004.
- [78] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad. *Security Patterns: Integrating Security and Systems Engineering*. Wiley, 2006.
- [79] Kendal Scott. *Unified Process Explained*. Addison-Wesley, 2002.
- [80] Securityfocus. *Securityfocus web page*. <http://www.securityfocus.com/> Last date accessed 2007-09-25.
- [81] Securityfocus. *Security Focus about the BugTraq mailing list*. <http://www.securityfocus.com/archive/1/description> Last date accessed 2007-09-25.
- [82] Guttorm Sindre and L. Opdahl. Eliciting security requirements with misuse cases. Technical report, Norwegian University of Science and Technology, Department of Computer and Information Science, 2000.
- [83] SINTEF, Telenor. *The CORAS UML Profile*. <http://coras.sourceforge.net/> Last date accessed 2007-10-26.
- [84] SmartDraw.com. *SmartDraw Business Graphics Software*. <http://www.smartdraw.com> Last date accessed 2007-09-15.
- [85] Christopher Steel, Ramesh Nagappan, and Ray Lai. Core security patterns. *Prentice Hall*, 2006.
- [86] SUN Microsystems. *Code Conventions for the Java Programming Language*. <http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html> Last date accessed 2007-11-20.
- [87] SUN Microsystems. *JavaServer Pages Technology*. <http://java.sun.com/products/jsp/> Last date accessed 2007-11-18.
- [88] Sun Microsystems. *J2SE 5.0*. <http://java.sun.com/j2se/1.5.0/> Last date accessed 2007-11-06.
- [89] SVG Working group. *Scalable Vector Graphics (SVG)*. <http://www.w3.org/Graphics/SVG/> Last date accessed 2007-10-31.
- [90] Frank Swiderski and Window Snyder. *Threat Modeling*. Microsoft Professional, 2004.
- [91] Symantec Corporation. *Symantec web page*. <http://www.symantec.com/index.jsp> Last date accessed 2007-09-25.
- [92] Telenor ASA. *About Telenor*. <http://www.telenor.com/about/> Last date accessed 2007-10-31.
- [93] The U.S. Commerce Department. *CVE Statistics*. <http://nvd.nist.gov/statistics.cfm> Last date accessed: 2007-09-27.

-
- [94] U.S. Commerce Department. *eXtensible Configuration Checklist Description Format (XCCDF)*. <http://nvd.nist.gov/xccdf.cfm> Last date accessed: 2007-10-06.
 - [95] The U.S. Commerce Department. *The Information Security Automation and The Security Content Automation Protocol*. <http://nvd.nist.gov/scap.cfm> Last date accessed 2007-10-01.
 - [96] U.S. Commerce Department. *National Vulnerability Database CVSS Scoring*. <http://nvd.nist.gov/cvss.cfm> Last date accessed: 2007-10-06.
 - [97] The U.S. Commerce Department. *National Vulnerability Database web page*. <http://www.nvd.nist.gov/> Last date accessed 2007-09-24.
 - [98] vim open source community. *vim online*. <http://www.vim.org> Last date accessed 2007-09-15.
 - [99] Visual Paradigm International. *Visual Paradigm for UML*. <http://www.visual-paradigm.com/product/vpuml/> Last date accessed 2007-11-14.
 - [100] Howard Wolpe. *GAO Report: Patriot missile defence*. <http://www.fas.org/spp/starwars/gao/im92026.htm> Last date accessed 2007-10-06.
 - [101] Xara Group Limited. *Xara Xtreme illustration and drawing software*. <http://www.xara.com/products/xtreme/> Last date accessed 2007-10-31.
 - [102] Joseph Yode and Jeffrey Barcalow. *Architectural patterns for enabling application security*, 1997.

Appendix A

Project directive appendix

A.1 Involved parties

See Table [A.1](#), Table [A.2](#), and Table [A.3](#) for contact info for the customer, the supervisors, and the team, respectively.

The customer	
Name:	SINTEF
Address:	Strindveien 4
Zip code:	7465
City:	Trondheim
Customer contact	
Name:	Per H. Meland
Phone:	+47 73 59 29 41
E-mail:	per.h.meland-at-sintef.no

Table A.1: Contact information for the customer.

Name	E-mail	Phone
Renate Kristiansen	renatek-at-idi.ntnu.no	+47 47 08 10 81
Basit Ahmed	basit-at-idi.ntnu.no	+47 73 55 04 35

Table A.2: Contact information for the supervisors.

Name	E-mail	Phone
Egil Trygve Baadshaug	egiltryg-at-stud.ntnu.no	+47 90 67 54 75
Eilev Hagen	eilev-at-stud.ntnu.no	+47 41 50 85 63
Kris-Mikael Krister	krismika-at-stud.ntnu.no	+47 97 15 26 92
Eirik Benum Reksten	eirikben-at-stud.ntnu.no	+47 90 65 09 81
Daniele Spampinato	spampina-at-stud.ntnu.no	+47 46 37 66 83
Ketil Sandanger Velle	ketilsan-at-stud.ntnu.no	+47 95 81 37 63

Table A.3: Contact information for the project team.

A.2 Abstract for the FP7-ICT project SHIELDS (215995)

Software systems continue to be crippled by security vulnerabilities. One of the reasons for this is that information on known vulnerabilities is not easily available to software developers, or integrated into the tools they use. The main objective of SHIELDS is to increase software security by bridging the gap between security experts and software practitioners and by providing the software developers with the means to effectively prevent occurrences of known vulnerabilities when building software. We will achieve this objective by developing novel formalisms for representing security information, such as known vulnerabilities, in a form directly usable by development tools, and accessible to software developers. This information will be stored in an internet-based Security Vulnerabilities Repository Service (SVRS) that facilitates fast dissemination of vulnerability information from security experts to software developers. We will also present a new breed of security methods and tools (some open source, some commercial) that are constantly kept up-to-date by using the information stored in the SVRS. In addition to the SVRS, and new security tools, we will create a SHIELDS Compliant certification for tools and a SHIELDS Verified logo program for software developers that will offer an affordable and yet technically effective evaluation and certification method in the fight against common security vulnerabilities. Commercial exploitation will be through these programs, the tools, and through subscriptions to the repository (parts will be free). The consortium consists of two universities and three major research institutes, with complementary leading expertise in the technical areas of the project, one large software developer, and two SMEs that specialise in security consulting, security evaluations and development of secure software.

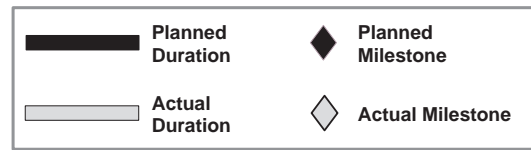


Figure A.1: The legend used in the Gantt diagrams.

A.3 Concrete project work plan

A.3.1 Overall work plan

After planning the work the first time, the result became as shown in Figure A.2. After completion of the first iteration, some changes were made to reflect the actual time used. The new work plan is shown in Figure A.3. The most notable changes is less time for testing since this phase was smaller than originally thought, and also the addition of “Installation manual” that was forgotten in the initial diagram. The legend used in the Gantt diagrams is shown in Figure A.1.

