Organizational Patterns for Multidisciplinary Development of Mechatronic Systems

Master's thesis. Computer Science

April 14. 2015

Mark den Hollander - 838099431

ORGANIZATIONAL PATTERNS FOR MULTIDISCPLINARY DEVELOPMENT OF MECHATRONIC SYSTEMS

Open Universiteit Nederland, faculteit Informatica
Masteropleiding Computer Science of Software Engineering

GRADUATION COMMITTEE

Prof. dr. A. Bijlsma
Chairman / Secretary, Open Universiteit Nederland
Valkenburgerweg 177
6419 AT Heerlen
The Netherlands
e-mail: lex.bijlsma@ou.nl

Ir. P. Oord
Supervisor, Open Universiteit Nederland
Valkenburgerweg 177
6419 AT Heerlen
The Netherlands
e-mail: paul.van.oord@ou.nl

COURSE

T76318: Afstudeeropdracht Computer Science

AUTHOR / STUDENT

Ing. M. den Hollander
e-mail: mark.den.hollander@gmail.com

Acknowledgement

At the start of this research, I was familiar with the design patterns of the Gang of Four (Gamma, Helm, Johnson & Vlissides, 1994). By reading (Buschmann, Henney & Schmidt, 2007) I learned the criteria to which patterns should comply, and I learned about organizational patterns, sequences, and pattern languages. This book led me to the organizational patterns and pattern languages of agile software development (Coplien & Harrison, 2005). This became my inspiration in creating new patterns for the multidisciplinary development of mechatronic systems.

At the beginning of this research, I found many problems which I wanted to investigate. Thanks to the guidance of Paul Oord and Lex Bijlsma, I was able to formulate a research question that would have social and academic relevance, and also my strong personal interest. Throughout the research, they kept me from wandering away from my chosen topic. One of my pitfalls was to continuously seek in-depth knowledge on every subject that I encountered. Paul Oord and Lex Bijlsma also provided constructive feedback on the deliverables produced during the research. I would like to thank them for their support.

A fundamental part of my research is the field data that I received from an organization that develops and produces mechatronic systems. They have conscientiously gathered and archived data from their projects and teams for a long time. These data helped me greatly to create patterns strongly related to the real world. I would like to thank this organization for making such valuable data available.

The conclusion of this research could have been that it is not possible to create organizational patterns. I am pleased to report that **is** possible. I am even more pleased with the feedback received from the reviewers and authors of the book *Organizational patterns of agile software development* (Coplien & Harrison, 2005), who granted me their time and provided good feedback. Such feedback confirms the relevance of the patterns, and gives an indication of the quality of my work. I would like to thank them for closing the loop between academic research and the working environment in which problems have to be solved daily.

Studies at the Open University require an investment of time and money. I had the privilege of having employers (ICT Automatisering N.V. and PROMEXX B.V) who were willing to pay the costs of the modules and offered me days off when I had to write exams. I would like to thank them for making the study of Computer Science accessible.

I would like to thank all my friends, students, teachers, and colleagues for their feedback and inspiration.

Finally, I would like to thank my family for being the greatest supporters that I could ever wish for.

Abstract

This thesis describes three new organizational patterns for developing a mechatronic system concurrently with a multidisciplinary team. These patterns are mined from the literature and field data of an organization that produces mechatronic systems. The patterns described in this work are related to other patterns expressed in a pattern language.

The development of a mechatronic system requires an intense collaboration between disciplines. This way of working introduces dependencies between disciplines, which introduces problems. These problems can lead to system integration issues and project delays. This problem statement leads to the following research question:

> "Is it possible to formulate organizational patterns that can be used for the development and integration of a mechatronic system in a multidisciplinary environment developed concurrently?"

There are organizations that have encountered and addressed these problems. Their solution can be seen as best or common practice. This research mines these practices in a literature study that contains journal articles, proceedings, and books. Practices are also mined from field data made available by an organization that develops and produces mechatronic systems. These field data contain Failure Methods and Effects Analysis (FMEA) worksheets and retrospective reports. Only a selection of all these practices that were mined are detailed in a pattern description. A practice is selected when it specifically addresses the following subjects:

- Development of a mechatronic system
- Multidisciplinary development
- Concurrent engineering

The result of this research is three new patterns that are integrated into pattern languages. The patterns are:

- *Common Plan*
- *Hardware in the Loop*
- *Simulator in the Loop.*

Individuals who work in the field of mechatronic systems and multidisciplinary projects reviewed the patterns. The amount of problems from one organization that are covered by these patterns is also determined.

The conclusion of this research is that the research question is answered positively. This research has addressed problems that are encountered during the concurrent development of a mechatronic system by a multidisciplinary team. These problems are:

- Collaboration between disciplines
- Integration of deliverables
- Dependency between disciplines

These problems generally lead to project delays. The results of this research are new organizational patterns that provide practical solutions to counter these problems. It is the first time that these solutions are formalized and presented as an organizational pattern and integrated into the organizational pattern language of (Coplien & Harrison, 2005). This result can benefit a multidisciplinary team that develops a mechatronic system concurrently. It will make them aware of: a solution to their problem, when to apply it, the forces and trade-offs of the solution, and how the solution can be implemented. These patterns will empower a multidisciplinary team to solve the stated problems.

Samenvatting

Dit proefschrift beschrijft drie nieuwe organisatorische patronen voor de ontwikkeling van een mechatronisch systeem door een multidisciplinair team. Deze patronen zijn gevonden in de literatuur en in veldgegevens van een organisatie die mechatronische systemen produceert. De in dit werk beschreven patronen zijn gerelateerd aan andere patronen die onderdeel zijn van een patroontaal.

De ontwikkeling van een mechatronisch systeem vereist een intensieve samenwerking tussen disciplines. Deze manier van werken creëert afhankelijkheden tussen disciplines en dat introduceert problemen. Deze problemen kunnen leiden tot systeem integratie problemen en vertragingen op het project. Deze probleemstelling leidt tot de volgende onderzoeksvraag:

> "Kan een organisatorisch pattern opgesteld en toegepast worden voor concurrent engineering in een multidisciplinaire omgeving voor het ontwikkelen en integreren van een mechatronisch systeem?"

Er zijn organisaties die deze problemen hebben ondervonden en deze hebben aangepakt. Hun oplossing kan worden gezien als een gangbare praktijk. Dit onderzoek zoekt deze praktijken in een literatuurstudie. Praktijken worden ook gezocht in veldgegevens die beschikbaar werden gemaakt door een organisatie die mechatronische systemen ontwikkelt en produceert. Deze veldgegevens bevatten Failure Methods and Effects Analysis (FMEA) werkbladen en retrospectieve rapporten. Slechts een selectie van al de gevonden gangbare praktijken zijn uiteindelijk gedetailleerd beschreven in een patroon. Een praktijk is geselecteerd wanneer het specifiek ingaat op de volgende onderwerpen:

- Ontwikkeling van een mechatronisch systeem
- Multidisciplinaire ontwikkeling
- Parallel ontwikkelen (concurrent engineering)

Het resultaat van dit onderzoek zijn drie nieuwe patronen die zijn geïntegreerd in patroontalen. De patronen zijn:

- *Common Plan*
- *Hardware in the Loop*
- *Simulator in the Loo*p

De patronen worden beoordeeld door personen die werken in het gebied van mechatronische systemen. Daarnaast wordt bepaald in welke mate deze patronen de problemen van één organisatie afdekken.

De conclusie van dit onderzoek is dat de onderzoeksvraag positief wordt beantwoord. Dit onderzoek heeft problemen geadresseerd die worden ondervonden tijdens de parallelle ontwikkeling van onderdelen van een mechatronisch systeem door een multidisciplinair team. Deze problemen zijn:

- Samenwerking tussen disciplines
- Integratie van de resultaten
- Afhankelijkheid tussen de disciplines

Deze problemen leiden in het algemeen tot vertraging van een project. De resultaten van dit onderzoek zijn nieuwe organisatorische patronen die praktische oplossingen bieden voor deze problemen. Het is de eerste keer dat deze oplossingen worden geformaliseerd en gepresenteerd als een organisatorische patroon en geïntegreerd in de organisatiestructuur patroon taal van (Coplien & Harrison, 2005). Met dit resultaat kan een multidisciplinair team dat een mechatronisch systeem gelijktijdig ontwikkelt profiteren. Het maakt hen bewust van: een oplossing voor hun probleem, wanneer het toegepast kan worden, de krachten en de compromissen van de oplossing, en hoe de oplossing kan worden geïmplementeerd. Deze patronen bieden multidisciplinaire teams de kans om de genoemde problemen op te lossen.

Table of contents

Acknowledgement 4
Abstract 5
Samenvatting 7
1 Introduction 14
1.1 Thesis 14
1.2 Mechatronic system development 14
1.3 Problem statement and research question 15
1.4 Benefits and relevance 15
1.5 Document layout 16
2 Background 17
2.1 Mechatronic system development 17
2.2 Existing solutions 18
2.3 Practice, patterns, and pattern language 18
3 Research description 20
3.1 Introduction 20
3.2 Purpose 20
3.3 Literature study 23
3.3.1 Introduction 23
3.3.2 Purpose 24
3.3.3 Sources 24
3.3.4 Practice mining 25
3.3.5 Result 25
3.4 FMEA worksheets analysis 26
3.4.1 Introduction 26
3.4.2 Purpose 28
3.4.3 Sources 30
3.4.4 Practice mining 31
3.4.5 Result 31
3.5 Retrospective report analysis 33
3.5.1 Introduction 33
3.5.2 Purpose 33
3.5.3 Sources 34
3.5.4 Practice mining 35
3.5.5 Result 35
3.6 Writing patterns 37
3.6.1 Introduction 37
3.6.2 Purpose 37
3.6.3 Sources 38
3.6.4 Result 39
3.7 Pattern evaluation 40
3.7.1 Introduction 40

3.7.2 Purpose ... 40
3.7.3 Result ... 41
3.8 Result ... 44
4 Results ... 45
4.1 Introduction ... 45
4.2 Patterns ... 45
4.2.1 Introduction ... 45
4.2.2 Pattern language ... 45
4.2.3 Pattern: Common Plan ... 48
4.2.4 Pattern: Hardware in the Loop ... 52
4.2.5 Pattern: Simulator in the Loop ... 56
4.2.6 Conclusion ... 59
4.3 Pattern evaluation ... 59
4.3.1 Introduction ... 59
4.3.2 Review feedback on the patterns ... 59
4.3.3 Coverage of failure categories ... 62
4.3.4 Conclusion ... 63
4.4 Conclusion ... 63
5 Discussion ... 65
5.1 Patterns ... 65
5.1.1 Commonalities between patterns ... 65
5.1.2 General pattern applicability ... 65
5.1.3 Pattern language ... 65
5.2 Evaluation ... 66
5.2.1 Coverage ... 66
5.2.2 Review ... 66
5.3 Research approach ... 66
5.3.1 Influence of personal knowledge ... 66
5.4 Recommendations and future work ... 67
5.4.1 Pattern description ... 67
5.4.2 Pattern language ... 67
5.4.3 Solution for top most failure categories ... 68
5.4.4 Mining different sources ... 68
5.4.5 Relationship between standards and language ... 68
6 Conclusion ... 69
7 Reference ... 71
Appendix A Acronyms ... 75
Appendix B Glossary ... 76
Appendix C Confidential sources ... 78
Appendix D Literature study: practices ... 79
Appendix E FMEA: Worksheet layout ... 83
Appendix F FMEA: process for creating failure category ... 84
Appendix G Result of FMEA worksheet analysis ... 90

Appendix H Failure category statistics (based on FMEA and Retrospective reports)..... 95
Appendix I Positive formulation of failure category..... 97
Appendix J Result of retrospective report analysis 98
Appendix K Practices with context and forces 104
Appendix L Pattern sequences..... 105
Appendix M Pattern feedback 107
Appendix N Mapping literature practices to failure categories..... 110
Appendix O Referenced patterns..... 111
Appendix P Pattern Language..... 113
Appendix Q Examples of form layout..... 115

Table of figures

Figure 3.2-1: Form layout for writing a pattern (based on (OrgPatterns, 2001)) 21
Figure 3.2-2: Research setup 22
Figure 3.4-1: Types of FMEA (Haapanen & Helminen, 2002) 28
Figure 4.2-1: Pattern sequences as part of two pattern languages 46
Figure 4.3-1: Review feedback: Is pattern description clear? 60
Figure 4.3-2: Standard deviation for pattern rating 61
Figure 4.3-3: Standard deviation for Simulator in the Loop 61
Figure E-1: FMEA worksheet example 83
Figure F-1: Process description for the classification of potential failure modes 84
Figure F-2: Process description for creating failure categories 86
Figure P-1: Integration of patterns (gray squares) in people and code pattern language 113
Figure P-2: Integration of patterns (gray squares) in project management pattern language 114

Table of tables

Table 3.2-1: Relationship of sources to pattern description 21
Table 3.3-1: Keywords used to find literature 24
Table 3.3-2: List of publications 25
Table 3.3-3: Mining practices in publications 26
Table 3.3-4: Practices mined from publications 26
Table 3.4-1: FMEA severity ranking 29
Table 3.4-2: Examples of failure category overview 32
Table 3.4-3: Practices mined during FMEA worksheet analysis 32
Table 3.5-1: Examples of remarks mapped onto failure category 36
Table 3.5-2: Most important failure categories in reports 36
Table 3.5-3: Practices mined during retrospective report analysis 36
Table 3.6-1: Practices found in the research 38
Table 3.6-2: Practice with problem statement: “How can…” 39
Table 3.6-3: Creating a pattern sequence based on unresolved forces 40
Table 3.7-1: Feedback form used for pattern review 43
Table 3.7-2: Failure categories related to the new patterns 44
Table 4.3-1: Coverage of failure categories by the new patterns 62
Table 5.4-1: Existing patterns referred in new patterns 68
Table D-1: Practices mined from literature 79
Table F-1: Process description for classification of potential failure modes (Figure F-1) 85
Table F-2: Examples of classification of the potential failure modes 85
Table F-3: Process description for creating failure category 87
Table G-1: Failure category overview based on FMEA worksheets 90
Table G-2: Mapping of practice name to solution 94
Table H-1: Failure category statistics 96
Table I-1: Translation of failure category to success category 97
Table J-1: Failure category overview based on retrospective reports 98
Table J-2 Mapping of practice name to solution 102
Table K-1: Pattern description with context, problem, and forces 104
Table L-1: Pattern sequences based on unresolved forces 105
Table N-1: Mapping practices to failure category 110
Table O-1: Patlets of referred patterns 111

1 Introduction

1.1 Thesis

This thesis considers the problems encountered when developing a mechatronic system. The objective of this thesis is to write the existing solutions to these problems in the format of organizational patterns.

The terms relevant for understanding this chapter are briefly introduced here (see also Appendix B for the Glossary):

- **Concurrent engineering.** A methodology used in product development based on the concept of tasks executed simultaneously. Some examples include the parallel development of a system, subsystem, or module.
- **Failure Methods and Effects Analysis (FMEA)**. This is a process whose objective is to prevent or reduce the impact of a potential failure.
- **Mining**. The process of analyzing data and summarizing such data into useful information. Within the context of this thesis, such useful information (practice) contains a problem description and the description of the solution that solves the problem.
- **Pattern language**. A network of interrelated patterns that define a process for resolving development problems systematically.
- **Practice.** A way to solve a problem. A practice describes the problem and its solution, and has a descriptive name based on the solution.
- **Retrospective**. A retrospective is a team activity in which the team reflects on the past period of development. The objective is to learn from the past period and use this knowledge to increase the quality of the product and work life of team members.

Section 1.2 introduces the development of a mechatronic system. Section 1.3 presents the problem statement and research questions. This is followed by the benefits and relevance of this research in section 1.4. The last section (section 1.5), describes an outline of this document.

1.2 Mechatronic system development

This section describes mechatronic systems and how they are developed. More information on this subject appears in section 2.1.

The term *mechatronic* was introduced in 1969. Over the past 40 years, many definitions were presented (Colorado State University, 2012). Thus, defining a mechatronic **system** precisely becomes even more difficult. For this document, the definition of a *mechatronic system* is:

> "A computer-controlled mechanical system [that includes] both an electronic computer and electromechanical components" (Wikipedia - Mechatronics, 2015)

In the literature, other names are used for this definition as well:

- Automatic machinery
- Complex manufacturing systems
- High-tech system
- High-end system

The development of a mechatronic system is traditionally done sequentially. First, a mechanical design is made, followed by an electronic design, and then software is developed to control the system (Boucher & Houlihan, 2008; Alvarez Cabrera et al., 2010). Because there is a demanding need for earlier introduction of these systems into the market, another strategy is chosen by manufactures. In this strategy, system development is done in parallel (concurrent engineering) by the disciplines. All disciplines work on their part of the system simultaneously. These parts are then integrated to become one system. This strategy can accelerate the process of developing a system with half a year (Teich, 2012). Another advantage of this strategy is that a multidisciplinary technical solution can be considered because no design is "frozen". This can lead to better overall system behavior because the interaction among mechanical, electronics, and control behavior can be addressed (Alvarez Cabrera et al., 2010).

1.3 Problem statement and research question

Development of a mechatronic system, subsystems, and even modules of these subsystems can be done in parallel by different disciplines. Through an integration process, these modules and subsystems are joined to become one system. To accomplish this, intense collaboration between disciplines is required. This way of working introduces dependencies between disciplines that can introduce problems. Such problems can then lead to integration issues and project delays (Schafer & Wehrheim, 2007; Bradley, 2010).

This problem statement leads to the following research question:

> "Is it possible to formulate organizational patterns that can be used for the development and integration of a mechatronic system in a multidisciplinary environment developed concurrently?"

1.4 Benefits and relevance

The result of this research can be an addition to the current set of patterns and expansion of the existing pattern languages (Coplien & Harrison, 2005). Once the patterns are published, the organizations that develop mechatronic systems can benefit from the proven solutions. Such organizations will know how to apply the solution and what the resulting context will be. An opportunity for publication is to submit these patterns to the ScrumPLoP community (ScrumPLoP, 2015), an active community with the mission of building a body of pattern literature around Scrum and Agile that can be shared easily.

This research also leads to new questions that will become recommendations for future research work. Other students or researchers can use these questions as research topics.

1.5 Document layout

After the introduction of Chapter 1, Chapter 2 (“Background”) provides some background information on developing a mechatronic system in a multidisciplinary team, and the problems that are encountered.

Chapter 3 (“Research description“) describes the research setup and presents the intermediate results. The research is divided in five steps. Each section describes the expected result, used sources, and results achieved. The last section provides a summary of this research.

Chapter 4 (“Results”) presents the three new organizational patterns and their related pattern languages. It also presents the results from the evaluation of the reviewers, and how many of the problems of one organization are covered by the new patterns.

Chapter 5 (“Discussion”) discusses the research, results, and evaluation. It also proposes topics for future research.

Chapter 6 (“Conclusion”) answers the research questions and the problem statement.

2 Background

2.1 Mechatronic system development

Mechatronic system development (see also section 1.2) requires a multidisciplinary approach. The disciplines that have to work together are electronics, mechanics, and software. The increasing complexity of the systems and integration of different technologies makes it essential for disciplines to work in close collaboration. This requires intense communication during development. This human interaction is an important aspect that should be considered (Boucher & Houlihan, 2008; Bradley, 2010; Bonnema, Borches & Houten, 2010).

To increase the time to market of mechatronic systems, they are developed with concurrent engineering, which is the parallel development of a system, subsystem, or module. With such concurrent engineering, mechatronic systems can be available in the market up to six months earlier than the classical design flow in which the system is developed sequentially (Teich, 2012). This acceleration occurs partly because the integration of the components delivered by the disciplines (e.g. software module and hardware components) starts early in the development cycle. This helps to:

- Start testing the parts (e.g., hardware and software) together in an early stage. This makes any integration issues apparent. For example, when a root cause is a hardware problem, and it is discovered late in the development cycle, it might not be possible to solve such problem in the hardware because of the long hardware lead times. This long lead time might cause the project to miss its delivery deadline. In order to meet the deadline, a solution is then created in the software because it has no long lead time. The early insight of integration issues has the advantage that problems can still be solved at the root cause. This can prevent implementation of software solutions for underlying hardware problems, thus making the software less complex and more maintainable.
- Determine how the subsystem is to be integrated in the system.
- Determine the user experience of the subsystem behavior. This has the advantage that new or changed user requirements can be included relatively easy, thus preventing "under" or "over design."

To allow concurrent engineering, the design from the various disciplines should be shareable between such disciplines at an early stage. In addition, each discipline should have the ability to test its functionality independently as much as possible. The project team should then integrate all the functionality pieces efficiently.

The system stakeholders are customers, production department, service department, and marketing. Other stakeholders that might be involved are suppliers and authorities (e.g., the U.S. Food and Drug Administration (FDA)) (Graaf, Lormans & Toetenel, 2003).

2.2 Existing solutions

Design patterns are available for the development of mechatronic systems. Some examples are safety patterns (Armoush, Salewski & Kowalewski, 2008; Wagner, Schatz, Puchner & Kock, 2010), patterns to improve the model quality (Kim, Kim & Hong, 2009), system control patterns (Pont & Banner, 2004; Fantuzzi, Bonfe, Secchi, e Reggio & Emilia, 2009; Garro, Ordinez & Alimenti, 2011), and patterns for parallel processes (Keutzer, Massingill, Mattson & Sanders, 2010). These patterns are mainly mono-disciplinary and do not consider concurrent engineering.

Various methods and frameworks are defined to support the development of mechatronic systems. Some examples are a framework for multidisciplinary teams (Alvarez Cabrera et al., 2010), a framework for designing architecture (Chen & Torngren, 2001), and a model for the creation of system architecture (Heemels, van de Waal & Muller, 2006). The frameworks provide solutions for certain aspects of concurrent engineering and collaboration between different disciplines. The frameworks consist of an integrated solution with tools, processes, and methodologies. Its introduction requires an investment from the organization. The organization in which developers are doing their job will not always have access to such a framework. A reason can be that the organization does not have the resources to do such an investment. Another reason might be that the organization has defined their own specific way of working to develop mechatronic systems (see also 2.3).