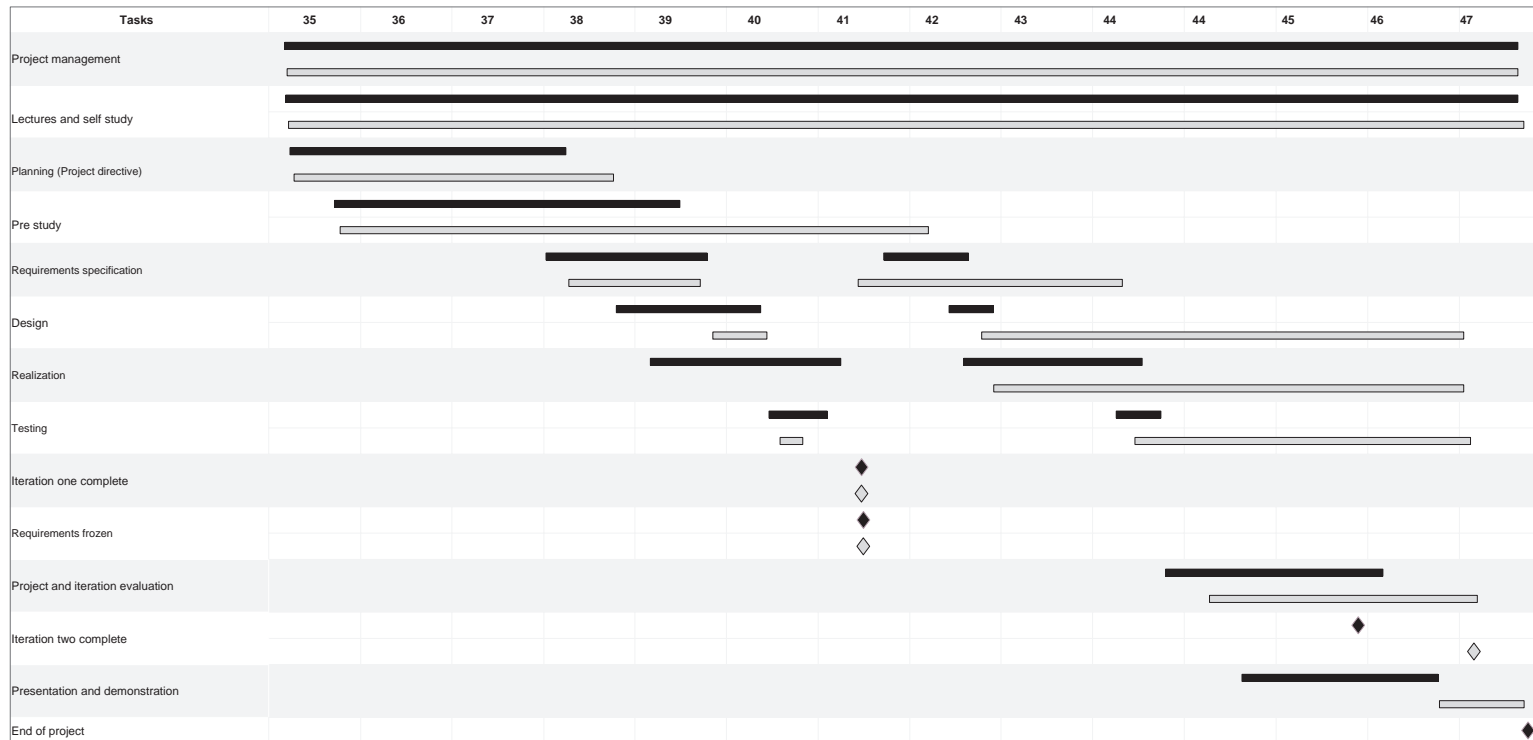


Figure A.2: Estimated plan of work in form of a Gantt diagram.

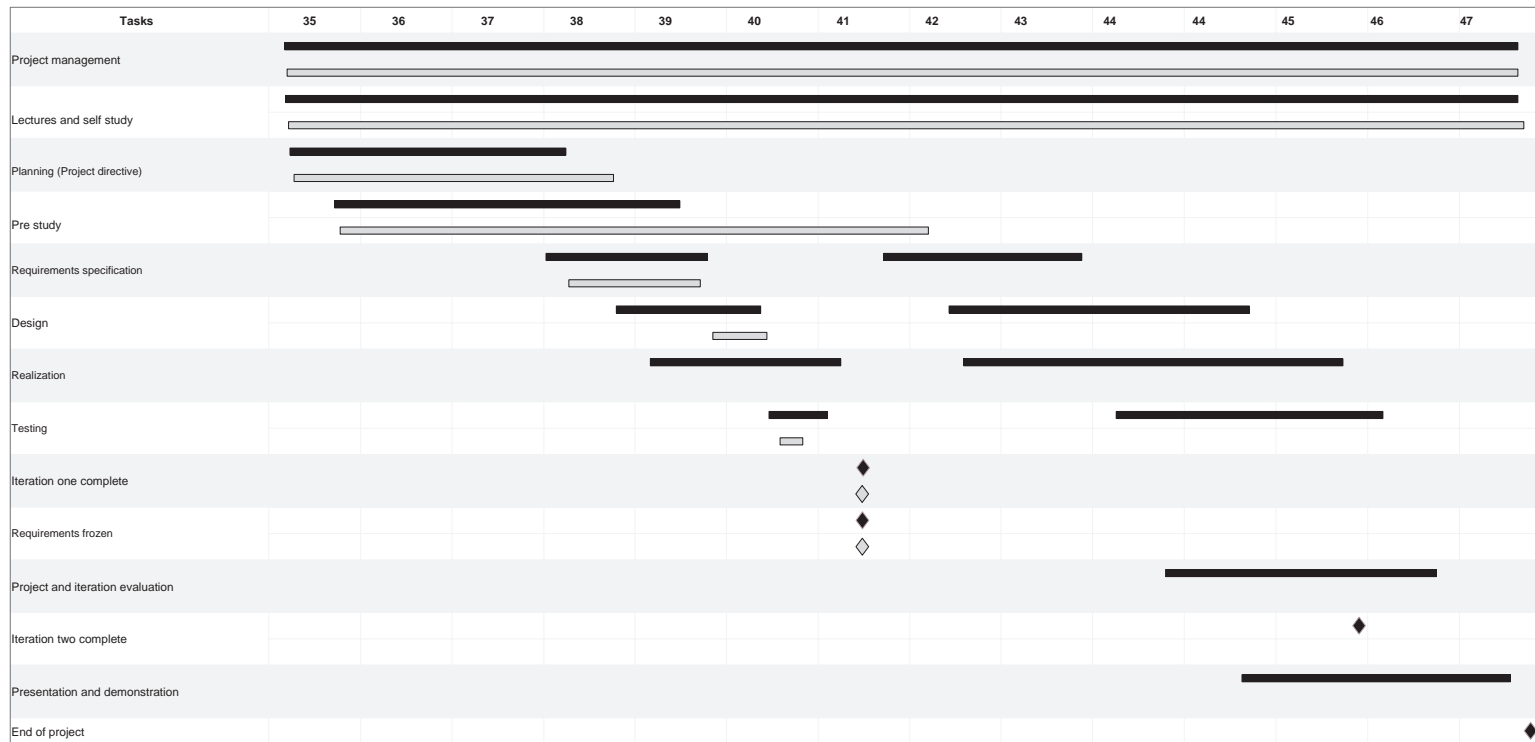


Figure A.3: Work plan in form of a Gantt diagram after iteration one.

A.3.2 Detailed work plan

Gantt diagrams were created in detail for some of the phases. Figure A.4 shows the planned distribution of time for the Project directive phase, while Figure A.5, Figure A.6 and Figure A.7 shows for Preliminary studies, Requirements specification and Testing, respectively.

A.3.3 Risk analysis for the project.

Some risks are listed in Table A.4. The probabilities are rated high, medium and low. High will have a probability of over 70% for happening, medium 30-70% and low under 30%. The consequences are also rated high, medium and low. High will cause big problems for the project and can be very critical. Medium is problems that can be handled, but can cause extra work. Low will only have small consequences for the project.

The strategy and actions part is divided into these strategies: avoid, transfer, reduce and accept. Avoid means that the team members will try to avoid the risk for happening. Transfer means that the work affected by the risk will be transfered to another team member. Reduce means that the team will reduce the consequences of the risk when it happens. If nothing can be done to avoid or fix the problem, actions must be taken accordingly.

The risk analysis is illustrated in a risk matrix in Figure A.8. The upper right part of the matrix is the most serious risks, and the lower left is the least serious risks.

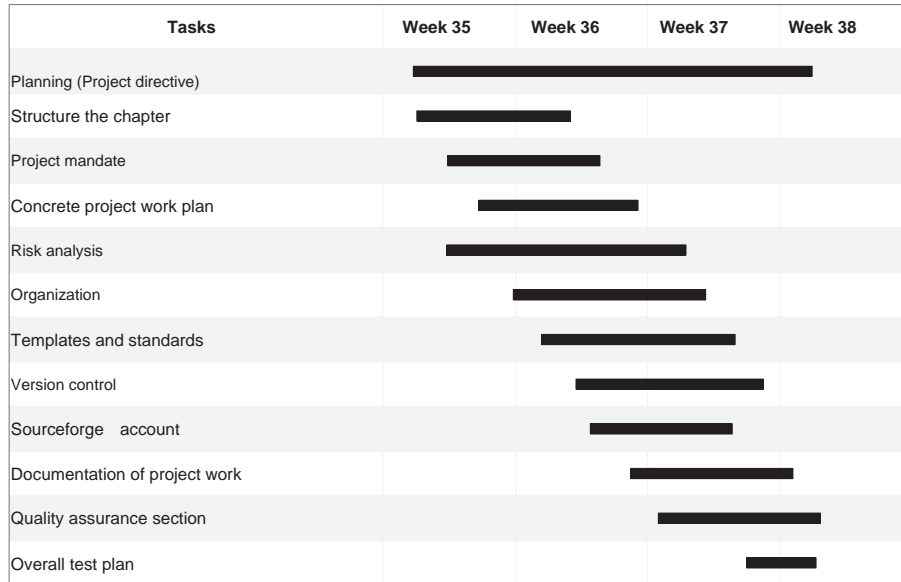


Figure A.4: Work plan for the Project directive phase in form of a Gantt diagram.