The problems that continue to exist with the existing solutions are:

- No good coordination between the various disciplines, which can lead to wrong solutions for system problems, or to the system not meeting technical requirements. For example, a particular hardware engineer might decide to reduce the amount of sensors on a system because of costs or lack of physical space. However, without these sensors, it might be difficult for the software to determine the system status. Consequently, the software has to implement derivative logic to determine the system state, thus causing the software to become complex and introduce the risk of bugs and maintenance burden. Another problem can be that the software cannot guarantee the real system status.
- No early testing and verification, which can cause inconsistent behavior at the system level, or to the end result not meeting user needs and/or technical requirements.

2.3 Practice, patterns, and pattern language

There are organizations that have encountered and addressed the issues described in section 2.2. Their solution most likely considers the organization, current processes, and the product. Such solution can be a best or common practice that might not resolve the same problems within other organizations because it might not be generally applicable. For a practice to become a pattern, it should first solve all its conflicting interests (forces). Next, the solution should improve the situation and not make it comparable or worse. Finally, the solution should have a proven track record.

A definition of the term *pattern* used commonly is:

> "Each pattern describes a problem that occurs [repeatedly] in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."(Christopher, Ishikawa & Silverstein, 1977).

A pattern describes its solution generically. This makes it possible to customize the use of the pattern to the environment in which it is to be implemented. Such flexibility results in many different applications of the same pattern. Almost every software engineer knows the design patterns of the Gang of Four (Gamma, Helm, Johnson & Vlissides, 1994). Non-design patterns, such as organizational and pedagogical patterns, are also commonly used (Buschmann, Henney & Schmidt, 2007), but are less known.

A pattern should be able to describe the following items (Coplien & Harrison, 2005):

- Pattern name (Title)
- Context in which the problem is found
- Pattern forces or tradeoffs
- Solution
- Explanation of why the pattern works

Patterns can be viewed as standalone solutions or as a collection of solutions. Different types of collections exist. Examples include pattern complements, compounds, sequences, and languages (Buschmann, Henney & Schmidt, 2007).

A pattern language can be seen as a roadmap. The exact path that an organization should take depends on circumstances and on the progress such organization makes to mature its development process. A pattern language is a language that comprises patterns and rules to combine patterns in meaningful ways and in a particular sequence. It describes how to create an integrated system (Coplien & Harrison, 2005). For additional information about this topic, refer to (Buschmann, Henney & Schmidt, 2007).

3 Research description

3.1 Introduction

This chapter describes the research that was conducted to obtain an answer for the research question (section 1.3).

This research mines practices by executing a literature study and analyzing field data. The literature study searches for journal articles, proceedings, and books. Within these publications, practices are mined. Practices are also mined in field data made available by an organization that develops and produces mechatronic systems. Only a selection of these practices is detailed in a pattern description and integrated into a pattern language. These patterns are then evaluated to determine whether they can answer the research question positively.

The terms relevant for understanding this chapter are briefly introduced here (see also Appendix B for the Glossary):

- **Failure category**. A generalization of failures that can potentially occur during the development, deployment, or use of a mechatronic system. These *failure categories* are based on potential failures from the FMEA worksheets, and they are used to allow for the processing of the great diversity of information available in such worksheets, as well as to obtain an insight on the size and severity of the field problems.
- **FMEA worksheet**. This is a recording of FMEA activity. An example of such a worksheet can be found in Appendix E.
- **Potential failure mode (also known as Failure mode)**. This term is used within FMEA worksheets (section 3.4), and it is a description of a specific failure that may occur within the project or with the system and its functions (NASA Academy of Aerospace Quality, 2014, DFMEA continued).
- **Retrospective report.** This is a recording of a retrospective. This report usually contains positive, negative, and improvement remarks.

Section 3.2 provides a description of the purpose of this research and the expected results. The following five sections (sections 3.3 to 3.7) describe each research step in detail. Each section describes the expected result, the sources used, and results achieved. Sections 3.3 to 3.5 also describe the process for mining the practices. The final section (section 3.8) provides a summary of the results.

3.2 Purpose

This section describes the expected result from this research, which is to answer the research question (section 1.3) positively. This means that this research should produce those patterns that can be used for the development and integration of a mechatronic system in a multidisciplinary environment developed concurrently.

This result can be achieved when the following activities are executed:

- Description of the patterns according to the form layout of Figure 3.2-1
- Description of the patterns based on verifiable sources
- Integration of the patterns into a pattern language
- Evaluation of the patterns

The patterns are described according to the form presented in Figure 3.2-1. This form is chosen because the new patterns will become part of the pattern language of (Coplien & Harrison, 2005). Two examples of pattern descriptions are provided in Appendix P.

Title *(can be descriptive and sometimes evocative)*

A picture *(should underscore the human dimension and should help make the pattern memorable)*
Prologue that describes the context in which the problem is found

❖ ❖ ❖

Problem statement
Discussion on the pattern forces or tradeoffs
Solution presentation *(sentence should start with "Therefore:")*

❖ ❖ ❖

Discussion of why the pattern works
Optionally, a description of:

- Related patterns
- Examples
- Sample situation
- Principles involved
- Related reading

Figure 3.2-1: Form layout for writing a pattern (based on (OrgPatterns, 2001))

The pattern description should be based on the information gathered during this research. Verifiable sources should be used to convincingly prove that the solution is used for real problems. The relationship between the patterns, practices, and sources used is expressed in Table 3.2-1.

Table 3.2-1: Relationship of sources to pattern description

Pattern	Practice	Literature	FMEA worksheet	Retrospective report
Title	Name			
Problem statement	Problem	Problem	Failure category and examples of potential failure mode(s)	Failure category and examples of negative remarks
Solution presentation	Solution	Solution	Examples of current design control or recommended action.	Examples of positive or improvement remarks

The first column describes the information required for the pattern description (Figure 3.2-1). The form layout shows that more information is required for a pattern description. For the clarity of this view, this is not included in Table 3.2-1. The second column shows the information required to

describe a practice. Such best or common practice describes the solution to a problem, but it might not resolve the same problems within other organizations because it is not generally applicable. The next three columns relate to the sources used in the research "step 1," "step 2," and "step 3" (Figure 3.2-2). These three steps produce practices as input for writing patterns in "step 4." The examples given in these columns are retrieved from the sources. Details on this information can be found in the sections that describe that step.

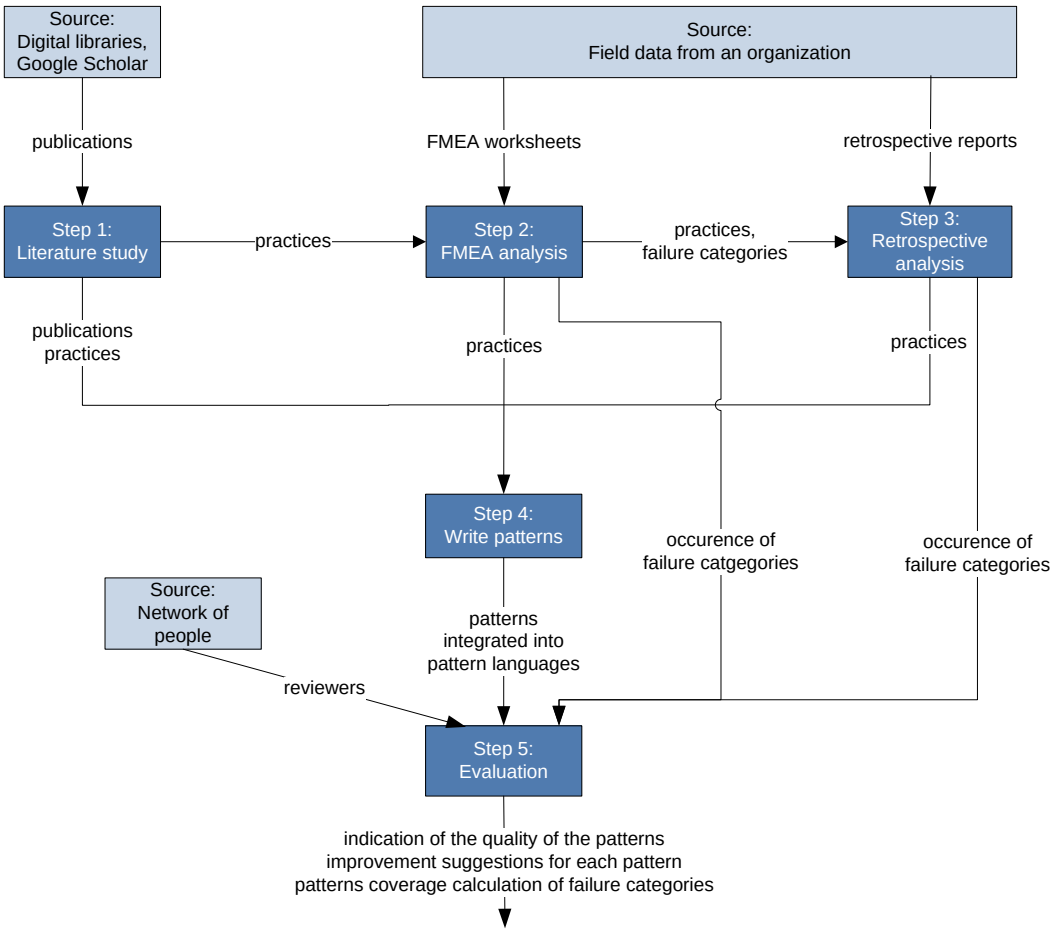


Figure 3.2-2: Research setup

In "step 1," the practices are mined from publications. In "steps 2" and "3," the practices are mined from field data. In "step 2," the same is done on FMEA worksheets, and in "step 3," on retrospective reports. The practices mined from these steps are given a practice name, which is inspired by the solution described for the practice. To prevent unnecessary name variation, the practice names already defined in a previous step can be used (indicated by the lines between "steps 1, "2," and "3").

In "steps 2," *failure categories* are defined. These are failure generalizations that can potentially occur during the development, deployment, or use of a product. These categories are used as the

problem description of a practice, and they are made to allow the processing of the great diversity of information retrieved from the FMEA worksheets, which cover all aspects from project initiation to system maintenance. These *failure categories* are also used in "step 3" to find solutions that solve the problem.

The *failure categories* can also provide insight on the size and severity of field problems. In "steps 2" and "3," the field data are categorized into the *failure categories*. This information is then used in the pattern evaluation ("step 5").

"Step 4" receives the practices from the previous steps. These data consist of the practice name and problem and solution description. Only some of these practices are detailed according to the form as used by (Coplien & Harrison, 2005) (see Figure 3.2-1). In this "step 4," references are added to the pattern description when they contribute to pattern readability and clarity. These references can be journal articles, books, or websites. These patterns are then integrated into the pattern language, which requires knowledge of all the patterns (Coplien & Harrison, 2005). Such integration should be based on the unresolved force(s) of the solution provided for a pattern. The problem statement of the other pattern should address the unresolved force(s). This integration places the patterns into the context of other problems and solutions. This way, patterns can be more easily understood and successfully applied.

The patterns are evaluated in "step 5." Individuals with experience developing mechatronic systems in a multidisciplinary environment conduct this evaluation. This evaluation is performed to determine whether a multidisciplinary team that develops a mechatronic system can use the patterns, and whether such patterns solve the problem they describe. It is also determined, by the obtained occurrences of the *failure categories*, how many of the problems of one organization are covered by these patterns. The result indicates the significance of the patterns in relationship to the *failure categories*.

3.3 Literature study

3.3.1 Introduction

In this literature study, publications (journal articles, proceedings, and books) are searched for the following subjects:

- Development and integration of mechatronic systems
- Development by a multidisciplinary project
- Concurrent engineering

These publications are analyzed and summarized into practices (problem/solution description).

The following sections describe the expected results, sources used, process for mining the practices, and results achieved.

3.3.2 Purpose

This section describes the expected result of the literature study, which is that the selected publications can be used as a source of information when the patterns are to be written. Another expected result is to have sufficient information to start writing patterns.

This result can be achieved when the following data is produced:

- List of publications
- List of practices mined from the literature

The practices are presented in an overview of the solutions to given problems. Each problem/solution pair should have a unique name (practice name) based on the solution description. Information from these practices is used for writing patterns (Table 3.2-1):

- The practice name is used for the *Title*
- The problem is used for the *Problem statement*
- The solution is used for the *Solution presentation*

To prevent unnecessary variation of practice names, one name can be assigned multiple times when the solutions appear to be equal. No name is created when a solution is already mentioned as a pattern (Coplien & Harrison, 2005) in order to prevent rewriting existing patterns.

3.3.3 Sources

This section describes the sources used for the literature study.

A search on journal articles, proceedings, and books is conducted with the scientific publishing sites Springer, ACM Digital Library, IEEE Digital Library, and ScienceDirect (Elsevier). Google Scholar is also used to search for books. The keywords used for searching are listed in Table 3.3-1.

Table 3.3-1: Keywords used to find literature

Keywords			
Automatic machinery	Embedded systems	Interdisciplinary	Patterns
Collaboration	Engineering disciplines	Manufacturing systems	Process
Complex systems	Hardware	Mechanical System	Product development
Concurrent	High-end system	Mechatronic	Project
Cross-functional	High-tech system	Multidisciplinary	Software
Development	Innovation teams	Multi-domain	System engineering
Domains	Integration	Organization	Teams

3.3.4 Practice mining

This section describes the process for mining practices from the publications, which is done in several steps. First, a summary is made of each publication (Table 3.3-2), and this consists of the abstract, problems, and solutions. The result of this summarization is a total of 147 solutions and 158 problems. The relationship between the problem and its solution is not always clear in the publication; therefore, only those solutions that could be related to a problem are selected, which results in problem/solution pairs. Then, these pairs are combined when they describe the same type of solution. For each pair, a practice name is created that is inspired by the described solution. Practice names that relate strongly to each other are reviewed. After the review, the names can be made more distinguishable, or the practices can be merged.

3.3.5 Result

This section describes the result of the search for valuable publications and the mining of the practices.

The result of the literature study is that the expected results (section 3.3.2) are achieved, as follows:

- List of publications. The result is presented in Table 3.3-2.
- List of practices mined from the literature. The result is presented in Table 3.3-3 and Table 3.3-4. The full overview is provided in Table D-1.

The publications that can possibly answer the research questions are searched as described in 3.3.3. The results of the search are:

- 15 journal articles/proceedings published between 2003 and 2013
- four books published between 1998 and 2013

The 19 publications found can provide insight on the problems that occur during development and integration of mechatronic systems, and the development by a multidisciplinary project. That makes these publications valuable sources of information when the patterns are to be written.

Table 3.3-2: List of publications

List of publications (books, journal articles, and proceedings)			
(Michalski, 1998)	(Heemels, van de Waal & Muller, 2006)	(Boucher & Houlihan, 2008)	(Alvarez Cabrera et al., 2010)
(Parker, 2003)	(Schafer & Wehrheim, 2007)	(Ratcheva, 2009)	(Nakata & Im, 2010)
(Graaf, Lormans & Toetenel, 2003)	(Vijaykumar & Chakrabarti, 2007)	(Moneva, Hamberg, Punter & Vissers, 2010)	(Zheng, le Duigou, Bricogne & Eynard, 2013)
(Muller, 2005)	(National Aeronautics and Space Administration, 2007)	(Kleinsmann, Buijs & Valkenburg, 2010)	(Jurgens-Kowal, 2013)
(Northrup & Northup, 2006)	(Beckers, Muller, Heemels & Bukkens, 2007)	(Bradley, 2010)	

Another result of this literature study is 39 practices. For each practice, one or more problems are described, along with one or more similar solutions, and each practice is given a unique practice name. Several examples can be found in Table 3.3-3. The full overview is provided in Table D-1. This

information is sufficient to start writing a pattern. An overview of all the practice names is provided in Table 3.3-4.

Table 3.3-3: Mining practices in publications

Problem	Solution	Practice name
Many integration problems at the end of project. (Alvarez Cabrera et al., 2010), (David Bradley, 2010) Fault detection and diagnoses can only be executed when final system is available. (Boucher & Houlihan, 2008)	Create setup that involves only critical hardware parts and software to be integrated. (Boucher & Houlihan, 2008), (David Bradley, 2010), (Alvarez Cabrera et al., 2010)	Hardware in the Loop
System level issues discovered late in design process. Consequently, design options are reduced because critical decisions are already made. (Boucher & Houlihan, 2008)	Simulate behavior at system level. With this simulation, virtual tests can be executed early in design cycle. This allows early identification of problems at system level. (Boucher & Houlihan, 2008)	Simulator in the Loop

Table 3.3-4: Practices mined from publications

Practice names			
Abstract design	Define dependencies	Involve key stakeholders	Only design critical parts
Alert design change	Define roles in team	Keep the schedule	Provide an incentive
Brainstorm on Design	Design walk through	Keep track of requirements	Shared leadership
Budget design	Empower the team	Key master	Simulator in the Loop
Build commitment	Encourage risk taking	Knowledge integration	Single point of contact
Central power	Establish a scoreboard	Learn by interaction	Small team
Clear priorities	Hardware in the Loop / Iron Bird	Learning community	Splitter
Clear tasks	Incorporate goals	Lessons learned	Team of experts
Common ground	Information flow	Manage team member expectations	Work in parallel
Common plan	Integrator	Mutual accountability	

The results of this literature study are used in the next sections. The publications are used to write the patterns (section 3.6). The practices are used to write the patterns (section 3.6).The practices are also used to prevent practice duplication during the retrospective analysis (section 3.4).

3.4 FMEA worksheets analysis

3.4.1 Introduction

To allow practice mining in a real organization that develops mechatronic systems, FMEA worksheet analysis is performed. Creating an overview of the failures that can occur during the development and use of a mechatronic system, as mentioned in the worksheets, allows practices to be mined.

FMEA is a process with the objective of preventing or reducing the impact of a *potential failure mode* (see terms below) identified by participants (e.g., project members, suppliers, or customers) during brainstorm sessions. The activity identifies the *potential failure modes* of a product

(Functional/Design FMEA) being developed, or that can occur during the development process (Process FMEA). For each *potential failure mode*, the following items are determined:

- potential failure effects
- potential cause(s)
- probability
- detection
- severity

An FMEA worksheet (Figure E-1) is the result of an FMEA activity.

The terms relevant for understanding FMEA worksheets are (see Appendix B for Glossary):

- **Potential failure mode (also known as Failure mode)**. The failure mode is a description of a specific failure that may occur within the project, or with the system and its functions (NASA Academy of Aerospace Quality, 2014, DFMEA continued).
- **Potential failure effect(s) (also known as Failure effect).** Description of the immediate consequence of a specific failure (NASA Academy of Aerospace Quality, 2014, DFMEA continued).
- **Severity**. Evaluation of the severity of the failure effect on the next system or the internal/external customer. Sometimes, large severity values can be reduced through design reviews that compensate or mitigate the resulting severity (NASA Academy of Aerospace Quality, 2014, DFMEA continued).
- **Current design control**. List the verification/validation design activities, or other related activities (NASA Academy of Aerospace Quality, 2014, DFMEA continued).
- **Recommended action**. Description of an action that can be taken to prevent the failure from happening, or eliminate/reduce the immediate consequences of the failure (NASA Academy of Aerospace Quality, 2014, DFMEA continued).

Corrective measures need to be taken in order to prevent or reduce the impact of a *potential failure mode* with high severity ranking. Such measures are called *current design controls* or *recommended actions* (Department of defense, 1980).

In this analysis, the term *failure category* is used as a generalization of *potential failure modes*. As part of this FMEA worksheet analysis, *failure categories* are defined. This categorization is made to allow the processing of the great diversity of information retrieved from the FMEA worksheets. It is not possible to use standardized *failure categories* because "there is no single list of failure modes that apply to all products. Some companies try to develop such lists for their specific products"[1]. Another reason is that the standardized *failure categories* (Chandler, Denson, Rossi & Wanner, 1991)

[1] This statement was made by Carl Carlson (author of (Carslon, 2012)) in an email conversation that I had with him (30-August-2014).A similar statement was made by Michael Herman (FMEA-FMECA.com, 2013) in an email conversation that I had with him (3-September-2014).

and (Flores & Malin, 2013, Appendix A) are not supportive in finding multidisciplinary problems for causes that include the following:

1. The standardized *failure categories* lists are very exhaustive. This makes it difficult to generalize the failure.
2. The standardized *failure categories* focuses on specific elements (fluid, sensor, I/O, etc.) and not on the product (mechatronic module or system).
3. The *potential failure mode*s from the received FMEA worksheets have great diversity. The worksheets cover all aspects from project initiation to system maintenance, which means that the *potential failure mode* can relate to the product or the process (see Figure 3.4-1).

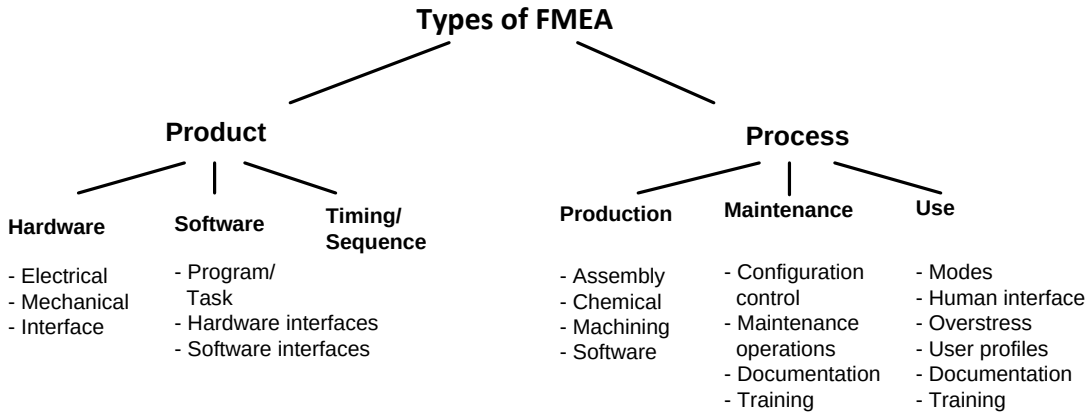


Figure 3.4-1: Types of FMEA (Haapanen & Helminen, 2002)

The following sections describe the expected results, sources used, process for mining the practices, and results achieved.

3.4.2 Purpose

This section describes the expected results of the FMEA worksheet analysis, which are to have sufficient information to start writing patterns, allow practice mining in retrospective reports, and determine how many of the problems of one organization are covered by these patterns. These results indicate the significance of the patterns to the *failure categories*.

These results can be achieved when the following information is produced:

- List of *failure categories,* sorted by average *severity*
- List of occurrences of *failure categories* in the FMEA worksheets
- List of practices mined from the FMEA worksheets

The *failure categories* should be defined based on the FMEA worksheets, and all *potential failure modes* need to be assigned to a *failure category*.