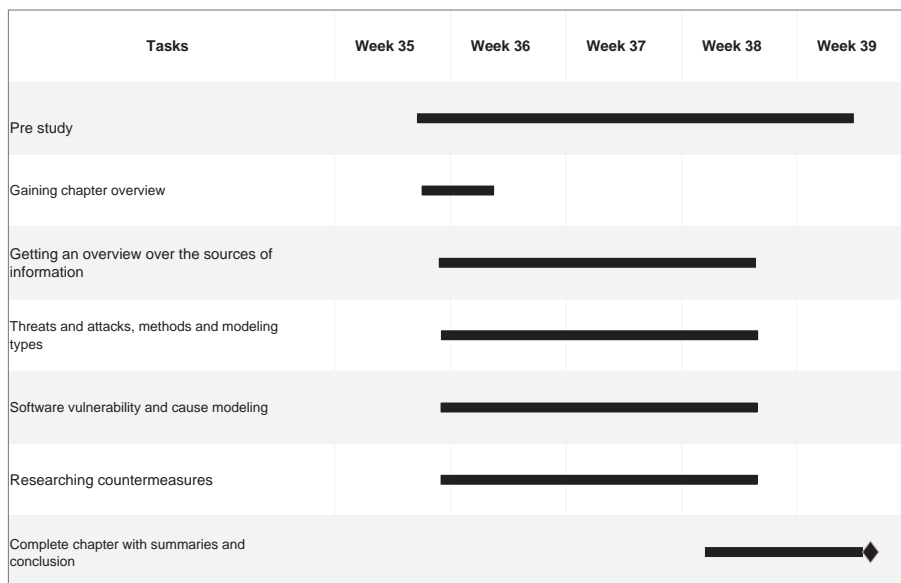


Figure A.5: Work plan for the Preliminary study phase in form of a Gantt diagram.

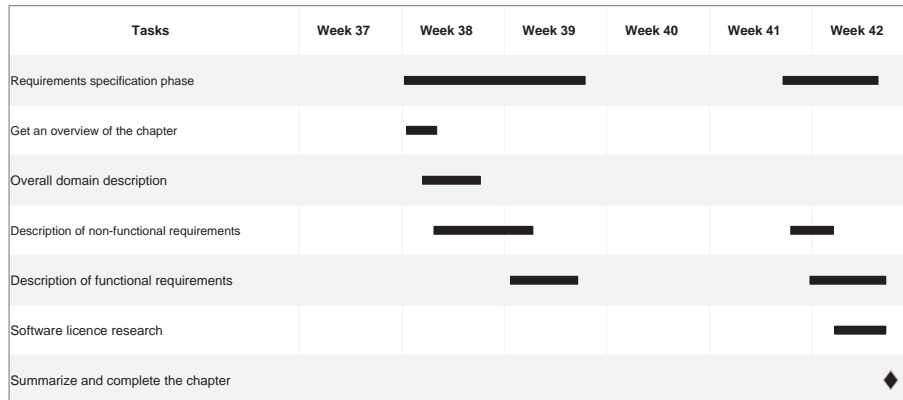


Figure A.6: Work plan for the Requirements specification phase in form of a Gantt diagram.

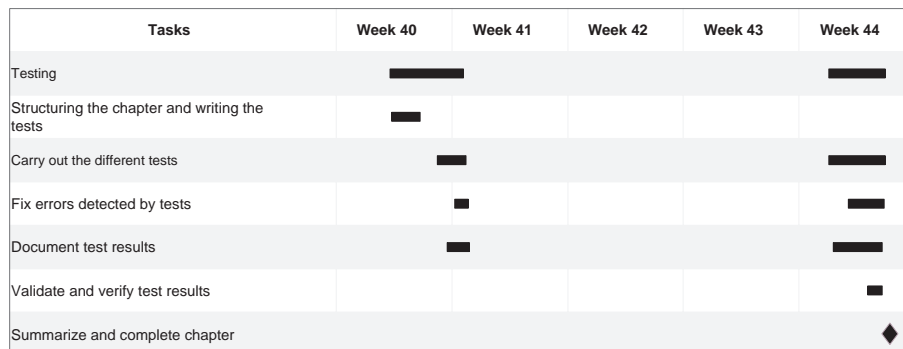


Figure A.7: Work plan for the Testing phase in form of a Gantt diagram.

NR	Activity	Risk factor	Consequence		Prob.	Strategy and actions	Deadline	Responsible
1	All	Illness among team members	M	Planned work may not be done in time	H	Transfer Other team members works more	Cont.	Egil
2	All	Illness or death in relatives or friends.	M	All the planned work may not be done in time	L	Transfer More work for the rest of the team.	Cont.	Egil
3	All	Technical problems, computer crash etc.	H	May lose big amounts of work and time.	M	Avoid Good backup routines, by using <code>svn</code> .	Cont.	Egil
4	All	Other courses consume a lot of time.	M	Less time for the project, and possibility of exceeding deadlines.	H	Reduce and Avoid Plan the usage of time better, and do not wait until the last deadline of delivering exercises.	Cont.	Egil
5	All	Documents are not managed properly.	M	May affect the quality of the work and the report.	H	Reduce and Avoid Good updating routines on the document, deciding concrete rules of writing and document standards.	Cont.	Kris-Mikael
6	Impl.	Bad design decisions.	H	System not corresponding to the specifications, or difficulties in implementing.	M	Avoid and Reduce Using standards i.e. UML, precise and formal requirements specification.	October	Eirik,Daniele

Table A.4: (continued)

NR	Activity	Risk factor	Consequence		Prob.	Strategy and actions	Deadline	Responsible
7	All	Ketil is working at UKA.	L	Less time for the project.	H	Reduce and Avoid Ketil works more before and after UKA, avoid giving Ketil extensive tasks in this period.	October	Ketil
8	All	Team members away for work/study/vacation over a long period.	H	The quality of the project may be affected.	H	Reduce and Accept Mitigate, working individually wherever possible. Help current team members with the tasks.	Cont.	Egil
9	All	Conflicts in the team.	M	Harder to work, lower quality on the work done.	M	Avoid, Reduce, Transfer Mitigate, having social events getting to know each other better, deal with and solve the problems together as a team.	Cont.	Egil
10	All	Team members quit the project.	H	Much more to do for the rest of the team.	L	Avoid, Transfer and Accept Shorten work delegation to on one person.	Cont.	Egil

Table A.4: (continued)

NR	Activity	Risk factor	Consequence		Prob.	Strategy and actions	Deadline	Responsible
11	All	Lack of communication.	M	Work done several times, done wrong, or not done at all.	M	Avoid and Reduce Force internal meetings more often.	Cont.	Egil
12	All	Customer contact at SINTEF quits his job.	M	Hard to get answers to questions regarding the project.	L	Accept There are two customer contacts at SINTEF, reducing the probability of impact.	Cont.	SINTEF
13	All	Bad choice of responsibility distribution.	M	Not be able to finish the planned deliverables.	H	Accept, reduce and avoid Redistribute the responsibilities if possible.	Cont.	Egil
14	All	Lack of necessary knowledge or experience.	M	Poor quality of the produced delivery.	H	Accept, reduce and avoid Try to learn and work extra with what is needed.	Cont.	Egil

Table A.4: Risk analysis

A.4 Documents and standards

A.4.1 Notice of customer meeting

Date	DD/MM-YYYY
Time	Start time - End time
Place	Location
Chairman	Egil Baadshaug

Sent to the following persons

SINTEF	Jostein Jensen Jostein.Jensen-at-sintef.no and Per H. Meland Per.H.Meland-at-sintef.no
Team	kpro8-at-idi.ntnu.no (all representatives from the team)

Agenda

- Approval of agenda
- Comments to the minutes of the last customer meeting
- Misc.

Notes

...

Appendix

...

A.4.2 Minute of customer meeting

Date	DD/MM-YYYY
Time	Start time - End time
Place	Location
Chairman	Egil Baadshaug

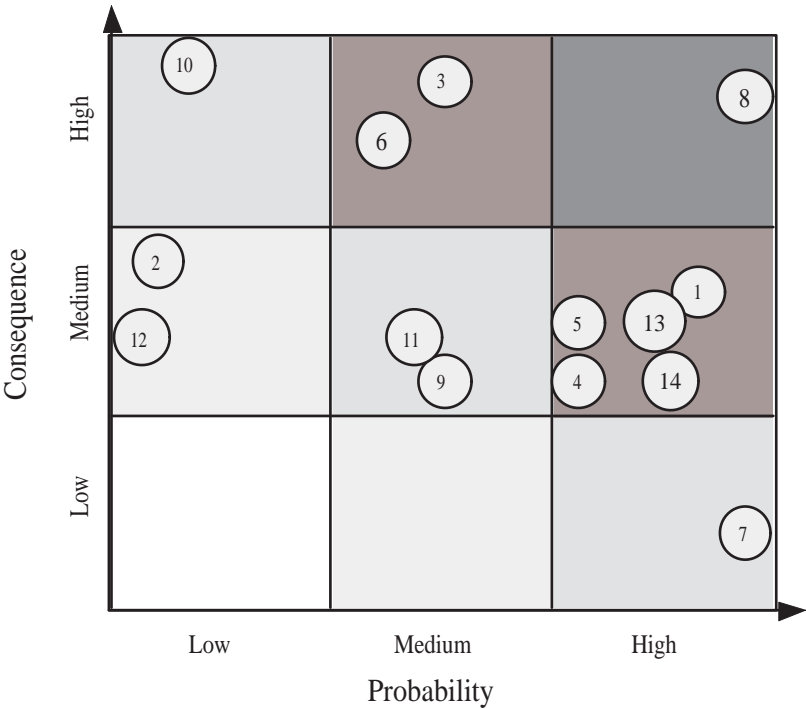


Figure A.8: Risk matrix.