One *failure category* should describe the following items:

- **Name of the *failure category***. This name should be descriptive of a problem. This helps when *potential failure modes* need to be assigned to the *failure category*.
- **Average *severity* ranking**. The organization that made the worksheets available customized the severity ranking standard SAE J1739 FMEA (Society of Automotive Engineers, 2000) to fit their needs. This is presented in Table 3.4-1. The ranking of the average *severity* can give an indication on the impact of the *failure category* on the project and mechatronic system.
- **Occurrences in worksheets.** For each set (*failure category*), the number of elements (*potential failure modes*) is counted. The amount of occurrences can provide an indication as to which failures are of biggest concern to the projects.
- **Examples of *potential failure mode(s)***. Examples of *potential failure modes* assigned to the *failure category*. These can be used as *failure category* examples (examples of the problem). A maximum of four *potential failure mode(s)* are defined in order to maintain the description generic.
- **Examples of *potential failure effect(s)*.** Generic examples of the *potential failure effect(s)*. The purpose of these examples is to provide more context to the *failure categories*. A maximum of four *potential failure effect(s)* are defined in order to maintain the description generic.
- **Examples of *current design control* and/or *recommended action*.** Generic examples of the *current design controls* and *recommended actions*, which can be used to define practices. A maximum of four *current design control* and/or *recommended action* are defined in order to maintain the description generic.

Table 3.4-1: FMEA severity ranking

Effect	Severity of Effect	Ranking
Hazardous without warning	Human safety issues	10
High	Production material damage or destruction Major tool/equipment damage	8
Medium	Minor tool/equipment damage system specifications compromised	5
Low	Brief production interruption non-critical specifications compromised	2
None	No measurable effect	0

Practice names should be given to a *failure category* when the solution solves the problem. To prevent unnecessary variation of practice names, one name can be assigned multiple times when the solutions appear to be the same. No name is created when a solution is already mentioned as a pattern (Coplien & Harrison, 2005) in order to prevent rewriting existing patterns. Another way to prevent unnecessary name variation is the use of the practice names already defined in the literature study (section 3.3). The practice names should only be used when the solutions matches.

Information from these *failure categories* are used for writing patterns (Table 3.2-1):

- The practice name is used for the *Title*
- The *failure category* and examples are used for the *Problem statement*
- The *current design control* or *recommended action* is used for the *Solution.*

The list of *failure categories* is limited to 30 categories in order to maintain the description generic. This list should also be able to indicate those *failure categories* that are covered by the new patterns.

3.4.3 Sources

This section describes the sources used for the FMEA analysis. The objective is to create an overview of all *potential failure modes* from the FMEA worksheets. With this overview, *failure categories* can be determined.

The FMEA worksheets (Figure E-1) used for this research were made available by an organization that develops mechatronic systems (Appendix C). This organization operates globally and has its own research and development department that is responsible for creating the worksheets. In particular, the project teams with the responsibility to deliver a module to the mechatronic system perform such creation. The project teams consist of several disciplines that in most cases include software, mechanics, electronics, product quality, factory integration, and service. During brainstorming sessions, the project team identifies the risks of potential module or development process failure. The risks are expressed by *potential failure mode* in a worksheet. The FMEA activity is executed several times during module development, and such modules can be new or replacements of existing modules.

The organization made 20 FMEA worksheets available that were created between 2006 and 2014. For each worksheet, a determination is made as to whether it can contribute to the research question. The selection criteria are:

- The project involves multidisciplinary activities
- The worksheet is written according to a formal layout. Examples are MIL-P-1629 by the US Armed Forces Military (Department of defense, 1980) and ARP4761 by the Society of Automotive Engineers (Society of Automotive Engineers, 1996)
- Duplicate worksheets are not used

The result is a selection of 17 worksheets that contain 1,185 *potential failure modes*.

3.4.4 Practice mining

This section describes the process for mining patterns from the FMEA worksheets, which is conducted in several steps. In the first step, *failure categories* are defined (Appendix F). This process starts with a classification of all the *potential failure modes* (see Figure F-1) based on the definitions of Figure 3.4-1. Next, the list of *failure categories* is defined based on the classification list (see Figure F-2).

In the second step, all the *current design controls* and *recommended actions* of each *failure category* are analyzed. First, an overview is created of all *current design controls* and *recommended actions* that belong to the same *failure category*. Relationships and overlapping descriptions can be rephrased in a generalized description. The descriptions also need to be rephrased to a more abstract description, which should remove the specific technical context that makes it more applicable for mechatronic systems in general. For each *failure category*, a maximum of four *current design controls* and *recommended actions* are defined in order to maintain the description generic.

The generic examples of *current design control* and *recommended action* can be seen as a solution to the problem (*failure category*). The solutions are given a practice name inspired by the solution description.

3.4.5 Result

This section describes the result of the FMEA worksheet analysis, which is that the expected result (section 3.4.2) has been achieved. The results are:

- List of *failure categories* sorted by average *severity.* The result is presented in Table 3.4-2, first and second column. A full overview is given in Table G-1.
- List of occurrences of *failure categories* in the FMEA worksheets. The result is presented in Table 3.4-2, third column. A full overview is given in Table G-1.
- List of practices mined from the FMEA worksheets. The result is presented in Table 3.4-3. A full overview is given in Table G-1.

Table 3.4-2 (a full overview is given in Table G-1) contains sufficient information to start writing patterns. The first row relates to the information available in the worksheets. The second row relates to the information required to define a practice (relation is also expressed in Table 3.2-1). In the first and fourth column, the practice problem is described. The fifth column contains examples to provide more context to the *failure category*. The sixth column contains the solution. The practice name (seventh column) is provided during the mining process (section 3.4.4). The relationship between the solution (sixth column) and the practices name (seventh column) is provided in Table G-2.

Table 3.4-2: Examples of failure category overview

Failure category (The risk is that …)	Average severity ranking	Occurrences in worksheets	Examples of Potential Failure Mode(s)	Examples of Potential Failure Effect(s)	Examples of current design control and/or recommended action	
Problem			Example of problem		Solution	Practice name
Integrated system behavior is not considered	6.2	153	Collision between components Pre-conditions for operation are not met Disturbances (e.g., vibration, temperature variation) influence performance	Reduced system performance Component or system becomes damaged	Improve design 3D design check Bench testing	Mock-up Hardware in the Loop
System deteriorates over time	5.8	107	System becomes contaminated Parts wear out Corrosion on components Parts become loose (e.g., because of aging glue, cracked solder joints)	System becomes unreliable Component or system becomes damaged Safety issues may arise	Reduce load on component (e.g., restrict number of retries) Execute lifetime tests of component at supplier	Module Testing

Table 3.4-2 (columns two and three) contains information on the average *severity* of the practices and the amount of occurrences of the *failure categories* in the FMEA worksheets. The table shows the relationship between the *failure categories* (first column) and the practice (seventh column). When a given practice becomes a pattern, the coverage of that pattern can be determined. The *failure category* is a generalization of the *potential failure modes*. This means that a pattern only cover aspects of the *failure category*. An example of such is the practice *Hardware in the Loop* that becomes a pattern. This relates to the *failure category, Integrated system behavior is not considered*, which has 153 occurrences. From a total of 1,185 potential failure modes, this represents coverage of 12.9% of all *potential failure modes*. Depending on the pattern description, not all mentioned problems will be solved.

Table 3.4-3 shows the 11 practices mined during the FMEA worksheet analysis. Each practice is given a unique and descriptive name.

Table 3.4-3: Practices mined during FMEA worksheet analysis

Practice names		
Clear plan	Factory acceptance test	Review deliverables
Define up/downgrade strategy	Hardware in the Loop	System integration testing
Drop-In replacement	Mock-up	Test with dummies
Embed knowledge in system	Module Testing	

The result of this FMEA worksheet analysis is used in the next sections. The list of *failure categories* is used to assign retrospective remarks (section 3.5). The practices are used for writing patterns (section 3.6). The practices are also used to prevent practice duplication during the retrospective

analysis (section 3.5). The list of occurrences of *potential failure modes* is used to determine the *failure category* coverage of the new patterns (section 4.3.3).

3.5 Retrospective report analysis

3.5.1 Introduction

In order to allow practice mining in a real organization that develops mechatronic systems, a retrospective report analysis is conducted. This mining is allowed by creating an overview of retrospective remarks related to *failure categories* (section 3.4). The practices are mined based on this overview.

A retrospective is a meeting in which a team reflects on the past period of development. A retrospective report is the minutes of such meeting. The objective of the retrospective is to increase product quality and the work life of team members. This is accomplished by incorporating the successes and improvements in the next period. Retrospective can be performed in many different ways (Derby & Larsen, 2006).

The terms that are relevant for understanding retrospective reports are (see also Appendix B):

- **Positive remarks**. Remarks on product development of which an individual or team is proud, and that the individual or team want to continue doing. These remarks are reported as: proud, positive, '+', continue to do, etc.
- **Negative remarks**. Remarks on product development of which the individual or team is dissatisfied. These remarks are reported as: negative, sorry, '-', sad, needs improvement, etc.
- **Improvement remarks**. Action assigned in order to improve a negative remark. These remarks are reported as: action point, agreement on what should be changed, etc. Such remarks are differentiated from the positive/negative remarks category because actions are assigned to them; furthermore, these actions are to be executed in the upcoming period to improve the current situation. Most of the remarks found in the reports describe an action on how to improve.

The following sections describe the expected results, sources used, process for mining the practices, and results achieved.

3.5.2 Purpose

This section describes the expected results of the retrospective report analysis, which are to have sufficient information to start writing patterns, and be able to determine the coverage of *failure categories* by the new patterns.

These results can be achieved when the following information is produced:

- List of occurrences of *failure categories* in the retrospective reports
- List of practices mined from the retrospective reports

The information is retrieved from the retrospective reports. Only the relevant remarks should be assigned to a *failure category*.

Practice names can be given to a *failure category* when the solution solves the problem. Unnecessary variation of practice names should be prevented (see 3.4.2).

One *failure category* should describe the following items:

- **Name of the *failure category.*** These are defined in section 3.4, see Table G-1.
- **Example of positive/negative/improvement remarks.** Generic examples of the remarks assigned to the *failure category*. The purpose of these examples is to provide more context to the *failure categories*. A maximum of four remarks each are defined in order to maintain the description generic.

Information from these *failure categories* can be used for writing patterns (Table 3.2-1):

- The practice name is used for the *Title.*
- The *failure category* and negative remarks are used for the *Problem Statement.*
- The positive and improvement remarks are used for the *Solution presentation.*

It should be possible to determine which *failure categories* are covered by the new patterns.

3.5.3 Sources

This section describes the sources used for the retrospective analysis. The objective is to create a categorized (positive/negative/improvement remarks) overview of all retrospective remarks for all reports.

The reports used for this research were made available by an organization that develops mechatronic systems (see Appendix C). This organization operates globally and has its own research and development department that is responsible for creating the reports. In particular, the department teams have the responsibility to deliver features to the system, and they have to maintain their software component(s). The organization made 59 reports available, which were created between 2008 and 2012.

A selection is made on the available reports. For each report, it is determined whether it can contribute to the research question. The selection criteria are:

- The project involves multidisciplinary activities.
- The report contains retrospective remarks (positive, negative, and improvements).

The result of this step is a selection of 58 reports. The content (1,229 remarks) of the reports can be categorized into the three retrospective categories (647 negative, 492 positive, 90 improvement remarks).

3.5.4 Practice mining

This section describes the process for practice mining from the retrospective reports, which starts with assigning relevant remarks to *failure categories* (section 3.4), and interpreting each remark. Based on such interpretation, the remark can be assigned to a *failure category*. In order to assign the positive remarks to the *failure categories*, a translation is made of the *failure categories* to the *success category* (Table I-1). Using the *success category*, all positive remarks do not need to be translated into the negative context of the *failure category*. This makes it easier to assign a positive remark to a *failure category*.

This assignment results in an overview where a *failure category* has the description of a problem (negative remarks) and a solution (positive or improvement remarks). The solutions are given a practice name inspired by the solution description. Duplication of practice names should be prevented.

3.5.5 Result

This section describes the result of the retrospective report analysis, which is that the expected result (section 3.5.2) has been achieved. The results are:

- List of occurrences of *failure categories* in the retrospective reports. The result is presented in Table 3.5-2. A full overview is given in Table H-1.
- List of practices mined from the retrospective reports. The result is presented in Table 3.5-3. A full overview is given in Table J-1.

For the analysis, 58 reports are selected. These reports were created in the years 2008 to 2012 (section 3.5.3). The content of the selected reports are categorized into three retrospective categories: positive, negative, and improvement remarks. This results in 1,229 remarks found in all the selected reports. A total of 852 remarks could be assigned to a *failure category*. Because the *failure categories* are based on FMEA's and they relate to product (Functional/Design FMEA) or process (Process FMEA) remarks, a total of 204 remarks related to organization could not be assigned.

This retrospective analysis provides insight on the problems that software development teams encounter during the development and maintenance of a mechatronic system. The reports are valuable for mining practices. Several remarks are summarized in Table 3.5-1 (a full overview is available in Table J-1). The first row relates to the information available in the retrospective reports. The second row relates to the information required to define a practice (relation is also expressed in Table 3.2-1). This table provides an overview of the remarks related to the *failure categories*, and it is used to mine practices. The negative remarks can be seen as examples of problems that did occur. The positive and improvement remarks can be seen as examples of solutions proven to be successful. The practices are given a name inspired by their solution description. One practice can be assigned to

multiple *failure categories*, and one *failure category* can have multiple unique practices assigned to it. The relationship between the solution and its practices name is provided in Table J-2.

Table 3.5-1: Examples of remarks mapped onto failure category

Failure category	Negative remarks	Positive remarks	Improvement remarks	
Problem	Example of problem	Solution	Solution	Practice name
Project plan is not managed	Focus is only on solving issues Roadmap is not clear Project is poorly planned	Create roadmap (define features and milestones) Create planning/work breakdown and track plan Communicate project status	Organize regular meetings with stakeholder to discuss feature priorities and requirements	Clear plan Common plan Prioritize for focus
Test coverage is too low	Not all high risk issues were verified No testing is performed on a real system No test plan available	Create unit tests Execute automated tests (e.g., weekend runs, nightly runs, smoke tests) Remote testing when specific hardware is on another location	Write unit tests Use simulation for (offline) testing Test deliverables Improve facilities for local testing	Duration runs Hardware in the Loop Simulator in the Loop Tester in team

All the remarks assigned to a *failure category* are counted and ranked. Those f*ailure categories* mentioned most frequently are listed in Table 3.5-2. The columns with the remarks contain the amount of occurrences and their ratio, which is calculated based on all the remarks of the same set of remarks (positive/negative/improvement). This ratio provides insight on the relevance of the *failure category* within the retrospective reports. A full overview is provided in Table H-1.

Table 3.5-2: Most important failure categories in reports

Failure categories	Positive remarks		Negative remarks		Improvements	
Production is not efficient	108	29.3%	151	26.8%	25	27.8%
Project plan is not managed	50	13.6%	79	14.0%	18	20.0%
Requirement is not met	32	8.7%	21	3.7%	4	4.4%
Test coverage is too low	29	7.9%	54	9.6%	8	8.9%

Table 3.5-3 lists the 30 practices mined during the retrospective report analysis. Each practice has a unique and descriptive name.

Table 3.5-3: Practices mined during retrospective report analysis

Practice names			
Automate repetitive work	Debt management	Increase system knowledge	Short lines
Boundary involvement	Deliver or Delay	Incremental architecture	Simulator in the Loop
Clear plan	Design by team	Incremental improvement	System monitor
Clear specification	Document overview / interface	Keep it simple	Tester in team
Common plan	Duration runs	Knowledge transfer	Unit testing
Constructive disagreement	Early confrontation	Prioritize for focus	Work in parallel
Co-ownership	Empower the team	Product owner	
Customer centric development	Hardware in the Loop	Review deliverables	

The result of this retrospective report analysis is used in the next sections. The list of occurrences of *remarks* is used to determine the *failure category* coverage of the new patterns (section 4.3.3). The practices are used for writing patterns (section 3.6).

3.6 Writing patterns

3.6.1 Introduction

To answer the research question, organizational patterns need to be written. Such patterns will be integrated into existing pattern languages.

Patterns can be used as stand-alone solutions in order to solve small and local problem. When a pattern is applied, it usually relates to other patterns, and this can be expressed in a pattern language. Such language combines the patterns based on context, and places them in a certain sequence. Those organizations that want to improve their development process can choose their own path through the language. This way, the organization can establish its growth by considering its circumstances and progress.

The following sections describe the expected results, sources used, and results achieved.

3.6.2 Purpose

This section describes the expected results of writing the patterns, which is that several patterns are written according to the form layout of Figure 3.2-1. Another expectation is that these patterns are integrated into the pattern languages of (Coplien & Harrison, 2005).

To answer the research question (section 1.3), a pattern should specifically address the following subjects:

- Development of a mechatronic system
- Multidisciplinary development
- Concurrent engineering

A pattern should be able to describe the following items (Figure 3.2-1):

- Pattern name (Title)
- Context in which the problem is found
- Pattern forces or tradeoffs
- Solution
- Explanation of why the pattern works

The pattern should be based on one of the practices (Table D-1, Table G-1, and Table J-1). All the information retrieved in the previous steps (publications, FMEA worksheets, and retrospective reports) can be used as input for writing the patterns. Additional references can be added when they contribute to the pattern readability and clarity. Such references can be journal articles, proceedings, books, or websites. The book (Coplien & Harrison, 2005) should be used because the patterns need to be integrated in their pattern language. The pattern description should refer to their patterns

when applicable. Two examples on how a pattern description should appear are provided in Appendix P.

The pattern should be integrated into the pattern language (Coplien & Harrison, 2005), which should put the patterns into the context of other problems and solutions. This way, patterns can be more easily understood and successfully applied.

3.6.3 Sources

This section describes the sources used to write the patterns.

In the previous steps, the practices were mined, and these can be found in the following tables:

- Table D-1: Practices mined from literature
- Table G-1: Failure category overview based on FMEA worksheets
- Table J-1: Failure category overview based on retrospective reports

These tables are merged together, which results in an overview of 73 unique practices. These practices will be used for selecting practices.

Table 3.6-1: Practices found in the research

Practice names				
Abstract design	Co-ownership	Encourage risk taking	Key master	Shared leadership
Alert design change	Customer centric development	Establish a scoreboard	Knowledge integration	Short lines
Automate repetitive work	Debt management	Factory acceptance test	Knowledge transfer	Simulator in the Loop
Boundary involvement	Define dependencies	Hardware in the Loop	Learn by interaction	Single point of contact
Brainstorm on Design	Define roles in team	Hardware in the Loop / Iron Bird	Learning community	Small team
Budget design	Define up/downgrade strategy	Incorporate goals	Lessons learned	Splitter
Build commitment	Deliver or Delay	Increase system knowledge	Manage team member expectations	System integration testing
Central power	Design by team	Incremental architecture	Mock-up	System monitor
Clear plan	Design walk through	Incremental improvement	Module Testing	Team of experts
Clear priorities	Document overview / interface	Information flow	Mutual accountability	Test with dummies
Clear specification	Drop-In replacement	Integrator	Only design critical parts	Tester in team
Clear tasks	Duration runs	Involve key stakeholders	Prioritize for focus	Unit testing
Common ground	Early confrontation	Keep it simple	Product owner	Work in parallel
Common plan	Embed knowledge in system	Keep the schedule	Provide an incentive	
Constructive disagreement	Empower the team	Keep track of requirements	Review deliverables	

3.6.4 Result

This section describes the result of writing patterns, which is that the expected result (section 3.6.2) has been achieved. The results are:

- Three patterns are written according to the form layout of Figure 3.2-1. The result is presented in section 4.2.
- Patterns are integrated into a pattern language. The result is presented in section 4.2.2.

This result is achieved by selecting three practices out of the 73 practices (Table 3.6-1). These three practices specifically address the concurrent development of a mechatronic system in a multidisciplinary environment.

The context and forces are described based on the information retrieved in previous research steps. An example is provided in Table 3.6-2. A full overview of this table is available in Table K-1.

Table 3.6-2: Practice with problem statement: "How can…"

Practice	Context	Problem	Forces
Hardware in the Loop	Get as close to actual operation concept as possible to support verification and validation when operational environment is difficult or expensive to recreate (NASA, 2007, p. 96) Development of mechatronic systems requires collaboration among experts from different design domains (Alvarez Cabrera et al, 2010)	How can integration problems during end-of-project be prevented?	Fault detection and diagnoses can only be executed when final system is available (Boucher & Houlihan, 2008) Models and simulations do not exactly represent real system. Therefore, system verification cannot be executed (Alvarez Cabrera et al, 2010)
Simulator in the Loop	Provide insight into trends and tendencies of system and subsystem performance that might not otherwise be possible because of hardware limitations (NASA, 2007, p. 96) Early testing to identify problems at system level (Boucher & Houlihan, 2008)	How can integration start when not all disciplines have delivered their product part?	Resources (hardware) are scarce (Retrospective Analysis, section 3.5)

The description of Table K-1 (subset is shown in Table 3.6-2) is provided in detail according to the form layout of Figure 3.2-1. This results in the following three pattern descriptions:

- *Common Plan* (section 4.2.3)
- *Hardware in the Loop* (section 4.2.4)
- *Simulator in the Loop* (section 4.2.5)

The pattern descriptions contain sidebars on the left side that relate to the template subjects presented in Figure 3.2-1. This addition is to increase readability for individuals not familiar with organizational patterns. Pattern descriptions strongly relate to other patterns, and their names appear in *italic* font. A short description of all such patterns is available in Appendix O.

These patterns are integrated into the pattern language based on the unresolved forces of the solution provided to a pattern. The problem statement of the other patterns should address such unresolved force(s). In Table 3.6-3, two patterns are combined, thus forming a pattern sequence. The *Hardware in the Loop* pattern addresses the unresolved forces of the *Incremental Integration* pattern. See Table L-1 for all other sequences.

Table 3.6-3: Creating a pattern sequence based on unresolved forces

Hardware in the Loop	
Incremental Integration → Hardware in the Loop	
Solution	"Provide a mechanism to allow developers to build all of the current software periodically. Developers should be discouraged from maintaining long intervals between check-ins. Developers should at any time also be able to build against any of the Named Stable Bases or the newest check-in software."
Unresolved forces	When developing a mechatronic system, the Named Stable Basis can be seen as a mechatronic system (hardware and software). This is only available at the end of the project because hardware is still under development. That means that frequent integration is not possible.
Problem statement	"It is important to identify multidisciplinary integration problems early in the development cycle."

The result is that all three patterns are integrated into a pattern language. The new pattern sequence is presented in Figure 4.2-1. The pattern languages in which they are integrated are presented in Appendix P.

The result of this pattern writing is used in the next sections. The three patterns are evaluated in section 3.7. The integration of the patterns into pattern languages is used to place the patterns into context, which will be supportive during the pattern review (section 3.7).

3.7 Pattern evaluation

3.7.1 Introduction

Evaluation feedback on the patterns (section 4.2) is received from practitioners who work in the field of mechatronic systems and multidisciplinary projects. The feedback is used to determine how the patterns are received, and whether they are applicable in their organization. It is also used for suggestions on future research. The feedback of all reviewers is summarized and added as an Appendix. The extent to which the patterns cover the *failure categories* is also evaluated.

The following sections describe the expected and achieved results.

3.7.2 Purpose

This section describes the expected result of the evaluation, which is that feedback is received on the patterns by at least eight people. With this feedback, the quality of the pattern description can be determined and improvements can be considered. Another expected result is that the coverage of the *failure categories* can be determined. Such result should indicate the significance of the patterns to the *failure categories*.

This result can be achieved when the following data are produced:

- An overview (e.g., bar graph) that indicates the quality of each pattern description (given by reviewers)
- Improvement suggestions for each pattern (given by reviewers)
- A table that indicates how much (expressed in percentage) the patterns cover the *failure categories* mentioned in the publications, FMEA worksheets, and retrospective reports.

The reviewer evaluation consists of the three patterns (section 4.2), and the reviewer should comply with the following criteria:

- Familiar with concurrent development of mechatronic systems in a multidisciplinary project.
- Not a colleague within the current project (of the author) in order to prevent feedback from being influenced by project relationships, and to prevent a single perspective on the patterns.

The group of reviewers should represent a wide variety of roles. This way, feedback can be provided from different perspectives, which in turn, can make the feedback complementary to each other. The review is conducted anonymously, and the name of the reviewers should not be mentioned in this or any resulting documents in order to encourage straight feedback.

The patterns can be reviewed separately because they are stand-alone solutions. The assumption that the reviewers are familiar with the organizational patterns and the pattern language of (Coplien & Harrison, 2005) cannot be made. Therefore, additional information should be given to the reviewer. For this reason, the reviewer is not asked to review pattern integration into the pattern language. For the reviewers to provide good feedback, they need to thoroughly understand all the patterns and the language, which cannot be expected from this group of reviewers.

The reviewers should judge the quality of all the aspects of the pattern description, which are:

- Context (Prologue)
- Problem
- Forces/Trade-offs
- Solution
- Discussion on why pattern works

The reviewer should also be asked for additional comments on the patterns. This feedback can be used to improve the patterns in the future. The result should be presented in an accessible manner.

The significance of the patterns for solving *failure categories* should be determined.

3.7.3 Result

This section describes the result of the evaluation, which is that the expected result (section 3.7.2) has been achieved. The results are:

- An overview (e.g., bar graph) that indicates the quality of each pattern description (provided by reviewers). The result is presented in section 4.3.2.
- Improvement suggestions for each pattern (provided by reviewers). The result is presented in Appendix M.
- A table that indicates how much (expressed in percentage) the patterns cover the *failure categories* mentioned in the FMEA worksheets and retrospective reports. The result is presented in section 4.3.3.

This result is achieved by inviting people to review the patterns. Nine people provided feedback on the three patterns, all of whom work in different companies. Most reviewers have experience with multidisciplinary development of mechatronic systems and concurrent engineering. The experienced varied between eight to 30 years. One reviewer has no experience with mechatronic systems, but has over eight years of experience in multidisciplinary development of hardware and software. This experience is also recognized as relevant for reviewing the patterns. The experience of the reviewers was obtained in the disciplines of software, electronics, optics, and mechanics. The current occupations/roles they have are:

- Electronics engineer
- Hardware engineer
- Company owner (for software contractors)
- Software architect
- Software engineer
- Software project leader
- Teacher Software Engineering
- Test engineer

Almost none of the reviewers are familiar with organizational patterns. Approximately half of the reviewer group is familiar with pattern languages.

To support the reviewers in executing their task, two documents are created. The first document contains the patterns, pattern languages, and a description of what is expected from the reviewer. For those reviewers not familiar with organizational patterns and pattern language, some context is provided. The second document contains the three feedback forms (Table 3.7-1) used to obtain feedback on the patterns (one form per pattern).

Table 3.7-1: Feedback form used for pattern review

<Pattern Name>	
Review feedback on pattern	**Reviewer's feedback**
How long did the review take? (e.g., 15 min)	
Is the pattern description clear? *Rating 1...10 (1 is bad; 10 is excellent)*	
Context (Prologue)	
Problem	
Forces/Trade-offs	
Solution	
Discussion on why pattern works	
Pattern used	
Are there other forces/trade-offs? (Yes/No) (Please explain your reasoning and rational for your view)	
Do you think that the proposed solution will solve the problem? (Yes / No) (Please explain your reasoning and rational for your view)	
Reviewer comments	
Do you know another pattern name (alias) that also covers the pattern description?	
Additional feedback (if applicable)	

All feedback is summarized (Appendix M). The pattern quality is based on average ratings shown in a bar graph (Figure 4.3-1).

An investigation is conducted to determine which *failure categories* are actually covered by the patterns. Table G-1 and Table J-1 demonstrate the relationship between the *failure categories* and practices. The practices mined during the literature study (Table D-1) are not related to the *failure categories*. To obtain a complete overview of the coverage of the *failure categories* by the new patterns, a mapping is made (see Appendix N). The relationship between the *failure categories* and patterns found in the sources is presented in Table 3.7-2. One source letter expresses such relationship. For example, the *failure category* "Resource (hardware) are scarce" is covered by the patterns *Hardware in the Loop* and *Simulation in the Loop* found in the FMEA worksheets (F) and retrospective reports (R). The table is sorted by *failure category*.

Table 3.7-2: Failure categories related to the new patterns

Failure category	Common Plan	Hardware in the Loop	Simulator in the Loop
Integrated system behavior is not considered		PF	P
Production is not efficient		P	
Project intake is not managed	R		
Project plan is not managed	R		
Resources (hardware) are scarce		FR	R
Responsibility is not assigned	P		
Test coverage is too low		R	R

Legend:
P = Publications
F = FMEA worksheets
R = Retrospective reports

The coverage calculation overview is presented in Table 4.3-1. This result indicates the significance of the patterns to the *failure categories*.

The result of this evaluation is used in the next sections. The bar graph, textual/verbal feedback, and the coverage calculation are used to determine whether the research question was answered (section 4.3.3, Chapter 6). The review feedback can also be used to improve pattern descriptions (section 5.4.1).

3.8 Result

This section describes the result of the research, which is that the expected result (section 3.2) is achieved. The results are:

- Pattern description according to the form layout of Figure 3.2-1. The result is presented in section 4.2.
- Description of the patterns based on verifiable sources. These are publications (section 3.3.3), FMEA worksheets (section 3.4.3), and retrospective reports (section 3.5.3).
- Integration of the patterns into a pattern language. The result is presented in section 4.2.
- Pattern evaluation. The result is presented in section 4.3.

4 Results

4.1 Introduction

This chapter describes the results of the research described in chapter 3.

First, the pattern languages in which the patterns are integrated are presented (section 4.2.2). This integration is expressed with pattern sequences. Next, the three pattern descriptions are presented (sections 4.2.3 to 4.2.5). The patterns are written according to the template shown in Figure 3.2-1. Finally, the pattern evaluation is presented (section 4.3). Such evaluation describes the pattern's coverage on the *failure categories,* and the feedback from the reviewers. This result provides insight on the description quality and possible pattern applicability. Conclusions are presented in 4.2.6 and 4.3.4. A general conclusion on the patterns is available in section 4.4.

4.2 Patterns

4.2.1 Introduction

This section presents the patterns and how they are integrated into the pattern languages (Coplien & Harrison, 2005).

The pattern descriptions are the result of the writing process described in section 3.6. The three patterns are:

- *Common Plan*
- *Hardware in the Loop*
- *Simulator in the Loop*

To understand the relationship between the patterns, refer to section 4.2.2. The new patterns can be identified by gray squares (refer to Figure 4.2-1). The next sections contain the pattern descriptions. The last section concludes with the added value of the new patterns to the existing pattern language.

A short description of the referenced patterns can be found in Appendix O.

4.2.2 Pattern language

This section describes the pattern sequences constructed with the new patterns, and are integrated into two pattern languages.

A pattern language is a network of interrelated patterns that define a process for resolving problem development systematically. The language comprises the patterns and rules required to combine such patterns in meaningful ways and in a particular sequence. From the perspective of a pattern language, a pattern sequence represents a particular path through the language, and it describes how to create an integrated system (Buschmann, Henney & Schmidt, 2007).

The integration of the new patterns into the pattern language is based on the unresolved force(s) of the solution provided by a pattern. The problem statement of the other patterns should address the unresolved force(s). This integration places the patterns into the context of other problems and

solutions. This way, the new patterns can be more easily understood and applied successfully in the correct order.

The new pattern sequences are presented in Figure 4.2-1. This figure is a subset of the people and code (Figure P-1) and project management pattern languages (Figure P-2). The gray squares represent the new patterns. The patterns with the dashed line belong to the people and code pattern language. The pattern sequences are described below Figure 4.2-1. A detailed explanation on the integration of the new patterns into the language is provided in Appendix L.

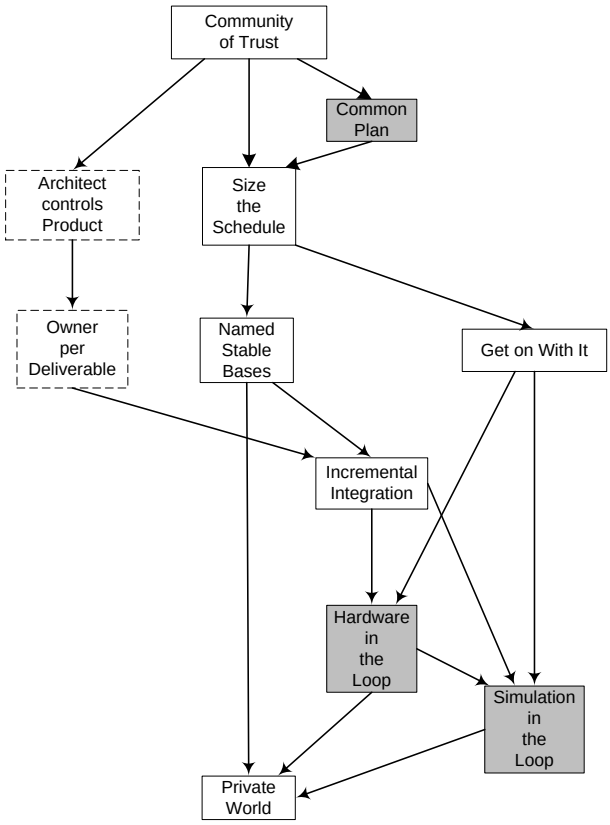


Figure 4.2-1: Pattern sequences as part of two pattern languages

The pattern sequence *Community of Trust* with *Common Plan* is constructed because deliverable orchestration is required in order to manage the schedule. The deliverables expected between the disciplines and their responsibilities need to be clear. Trust alone is insufficient for managing a project because agreements can be influenced by external (e.g., third-party deliverables) and internal factors (e.g., resources allocation). These factors can be triggered from outside the *Community of Trust*.

The pattern sequence *Common Plan* with *Size the Schedule* is constructed because overly ambitious and generous schedules are cumbersome for either the developers or the customers. A plan that considers all the deliverables between disciplines might result in an unrealistic schedule.

The pattern sequence *Incremental Integration* with *Hardware in the Loop* is constructed because identifying multidisciplinary integration problems early in the development cycle is important. The

pattern sequence *Incremental integration* with *Simulator in the Loop* is constructed because the ability to verify system behavior before integration on a real system begins is important. When developing a mechatronic system, the *Named Stable Bases* can be seen as a mechatronic system (hardware and software). This might only be available at the end of the project because hardware is still under development, which means that frequent integration is not possible as proposed in the solution for *Incremental Integration*. Another reason could be that it is too expensive to have a mechatronic system available for *Incremental Integration.*

The pattern sequence *Get On With It* with *Hardware in the Loop* is constructed because identifying multidisciplinary integration problems early in the development cycle is important. When developing a mechatronic system, high confidence on the software direction might exist. This software development is blocked when hardware that still has a low confidence is required. For example, the third-party Application Programming Interface (API) that needs to be integrated might be clear, but the hardware requirements for that third-party product might still be unclear.

The pattern sequence *Get On With It* with *Simulator in the Loop* is constructed because the ability to verify system behavior before integration on a real system begins is important. When developing a mechatronic system, high confidence on the software direction might exist. This software development is blocked when hardware that still has a low confidence is required. For example, the behavior that modules should have might be clear, but building a hardware setup might be too expensive.

The pattern sequence *Hardware in the Loop* with *Simulator in the Loop* is constructed because the ability to verify system behavior before integration on a real system begins is important. Multiple teams can develop a mechatronic system, and such teams can be on different locations and time zone; consequently, supporting these teams with hardware setups can become expensive.

The pattern sequence *Hardware in the Loop* / *Simulator in the Loop* with *Private World* is constructed because there is the desire for preventing developers from experiencing undue grief by having development dependencies change from underneath them. The unresolved force(s) of the solution provided by a pattern is:

- *Hardware in the Loop:* When developing with a multidisciplinary team, each discipline can have its own preferences on the setup. For example, the software discipline might want to test their latest features and fixes, whereas service engineering might want to verify features against a previous configuration (that can include other hardware).
- *Simulator in the Loop*: During development of a simulator, its behavior might change over time. For example, when a simulator is implemented incrementally, the initial behavior might only support happy flow. As the simulator matures, it can verify parameters, autonomously send events, or have a completely different start-up procedure.

4.2.3 Pattern: Common Plan

Title *(can be descriptive and sometimes evocative)*

Common Plan

alias: Integration Plan

A picture (should underscore the human dimension and should help to make the pattern memorable)

(Texan businessman WC bellow looking over schematics / Dimitri Kessel / Getty Images)

Prologue that describes the context in which the problem is found

The initial architecture is defined by the *Architecture Team*. During the concurrent development of the multidisciplinary project, there will be dependencies between disciplines. Each discipline might have its own strategy concerning how its deliverables are developed and released.

❖ ❖ ❖

Problem statement

Orchestration of deliverables is required to manage the schedule.

Discussion of the forces or tradeoffs of the pattern

The disciplines are strongly dependent on each other because their joint efforts will result in the final deliverable. If they are not coordinated, then good cross-disciplinary design decisions cannot be made because these decisions cannot be evaluated effectively (Graaf, Lormans & Toetenel, 2003; Heemels, van de Waal & Muller, 2006; Alvarez Cabrera et al., 2010). As a consequence, software is sometimes used for last-minute fixes to solve hardware problems due to the shorter lead time of these solutions (Graaf, Lormans & Toetenel, 2003; Alvarez Cabrera et al., 2010). However, this approach often leads to delays and errors (Bradley, 2010).

Each discipline can have its own development model (Schafer & Wehrheim, 2007). For instance, the Vee-model (Forsberg & Mooz, 1991) can be selected for development that involves long lead times. A methodology like Agile (Schwaber & Beedle, 2002) can be chosen when developers want to have short feedback loops. The use of different methodologies in a project can lead to conflicting interests.

Multidisciplinary development is sometimes characterized as the “throw it over the wall” approach (Schafer & Wehrheim, 2007). When a deadline is not met, the response of an individual team member is often: “Well, I got my work done on time” (Parker, 2003).

When a discipline makes a delivery, it needs to be clear about what the status is. As an example, the deliverable could be a proof of concept, partial delivery of a feature, or even an official release. The status of the delivery is of great importance to the other disciplines. They need to determine what the impact might be on their component of the delivery.

Conflicts among team members can arise when the responsibilities of delivery, intake and verification are not clearly defined (Nakata & Im, 2010).

Presentation of the solution
(sentence starts with "Therefore:")

Therefore:

Create a common plan of the deliverables that have dependencies between disciplines.

Based on the initial architecture, the dependencies between the disciplines are known. For each dependency, it should be made explicit who is involved. Responsibilities should be assigned for delivering, integrating and verifying the deliverables. The plan should not be expressed in excessive detail, as plans that are too detailed are difficult to manage.

When the product is understood and the project size has been estimated, a schedule can be created (*Size the Schedule*). The schedule should contain the dependencies between the disciplines and the milestones of the deliverables. The timing should be negotiated with the customer (*Engage Customers*).

The common plan should be agreed upon by all disciplines (*Unity of Purpose*). During execution of the plan, alignment between them is required. The team can align by arranging a multidisciplinary *Stand-up Meeting*.

The plan is owned by the person who is responsible for the project deliverables (*Owner per deliverable*).

A *Patron Role* can be assigned to the project when disciplines are involved in other projects. The Patron should resolve organizational conflicts.

❖ ❖ ❖

Discussion on why the pattern works

Creating a common plan and alignment of the dependencies enforces communication between disciplines (*Shaping Circulation Realms*).

Once the schedule has been agreed upon, each discipline can select the development model that fits their needs.

Conflicts within the team will be reduced when responsibilities concerning integration and testing deliverables are clear. Team members will not be wasting time on investigating problems because of untested deliverables of other disciplines. Verification by others is also highly inefficient because they lack the necessary expertise.

When the organization is not aligned with the product architecture, *Conway's Law* should be applied. If the team is geographically distributed, then *Organization follows Location* should be used.

Dependencies between disciplines can hamper development. This can be resolved by *Hardware in the Loop* or *Simulator in the Loop* strategies.

When the project team is working on a subsystem, a plan needs to be defined concerning how this will be integrated in the system and what the responsibilities of the disciplines are.

Optional descriptions

Examples:

Examples addressing how to create a common plan are network schedules (National Aeronautics and Space Administration, 2008) and Gantt charts (Wallace, 1922).

Concurrent engineering design activities at NASA involve a management or leadership team, a multidisciplinary engineering team, a stakeholder team, and a facility support team. In their experience of concurrent engineering, it needs to be clear what the level of maturity of the incoming design is, the stated goals and objectives of the engineering activity are, etc. (National Aeronautics and Space Administration, 2007). NASA states in their best practices on planning:

> "Planning: Proper planning and preparation are crucial for efficient CACE (*Capability for Accelerated Concurrent Engineering)* study execution. Customers wanting to forgo the necessary pre-study activity or planning and preparation must be aware of and accept the risk of a poor or less-than-desired outcome." (National Aeronautics and Space Administration, 2008).

A recommended action, in an organization that produces mechatronic systems, is to create a clear plan to prevent potential failures. Potential failures can include multiple versions of hardware needing to be released, lack of expertise and resources during a project phase, and project entanglement. The consequences can include project delays, integration problems, dependency on other projects being introduced, and unreleased products being shipped (see Table G-1).

Related patterns:

Developer controls Process orchestrates the activities of a given feature. This pattern does not describe how to coordinate the integration of multiple deliverables.

Holistic Diversity can be used to create a multidisciplinary team that is responsible for delivering a feature.

4.2.4 Pattern: Hardware in the Loop

Title *(can be descriptive and sometimes evocative)*

Hardware in the Loop

alias: Iron Bird

A picture (should underscore the human dimension and should help to make the pattern memorable)

(Velocipede on railroad track / John Hayford Album / NOAA's Historic Coast & Geodetic Survey (C&GS) Collection)

It is not a train but they are on the right track

Prologue that describes the context in which the problem is found

During concurrent development of a multidisciplinary project, hardware components of the system can be unavailable until late in the development cycle. Also, software is under development and may not yet be able to control a complete system. This dependency between hardware and software delays *Incremental Integration* and *Get on With It*.

❖ ❖ ❖

Problem statement

It is important to identify multidisciplinary integration problems early in the development cycle.

Discussion on the forces or tradeoffs of the pattern

There are several reasons why hardware may not be available. If the project team develops a new system, hardware may still be in development (Boucher & Houlihan, 2008). Another reason is that it may be too expensive to have it available for development and integration (Alvarez Cabrera et al., 2010).

Integration is a multidisciplinary collaboration. To be successful, each discipline should provide its deliverables, be available to share its knowledge, and collaborate with the other disciplines (Alvarez Cabrera et al., 2010).

During development, modifications of the hardware and software will be made. The modifications might be required because of non-compliance to the initial requirements or due to changing requirements. The current status of the project should be clearly stated.

Threats to the setup include people who want to borrow parts for their own testing purposes or who seek to use parts as a spare parts for manufacturing or servicing customer systems. Modifications for testing purposes (e.g. software patch, disconnected sensors) constitute another threat.

Integration of a setup does not include system integration.

Presentation of the solution (sentence starts with "Therefore:")

Therefore:

Build a hardware setup that can be used to verify the current state of the development.

The hardware setup does not need to have the exact specifications of the (sub-) system that is being developed (see *Get on With It*). The project managers should seek alternatives to cover their use cases and deal with limited resources. They can create a setup with alternative parts that are already available. This makes it possible to initiate testing before the final parts are available. To reduce the cost, they can also select cheaper parts.

Building and maintaining the setup requires the collaboration of all disciplines. This can be achieved by *Unity of Purpose*. When the participants from the different disciplines are located in different geographical regions, they should have a *Face to Face before Working Remotely*.

The setup requires an owner. This is because "Something that is everybody's responsibility is really no one's responsibility" (*Code Ownership*). The person who is assigned as owner should be the one who benefits from a representative setup. This could be the software engineer who wants *Incremental Integration* or *Get on With It*. It could also be the test engineers who can *Engage Quality Assurance*. The owner is responsible for keeping the setup operational. To remain aware of the current state of development, the owner should be informed of all design modifications. The owner can be informed by joining the *Stand-up Meeting*.

After the successful integration on a setup, system integration tests still need to be executed. These tests can be performed on a prototype (*Build Prototypes*) or on the final product. These tests can be executed by *Group Validation* and subsequently with the customer (*Engage Customer; Surrogate Customer*).