Sent to the following persons

SINTEF

Jostein Jensen Jostein.Jensen-at-sintef.no and Per H. Meland Per.H.Meland-at-sintef.no

Team

kpro8-at-idi.ntnu.no (all representatives from the team)

People present

From SINTEF

...

From the team

...

Not present

From SINTEF

...

From the team

...

Agenda

• Approval of agenda

- Comments to the minutes of the last customer meeting approved by SINTEF the DD/MM-YYYY
- ⋮
- Misc.

A.4.3 Notice of supervisor meeting

Date DD/MM-YYYY
 Time Start time - End time
 Place Location
 Chairman Egil Baadshaug

Sent to the following persons

NTNU - IDI Renate Kristiansen renatek-at-idi.ntnu.no and Basit Ahmed Khan basit-at-idi.ntnu.no
 Team kpro8-at-idi.ntnu.no (all representatives from the team)

Agenda

- Approval of agenda
- ⋮
- Misc.

Notes

...

Appendix

- Time usage
- Status report
- Minutes of the last supervisor meeting
- Minutes of the last customer meeting
- ⋮

A.4.4 Minute of supervisor meeting

Date	DD/MM-YYYY
Time	Start time - End time
Place	Location
Chairman	Egil Baadshaug

Sent to the following persons

NTNU - IDI	Renate Kristiansen renatek-at-idi.ntnu.no and Basit Ahmed Khan basit-at-idi.ntnu.no
Team	kpro8-at-idi.ntnu.no (all representatives from the team)

People present
From the team ...

Not present
From the team ...

Agenda

- Approval of agenda
- Misc.

A.4.5 Status report from SeaMonster

Date	DD/MM-YYYY
Week	XY
Author	Ketil Sandanger Velle

Summary
...

Work done	
TRECQ	
<ul style="list-style-type: none"> • Time: ... • Risk: ... • Extent: ... • Cost: ... • Quality: ... 	
Documents	...
Meetings	
Customer meeting	DD/MM-YYYY
Internal meeting	DD/MM-YYYY
Supervisor meeting	DD/MM-YYYY
Activities	...
Problems	...
Planned work for next period	...

A.4.6 Tree structure of files

The file structure tree for the report is listed in Listing A.1.

```

|-- 0_bibliography
|   |-- bib.bib
|   '-- bib.bib.bak
|-- 0_images
|   |-- 0_seamonster_modeling.eps
|   |-- 1_A_RiskMatrix.eps
|   |-- 1_A_decitionFlowchart.eps
|   '-- 1_A_gantt-planning.eps

```

```

|-- 1_A_gantt-preStudy.eps
|-- 1_A_gantt-requirements-specification.eps
|-- 1_A_gantt-testing.eps
|-- 1_A_gantt.eps
|-- 1_A_gantt2.eps
|-- 1_A_ganttlegend.eps
|-- 1_orgchart.eps
|-- 1_unified_process_lifecycle.eps
|-- 2-pre-studyattacktreeplus.eps
|-- 2-pre-studysecuritytree.eps
|-- 2-prestudy-Microsoft1.eps
|-- 2-prestudy-Microsoft2.eps
|-- 2-prestudy-coras.eps
|-- 2-prestudy-coras.pdf
|-- 2_prestudy-at-1.eps
|-- 2_prestudy-at-2.eps
|-- 2_prestudy-at-3.eps
|-- 2_prestudy-at-4.eps
|-- 2_prestudy-at-5.eps
|-- 2_prestudy-at-6.eps
|-- 2_prestudy-cve1.eps
|-- 2_prestudy-cve2.eps
|-- 2_prestudy-cwe1.eps
|-- 2_prestudy-exg.eps
|-- 2_prestudy-sag-1.eps
|-- 2_prestudy-sag-2.eps
|-- 2_prestudy-sag-3.eps
|-- 2_prestudy-securitypatterns-1.eps
|-- 2_prestudy-securitypatterns-2.eps
|-- 2_prestudy-ucd-1.eps
|-- 2_prestudy-ucd-2.eps
|-- 2_prestudy-ucd-3.eps
|-- 2_prestudy-ucd-4.eps
|-- 2_prestudy-umlsec-1.eps
|-- 2_prestudy-vcg-example.eps
|-- 2_prestudy-vcg-notation.eps
|-- 2_prestudy-vcgsag.eps
|-- 2_prestudy_xccdf-oval.eps
|-- 3_Sequence_Diagram_sd_switching.eps
|-- 3_Sequence_Diagram_se_model_creation.eps
|-- 3_common_use_case.eps
|-- 3_consult_cwe_cve_dbs.eps
|-- 3_copy_and_paste_between_models.eps
|-- 3_create_a_new_model.eps
|-- 3_model_a_security_entity.eps
|-- 3_open_a_model.eps
|-- 3_print_a_model.eps
|-- 3_product_perspective.eps
|-- 3_relations.eps
|-- 3_reqspec_misusecase1.eps
|-- 3_save_a_model.eps
|-- 3_scenario1.eps
|-- 3_scenario2.eps
|-- 3_scenario3.eps

```

```

|-- 3_scenario4.eps
|-- 3_scenario5.eps
|-- 3_sd_use_case.eps
|-- 3_se_use_case.eps
|-- 3_traceability_matrix.eps
|-- 3_user_interface.eps
|-- 3_view_an_existing_model.eps
|-- 3_view_modify_an_existing_model.eps
|-- 3_view_modify_model.eps
|-- 4_at_notation.eps
|-- 4_data_model_attack_tree.eps
|-- 4_data_model_misuse_case.eps
|-- 4_data_model_toplevel.eps
|-- 4_data_model_vcg.eps
|-- 4_deployment_diagram.eps
|-- 4_design-at-1.eps
|-- 4_gmf_process.eps
|-- 4_gmf_structure.eps
|-- 4_mc_notation.eps
|-- 4_vcg_notation.eps
|-- 6_testmethod.eps
|-- 6_v-model.eps
|-- 8_cause-effect-delays-phase-deliveries.eps
|-- 8_piechart-estimated-work-distribution.eps
|-- 0_other-tex-files
|-- 0_glossarylist.tex
|-- 0_preface.aux
|-- 0_preface.tex
|-- 0_project-introduction.tex
|-- 10_appendix
|-- 10_A_project-directive.aux
|-- 10_pre-study.tex
|-- 10_project-directive.tex
|-- 10_specific-requirements.tex
|-- 10_testing.tex
|-- 10_conclusion
|-- 10_conclusion.tex
|-- 1_planning
|-- 1_1_introduction.tex
|-- 1_2_project-mandate.tex
|-- 1_3_concrete-project-work-plan.tex
|-- 1_4_organization.tex
|-- 1_5_templates-and-standards.tex
|-- 1_6_version-control.tex
|-- 1_7_documentation-of-project-work.tex
|-- 1_8_quality_assurance.tex
|-- 1_9_overall-test-plan.tex
|-- 2_prestudy
|-- 2_10_conclusion.tex
|-- 2_1_introduction.tex
|-- 2_2_business-requirements.tex
|-- 2_3_current-solution.tex
|-- 2_3_nvdgrowth.dat
|-- 2_3_nvdgrowth.p

```

```

|-- 2_3_nvdgrowth.tex
|-- 2_4_state-of-the-art.tex
|-- 2_4_x_countermeasures.tex
|-- 2_4_x_threats-and-attacks.tex
|-- 2_4_x_vulnerability-and-cause-modeling.tex
|-- 2_5_desired-solution.tex
|-- 2_6_market-analysis.tex
|-- 2_7_alternative-solutions.tex
|-- 2_8_evaluation-criteria.tex
|-- 2_9_solution-evaluation.tex
'-- 2_prestudy.tex
-- 3_requirements_specification
|-- 3_1_introduction.aux
|-- 3_1_introduction.tex
|-- 3_2_overall-description.aux
|-- 3_2_overall-description.tex
|-- 3_3_functional-requirements.tex
|-- 3_3_specific-requirements.aux
|-- 3_4_non-functional-requirements.tex
|-- 3_5_conclusion.tex
|-- 3_5_summary.tex
|-- 3_requirements_specification.aux
'-- 3_requirements_specification.tex
-- 4_design
|-- 4_1_introduction.tex
|-- 4_2_system_architectural_design.tex
|-- 4_3_detail_description_of_components.tex
|-- 4_4_user_interface_design.tex
|-- 4_construction.aux
'-- 4_construction.tex
-- 5_programming
|-- 5_programming.aux
'-- 5_programming.tex
-- 6_testing
|-- 6_1_introduction.tex
|-- 6_2_overall-testplan.tex
|-- 6_3_test-plan.tex
|-- 6_4_test-results.tex
|-- 6_5_tracking-of-tests.tex
|-- 6_6_conclusion.tex
|-- 6_testing.aux
'-- 6_testing.tex
-- 7_userinstallationguide
|-- 7_1_introduction.tex
|-- 7_userinstallationguide.aux
'-- 7_userinstallationguide.tex
-- 8_evaluation
|-- 8_evaluation.aux
'-- 8_evaluation.tex
-- 9_presentation
|-- 9_presentation.aux
'-- 9_presentation.tex
-- ReportCorrection
'-- report.pdf

```

```
|-- divfiles  
|-- report.aux  
|-- report.bbl  
|-- report.blg  
|-- report.dvi  
|-- report.glg  
|-- report.glo  
|-- report.gls  
|-- report.ist  
|-- report.lof  
|-- report.log  
|-- report.lot  
|-- report.out  
|-- report.pdf  
|-- report.ps  
|-- report.tex  
|-- report.toc  
|-- svnpropset.sh
```

Listing A.1: Report structure including files.