❖ ❖ ❖

Discussion of why the pattern works

Having a hardware setup allows disciplines to work in *Private Worlds*. The state of the setup can be frozen (*Named Stable Bases*). This allows developers to make progress (*Programming Episode*). Upgrades on the hardware or software can be planned and prepared for. This creates the opportunity for multidisciplinary *Incremental Integration*.

The setup allows the execution of tests to be initiated early in the development cycle (Boucher & Houlihan, 2008) (*Engage Quality Assurance*). For instance, team members can start executing runs on evenings or weekends to test stability and reproducibility.

Optional descriptions

Principle involved:

The costs of fixing problems increase when they are discovered late in the development cycle (Tassey, 2002). Solving integration issues late in the development cycle might also lead to poor quality of the system. This is the case when there is no time remaining to solve hardware problems. In contrast, correcting problems in the software seems relatively easy (Alvarez Cabrera et al., 2010; Bradley, 2010).If the problem is not solved at its source the solution might not be optimal. Furthermore, the software needs to maintain the solution for the complete lifetime of the hardware.

Related pattern:

Building and maintaining the hardware setup by the team enforces communication between disciplines (*Shaping Circulation Realms*).

Example:

A manager describes his experience with *Hardware in the Loop* testing:

> "Going to HIL testing was a natural outgrowth of our concurrent development strategy and our efforts to model devices. The ability to simulate the system and test software has greatly increased first-time software quality and allowed developers to have much more immediate and meaningful feedback (by Senior VP Technical Services, Industrial Equipment Manager)" (Boucher & Houlihan, 2008).

In an organization that produces mechatronic systems, having no *Hardware in the Loop* is mentioned as a potential failure point for the projects. The consequences can include insufficient hours for Mean Time Between Failure (MTBF) measurement, integration problems, and discovering issues late in the development cycle (e.g. during production or in the field) (see Table G-1).

Reading:

The use of *Hardware in the Loop* (HIL or HWIL) is presented in (National Aeronautics and Space Administration, 2007; Boucher & Houlihan, 2008; Bradley, 2010).

The use of *Iron Bird* is described in (Alvarez Cabrera et al., 2010)

4.2.5 Pattern: Simulator in the Loop

Title *(can be descriptive and sometimes evocative)*

Simulator in the Loop

A picture (should underscore the human dimension and should help to make the pattern memorable)

(Marcel Marceau, 1923 – 2007 / Ahmad Kavousian / Getty Images)

Mime player who creates the suggestion of a real world.

Prologue that describes the context in which the problem is found

The use cases of a mechatronic system are defined. The current development state cannot be verified because there is no system available. A reason for this might be that it is too expensive to have multiple systems available to support all development teams, which might have different locations and be situated in different time zones. This dependency delays *Incremental Integration* and *Get on With It*.

❖ ❖ ❖

Problem statement

It is important that system behavior can be verified before integration on a real system begins.

Discussion on the forces or tradeoffs of the pattern

To be able to verify system behavior, simulating (parts of) its behavior must be possible.

System behavior can be verified by using simulators. Development of these simulators is an investment.

The use of a simulator can introduce constraints. Some use cases cannot be supported. In these cases, a different flow in the code might be required to deal with this omission. Another constraint could be the timing of the behavior.

System-level simulation can be achieved when all simulators of the subsystems work together. Hybrid system-level simulation is possible when simulators work in combination with system.

Presentation of the solution (sentence starts with "Therefore:")

Therefore:

Build simulator(s) that can simulate (sub-) system behavior.

A simulator is a product. It should therefore be developed like any other product that is being developed by the organization (National Aeronautics and Space Administration, 2007). The simulator can be a software simulator or a hardware simulator.

The requirements should make clear which behavior is to be simulated, which architectural constraints are to be applied, what the performance should be, and other factors (*Surrogate Customer, Architect controls Product*). A software simulator can be a stub with limited intelligence or a true representation of system behavior. The architect chooses what type to use based on the requirements and the budget. When the complexity of the system increases, the more benefits will be gained by using representative simulators.

During the design process, it should become clear how to deal with deviations between the simulator and the real world. Preferably these are kept to a minimum. Integration of the simulator into the product should be addressed, as well. As an example: It should be clear how a simulated subsystem can work as part of a whole. That could necessitate that some parts of the system are simulated, while other parts should contain real subsystems. During the design process, other disciplines might get involved to provide their input on (sub-) system behavior.

After implementation (*Programming Episode*), the simulator needs to be tested (*Engage Quality Assurance*).

A simulator can be developed iteratively (*Incremental Integration*), and ownership of the delivery should be assigned (*Owner per Deliverable*).

❖ ❖ ❖

Discussion of why the pattern works

Having simulator(s) creates the opportunity to work in *Private Worlds*. This allows developers and testers to make progress (*Programming Episode; Application Design is Bounded by Test Design*) in an isolated environment. Early in the development cycle, behavioral requirements can be validated (Boucher & Houlihan, 2008) by QA (*Engage Quality Assurance*) and/or customer feedback (*Engage Customers*). This validation can be performed as a standalone process. This early feedback can be used to improve the design before it is verified on a prototype (*Build Prototypes*) or real system. This can also prevent material from being damaged.

From the development perspective, simulators have several advantages in comparison with real systems. First, a real system needs to become operational. This might require training of the developers or support from an expert. Next, real systems have to manage many variables (environment, mechanical tolerances, electronic signals, etc.). With all of these variables, it can be difficult to reproduce behavior in a specific use case. That can make it difficult to troubleshoot problems and to identify the root cause. Finally, real systems are typically expensive. As a consequence, they are scarce and may need to be shared among different groups. That means that only limited time may be available to validate the implementation.

Simulators can be used to increase development speed. Use cases can be verified much more quickly when the simulation speed can be increased. The developer can quickly establish preconditions before testing his or her own implementation. With a simulator it can also be much easier to trigger and verify extreme use cases, like error handling of error situations that occur seldom in real life environment.

Optional descriptions

Example:

Recommendation for companies to become more effective in developing mechatronic products is given by (Boucher & Houlihan, 2008):

> "Identify system level problems early in the design process by leveraging simulation to validate system behavior."

Related pattern:

Simulators can be used to maintain code quality. In a simulated environment, automated tests can be executed. The results of the automated test should be reproducible. The tests support the creation of *Named Stable Bases*.

Reading:

The use of simulators is discussed in (National Aeronautics and Space Administration, 2007; Boucher & Houlihan, 2008).

4.2.6 Conclusion

This section describes the added value of the new patterns to the existing pattern language.

The pattern *Common Plan* allows the orchestration of deliverables from different disciplines required to manage the development schedule of the mechatronic system. With the current pattern language, the assumption is that all activities are performed by one group of developers. When developing a mechatronic system, this is not always the case. The development is influenced by external (e.g., third-party deliverables) and internal factors (e.g., resources allocation).

The pattern *Hardware in the Loop* and *Simulator in the Loop* makes the identification of multidisciplinary integration problems early in the development cycle possible. With the current pattern language, frequent integration is not possible because the mechatronic system (hardware and software) is only available at the end of the project. Starting with software development is not possible because such development is blocked when hardware that still has a low confidence is required on the project direction and requirements.

The pattern *Simulator in the Loop* makes verification of the system behavior before integration on a real system begins possible. The pattern allows the development and testing by multiple teams on different locations and time zones without the expense of hardware setups. It also allows large test coverage because testing different releases and configurations can be done relatively easy (in comparison to hardware setup).

4.3 Pattern evaluation

4.3.1 Introduction

An evaluation is executed (section 3.7) to obtain insight on the coverage of *failure categories*, quality, and pattern applicability.

Quality and applicability are determined based on reviewer feedback (section 4.3.2). Pattern coverage of the *failure categories* is determined based on the occurrence of *failure categories* in FMEA worksheets and retrospective reports (section 4.3.3). A conclusion is drawn based on these results (section 4.3.3).

4.3.2 Review feedback on the patterns

This section describes the results of the reviewer feedback on the patterns (section 3.7.3).

Nine people provided feedback on the three patterns (section 4.2). On average, 20 minutes were required to review each pattern. All patterns were received positively. The feedback is summarized in Appendix M. As indicated above, the reviewers were positive on the presented patterns.

On average, the patterns received the following ratings (*Rating 1...10; 1 is bad and 10 is excellent*):

- *Common Plan* 7.6
- *Hardware in the Loop* 7.8
- *Simulator in the Loop* 7.5

Figure 4.3-1 shows the average rating in more detail. The question presented to the reviewers was (Table 3.7-1): *Is the pattern description clear?*

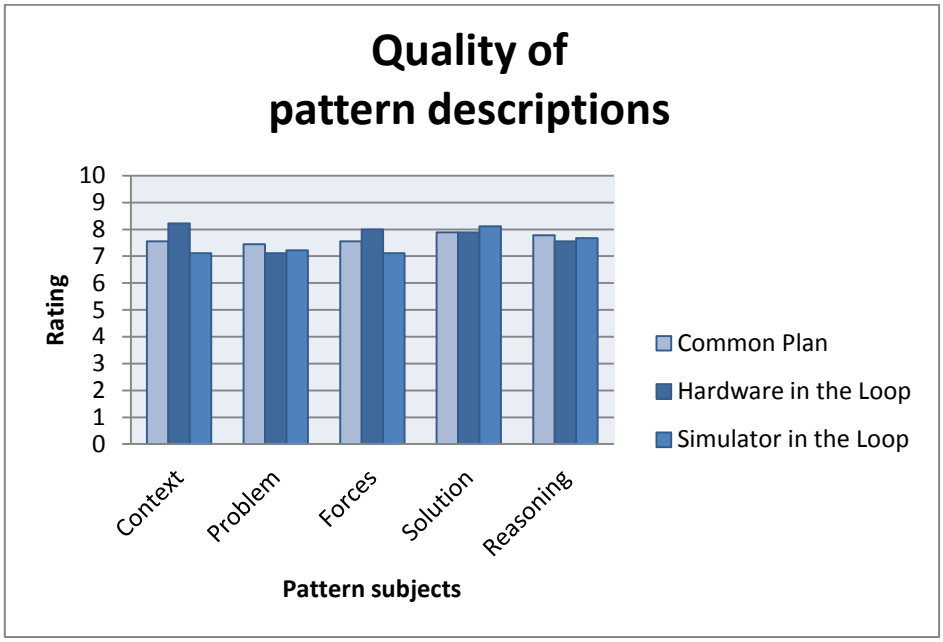


Figure 4.3-1: Review feedback: Is pattern description clear?

There was no real consensus on the rating. For example, the problem description for the *Simulator in the Loop* was rated with a four by one reviewer and nine by another. To quantify the amount of variation, a standard deviation overview is provided by Figure 4.3-2.

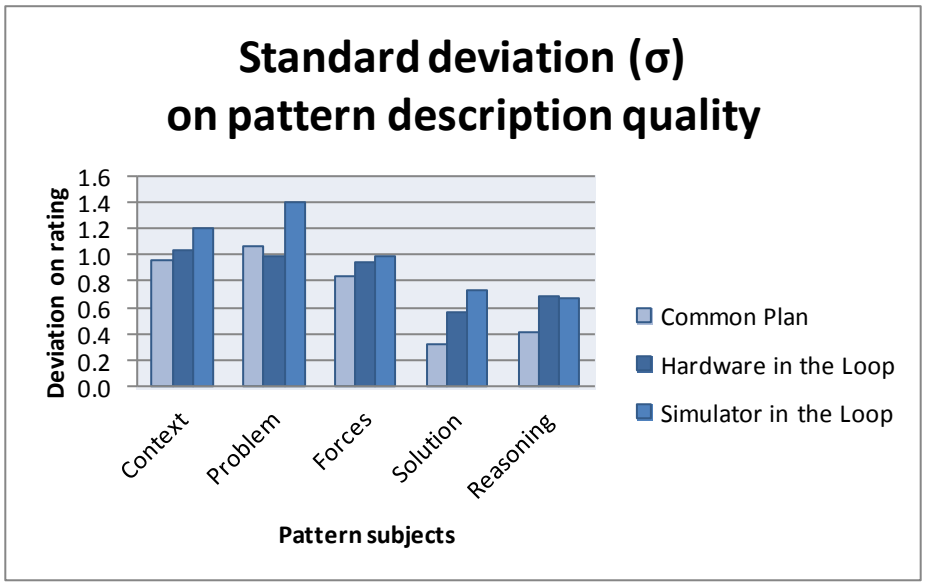


Figure 4.3-2: Standard deviation for pattern rating

No causal relationship could be made between the ratings provided by the reviewers and their role, experience, or organization for which they worked. In Figure 4.3-3, an attempt is made to reduce the standard deviation for the pattern *Simulator in the Loop*.

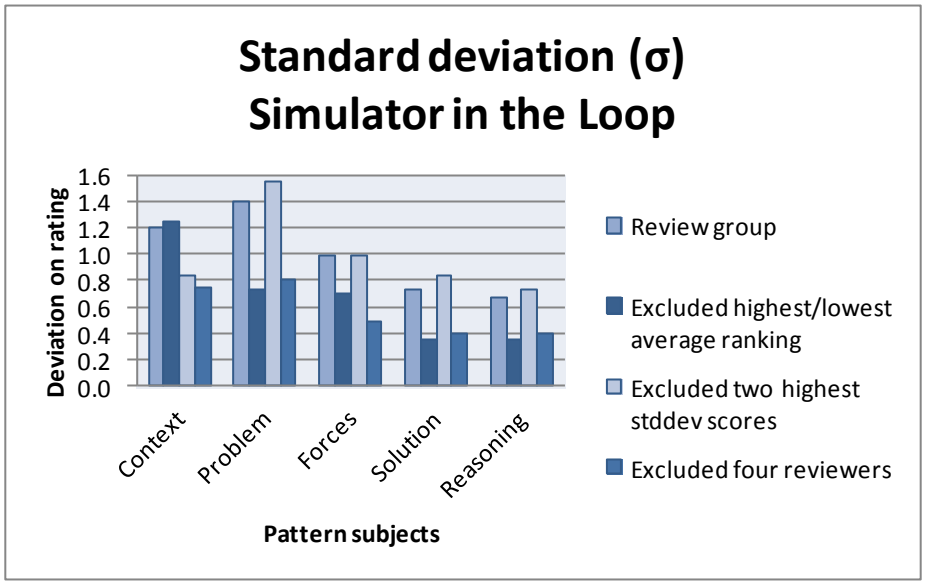


Figure 4.3-3: Standard deviation for Simulator in the Loop

In Figure 4.3-3, the "Review group" refers to the rating results from Figure 4.3-2 provided by the reviewers. In order to reduce the standard deviation, the reviewers with the highest (Hardware engineer) and lowest (Test engineer) average rating are excluded. This increases the standard deviation of the C*ontext* slightly. All other subjects are reduced. In order to reduce the standard deviation for *Context*, the two reviewers with the highest standard deviation (Software architect and Software project leader) are excluded. This reduces the standard deviation of *Context*, but increases

that of *Problem*, *Solution*, and *Reasoning*. When excluding the four reviewers, all the subjects have a reduced standard deviation. The excluded reviewers do not have a specific relationship to each other, or to the reviewers who are not excluded. By excluding these reviewers, the total reviewer group is reduced by almost half. The remaining group does not comply with the requirements set in section 3.7.2; therefore, the result is not representative.

The feedback on *Common Plan* is that it could also be influenced by standards, policies, and regulatory requirements. Another remark is that more aspects of Scrum Agile could be used, such as planning poker and backlog meeting.

Most of the feedback on the *Hardware in the Loop* is related to the pattern *Build Prototypes* (Appendix O). There is a relationship between these two patterns, although they serve different purposes. Such difference is that *Hardware in the Loop* has more focus on incremental (multidisciplinary) development, integration, and multidisciplinary collaboration, whereas the main purpose of *Build Prototype* is to gain knowledge. Such difference is not explained in the pattern.

Most of the feedback on the *Simulator in the Loop* relates to costs. The pattern description does not address this issue as a force or tradeoff.

4.3.3 Coverage of failure categories

This section describes the coverage of *failure categories* by the patterns (section 3.7.3).

To obtain insight on the coverage of the *failure categories* by the patterns, an overview is created (Table 4.3-1) that is a subset of Table H-1. Only the *failure categories* that relate (Table 3.7-2) to the patterns (section 4.2) are demonstrated. The table is sorted on severity ranking. These data relate only to the organization that made the field data available.

Table 4.3-1: Coverage of failure categories by the new patterns

FMEA: Failure categories	FMEA: Average severity ranking	FMEA: Occurrences of failure category		Retrospective: Positive remarks		Retrospective: Negative remarks		Retrospective: Improvements	
Integrated system behavior is not considered	6.2	153	12.9%	7	1.9%	23	4.1%	1	1.1%
Test coverage is too low	6.1	10	0.8%	29	7.9%	54	9.6%	8	8.9%
Project plan is not managed	5.8	24	2.0%	50	13.6%	79	14.0%	18	20.0%
Responsibility is not assigned	5.8	9	0.8%	2	0.5%	21	3.7%	1	1.1%
Resources (hardware) are scarce	5.1	14	1.2%	10	2.7%	40	7.1%	7	7.8%
Project intake is not managed	4.8	23	1.9%	7	1.9%	23	4.1%	4	4.4%
Production is not efficient	4.4	24	2.0%	108	29.3%	151	26.8%	25	27.8%
Total	5.5	257	21.6%	213	57.8%	391	57.8%	64	71.1%

The average severity ranking of this subset (5.5) is almost equal to the average of all *failure categories*, which is 5.4 (Table H-1). This ranking means that the patterns, on average, solve *failure categories* with medium impact (Table 3.4-1). The severity of the effect is “Minor tool/equipment

damage and system specification compromised." The coverage of FMEA failures is slightly above 20%. Because a pattern only covers aspects of a *failure category*, the conclusion is that the introduction of these patterns does not reduce the *potential failures modes* of a project significantly. The coverage of the retrospective remarks is above 57%. Although a pattern only covers aspects of a *failure category*, the conclusion is that the introduction of these patterns can help to solve the negative remarks (coverage of 69%) and provide support for the improvement actions (coverage of 71%).

4.3.4 Conclusion

This section describes the conclusion based on the results.

The reviewers were positive on the presented patterns. Almost everyone recognized the patterns and confirmed that they have seen them being used. Based on the rating (Figure 4.3-1) and textual/verbal feedback (Appendix M), it is concluded that the patterns can be used for the development and integration of a mechatronic system in a multidisciplinary environment developed concurrently.

The conclusion is that the patterns cover aspects of field problems (Table 4.3-1). Introduction of these patterns can help the organization that made the field data available to *improve production efficiency, manage the project plan*, and *increase the test coverage*.

4.4 Conclusion

This section describes how the multidisciplinary development problems of mechatronic systems are addressed with the new patterns presented in this chapter.

The conclusion of this chapter is that the patterns address the problems mentioned in section 1.3. In this section, the following problems were reported:

- Collaboration between disciplines
- Integration of deliverables
- Dependency between disciplines

The three patterns presented in this chapter address these problems (see also 4.2.6) as follows:

- Orchestration of deliverables from different disciplines
- Identification of multidisciplinary integration problems early in the development cycle
- Verification of system behavior before integration on a real system begins

The *Common Plan* patterns support collaboration, integration, and independency. Creating a common plan and alignment of the dependencies enforces communication between disciplines. Once the schedule has been agreed, each discipline can select the development model that fits their needs. Conflicts within the team are reduced when the responsibilities that concern integration and deliverable testing are clear.

The *Hardware in the Loop* pattern supports collaboration, integration, and independency. Building and maintaining a hardware setup requires the collaboration of all disciplines. Having a hardware

setup allows disciplines to work independently. The state of the setup can be frozen. This allows developers to make progress. Upgrades on the hardware or software can be planned and prepared. The setup allows the execution of tests to be initiated early in the development cycle. This supports early deliverable integration.

The pattern *Simulator in the Loop* supports collaboration, integration, and independency. Having simulator(s) creates the opportunity to work independently, which allows developers and testers to make progress. Simulators have several advantages in comparison with real systems, and such advantages support deliverable integration. Early in the development cycle, behavioral requirements can be validated. During the simulator design process, other disciplines might become involved in order to provide their input on (sub-)system behavior. This allows multidisciplinary collaboration.

The quality of the pattern descriptions is sufficient for practitioners to understand all pattern aspects, which are context, problem, forces, solution, and a discussion on why pattern works.

The patterns proposed a solution for the problems faced by the organization that made the field data available.

5 Discussion

5.1 Patterns

5.1.1 Commonalities between patterns

The patterns *Build Prototype*, *Hardware in the Loop*, and *Simulator in the Loop* have commonalities: they all allow early implementation, confrontation, and incremental development. The resolving of forces justifies the need for *Simulator in the Loop*:

- Multiplication can be done at low cost. This way, all developers can have access to a test environment that can be supportive of the development of large systems, multisite development, and automated testing.
- Behavioral requirements at the system level can be verified before a system is available.
- The use of a simulator does not require a system expert for operation.
- Problems are easier to reproduce because there are fewer variables (e.g., environment, mechanical tolerance).
- Easy to verify extreme use cases that occur seldom in the real-life environment.

Hardware in the Loop strongly relates to *Build Prototype*. The difference is that *Hardware in the Loop* has more focus on incremental (multidisciplinary) development, integration, and multidisciplinary collaboration, whereas the main purpose of *Build Prototype* is to gain knowledge. Another difference is that hardware setup evolves over time, and therefore, it represents the latest state of development. This setup does not need to be discarded; it can be used continuously for development and testing.

5.1.2 General pattern applicability

The pattern *Common Plan* resolves dependencies between disciplines, unassigned responsibilities, and misinterpretations on design. The pattern might also be applicable outside the mechatronic development domain. An indication of this is that the pattern covers general process development issues based on Table J-1 and Table N-1:

- Production is not efficient
- Project intake is not managed
- Project plan is not managed
- Responsibility is not assigned

Discussion with other people outside the mechatronic development domain is required to obtain confirmation of this statement.

5.1.3 Pattern language

The patterns are integrated into a language based on software development. Several patterns, such as *Parser Builder*, *Code Ownership*, and *Programming Episode* (Appendix O), strongly refer to this. Most patterns are more generally applicable. For example, the *Named Stable Bases* describes stabilizing the system architecture and provides a name that can be used to identify that version. This pattern can also be applied to hardware. The hardware engineers create stable bases with functional

models and Alpha and Beta versions that can have subversion when minor modifications are applied. Another example is the *Architect Controls Product*. Within the development of a mechatronic system, a system engineer fulfills this role, and the *Architecture Team* with representatives from all disciplines support the system engineer. Because most patterns can also be applied to mechatronic development, the new patterns can be seen as language extensions.