A.5 Common ground rules

Communication

- All team mail should be read by all members unless stated otherwise. SVN emails can be skipped.
- English is the primary language for all communication within the team.
- The project leader has to make sure everyone has a clear and equal understanding of all relevant issues at hand.
- All internal meetings should be announced at least 20 hours before the meeting.
- If the meeting is canceled, all the members of the team should be notified as soon as possible.
- Every misunderstood and conflicts should be discussed within the team.
- If someone will be absent for a significant period of time (more than two days) they have to inform the team as soon as they know and latest one week.

Decision making

See Figure A.9 for a flow chart to the decision making.

- The project manager has the final word in discussions.
- In non-trivial decisions, five out of seven have to agree. If less than five are present, all have to agree.

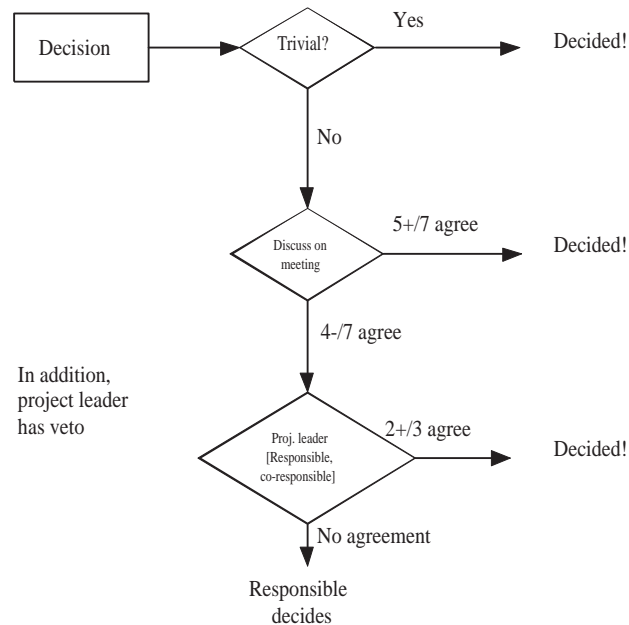


Figure A.9: Flow chart of decision making within the team.

- Voting could be used in difficult situation.
- The supervisors can be used for help in some case.

Process Rules

- Everyone has to meet for customer and supervisor meetings every Tuesday from 14:00 – 16:00 (CEST). If unable to attend the meeting, ensure the team get notified before the meeting.
- Everyone not being able to meet for internal meetings, must announce this to the project leader at least four hours before the meeting (including reason of absence).
- Everyone must fulfill their tasks according to the to-do-list. Fulfillment of these will be checked on an internal meeting Monday morning at 08:00 (CEST).
- All non-trivial tasks should have more than one person responsible for it.
- Everyone must keep track of working hours.
- Evaluation of progress and everyone's work is to be made every Monday on the internal meeting
- Everyone has to comply to all standards that has been set by the team.

Conflict Handling

- If there is a conflict this should be solved as soon as possible at a meeting.
- If there is a conflict splitting the team in two, contact with the supervisors should be done.

Reactions to someone breaking ground rules

- If someone breaks the ground rules, the conflict handling guidelines are to be used.

Meeting behavior

- The changing of a Ground Rule has to be unanimously decided by the team.

Appendix B

Pre-study appendix

B.1 Screenshots from different programs

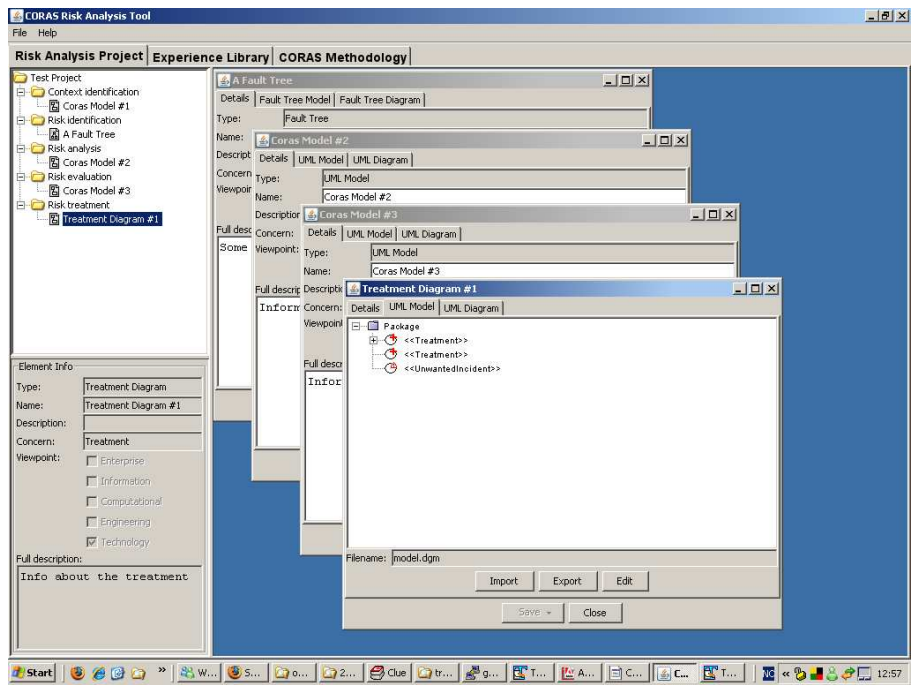


Figure B.1: Screenshot from the Coras tool.

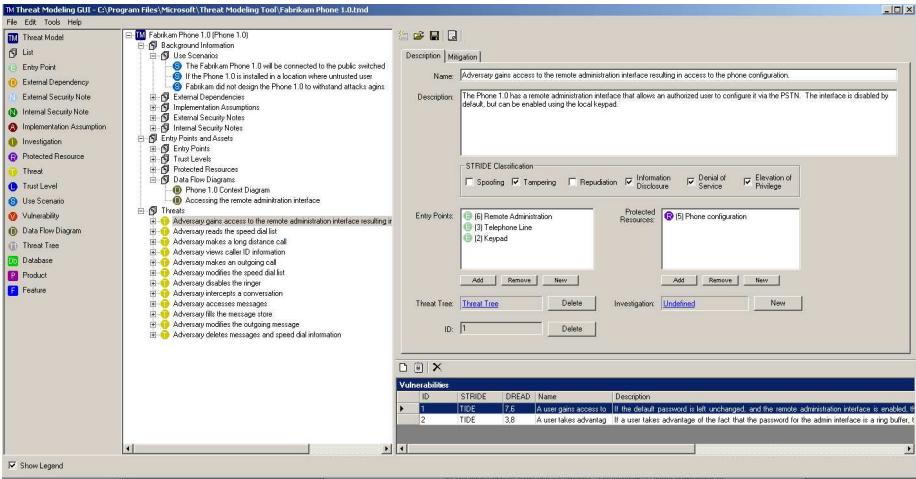


Figure B.2: Screenshot from the Microsoft threat modeling tool.

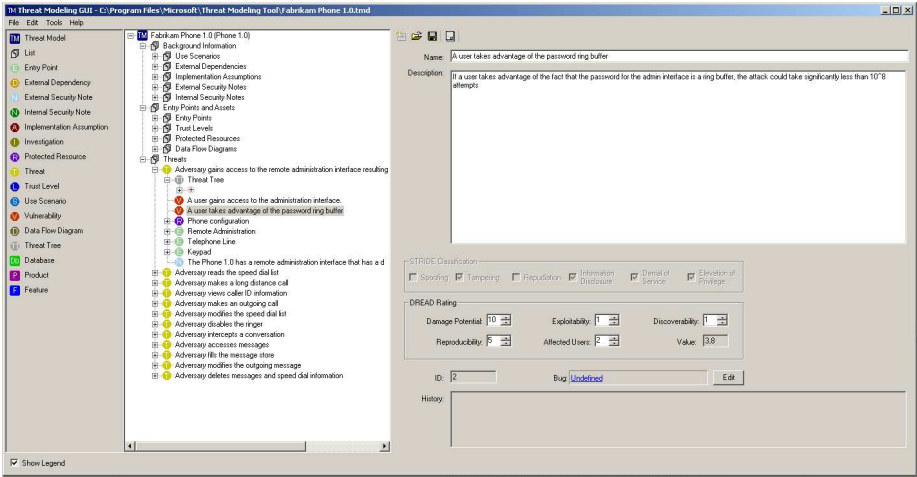


Figure B.3: Screenshot from the Microsoft threat modeling tool.

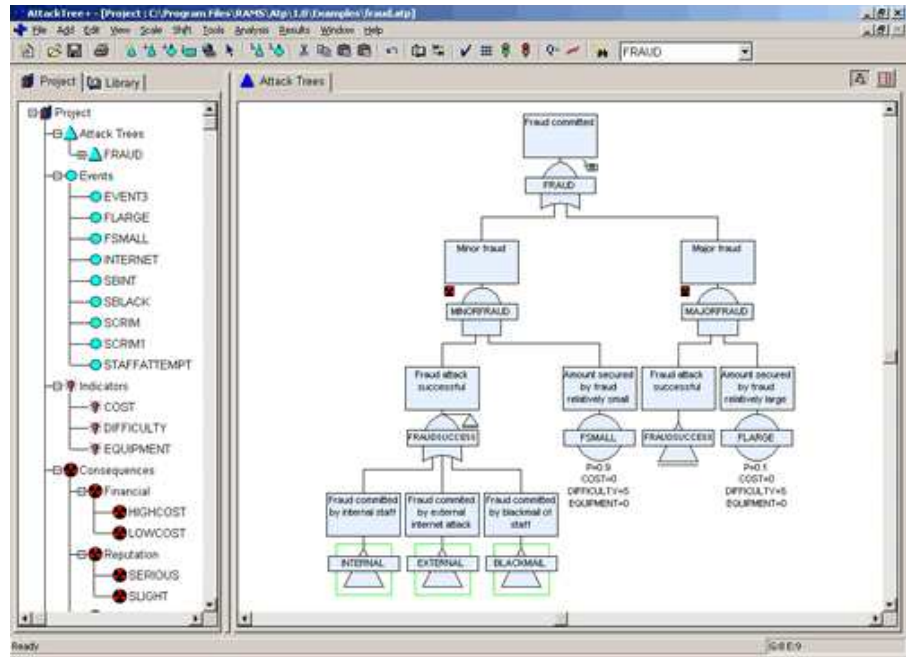


Figure B.4: Screenshot from Isograph's AttackTree+ fetched from their home-page [3].

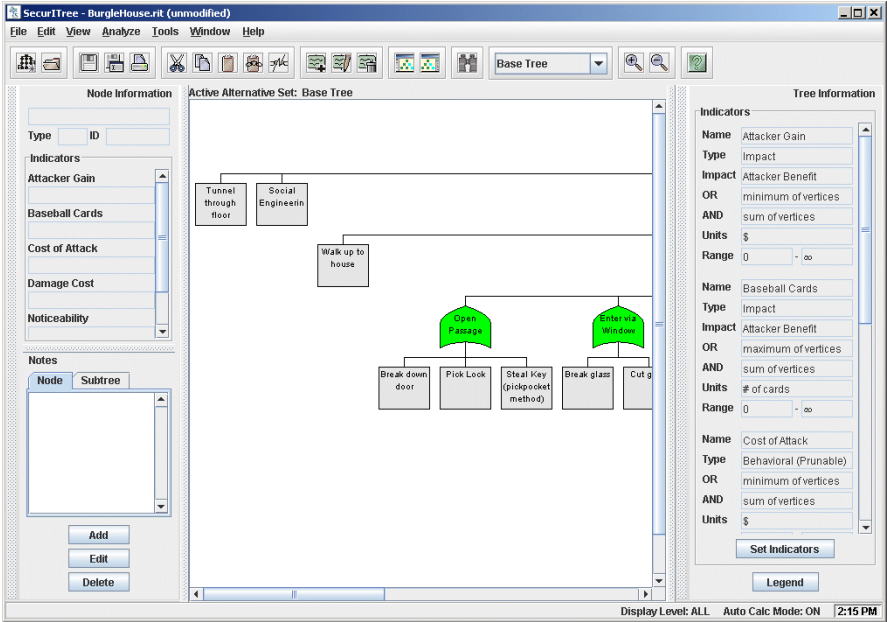


Figure B.5: Screenshot from Amenaza’s SecurITree fetched from their tool tutorial [50].

Appendix C

Specific requirements appendix

C.1 Functional Requirements

ID	Name	Description	Priority
EF.01	Import external definitions	The CWE/CVE databases has an export functionality. Content of these databases shall be possible to import into the tool for identification purposes.	L
EF.02	Link to external resources	The tool must provide functionalities for linking security model components to external resources.	M
EF.03	Import external information	The tool must provide functionalities for displaying information related to external resources.	M
IGF.01	Zooming functionalities	The tool must be able to zoom in and out over the modeling board.	M
IGF.02	Outline window	The tool must provide an outline window for better moving within the modelled diagram.	M

Table C.1: (continued)

ID	Name	Description	Priority
IGF.03	Automatic arranging components	When the diagram is composed of a considerable number of component managing an ordered diagram could become harder. The tool must provide the possibility of automatically arranging the components within the board.	M
IGF.04	Several diagrams open simultaneously	The tool must be able to take open more then one diagram contemporaneously.	H
IGF.05	Adding notes	The tool must leave the user free to add textual notes to the diagram.	M
ATF.01	Models of threats/attacks	The tool must be able to model threats/attacks with Attack Trees (See Section 3.5.2).	H
ATF.02	Automatic cost calculation	The tool must be able to automatically calculate the cost of an attack.	L
ATF.03	Marking resolved attack paths	The tool must allow the user to visibly mark resolved attack paths.	L
ATF.04	Attack Trees notation	The tool must implement the Attack Trees' notation displayed in Figure 5.15.	M
ATF.05	Attack Trees components' properties	Goals must have the following properties: cost, boolean, name and description.	M
ATF.06, VCGF.02	Preventing cycles	The tool must be able to automatically detect and forbid cycles in the diagram.	M
VCGF.01	Models of vulnerabilities	The tool must be able to model vulnerabilities with Vulnerability Cause Graphs (See Section 3.5.1).	H
VCGF.03	Vulnerability Cause Graphs notation	The tool must implement the Vulnerability Cause Graphs' notation displayed in Figure 5.13.	M
VCGF.04	Vulnerability Cause Graphs' properties	Every VCG component must have the following properties: name and description.	M

Table C.1: (continued)

ID	Name	Description	Priority
MCF.01	Models of counter-measures	The tool must be able to model countermeasures with Use/Misuse cases (See Section 3.5.3).	M
MCF.02	Use/Misuse Cases notation	The tool must implement the Vulnerability Cause Graphs' notation displayed in Figure 5.14.	M
MCF.03	Use/Misuse Cases' properties	Every Use/Misuse Cases component must have the following properties: name and description.	M
MF.01	Specialisation and generalisation	It must be possible to specialise and generalise objects allowing the creation of objects containing several sub-objects. In particular, using use/misuse cases diagrams the user must be allowed to add cases within the system area, while using VCG diagrams he must be allowed to create conjunction nodes containing simple and compound nodes.	M
MF.02	Pluralism of the diagrams	The tool must allow the user to associate more than one diagram to a security model.	M
MF.03	Visible properties	The tool must visibly manage components' properties, if any.	M
SF.01	UML 2.1 meta model	Data should be stored accordingly to the UML 2.1 meta model.	L
SF.02	Textual representation	The graphical security models drawn in the tool must be converted to an XML textual representation, describing the model completely, stored in a database (or similar functionality).	M
SF.03	Save and open	Models drawn in the graphical editor must be possible to save, and open at a later stage.	H

Table C.1: (continued)

ID	Name	Description	Priority
SF.04	Provident creation	When creating new diagrams, if some relevant information about the model are stored yet, the tool must allow the user to start the draw from them.	M
SF.05	Exporting as image	The tool must allow the user to export diagrams as image. In particular, it must provide the possibility to save the image both in bitmap and in vectorial format.	M
SF.06	Printing with preview	The tool must allow the user to print diagrams giving him/her the possibility to preview the job.	M
SECF.01	Validate XML input	The application must validate the XML input referring to a predefined XML schema.	M

Table C.1: Functional requirements.

C.2 Non-Functional Requirements

ID	Name	Description	Priority
INF.01	Eclipse	The tool should be based on the Eclipse framework.	H
INF.02	GMF	The Eclipse Graphical modeling Framework (GMF) should be used to develop the graphical editor.	H
INF.03	EMF	The Eclipse modeling Framework (EMF) should be used as the modeling framework.	M
INF.04	Java	Model logic should be programmed in Java.	H
UNF.01	Intuitive GUI	The GUI should be based on the Eclipse pattern allowing users to find a well-know environment.	H

Table C.2: (continued)

ID	Name	Description	Priority
UNF.02	Screen layout	The tool should present an intuitive and well structured GUI, with a palette, model selection pane and drawing board.	H
PNF.01	Platform independent	The tool must be independent of specific operating systems.	H
PKNF.01	System resources	The application must not require more than 50 MB of RAM, and 100 MB of hard drive space.	M
LNF.01	Open source	The project will be considered an open source project and available to a wider community for further development and suggestions. Use of Sourceforge as the project repository.	H

Table C.2: Non-Functional requirements.