5.2 Evaluation

5.2.1 Coverage

The evaluation shows the coverage of the patterns on the *failure categories* (Table 4.3-1), and such coverage is based on the field data of one organization. Therefore, this result cannot be extrapolated. Coverage within another organization that has, for example, a higher maturity level (e.g., Level 5 of the Capability Maturity Model (Humphrey, 1989)), could have a different outcome.

The evaluation also shows that the patterns do not cover the top most *failure categories* of the FMEA worksheets (Table 4.3-1). During this research, practices were mined that cover these top most *failure categories*, for example, *System integration/Module testing, Review deliverables, and Boundary involvement*. However, some of the practices were dropped mainly because they were not specific for multidisciplinary development. Therefore, no pattern was written to cover these *failure categories*. If the research question were formulated differently, they might be selected.

5.2.2 Review

The intention for the review is to see how practitioners receive the patterns, and obtain feedback on how they can be improved.

The group of reviewers consisted of nine individuals. For a first review, this group was sufficiently large because most of the comments had overlap. A significant spread on the description rating was determined; however, no explanation could be found for such spread. If more individuals were added to the group, a causal relationship might become apparent. Another approach would be to hold a discussion with all reviewers with regard to their ratings. This should provide an explanation for the rating, as well as more input for description improvements. Such an initiative was not undertaken because of the time constraint for this thesis.

5.3 Research approach

5.3.1 Influence of personal knowledge

This research is influenced by the author's personal knowledge and perspective, given that the author has worked for more than 15 years in the development of mechatronic systems. In those years, the author has executed different roles, such as developer, architect, team leader, and project leader, all of which were performed within the software discipline. As a contractor, the author has performed these roles in nine different companies active in different industries and of different size. Some of such companies have a maximum of 50 employees, and others over 4,000.

This personal knowledge has influenced:

- Definition of the practices (Table D-1, Table G-1, Table J-1)
- Definition and assignment of *failure categories* (Appendix F)

Such influence can be prevented if the research question were answered with a different approach. An option is to present the problem statement to the pattern community (EuroPLoP, 2015; ScrumPLoP, 2015) and mine patterns through discussion. However, such approach does not qualify for academic research because it is not an independent research that demonstrates research skills. It is also difficult to reproduce the discussion results because the result depends on the knowledge of the present participants.

The definition of practices and pattern writing (section 3.6) are performed based on common sense. This process can be done according to a pattern language for writing patterns (Meszaros, Doble, Martin, Riehle & Buschmann, 1997). However, for this thesis, this is a missed opportunity because the author learned of the existence of such language after writing the patterns.

5.4 Recommendations and future work

5.4.1 Pattern description

The pattern descriptions (section 4.2) are solely based on this research, and they can be improved by incorporating feedback from the reviewers. Another method is to brainstorm the pattern descriptions during a writer's workshop (ScrumPLoP, 2015; EuroPLoP, 2015). This is a platform to improve patterns through group exposure, and it can lead to new insights to refine the description. Such platform also allows the possibility of publication (e.g., Springer journal LNCS Transactions on Pattern Languages of Programming). When the patterns are published, they will become more visible to the public, which should increase the social relevance of this thesis.

Exposure of the *pattern description* to the writer's workshop does not contradict the statement made in 5.3.1, which is that *mining* practices through discussion does not qualify for academic research. The objective of the exposure of the *pattern description* to the writer's workshop is to refine it in order to allow the possibility of publication.

5.4.2 Pattern language

During the literature study, publications were found that describe development by multidisciplinary teams (Michalski, 1998; Parker, 2003). No publication was found of a pattern language that describes multidisciplinary development. The existence of such a language can contribute to the process improvements of a multidisciplinary team. The existing patterns (Coplien & Harrison, 2005) used for the new patterns can be considered when such a language is constructed. These patterns are based on section 4.2 and listed in Table 5.4-1. A short description of these patterns can be found in Appendix O.

Table 5.4-1: Existing patterns referred in new patterns

Pattern names			
Application Design Is Bounded By Test Design	Engage Customer	Named Stable Bases	Size the Schedule
Architect Controls Product	Engage Quality Assurance	Organization Follows Location	Stand-up Meeting
Architecture Team	Face to Face Before Working Remotely	Owner per Deliverable	Surrogate Customer
Build Prototypes	Get on with It	Patron Role	Unity of Purpose
Code Ownership	Group Validation	Private Worlds	
Conway's Law	Holistic Diversity	Programming Episode	
Developer Controls Process	Incremental Integration	Shaping Circulation Realms	

5.4.3 Solution for top most failure categories

No solution is provided for the top most *failure categories* of the FMEA worksheets. Research can be conducted to mine practices that address these categories. One recommendation is to execute a literature study first. Then, the individuals actively involved in the development of mechatronic systems can be interviewed. Next, the patterns can be written not as a *Solo Virtuoso*, but by *Developing in Pairs* (Table O-1). When the patterns are mature, they can be presented at a writers' workshop (EuroPLoP, 2015; ScrumPLoP, 2015).

5.4.4 Mining different sources

The field data used are FMEA worksheets and retrospective reports. There are other organizations that have also archived their development data and made them accessible to the public. An example of this is NASA's Lessons Learned (NASA - Lessons learned, 2015). These data can be used as source for mining practices.

5.4.5 Relationship between standards and language

When a business wants to improve its development practices, it can apply international standards (e.g., ISO/IEC, 2008) or achieve a higher level in the Capability Maturity Model (Humphrey, 1989). Another approach is to apply a development pattern language.

Research can be conducted to determine how the pattern languages of (Coplien & Harrison, 2005) relate to international standards (e.g., ISO/IEC, 2008). The possibility of formalizing the use of pattern languages, similar to international standards, can also be investigated. By executing an audit in an organization, the level to which such organization complies with a language can be determined. Such an audit can result in improvement advice, which in turn, can improve the processes within an organization and popularize the use of pattern languages.

6 Conclusion

The conclusion is that the research question was answered positively. The research question is as follows:

> *"Is it possible to formulate organizational patterns that can be used for the development and integration of a mechatronic system in a multidisciplinary environment developed concurrently?"*

The problem statement was described in section 1.3, and the following problems were reported:

- Collaboration between disciplines
- Integration of deliverables
- Dependency between disciplines

Such problems lead to project delays and integration issues. These problems were addressed with the new patterns (section 4.4). Project delays can be prevented because a *Common Plan* (section 4.2.3) orchestrates the deliverables from different disciplines. This *Common Plan* manages the development schedule of the mechatronic system. In addition, conflicts within the team can be reduced because the responsibilities that concern integration and testing deliverables are determined. *Hardware in the Loop* (section 4.2.4) and *Simulator in the Loop* (section 4.2.5) allow the possibility of disciplines to work independently, which permits concurrent engineering, and hence reduces project delays. A project can also be delayed because of integration issues, which can be reduced by *Hardware in the Loop* and *Simulator in the Loop*. These patterns allow early integration.

It is the first time that these solutions are formalized and presented as an organizational pattern and integrated into the organizational pattern language of (Coplien & Harrison, 2005). This result can benefit a multidisciplinary team that develops a mechatronic system concurrently. It will make them aware of: a solution to their problem, when to apply it, the forces and trade-offs of the solution, and how the solution can be implemented. These patterns will empower the multidisciplinary team to solve the stated problems.

This result was achieved by mining practices in publications (section 3.3) and in field data that consist of FMEA worksheets (section 3.4) and retrospective reports (section 3.5). Based on several criteria, three practices were selected to be written as a pattern (section 3.6) according to a form layout (Figure 3.2-1). For reading clarity, sidebars were introduced. The reviewers explicitly mentioned such sidebars as being very supportive to understanding the patterns. The patterns were integrated into two pattern languages (section 4.2.2): People and code, and Project management. With this integration, the new patterns can be understood more easily and applied successfully in the correct order.

The fact that the patterns cover aspects of the following *failure categories* within the organization that made the field data available (section 4.3.3) was determined:

- Integrated system behavior is not considered
- Production is not efficient
- Project plan is not managed
- Resources (hardware) are scarce
- Responsibility is not assigned
- Test coverage is too low

The introduction of these patterns can help the organization at least to *improve production efficiency*, *manage the project plan*, and increase *test coverage.*

Individuals who work in the field of mechatronic systems and multidisciplinary projects (section 4.3.2) reviewed the patterns. Such reviewers were positive on the presented patterns. Almost everyone recognized the patterns and confirmed that they have seen them being used. All the patterns rated 7.5 or higher (1 is bad and 10 is excellent). Based on this rating and textual/verbal feedback (Appendix M), it is concluded that the patterns can be used for the development and integration of a mechatronic system in a multidisciplinary environment developed concurrently.

Recommendations for future work (section 5.4) are as follows:

- Improve pattern description
- Construct dedicated pattern language for developing a mechatronic system
- Write patterns for top most *failure categories*
- Mine different sources to retrieve practices
- Investigate the relationship between standards and pattern language

7 Reference

Alvarez Cabrera, A., Foeken, M., Tekin, O., Woestenenk, K., Erden, M., & De Schutter, B., et al. (2010). Towards automation of control software: A review of challenges in mechatronic design. *Mechatronics, 20*(8), 876-886. doi:10.1016/j.mechatronics.2010.05.003

Armoush, A., Salewski, F., & Kowalewski, S. (2008). *Recovery block with backup voting: A new pattern with extended representation for safety critical embedded systems.* Paper presented at the International Conference on Information Technology (ICIT), (pp. 232-237), Bhubaneswar, India.

Beckers, J., Muller, G., Heemels, W., & Bukkens, B. (2007). *Effective industrial modeling for high-tech systems: The example of happy flow.* Paper presented at the International Council on System Engineering (INCOSE), San Diego, USA.

Bonnema, G., Borches, P., & Houten, F. (2010). *Communication: Key factor in multidisciplinary system design.* Paper presented at the 8th Conference on Systems Engineering Research (CSER), Hoboken, NJ, USA.

Boucher, M., & Houlihan, D. (2008). *System design new products development for mechatronics.* Boston, MA, USA: Aberdeen Group.

Bradley, D. (2010). Mechatronics – More questions than answers. *Mechatronics, 20*(8), 827-841. doi:10.1016/j.mechatronics.2010.07.011

Buschmann, F., Henney, K., & Schmidt, D. (2007). *Pattern-oriented software architecture, on patterns and pattern languages.* UK: John Wiley & Sons.

Chandler, G., Denson, W. K., Rossi, M. J., & Wanner, R. (1991). *Failure mode/mechanism distribution (ADA259655).* USA: Reliability Analysis Centre (RAC).

Chen, D. J., & Torngren, M. (2001). *Towards a framework for architecting mechatronics software systems.* Paper presented at the 7th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), (pp. 170-179), Skvde, Sweden.

Christopher, A., Ishikawa, S., & Silverstein, M. (1977). *A pattern language.* Oxford University Press.

Colorado State University (2012). *Definitions of "Mechatronics."* Retrieved 5 Feb 2015, from http://mechatronics.colostate.edu/definitions.html

Coplien, J. O., & Harrison, N. (2005). *Organizational patterns of agile software development.* USA: Prentice Hall.

Department of defense (1980). *Military standard: Procedures for performing a Failure Mode, Effects and Criticality Analysis (MIL-STD-1629A).* Washington, DC, USA: Department of defense.

Derby, E., & Larsen, D. (2006). *Agile retrospectives.* USA: Pragmatic Bookshelf.

Eppinger, S., & Salminen, V. (2001). *Patterns of product development interaction.* Paper presented at the International Conference on Engineering Design (ICED), Glasgow, UK.

EuroPloP (2015). *European conference on patterns and pattern languages*. Retrieved 21 Feb 2015, from http://www.europlop.net/

Fantuzzi, C., Bonfe, M., Secchi, C., e Reggio, U., & Emilia, D. (2009). *A design pattern for model based software development for automatic machinery.* Paper presented at the 13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM), Moscow, Russia.

Forsberg, K., & Mooz, H. (1991). *The relationship of systems engineering to the project cycle*. Paper presented at the First Annual Symposium of the National Council On Systems Engineering (NCOSE), Chattanooga, TN, USA.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns.* USA: Pearson Education.

Garro, R., Ordinez, L., & Alimenti, O. (2011). *Design patterns for cyber-physical systems: The case of a robotic greenhouse.* Paper presented at the Brazilian Symposium on Computing System Engineering (SBESC), (pp. 15-20), Florianopolis, Brazil.

Graaf, B., Lormans, M., & Toetenel, H. (2003). Embedded software engineering: The state of the practice. *Software, IEEE, 20*(6), 61-69.

Haapanen, P., & Helminen, A. (2002). *Failure mode and effects analysis of software-based automation systems.* Helsinki, Finland: STUK.

Heemels, W., van de Waal, E., & Muller, G. (2006). *A multi-disciplinary and model-based design methodology for high-tech systems.* Paper presented at the Conference on Systems Engineering Research (CSER), Los Angeles, USA.

Humphrey, W. S. (1989). *Managing the software process*. Boston, MA, USA: Addison-Wesley Professional.

ISO/IEC (2008). *ISO/IEC 15288:2008, Systems and software engineering—System life cycle processes*. Switzerland: International Organization for Standardization/International Electrotechnical Commission.

Keutzer, K., Massingill, B., Mattson, T., & Sanders, B. (2010). *A design pattern language for engineering (parallel) software: Merging the PLPP and OPL projects.* Paper presented at the Workshop on Parallel Programming Patterns (ParaPLoP), Carefree, AZ, USA.

Kim, D. H., Kim, J. P., & Hong, J. E. (2009). *Practice patterns to improve the quality of design model in embedded software development.* Paper presented at the 9th International Conference on Quality Software (QSIC), (pp. 179-184), Jeju, Korea.

Kleinsmann, M., Buijs, J., & Valkenburg, R. (2010). Understanding the complexity of knowledge integration in collaborative new product development teams: A case study. *Journal of Engineering and Technology Management, 27*(1-2), 20-32.

Meszaros, G., Doble, J., Martin, R. C., Riehle, D., & Buschmann, F. (1997). A pattern language for pattern writing. *Pattern languages of program design 3*. Boston, MA, USA: Addison-Wesley Professional.

Michalski, W. (1998). *40 Tools for cross-functional teams*. USA: Productivity Press.

Moneva, H., Hamberg, R., Punter, T., & Vissers, J. (2010). *Putting chaos under control on how modeling should support design*. Paper presented at the International Council on Systems Engineering (INCOSE), Chicago, USA.

Muller, G. (2005). *Do useful multi domain methods exist*? Paper presented at the Conference on Systems Engineering Research (CSER), Hoboken, NJ, USA.

Nakata, C., & Im, S. (2010). Spurring cross-functional integration for higher new product performance: A group effectiveness perspective. *Journal of Product Innovation Management, 27*(4), 554-571. doi:10.1111/j.1540-5885.2010.00735.x

NASA - Lessons learned (2015). *Public Lessons Learned System*. Retrieved 21 Feb 2015, from http://llis.nasa.gov/

NASA Academy of Aerospace Quality (2014). *Failure mode and effects analysis module objectives.* Retrieved 11 Jan 2015, from http://aaq.auburn.edu/node/501

National Aeronautics and Space Administration (2007). *Systems engineering handbook (NASA/SP-2007-6105 Rev1).* USA: NASA.

OrgPatterns (2001). *Pattern template.* Retrieved 9 Feb 2015, from http://web.archive.org/web/20061012032421/http://www.easycomp.org/cgi-bin/OrgPatterns?PatternTemplate

Parker, G. (2003). Cross-functional teams: *Working with allies, enemies, and other strangers*. USA: John Wiley & Sons.

Pont, M., & Banner, M. (2004). Designing embedded systems using patterns: A case study. *Journal of Systems and Software, 71*(3), 201-213.

Ratcheva, V. (2009). Integrating diverse knowledge through boundary spanning processes—The case of multidisciplinary project teams. *International Journal of Project Management*, 27(3), 206-215. doi:10.1016/j.ijproman.2008.02.008

Royce, W. (1970). *Managing the development of large software systems*. Paper presented at the IEEE Western Electronic Show and Convention 26 (WESCON), Los Angeles, USA.

Schafer, W., & Wehrheim, H. (2007). *The challenges of building advanced mechatronic systems.* Paper presented at the Future of Software Engineering (FOSE), (pp. 72-84), Minneapolis, MN, USA.

Schwaber, K., & Beedle, M. (2002). *Agile software development with scrum*. USA: Prentice Hall.

ScrumPLoP (2015). *Scrum pattern community.* Retrieved 9 Feb 2015, from http://www.scrumplop.org/

Society of Automotive Engineers (1996). *Standard ARP4761: Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment*. Washington, DC, USA: SAE International.

Society of Automotive Engineers (2000). *Standard J1739: Potential failure mode and effects analysis in design (Design FMEA), Potential failure modes and effects analysis in manufacturing and assembly processes (Process FMEA), and potential failure mode and effects analysis for machinery (Machinery FMEA).* Warrendale, PA, USA: SAE International.

Teich, J. (2012). Hardware/Software co-design: The past, the present, and predicting the future. *Proceedings of the IEEE, 100* (Special Centennial Issue), 1411-1430. doi:10.1109/JPROC.2011.2182009

Vijaykumar, & Chakrabarti (2007). *Understanding patterns of interaction between designers during design process.* Paper presented at the 16th International Conference on Engineering Design (ICED), Paris, France.

Wagner, S., Schatz, B., Puchner, S., & Kock, P. (2010). *A case study on safety cases in the automotive domain: Modules, patterns, and models.* Paper presented at the 21st International Symposium on Software Reliability Engineering (ISSRE), (pp. 269-278), San Jose, CA, USA.

Wallace, C. (1922). *The Gantt chart, a working tool of management*. New York, USA: Ronald Press.

Wikipedia - Mechatronics (2015). *Mechatronics.* Retrieved 5 Feb 2015, from http://en.wikipedia.org/wiki/Mechatronics

Zheng, C., le Duigou, J., Bricogne, M., & Eynard, B. (2013). *Survey of design process models for mechatronic system engineering*. Paper presented at the 10th Congres International de Genie Industriel (CIGI), La Rochelle, France.

Appendix A Acronyms

Acronyms	Meaning
CMM	Capability Maturity Model
FMEA	Failure Methods and Effects Analysis
ISO/ECI	International Organization for Standardization/International Electrotechnical Commission
PLoP	Pattern Languages Of Programs
NASA	National Aeronautics and Space Administration
SysML	System Modeling Language

Appendix B Glossary

Term	Definition/Context
Concurrent engineering	A methodology used in product development based on the concept of tasks being executed simultaneously. Some examples include parallel development of a system, subsystem, or module.
Current design control	Description of a design control (e.g., review, prototyping, etc.) already in place to prevent failure from occurring or eliminate/reduce the immediate consequences of the failure. This description is part of FMEA ((NASA Academy of Aerospace Quality, 2014), DFMEA continued)
Failure category	Abstraction of potential failure modes.
FMEA activity	This activity is a process with the objective of preventing or reducing the impact of a *potential failure mode*. The activity identifies the *potential failure modes* of a product (Functional/Design FMEA) being developed, or identifies the *potential failure modes* that can occur during the development process (Process FMEA). For each *potential failure mode*, the following items are determined: potential failure effects, potential cause(s), probability, severity, and detection. Corrective measures need to be taken to prevent or reduce the impact of a *potential failure mode* with high severity ranking. The measures are called current design controls, or recommended actions (Department of defense, 1980).
FMEA category	The FMEA categories are the subjects on which the FMEA (product or process) can be applied. • The FMEA categories for the product FMEA are: hardware, software, and timing/sequence. • The FMEA categories for the process FMEA are: production, maintenance, and use Retrieved from: (Haapanen & Helminen, 2002)
FMEA category examples	Examples of FMEA categories described by (Haapanen & Helminen, 2002) are: • Hardware category: “a system’s electrical, mechanical, and hydraulic subsystems and the interfaces between these subsystems.” • Software category: “programs and their execution as tasks that implement various system functions. This category also includes the program interfaces with the hardware and those between different programs or tasks.” • Timing/sequence category: “timing and sequence of various system operations.” • Production category: “the production of the hardware of a software-based system may involve chemical processes, machining operations, and the assembly of subsystem components. The software production includes the production routines reaching from requirement specification to the final testing of the software.” • Maintenance category: “preventive and corrective maintenance as well as configuration control.” • Use category: “all of the ways a product may be used; it includes operator or other human interfaces, effects of over-stress conditions, and possible misuses of the system.”
FMEA classification	Depending on the application, FMEA can generally be classified as either process or product. • “The product FMEA analyses the design of a product by examining the way that item’s failure modes affect the operation of the product.“ • “The process FMEA analyses the processes involved in design, building, using, and maintaining a product by examining the way that failures in the manufacturing or service processes affect on the operation of the product.” Retrieved from: (Haapanen & Helminen, 2002)
Force	“The features or characteristics of a situation that, when brought together, find themselves in conflict and create a problem. To consider any solution to the problem effective, the forces must be balanced.” (Buschmann, Henney & Schmidt, 2007)
Mechatronic system	“A computer-controlled mechanical system, including both an electronic computer and electromechanical components” (Wikipedia - Mechatronics, 2015)
Mining	The process of analyzing data and summarizing it into useful information. Within the context of this thesis, practice mining such useful information contains problem and solution descriptions that solve the problem.
Patlet	“A short summary of the problem and solution for a pattern.” (Coplien & Harrison, 2005)
Pattern	“Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.”(Christopher, Ishikawa & Silverstein, 1977)
Pattern language	A network of interrelated patterns that define a process for resolving development problems systematically. Definition based on (Buschmann, Henney & Schmidt, 2007)
Pattern sequence	“A sequence of patterns applied to create a particular architecture or design in response to a specific situation. From the point of view of a pattern language, a pattern sequence represents a particular path

	through the language." (Buschmann, Henney & Schmidt, 2007)
Potential effect(s) of failure	This term is used within FMEA worksheets, and it is a description of the immediate consequence of a specific failure ((NASA Academy of Aerospace Quality, 2014), DFMEA continued)
Potential failure mode	This term is used within FMEA worksheets, and it is a description of a specific failure that may occur within the project, or with the system and its functions ((NASA Academy of Aerospace Quality, 2014), DFMEA continued)
Practice	A way to solve a problem. A practice describes the problem and solution, and has a descriptive name based on the solution.
Product development domains	The product development domains are defined as follows (Eppinger & Salminen, 2001): • "Product" is what an organization produces. "A complex product or large system is decomposed into sub-systems, and these in turn may be further decomposed into sub-assemblies and/or components." • "Process" is the way an organization produces a product. "A full development process is decomposed into phases or sub-processes, and these in turn may be further decomposed into tasks, activities, and work units." • "Organization" produces a product through a process. "A large development organization is decomposed into teams, and these in turn may be further decomposed into working groups and individual assignments."
Project lifecycle phase	The project lifecycle phases are based on the waterfall model. These are (Royce, 1970): • Requirements specification • Design • Construction (Implementation) • Integration • Testing (and debugging) • Installation • Maintenance Note: This definition is only used to define the project phases and not the development process.
Recommended action	This term is used within FMEA worksheets, and it is a description of an action that can be taken to prevent failure from occurring, or eliminate/reduce the immediate consequences of the failure. ((NASA Academy of Aerospace Quality, 2014), DFMEA continued)
Retrospective	A retrospective is a team activity in which a team reflects on the past period of development. The objective is to learn from the past period and use this knowledge to increase product quality and the work life of team members. This is accomplished by incorporating the successes and improvements in the next period. Retrospective can be done in many different ways (Derby & Larsen, 2006)
Retrospective categories	During a retrospective, an individual or team can make remarks. Such remarks are categorized as follows: • Positive remark. Remarks on product development of which the individual or team is proud and wants to continue. • Negative remark. Remarks on product development of which the individual or team is dissatisfied. • Improvement remark. Action assigned to an individual or team in order to improve a negative remark.
Severity	This term is used within FMEA worksheets, and it is the evaluation of the severity of the failure effect on the next system or internal/external customer. Sometimes, large values of severity can be reduced through design reviews that compensate or mitigate the resulting severity (NASA Academy of Aerospace Quality, 2014, DFMEA continued).