C.3 Documentation Requirements

ID	Name	Description	Priority
D.01	Report	The project must provide a report containing scientific research and development of the application.	H
D.02	Language	The preferred documentation language is English.	M
D.03	State-of-the-art	Describe and compare the state-of-the-art in security modeling (both theory and existing tools).	H
D.04	Future work	Future work have to be well documented. What remains to be done according to requirements, and what should be done to improve the product?	H

Table C.3: (continued)

ID	Name	Description	Priority
D.05	Problems	Describe which problems were discovered during the project (implementation specific).	H
D.06	Documentation of graphical representation/notation	A comprehensive documentation of the graphical tool and the notation used.	H
D.07	API documentation	The API of the program logic shall be well documented.	H
D.08	Documentation of textual representation	Provide a structured view, e.g. by XML schema	M
D.09	Documentation of storage structures	Document the use of UML 2.1 meta model (if used), alternatively ER-diagrams if databases are used.	M
D.10	Design specification	Present a structured overview of the tool design, such as UML component diagrams, class diagrams and sequence diagrams with related textual descriptions.	M
D.11	Implementation specification	Describe the implementation specific solutions that were chosen.	M
D.12	Source code	Provide documented source code (in report appendix and on CD).	H
D.13	Requirement specification	Provide a list of final requirements.	H
D.14	Requirement fulfillment	Describe which and how the requirements have been fulfilled. This means that not fulfilled requirements must also be stated and why.	H
D.15	User manual	A user manual written in English should be provided.	M
D.16	Draft for a scientific paper	During the process you should prepare a draft for a scientific paper. SINTEF will be able to sponsor one or two students to present the paper on a suited conference.	L

Table C.3: Documentation requirements.

C.4 Process Requirements

ID	Name	Description	Priority
P.01	Milestones	The team shall prepare a detailed work plan with defined milestones.	H
P.02	Project coordinator	One person shall be selected to be project coordinator with responsibility to lead the team and report to SINTEF.	H
P.03	Code repository	A code repository with version tracking should be established and used throughout the project.	M
P.04	Bugtracking	The team must provide a method for the supervisors to report bugs and comments to the tool (e.g. Bugzilla or Traq).	M
P.05	Changes	The team must be open to minor changes and repriorisation of requirements during the project phase.	M
P.06	Customer meetings	Customer meeting shall be held once a week. A written of status and deviations from the original project plan is required.	H
P.07	Agile development	The development process shall be iterative, meaning that requirements can be added/alterd after a development iteration.	M
P.08	Risk analysis	Prepare a rough risk analysis describing task that may stagger the work process and define measures that will mitigate the risks. This should be revised throughout the project.	M

Table C.4: Process requirements.

Appendix D

Design appendix

D.1 Detailed class diagram

A more detailed class diagram for the MisuseCase part of the data model is shown in Figure [D.1](#).

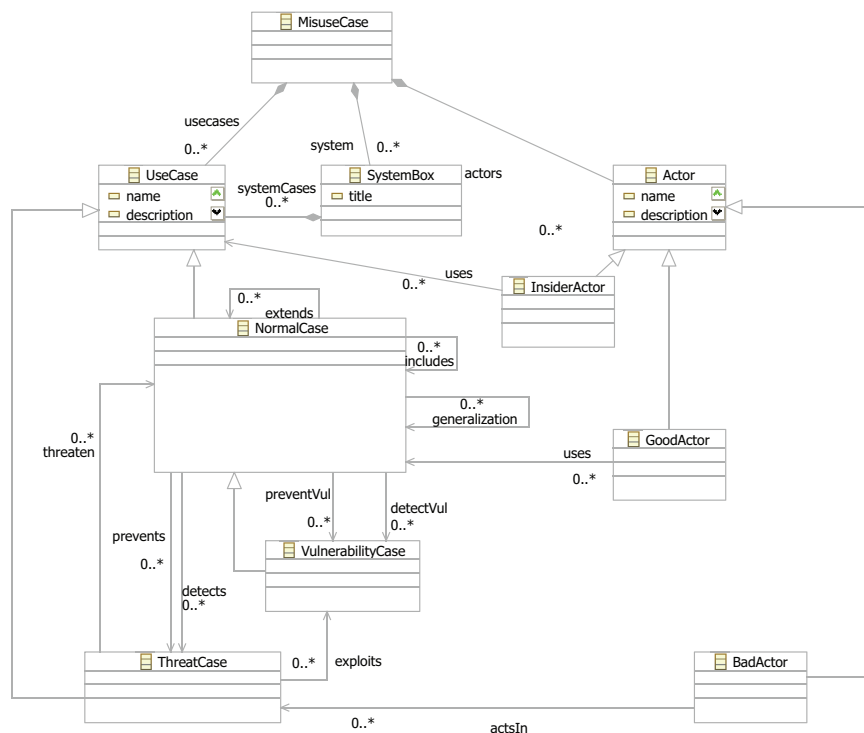


Figure D.1: Detailed class diagram for the MisuseCase part of the applications data model.

D.2 SeaMonster XMI schema

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema
  xmlns:seaMonster="http://www.example.org/seaMonster"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.org/seaMonster">
  <xsd:import namespace="http://www.omg.org/XMI"
    schemaLocation=
      "../..../plugin/org.eclipse.emf.ecore/model/XMI.xsd"/>
  <xsd:complexType name="SecurityModel">
  <xsd:choice maxOccurs="unbounded" minOccurs="0">
    <xsd:element name="contains"
      type="seaMonster:SubDiagram"/>
    <xsd:element ref="xmi:Extension"/>
  </xsd:choice>
  <xsd:attribute ref="xmi:id"/>
  <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
</xsd:complexType>
<xsd:element name="SecurityModel"
  type="seaMonster:SecurityModel"/>
<xsd:complexType name="AttackTree">
<xsd:complexContent>
  <xsd:extension base="seaMonster:SubDiagram">
  <xsd:choice maxOccurs="unbounded"
    minOccurs="0">
    <xsd:element name="contains"
      type="seaMonster:AttackTreeComponent"/>
  </xsd:choice>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:element name="AttackTree"
  type="seaMonster:AttackTree"/>
<xsd:complexType name="VCG">
<xsd:complexContent>
  <xsd:extension base="seaMonster:SubDiagram">
  <xsd:choice maxOccurs="unbounded"
    minOccurs="0">
    <xsd:element name="vulnerability"
      type="seaMonster:ExitNode"/>
    <xsd:element name="causeList"
      type="seaMonster:CauseNode"/>
  </xsd:choice>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:element name="VCG" type="seaMonster:VCG"/>
<xsd:complexType name="MisuseCase">
<xsd:complexContent>
  <xsd:extension base="seaMonster:SubDiagram">
  <xsd:choice maxOccurs="unbounded"

```

```

minOccurs="0">
  <xsd:element name="actors"
    type="seaMonster:Actor"/>
  <xsd:element name="usecases"
    type="seaMonster:UseCase"/>
  <xsd:element name="system"
    type="seaMonster:SystemBox"/>
</xsd:choice>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:element name="MisuseCase"
  type="seaMonster:MisuseCase"/>
<xsd:complexType abstract="true"
  name="AttackTreeComponent">
  <xsd:choice maxOccurs="unbounded" minOccurs="0">
    <xsd:element ref="xmi:Extension"/>
  </xsd:choice>
  <xsd:attribute ref="xmi:id"/>
  <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="cost" type="xsd:int"/>
</xsd:complexType>
<xsd:element name="AttackTreeComponent"
  type="seaMonster:AttackTreeComponent"/>
<xsd:complexType name="AndNode">
  <xsd:complexContent>
    <xsd:extension
      base="seaMonster:AttackTreeComponent">
      <xsd:choice maxOccurs="unbounded"
        minOccurs="0">
        <xsd:element name="predecessor"
          type="seaMonster:GoalNode"/>
      </xsd:choice>
      <xsd:attribute name="predecessor"
        type="xsd:string"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="AndNode"
  type="seaMonster:AndNode"/>
<xsd:complexType name="GoalNode">
  <xsd:complexContent>
    <xsd:extension
      base="seaMonster:AttackTreeComponent">
      <xsd:choice maxOccurs="unbounded"
        minOccurs="0">
        <xsd:element name="predecessor"
          type="seaMonster:AttackTreeComponent"/>
      </xsd:choice>
      <xsd:attribute name="description"
        type="xsd:string"/>
      <xsd:attribute name="predecessor"
        type="xsd:string"/>
    </xsd:extension>
  </xsd:complexContent>

```