Appendix C Confidential sources

The FMEA worksheets and Retrospective reports were made available by an organization that develops and produces mechatronic systems. However, such organization made these reports available under the strict condition that the information is kept confidential.

Information details:

- 1 A4 book (101 A4-pages)
    - 17 FMEA worksheets (24 A4-pages).
      Adjustments to the information:
        - Names are made anonymous
        - Financial information is removed
        - Removed severity level, probability, detectability, and risk priority number
    - 57 Retrospective reports (77 A4-pages).
      Adjustment to the information
        - Names are made anonymous

One hardcopy of the information is provided to the graduation committee. For those who want to see this information, please contact:

Open Universiteit Nederland
Department: Computer Science
Valkenburgerweg 177
6419 AT Heerlen
The Netherlands
e-mail: info@ou.nl
Phone: +31 45 576 2888

Appendix D Literature study: practices

This appendix provides an overview of the practices mined during the literature study.

In Table D-1, the first column describes the problem. The second column contains the solution to the problem. This problem and the solution can have multiple references. The third column contains the practice name, which is inspired by the solution. The table is sorted based on the practice name.

Table D-1: Practices mined from literature

Problem	Solution	Practice name
Communication between disciplines is hampering. The reason is that there is no common understanding (Beckers et al., 2007); Integration problems caused by the complexity of the dependencies between subsystems and developers (Alvarez Cabrera et al., 2010)	Create a simplified view of the required system behavior. This description can be based on the ideal world (Happy flow) (Beckers et al., 2007); Increase abstractions level of the design (Alvarez Cabrera et al., 2010)	Abstract design
The problem is that a design decision made in one discipline can have enormous impact on another discipline. A reason for this is that monodisciplinair design decisions are not communicated to other disciplines (Moneva et al., 2010) (Heemels, 2006) (Boucher & Houlihan, 2008)	Make a selection of the most critical aspects of the system. Manage the requested modification by a formal process (Moneva et al., 2010); Analyze the impact of the design decision on the system as soon as possible. A framework can support this analysis (Heemels, 2006); Implement a process that organizes the communication of changes to all disciplines (Boucher & Houlihan, 2008)	Alert design change
Collaboration between engineers of different disciplines hampers during initial design phase. This can result in poor system design (Alvarez Cabrera et al., 2010); The design work is not divided among the disciplines. This can result in poor system design (Alvarez Cabrera et al., 2010)	Organize an activity in which information is exchanged between the engineers of all disciplines (brainstorm, group reviews) (Alvarez Cabrera et al., 2010)	Brainstorm on design
A multidisciplinary design is too abstract to extract detailed requirements (Muller, 2005); No verification can be done based on a multidisciplinary design (Muller, 2005).	Create an explicit design based on budgets (e.g., Performance, use of resources, etc.) (Muller, 2005)	Budget design
There is no team synergy (Michalski, 1998, pp. 21, 28)	When making decisions, favor consensus over voting. When consensus is reached, all team members will support the outcome, even when it fails (Michalski, 1998, p. 28)	Build commitment
Team members fail to cooperate with each other and abdicate responsibility (Nakata & Im, 2010, p. 13)	Assign a central power (Nakata & Im, 2010, p. 13)	Central power
Team fails to achieve goal (Michalski, 1998, p. 23)	Set clearly stated mission, goals, and team objectives (Michalski, 1998, p. 23)	Clear priorities
Team fails to achieve goal (Michalski, 1998, p. 23)	Create a clear definition of tasks (Michalski, 1998, p. 12)	Clear tasks

The project engages unnecessary iterative loops (Kleinsmann, Buijs & Valkenburg, 2010, p. 22); Miscommunication occurs (Vijaykumar & Chakrabarti, 2007, p. 3); Lack of common background knowledge at the beginning of the projects (Ratcheva, 2009)	Iterative style of informal communication allows people to reach well-founded decisions and find common ground (Vijaykumar & Chakrabarti, 2007, p. 2); Develop a shared vocabulary by which to communicate (Vijaykumar & Chakrabarti, 2007, p. 4); Create shared understanding on development process and its content (Kleinsmann, Buijs & Valkenburg, 2010, p.21); One person must “mutually accept” another's references before the conversation proceeds (Vijaykumar & Chakrabarti, 2007)	Common ground
Projects suffer from overlapping responsibilities and ambiguous command chains, generating psychic distance and intense conflict among team members (Nakata & Im, 2010, p. 7)	Create a common project plan (Nakata & Im, 2010, p. 7)	Common plan
Tasks of disciplines are not aligned. This leads to inefficient execution of the project (Zheng et al., 2013)	Inform the status of the project to all disciplines. Make clear which parts have impact on each other (Zheng et al., 2013)	Define dependencies
There is no team synergy (Michalski, 1998, p. 27)	Define team roles (Michalski, 1998, p. 27)	Define roles in team
There is no cross-functional knowledge. This can lead to unawareness of the impact of modifications. This can also lead to a non-optimal solution for a multidisciplinary problem (Boucher & Houlihan, 2008); The consequence of a monodisciplinair design decision on other disciplines is unclear (Heemels, 2006)	Inform other disciplines of the design and design choices. This will actively involve other disciplines to interact and benefit the overall system design (Boucher & Houlihan, 2008); Invest on the consequences of a design decision for all disciplines as soon as possible (Heemels, 2006)	Design walkthrough
Development of a new product takes too long (Parker, 2003, p. 15); Group does not working effectively (Nakata & Im, 2010, p. 558)	Grant team authority to make decisions on functionality, cost, production, and appearance (Parker, 2003, p. 15, chapter 5); (Michalski, 1998, p. 27; Nakata & Im, 2010, p. 558)	Empower the team
Team is not well integrated (Parker, 2003, p. 28) (Nakata & Im, 2010, pp. 554, 566)	As management, accept failures. This motivates teams to pursue high potential projects (Nakata & Im, 2010, p. 560)	Encourage risk taking
Team does not meet goals (Parker, 2003, p. 90)	Introduce feedback loop on the team’s activities (e.g., plan versus actual) (Parker, 2003, p. 90)	Establish a scoreboard
Models and simulations do not exactly represent the real system. Therefore, system verification cannot be executed (Alvarez Cabrera et al., 2010); Many integration problems at the end of the project (Alvarez Cabrera et al., 2010), (David Bradley, 2010); Fault detection and diagnoses can only be executed when the final system is available (Boucher & Houlihan, 2008)	Create a setup that involves only the critical hardware parts and software to be integrated (Boucher & Houlihan, 2008), (David Bradley, 2010), (Alvarez Cabrera et al., 2010)	Hardware in the Loop
Lack of support by functional department (Parker, 2003, pp. 89, 95)	Incorporate team goals into goals of the functional department (Parker, 2003, pp. 89, 95)	Incorporate goals
People do not understand the big picture, do not know what to know, experience information distortion, and have different interpretation of representations (Vijaykumar & Chakrabarti, 2007, p. 2)	Create appropriate information flow (Vijaykumar & Chakrabarti, 2007)	Information flow
There is no good mechanism to coordinate the varied efforts (Parker, 2003, p. 32)	Assign a role whose job it is to facilitate the coordination of the various allies, enemies, and strangers among the project (Parker, 2003, p. 32)	Integrator
Development of a new product takes too long (Parker, 2003, p. 15)	Compose a team with all relevant departments, including marketing and purchase (Parker, 2003, p. 15).	Involve key stakeholders
Development of a new product takes too long (Parker, 2003, p. 15)	Enforce the discipline necessary to maintain the schedule. This can be done by setting strict deadlines and hold to them (Parker, 2003, pp. 15-16)	Keep the schedule

Impact of a new requirement on the design cannot be determined because the relationship between design and requirements are unclear. This is especially difficult for multidisciplinary projects because requirement implementations can be divided among disciplines (Graaf, 2003)	Keep track of requirements during design by implementing requirements management (Graaf, 2003)	Keep track of requirements
The problem is that requirements are not implemented. Top-level requirements are divided over the disciplines. When dividing requirements, details can be lost. In addition, during development, new requirements emerge. This may not be addressed correctly (Boucher & Houlihan, 2008)	Assign one person who becomes owner of all requirements. This person has the oversight of the requirements (Boucher & Houlihan, 2008)	Key master
Lack of ability to integrate knowledge can keep the team from gaining any benefits from resource pool (Ratcheva, 2009, p. 208); Diverse knowledge is not integrated in team (Ratcheva, 2009, p. 208); Knowledge integration is difficult because team members often have different interests and perspectives on new methods to develop the product (Kleinsmann, Buijs & Valkenburg, 2010, p. 21); A domain expert cannot find a solution in another domain. This may lead to an inefficient solution (David Bradley, 2010); For the engineers, it is difficult to foresee the consequence of becoming involved in a task outside their direct scope (Kleinsmann, Buijs & Valkenburg, 2010, p. 26); Disciplines are caught in knowledge silos. As a consequence, design decisions made by one discipline can have a negative impact on another discipline in a next development phase (Boucher & Houlihan, 2008)	Maintain the team together over time. Behaviors are learned over time from working in a specific setting (Ratcheva, 2009, p. 208); Create transitive memory. This makes it possible to develop complex products with actors from different disciplines without having too much redundancy of knowledge (Nakata & Im, 2010, p. 22); Increase interpersonal interactions and relational capital developed among members (Ratcheva, 2009); Face-to-face communication (Kleinsmann, Buijs & Valkenburg, 2010); Firms should develop an organizational context that allows collective action (Kleinsmann, Buijs & Valkenburg, 2010, p. 29); Case-based reasoning can lead to an existing solution or to granular ideas of a solution. This idea can be taken to the domain expert (David Bradley, 2010); Make clear the allocation of tasks and responsibilities (Kleinsmann, Buijs & Valkenburg, 2010, p. 26); Increase multidisciplinary knowledge of the engineers. This can be accomplished by cross-training. (Boucher & Houlihan, 2008)	Knowledge integration
Team is not effective (Ratcheva, 2009)	Let a new way of working emerge and develop through intense interactions (Ratcheva, 2009)	Learn by interaction
Teams do not benefit from team diversity (Parker, 2003, p. 28)	Conduct technical training (Parker, 2003, p. 31); Provide team training (Parker, 2003, p. 31)	Learning community
Lack of common understanding between multidisciplinary engineers (Heemels, 2006); No idea on how to approach the challenges of the development of the system (David Bradley, 2010); Not all disciplines are aware of the current multidisciplinary design conflicts. The conflicts should be considered when new design decisions are made. If this out of scope, decisions can have negative consequences (Boucher & Houlihan, 2008)	Write the most important insights on the design, development, and practices of the previous systems. Successful solutions can be reapplied (Heemels, 2006); Formally document the multidisciplinary integrations issues. This improves visibility of the design conflicts (Boucher & Houlihan, 2008)	Lessons learned
Team members have different personal objectives and motivations for participation, which do not align with the project or organizational objectives (Ratcheva, 2009)	Managing expectations is of paramount importance for successfully accomplishing a project. Therefore, project teams are required to adopt a much wider set of responsibilities beyond the immediate technical aspects of the project, and be granted greater autonomy (Ratcheva, 2009, p. 214)	Manage team member expectations
Team fails to achieve goal (Michalski, 1998, p. 23); Team members are less motivated to put effort in the project because they are not evaluated against project objectives (Parker, 2003, p. 95)	Assign challenging, but fair, set of measures directly linked to the team's goal and objectives (Michalski, 1998, p. 23); Incorporate team goals into the goals of the team member (Parker, 2003, pp. 89, 95); Evaluate team members on their performance in the project (Parker, 2003, p. 95)	Mutual accountability

Designing all aspects of a system is not cost effective (Moneva et al., 2010)	Only design the critical parts of a system (Moneva et al., 2010)	Only design critical parts
Team underperforms because it is not highly motivated (Parker, 2003, p. 90)	Set a compelling goal for the team (Parker, 2003, p. 90)	Provide an incentive
Team fails to achieve goal (Michalski, 1998, p. 12)	Share leadership across members (Michalski, 1998, p. 12)	Shared leadership
System level issues are discovered late in the design process. As a consequence, design options are reduced because critical decisions are already made (Boucher & Houlihan, 2008)	Simulate behavior at system level. With this simulation, virtual tests can be executed early in the design cycle. This allows early identification of problems on system level. (Boucher & Houlihan, 2008)	Simulator in the Loop
It is difficult to determine where to go for information regarding the project (e.g., status, technical details). (Parker, 2003, pp. 31-36)	Assign a role (e.g., team leader) whose job is to communicate team information to the stakeholders (Parker, 2003, pp.31-36)	Single point of contact
In larger projects, the design and integration of a system requires a significant amount of time. This can be caused by organizational issues, politics, and projects that are not aligned (Heemels, 2006)	Form small multidisciplinary teams. A small team can work efficiently (Heemels, 2006)	Small team
The problem is that requirements are not implemented. It is unclear how the responsibilities of the system are divided (Boucher & Houlihan, 2008), (Zheng et.al, 2013); Integration of a mechatronic system is difficult (David Bradley, 2010)	Create one role that divides the system into manageable parts (such as subsystems and components) over all the disciplines (Boucher & Houlihan, 2008)	Splitter
Problems are too complex to be solved by a series of functional teams (Parker, 2003, pp. 17-19, Chapter 8); No single person can do it alone (Parker, 2003, pp. 17-19, Chapter 8)	Bring together scientist and engineers from a variety of backgrounds and diverse training (Parker, 2003, pp. 17-19, Chapter 8)	Team of experts
Development of a new product takes too long (Parker, 2003, p. 15)	Conduct many tasks in parallel (Parker, 2003, p. 16)	Work in parallel

Appendix E FMEA: Worksheet layout

This appendix shows the layout of the FMEA worksheet made available by an organization that produces mechatronic systems (see also Appendix C). In this layout, a description is provided on the columns that need to be completed (red squares).

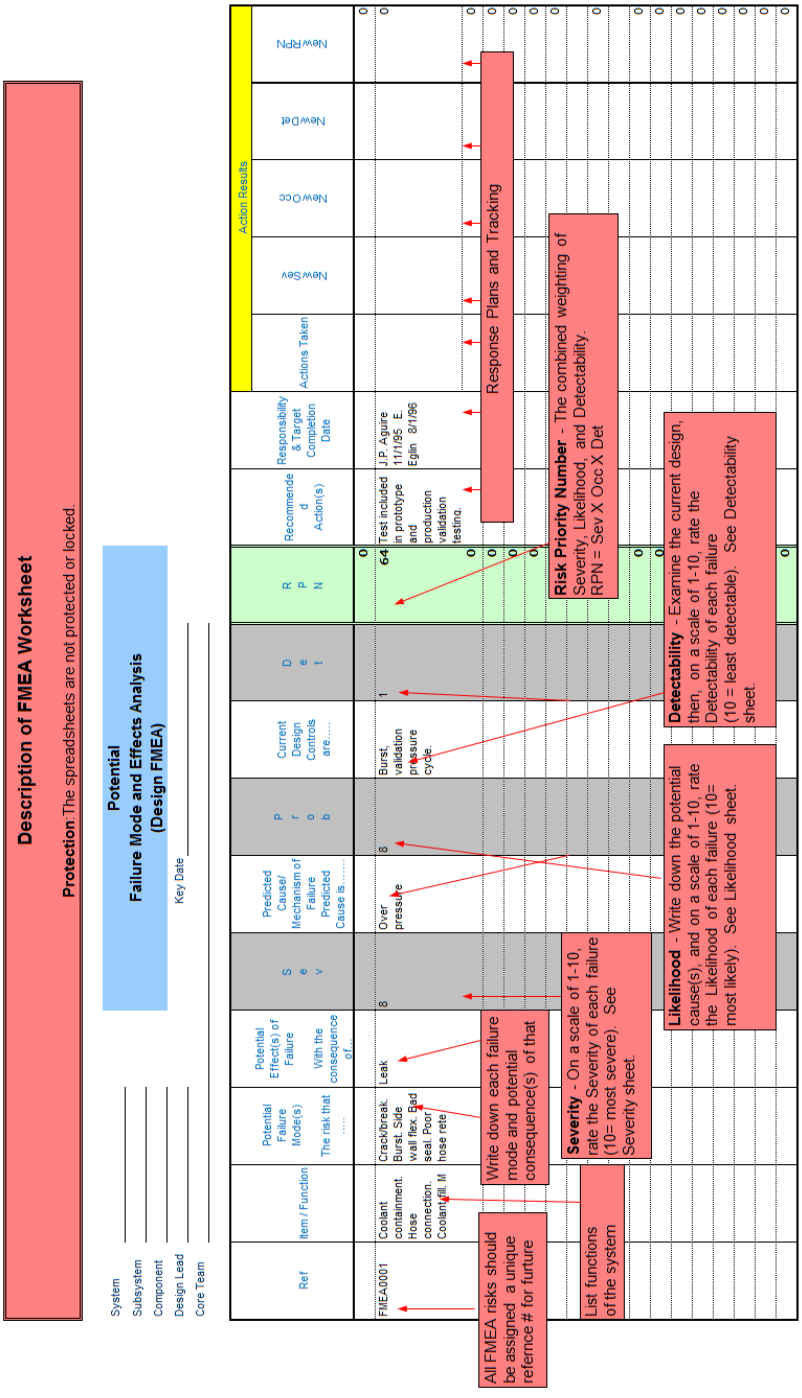


Figure E-1: FMEA worksheet example

Appendix F FMEA: process for creating failure category

This appendix describes the process for creating *failure categories*, which is performed in two stages. First, a classification is made of all the *potential failure modes* (see Figure F-1). In the second stage, a list of *failure categories* are defined based on the classification list (see Figure F-2). The purpose is to create a list of (maximum 30) *failure categories*.

Figure F-1 shows the process for classifying the *potential failure modes*. On the left side, the process flow is visualized. On the right side, assignment options for each step are defined. The FMEA assignment options are based on the definitions of Figure 3.4-1. The process steps are described in more detail in Table F-1. An example of the output of the process is provided in Table F-2.