```

    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="GoalNode"
  type="seaMonster:GoalNode" />
<xsd:complexType abstract="true" name="Node">
  <xsd:choice maxOccurs="unbounded" minOccurs="0">
    <xsd:element ref="xmi:Extension" />
  </xsd:choice>
  <xsd:attribute ref="xmi:id" />
  <xsd:attributeGroup ref="xmi:ObjectAttribs" />
  <xsd:attribute name="name" type="xsd:string" />
  <xsd:attribute name="description"
    type="xsd:string" />
</xsd:complexType>
<xsd:element name="Node" type="seaMonster:Node" />
<xsd:complexType name="ExitNode">
  <xsd:complexContent>
    <xsd:extension base="seaMonster:Node">
      <xsd:attribute name="CVLink"
        type="xsd:string" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="ExitNode"
  type="seaMonster:ExitNode" />
<xsd:complexType name="CauseNode">
  <xsd:complexContent>
    <xsd:extension base="seaMonster:Node">
      <xsd:choice maxOccurs="unbounded"
        minOccurs="0">
        <xsd:element name="successor"
          type="seaMonster:Node" />
      </xsd:choice>
      <xsd:attribute name="successor"
        type="xsd:string" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="CauseNode"
  type="seaMonster:CauseNode" />
<xsd:complexType name="CompoundNode">
  <xsd:complexContent>
    <xsd:extension base="seaMonster:CauseNode" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="CompoundNode"
  type="seaMonster:CompoundNode" />
<xsd:complexType name="SimpleNode">
  <xsd:complexContent>
    <xsd:extension base="seaMonster:CauseNode" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="SimpleNode"

```

```

type="seaMonster:SimpleNode"/>
<xsd:complexType name="ConjunctionNode">
<xsd:complexContent>
  <xsd:extension base="seaMonster:CauseNode">
    <xsd:choice maxOccurs="unbounded"
      minOccurs="0">
      <xsd:element name="conjunctionCauses"
        type="seaMonster:CauseNode"/>
    </xsd:choice>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:element name="ConjunctionNode"
  type="seaMonster:ConjunctionNode"/>
<xsd:complexType name="GoodActor">
<xsd:complexContent>
  <xsd:extension base="seaMonster:Actor">
    <xsd:choice maxOccurs="unbounded"
      minOccurs="0">
      <xsd:element name="uses"
        type="seaMonster:NormalCase"/>
    </xsd:choice>
    <xsd:attribute name="uses" type="xsd:string"/>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:element name="GoodActor"
  type="seaMonster:GoodActor"/>
<xsd:complexType name="BadActor">
<xsd:complexContent>
  <xsd:extension base="seaMonster:Actor">
    <xsd:choice maxOccurs="unbounded"
      minOccurs="0">
      <xsd:element name="actsIn"
        type="seaMonster:ThreatCase"/>
    </xsd:choice>
    <xsd:attribute name="actsIn"
      type="xsd:string"/>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:element name="BadActor"
  type="seaMonster:BadActor"/>
<xsd:complexType name="InsiderActor">
<xsd:complexContent>
  <xsd:extension base="seaMonster:Actor">
    <xsd:choice maxOccurs="unbounded"
      minOccurs="0">
      <xsd:element name="uses"
        type="seaMonster:UseCase"/>
    </xsd:choice>
    <xsd:attribute name="uses" type="xsd:string"/>
  </xsd:extension>
</xsd:complexContent>

```

```

</xsd:complexType>
<xsd:element name="InsiderActor"
type="seaMonster:InsiderActor" />
<xsd:complexType name="Actor">
<xsd:choice maxOccurs="unbounded" minOccurs="0">
  <xsd:element ref="xmi:Extension" />
</xsd:choice>
<xsd:attribute ref="xmi:id" />
<xsd:attributeGroup ref="xmi:ObjectAttribs" />
<xsd:attribute name="name" type="xsd:string" />
<xsd:attribute name="description"
type="xsd:string" />
</xsd:complexType>
<xsd:element name="Actor" type="seaMonster:Actor" />
<xsd:complexType name="UseCase">
<xsd:choice maxOccurs="unbounded" minOccurs="0">
  <xsd:element ref="xmi:Extension" />
</xsd:choice>
<xsd:attribute ref="xmi:id" />
<xsd:attributeGroup ref="xmi:ObjectAttribs" />
<xsd:attribute name="name" type="xsd:string" />
<xsd:attribute name="description"
type="xsd:string" />
</xsd:complexType>
<xsd:element name="UseCase"
type="seaMonster:UseCase" />
<xsd:complexType name="NormalCase">
<xsd:complexContent>
  <xsd:extension base="seaMonster:UseCase">
    <xsd:choice maxOccurs="unbounded"
minOccurs="0">
      <xsd:element name="detects"
type="seaMonster:ThreatCase" />
      <xsd:element name="extends"
type="seaMonster:NormalCase" />
      <xsd:element name="generalization"
type="seaMonster:NormalCase" />
      <xsd:element name="includes"
type="seaMonster:NormalCase" />
      <xsd:element name="detectVul"
type="seaMonster:VulnerabilityCase" />
      <xsd:element name="prevents"
type="seaMonster:ThreatCase" />
      <xsd:element name="preventVul"
type="seaMonster:VulnerabilityCase" />
    </xsd:choice>
    <xsd:attribute name="detects"
type="xsd:string" />
    <xsd:attribute name="extends"
type="xsd:string" />
    <xsd:attribute name="generalization"
type="xsd:string" />
    <xsd:attribute name="includes"
type="xsd:string" />

```

```

    <xsd:attribute name="detectVul"
      type="xsd:string"/>
    <xsd:attribute name="prevents"
      type="xsd:string"/>
    <xsd:attribute name="preventVul"
      type="xsd:string"/>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:element name="NormalCase"
  type="seaMonster:NormalCase"/>
<xsd:complexType name="ThreatCase">
<xsd:complexContent>
  <xsd:extension base="seaMonster:UseCase">
    <xsd:choice maxOccurs="unbounded"
      minOccurs="0">
      <xsd:element name="exploits"
        type="seaMonster:VulnerabilityCase"/>
      <xsd:element name="threaten"
        type="seaMonster:NormalCase"/>
    </xsd:choice>
    <xsd:attribute name="exploits"
      type="xsd:string"/>
    <xsd:attribute name="threaten"
      type="xsd:string"/>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:element name="ThreatCase"
  type="seaMonster:ThreatCase"/>
<xsd:complexType name="VulnerabilityCase">
<xsd:complexContent>
  <xsd:extension base="seaMonster:NormalCase"/>
</xsd:complexContent>
</xsd:complexType>
<xsd:element name="VulnerabilityCase"
  type="seaMonster:VulnerabilityCase"/>
<xsd:complexType name="SystemBox">
<xsd:choice maxOccurs="unbounded" minOccurs="0">
  <xsd:element name="systemCases"
    type="seaMonster:UseCase"/>
  <xsd:element ref="xmi:Extension"/>
</xsd:choice>
<xsd:attribute ref="xmi:id"/>
<xsd:attributeGroup ref="xmi:ObjectAttribs"/>
<xsd:attribute name="title" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="SystemBox"
  type="seaMonster:SystemBox"/>
<xsd:complexType abstract="true" name="SubDiagram">
<xsd:choice maxOccurs="unbounded" minOccurs="0">
  <xsd:element name="relation"
    type="seaMonster:SubDiagram"/>
  <xsd:element ref="xmi:Extension"/>

```

```
</xsd:choice>
<xsd:attribute ref="xmi:id"/>
<xsd:attributeGroup ref="xmi:ObjectAttribs"/>
<xsd:attribute name="title" type="xsd:string"/>
<xsd:attribute name="relation" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="SubDiagram"
  type="seaMonster:SubDiagram"/>
<xsd:complexType name="DocumentRoot">
<xsd:choice maxOccurs="unbounded" minOccurs="0">
  <xsd:element name="model"
    type="seaMonster:SecurityModel"/>
  <xsd:element ref="xmi:Extension"/>
</xsd:choice>
<xsd:attribute ref="xmi:id"/>
<xsd:attributeGroup ref="xmi:ObjectAttribs"/>
</xsd:complexType>
<xsd:element name="DocumentRoot"
  type="seaMonster:DocumentRoot"/>
</xsd:schema>
```

Listing D.1: SeaMonster XMI schema.

Appendix E

Testing appendix

E.1 Early system test

Table [E.1](#) contains a system test made for the early prototype, that was demonstrated to the customer. Almost all tests were approved, but with minor details missing. ST-1 is marked with an (PASSED), because the model components does not “show up”, only two buttons with the name “Goal” and “Subgoal”. But this will be changed in further development, and is not an important absence. ST-2 is not approved because of the absence of the AND notation. Overall, the testing for the early prototype are satisfactory.

Testing done by	<i>Eirik</i>	
Date	<i>2007-10-15</i>	
Responsible	Eilev	
Test name	Attack tree modeling	
Requirements tested	ATF.01, MF.01, SF.03	
Test case	Expected result	Result
ST-1		
Choose to model a new attack tree diagram.	Model components for the attack tree should show up	<i>(PASSED)</i>
ST-2		
Create the attack tree shown in the example in Figure 3.6.	All model components should be available. Components should be placed, and be connected together without any errors.	<i>FAILED</i>
ST-3		
Click on different component boxes and move them around. Select several boxes at once and move them around.	All connectors to the selected box should follow the movement.	<i>PASSED</i>
ST-4		
Drag one end of a connector from one component to another	Connector should connect to the new component without any errors.	<i>PASSED</i>
ST-5		
Delete one of each component.	The component should disappear.	<i>PASSED</i>
ST-6		
Save the model, then close the application, start the application again and open the saved model	The model should be saved at the specified location. The model should be identical before the saving and after opening it again.	<i>PASSED</i>

Table E.1: System test: Attack tree.