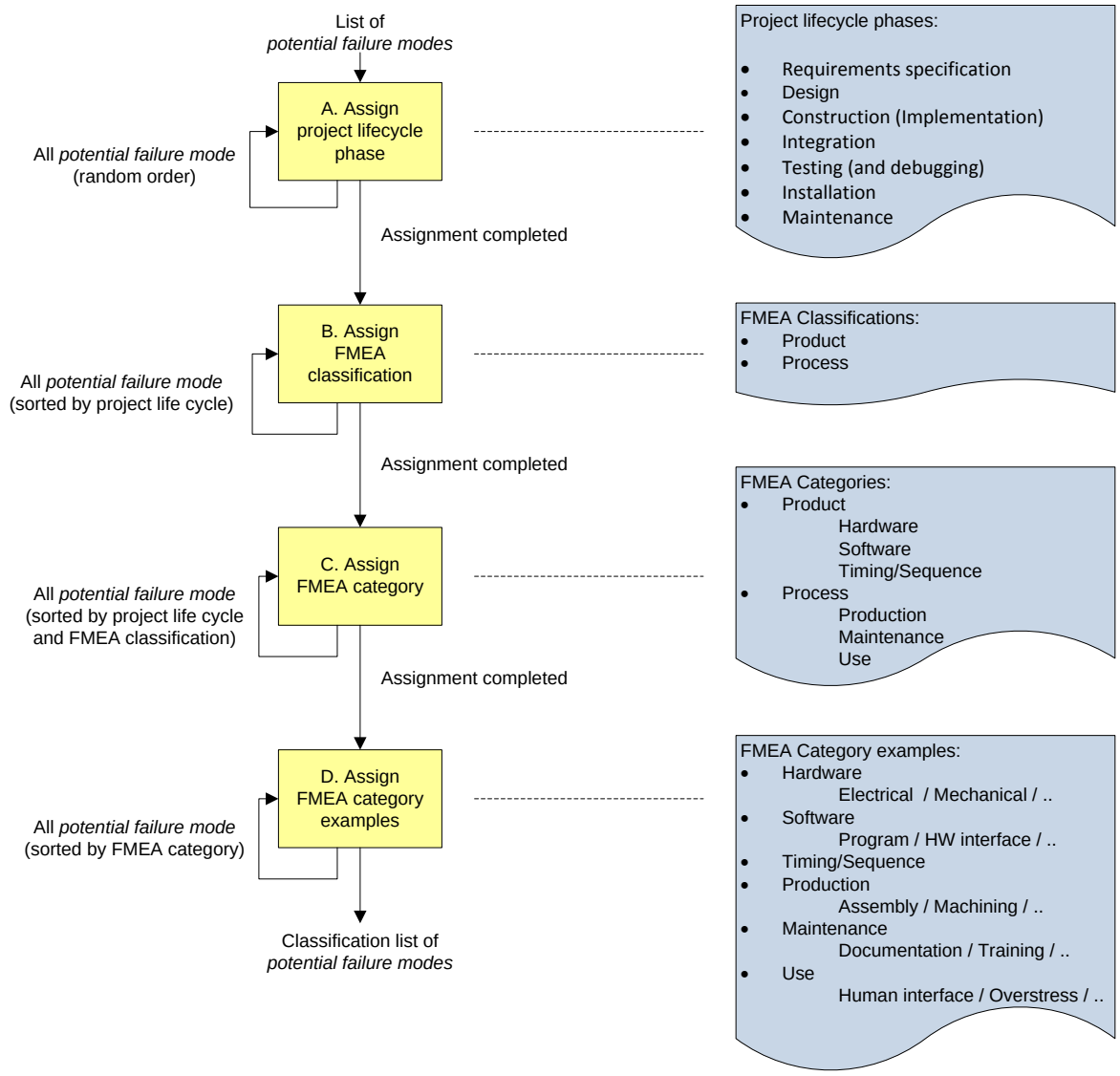


Figure F-1: Process description for the classification of potential failure modes

Table F-1: Process description for classification of potential failure modes (Figure F-1)

Process step		Process description
A. Assign project lifecycle phase	Input	List of *potential failure modes*. This list contains all the *potential failure modes* of all the FMEA worksheets.
	Process	One *potential failure mode* is read. Based on the available information, one project lifecycle phase is assigned to that specific *potential failure mode*. This process continues until all *potential failure modes* are assigned to a project lifecycle phase. Examples are requirement analysis, design, and construction.
	Output	List of *potential failure modes* assigned to a project lifecycle phase.
B. Assign FMEA classification	Input	List of *potential failure modes* assigned to a project lifecycle phase. This list is sorted by the project lifecycle phases. Such sorting can accelerate the process when duplicate descriptions are found.
	Process	Process is similar to process "step A." The difference is that this process assigns FMEA classifications, which are process and product.
	Output	List of *potential failure modes* assigned to FMEA classification.
C. Assign FMEA category	Input	List of *potential failure modes* assigned to a project lifecycle phase and one FMEA classification. This list is sorted by project lifecycle phases and FMEA classification. Such sorting can accelerate the process when similar or duplicate descriptions are found.
	Process	Process is similar to process "step A." The difference is that this process assigns a FMEA category. Examples are: hardware and software.
	Output	List of *potential failure modes* assigned to a FMEA category.
D. Assign FMEA category examples	Input	List of *potential failure modes* assigned to a project lifecycle phase, FMEA classification, and FMEA category. This list is sorted by FMEA category. Such sorting can accelerate the process when similar or duplicate descriptions are found.
	Process	Process is similar to process "step A." The difference is that this process assigns a FMEA category example. Examples are: human interface, overstress.
	Output	List of *potential failure modes* assigned to FMEA category example.

Table F-2 lists examples of the output for this step. The data used are real data from the worksheets.

Table F-2: Examples of classification of the potential failure modes

Potential Failure Mode(s) (real data)	Project lifecycle phase when risk arises	FMEA Classification	FMEA Category	FMEA Category example
Cable routing causes interference on system performance	Maintenance	Product	Hardware	Electrical
Difficult to validate that a correct Wizard is included in the built	Installation	Process	Production	Software
Module configurations in the field are unknown	Maintenance	Process	Maintenance	Configuration control
No offline test environment is available	Testing	Product	Software	Hardware interface
Still gases present when cleaner is switched on	Maintenance	Product	Timing/ Sequence	-
User interface does not adapt to larger display	Maintenance	Process	Use	Human Interface

Figure F-2 shows the process of creating *failure categories*.

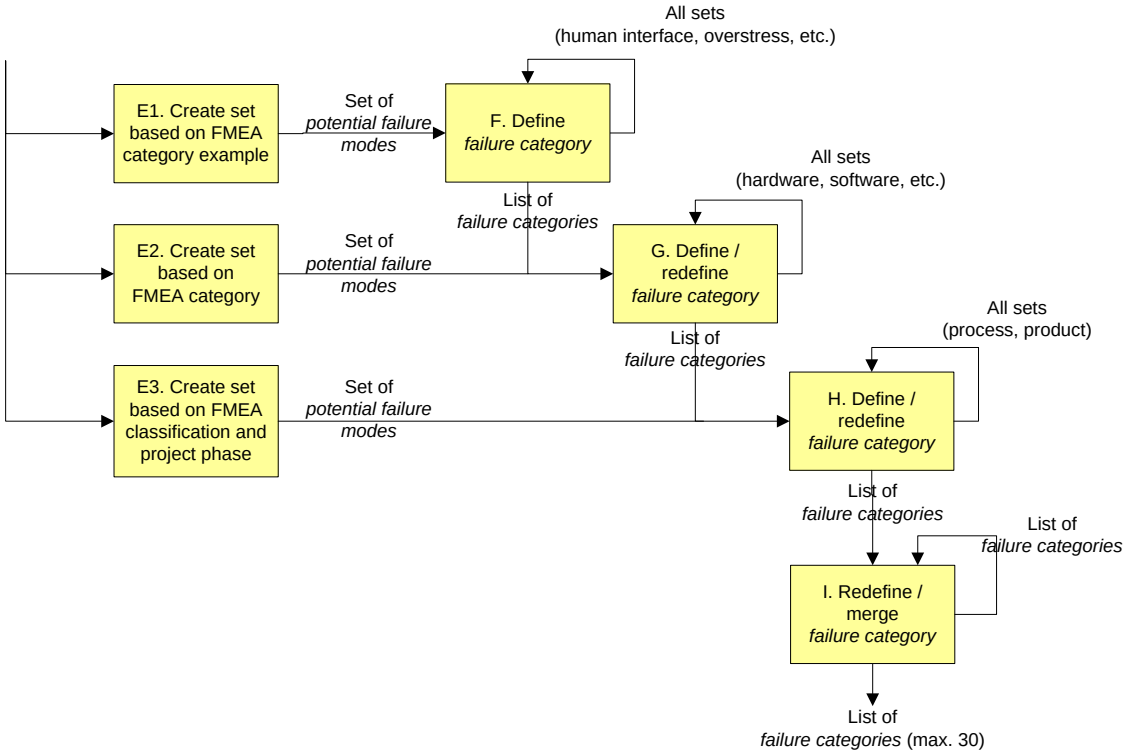


Figure F-2: Process description for creating failure categories

The process for creating the *failure categories* starts with all the *potential failure modes* that are classified (Figure F-1). The three classifications are:

- FMEA classification and project lifecycle phase
- FMEA category
- FMEA category examples

The "steps E1," "E2," and "E3" use the classified *potential failure modes* as input. In "step F," *failure categories* are created based on the set of *potential failure modes*. This creation starts with fine-grained *failure categories*. After several more generic steps, coarse-grained *failure categories* are created. At the end of the process ("steps G," "H," and "I"), a list of maximum 30 *failure categories* is defined. This limitation helps maintain the description generic. In Table F-3, the process steps are described in more detail. An example of this process is provided in the next section (under Table F-3).

During this process, the author used domain knowledge to define *failure categories* and assign these to a *potential failure*. The domain knowledge used is: abbreviations, technical context, organization knowledge, and supplier knowledge.

Table F-3: Process description for creating failure category

Process step		Process description	
E.	Create sets	Input	All *potential failure modes* that are classified
		Process	Create sets based on: • FMEA category examples • FMEA category • FMEA classification and project lifecycle phase These are made by executing a query on the classified *potential failure modes* (see Table F-2)
		Output	Each sub-step has its own specific output: • Step E1. A total of 15 sets of *potential failure modes* based on FMEA category examples (examples are: human interface, overstress, etc.) • Step E2. Six sets of *potential failure modes* based on FMEA category (examples are: hardware, software, etc.) • Step E3. Two sets of *potential failure modes* based on FMEA classification (process, product)
F.	Define *failure category*	Input	Sets of *potential failure modes* (human interface, overstress, etc.)
		Process	1) Similar *potential failure modes* are searched within the set. 2) A f*ailure category* is created based on similar *potential failure modes*. 3) This new *failure category* is compared with the existing *failure categories*. When there is a similar *failure category*, a new *failure category* is created to cover both, or they are redefined to distinguish them better. Each set can have multiple *failure categories*. 4) This process ends when all sets are processed. An example of this process is described below this table.
		Output	A list of *failure categories*
G.	Define/redefine *failure category*	Input	Sets of *potential failure modes* (hardware, software, etc.) A list of *failure categories*
		Process	Process is similar to the process for “step F.”
		Output	A list of *failure categories*
H.	Define/redefine *failure category*	Input	Sets of *potential failure modes* (process, product) A list of *failure categories*
		Process	Process is similar to the process for “step F.”
		Output	A list of *failure categories*
I.	Redefine/merge *failure category*	Input	A list of *failure categories*
		Process	This process only starts or continues when there are more than 30 *failure categories*. In such a case, similar *failure categories* are searched. These similar *failure categories* should be redefined to cover both.
		Output	A list of 30 or fewer *failure categories*

This section describes an example of the creative process for defining a *failure category*. This example is based on "step F" of Figure F-2. The description of the sub-steps can be found in Table F-3. The data used for this example are real data retrieved from the FMEA worksheets.

Step F. Process 1

- The set "Process.Production.Software" (see Figure 3.4-1) is selected.
- All the *potential failure modes* are read. A manual search is performed on similar subjects (can be failure modes, possible root causes, or similar description). One example is several *potential failure modes* that consider requirements.

Potential failure mode (real data obtained directly from the source)
Limited feedback from customers about UI
New system will just be a set of point solutions, not clear what customer expects
Software automation, consequences unknown because it is not yet defined
The risk that use cases are not fully analyzed and linked to requirements
Unclear how much customer feedback is expected (i.e., changes to requirements)
Uptime requirements? System SW does not guarantee uptime!

Step F. Process 2

- Define or assign a *failure category* for each *potential failure mode*.

Potential failure mode (real data obtained directly from the source)	**Failure category**
Limited feedback from customers about UI	User feedback not considered
New system will just be a set of point solutions, not clear what customer expects	Expectations of customer unclear
Software automation, consequences unknown because it is not yet defined	Not clear what will be created
The risk that use case are not fully analyzed and linked to requirements	Requirement cannot be traced
Unclear how much customer feedback is expected (i.e. changes to requirements)	Expectations of customer unclear
Uptime requirements? System SW does not guarantee uptime!	No requirement defined

- The six *potential failure modes* result in five unique *failure categories*. Such five unique *failure categories* should be reduced to one *failure category*. This is done by asking the following questions:
    - What is the possible root cause?
    - What is the impact on the system/customer of this root cause?
    - What could be done to prevent/improve this impact?

Potential failure mode (real data)	Possible root cause	Impact on system/customer	Prevent/Improve
Limited feedback from customers about UI	R&D is focused inside out. No active feedback from customer gathered.	System does not comply with customer expectations.	Define requirements
New system will just be a set of point solutions, not clear what customer expects	R&D is focused inside out. Never asked customer about his requirements.	System does not comply with customer expectations.	Define requirements
Software automation, consequences unknown because it is not yet defined	A project starts without clear requirements.	System development delays	Define requirements
The risk that use case are not fully analyzed and linked to requirements	No focus on good requirements management.	Features are not available to the customer.	Define requirements
Unclear how much customer feedback is expected (i.e. changes to requirements)	R&D is focused inside out. No customer feedback was gathered during development.	System does not comply with customer expectations.	Define requirements
Uptime requirements? System SW does not guarantee uptime!	No focus on determining the expected behavior of the system.	System availability can be lower than expected by the customer.	Define requirements

Now, sentences can be made (If…then…To achieve this….):

- **If** requirements are defined, **then** a feature will be available for the customer. **To achieve this**, good requirement management is required.
- **If** requirements are defined, **then** the system can comply with customer expectations**. To achieve this**, R&D needs to focus outside-in and actively gather customer feedback.

If the sentences make sense, the *failure category* can be defined based on "prevent/improve." The *failure category* is a negative formulation of "prevent/improve." In this example, the "failure category" becomes "Requirement is not defined."

Note: during this process, the author used the following domain knowledge:

- Abbreviations (e.g., HM, UEC, TAD, HAL, etc.)
- Technical context (e.g., names of modules, application)
- Organization (e.g., responsibilities)
- Suppliers (e.g., supplied modules)

Step F. Process 3

- When *failure category* descriptions are similar, the same process as "step F.2" is executed. The result could be that two *failure categories* become one. The result could also be that the *failure categories* are redefined to a more distinct description. If the *failure categories* are redefined, all assigned *potential failure modes* should be analyzed again to determine whether the *failure category* still fits.

Appendix G Result of FMEA worksheet analysis

This appendix contains an overview of all the *failure categories* identified during the FMEA worksheet analysis (chapter 3.4). In some cases, a practice name is assigned to a *failure category*. The name of the practice is inspired by the solution. The same name is used when solutions appears to be the same.

Table G-1 lists the result of that analysis. The table is sorted on average severity ranking (second column). The ranking of each *failure category* is based on Table 3.4-1.The third column contains the occurrences of *potential failure mode* found in all the worksheets. The fourth, fifth, and sixth columns contain examples, which are examples of *potential failure mode(s)*, *potential effects of failure,* and *current design control* and/or *recommended action*.

The table contains information to start writing patterns (section 3.6). The problem is described in the first and fourth columns (see second row of the table). The fifth column contains examples that can become part of the context or the forces/tradeoffs description. The sixth column contains the solution, and the seventh column the practice name. Refer to Table G-2 for details on the mapping between solution and practice name.

Table G-1: Failure category overview based on FMEA worksheets

Failure category (The risk is that …)	Average severity ranking	Occurrences in worksheets	Examples of Potential Failure Mode(s)	Examples of Potential Failure Effect(s)	Examples of current design control and/or recommended action	
Problem			Example of problem		Solution	Practice name
Human safety is not considered	8.6	83	X-Ray leak Pinch hazard Lack of safety expertise Parts are not explosive-proof	User dies System cannot be shipped Component or system becomes damaged	Review the design Add safety instructions to work instructions Determine safety responsibility (customer or supplier)	Review deliverables System integration testing
Customer expectation is not managed	6.7	12	System specification cannot be met in case of some system options Certain combinations of system options are not possible Immature feature is sold Deviations in service contract are not communicated	System is not accepted by customer Customer complaints Customer ignorance causes more service actions	-	
Requirement is not defined	6.2	27	Exact customer specifications are unknown Crucial feature requirements are unknown Cost of good (COGS) target is unknown	Project delay Redesign required System does not meet specifications	Discuss and decide requirements with stakeholders Discuss and decide requirements with suppliers Determine responsibility	

Integrated system behavior is not considered	6.2	153	Collision between components Pre-conditions for operation are not met Disturbances (e.g., vibration, temperature variation) influences performance	Reduced system performance Component or system becomes damaged System cannot be used	Improve design (e.g., add fail-safe mechanism, improve system robustness) 3D design check Bench testing	Mock-up Hardware in the Loop
Behavior of system is unknown	6.2	25	Customer uses system with non-standard materials Untested effects of functionality on system Behavior after failure is unknown	Component or system becomes damaged Reduced system performance Customer complaints	Execute system integration tests Verify behavior with engineers	System integration testing
Test coverage is too low	6.1	10	No test plan available Tests are not representative for end-user qualification Low testability during system buildup	Component or system becomes damaged Service action required Unstable or unreliable system behavior	Extend factory acceptance test (FAT) with tests for new components Design and verify qualification tool in the supply chain Align quality control system with supplier	
System performance varies	5.9	23	Mechanical tolerances Repeatability of system is low	Reduced system performance System performance not within specifications Lifetime decreases	Measure performance Redesign component Automate procedure (e.g., startup procedure for better conditioning)	
System deteriorates over time	5.8	107	System becomes contaminated Parts wear out Corrosion on components Parts become loose (e.g., because of aging glue, cracked solder joints)	System becomes unreliable Component or system becomes damaged Safety issues may arise	Component redesign (e.g., use other materials) Reduce load on component (e.g., restrict number of retries) Execute lifetime tests of a component at the supplier	Module Testing
Responsibility is not assigned	5.8	9	Not clear if customer or supplier has a responsibility on safety issues Not clear who is responsible for maintaining non released products	Legal discussions with customer Version problems Hardware/firmware Safety issue overseen	Assign responsibility to specific person Only support released products	
Project plan is not managed	5.8	24	No release plan No integration plan Project status not clear Creating unnecessary project dependencies	Project delay Shipping of unreleased products Introduction of dependency of other projects	Make clear plan Requirements/design review Discuss project risks with marketing	Clear plan Review deliverables
Project scope is not managed	5.8	8	Not clear which requirements are the responsibility of the project Not clear how to manage conflicting requirements	Additional costs and resources involved Project delay Customer complaints	Make decision on project scope	

Requirement is not met	5.7	92	Requested feature is not available when system is released System performance not within specifications Feature does not work as expected No certification	Reduced system performance System cannot be shipped Missing system functions	Improve design Extend factory acceptance test (FAT) Implement missing functionality	Factory acceptance test
Failure can occur during production	5.6	103	Assembly mistakes Wrong material used Damage during component handling	Lifetime decreases System cannot be used System does not meet specification	Define/review factory acceptance test (FAT) Improve awareness (e.g., cleanliness) Improve production process	Factory acceptance test System integration testing
Status of system cannot be monitored	5.5	33	No signal readout available Insufficient diagnostic information available Mismatch between real and set value	Unknown safety risks Problem cannot be determined Unstable system	Implement tool that can diagnose the system Implement functionality that logs current status (e.g., active errors, operator input) Improve current monitor solution (e.g., sensor read-out)	
Vendor is locked-in	5.5	17	Lead-time dependence for parts No fast switching possible when supplier disappears	Risk in the supply chain No upgrades possible Redesign required	Introduce second supplier	
Design is not feasible	5.4	95	Feature cannot be integrated in system Feature does not cover requirements	Component or system becomes damaged System cannot be assembled Redesign required	Make return of investment (ROI) analysis Execute system test to validate critical specifications Redesign	Mock-up Review deliverables System integration testing Drop-In replacement
System is not reliable	5.3	125	Component lifetime is not met Component parts do not operate (e.g., vans) Software crashes Software cannot control the hardware (e.g., failed to upload firmware, no communication with hardware)	System cannot be used System performance not within specifications Service action required	Redesign component Organize lifetime test Improve testability of real problem	Module Testing
Resources (hardware) are scarce	5.1	14	No system or system options available for integration and verification No test setup available for implementation and integration	Project delay Component or system not tested sufficiently Component or system becomes damaged	Create test plan Use test bench Investigate what functionality is not tested on hardware	Hardware in the Loop
Configuration management is inaccurate	5.0	10	System configuration in field is unknown Wrong firmware is installed Wrong system option is selected	System cannot be used Incompatible field upgrades Troubleshooting takes too long	Train people Overview present configuration Restrict functionality based on configuration	Define up/downgrade strategy Embed knowledge in system

Failure can occur during use	5.0	23	User provides wrong input Procedure not followed Errors made in expert mode	Customer complaints Component or system becomes damaged Unreliable results	Check validity of input values Restrict possible parameters that can be modified (e.g., by reading values from system) Improve procedure	Embed knowledge in system
Resources (people) are scarce	4.9	14	No software developer available for support (e.g., solving bugs) No test engineer available for verification	Project delay Loss of knowledge over time Time delays in solving issue	Request for resource capacity or adjust project plan accordingly Create resource allocation plan Define responsibilities	Clear plan
Project intake is not managed	4.8	23	Third-party software not under version control Unknown semantic changes made in firmware can lead to system failures Intake not delivered on time	Project delay Unstable or slow system Unplanned upgrades required	Requirement/design review of deliverables from supplier Make deliverables explicit in supplier contract Align quality control system with supplier	
Maintenance is not efficient	4.8	53	Diagnose time is too high Service procedures not available Component difficult to replace Upgrade tooling not available	Increased downtime Only limited service actions possible Higher cost for service action	Pre-assessment of system before upgrade Write procedures Make return of investment (ROI) analysis	Clear plan
Maintainability of the system becomes difficult	4.5	15	Parts become obsolete No backwards compatibility Upgrade issues	Production problems when parts are not available Higher cost for service actions No more systems sold, no revenue	Write supplier agreement Competence management	
Production is not efficient	4.4	24	Logistic problems Tuning problems Geographical separation of software development Cycle time in factory too high	Production delay System is not accepted Time delays in solving issue	Increase expertise of components Prevent wrong assembly of component (e.g., by using different screws) Investigate the need for additional production tools	
Utilization of the system is not optimal	4.3	23	Procedures (e.g., calibration, install time) takes too long Time before system is stable is too long No abort functionality available	Increased downtime Not acceptable waiting time Reduced system performance	Implement a standby-modus Implement functionality to restore system status Use pre-measured values	
Design documentation is not available	4.3	10	Impact of modification on the system is unknown (e.g., volume claims) Non-documented behavior More time required to solve issues in production or in the field	More time required to solve issues in production or in the field Difficult to extend functionality or hardware in the future Unforeseen failures	Test component before system integration Create design solution	

Failure can occur during maintenance	2.8	30	Damage made to part of the system Components not correctly connected Transportation causes damage	System cannot be used Component or system becomes damaged Delay in upgrade	Add instruction to service manual Improve component handling (e.g., transportation lock) Improve service tooling	Test with dummies

The first column in Table G-2 describes the problem. Such problems are the *failure categories* determined through FMEA Analysis (section 3.4). The second column contains the solution to the problem. These solutions are retrieved from the worksheets (*current design controls* and *recommended actions*). Most can be found in Table G-1, the others are copied from the worksheets. The third column contains the practice name, which is inspired by the solution. The same name is used when solutions appears to be the equal.

Table G-2: Mapping of practice name to solution

Problem (Failure category)	Solution	Practice name
Behavior of system is unknown	Execute system integration tests	System integration testing
Configuration management is inaccurate	Latest firmware should become the new standard	Define up/downgrade strategy
	Overview of the present configuration should be given	Embed knowledge in system
Design is not feasible	3D design check	Mock-up
	Design review	Review deliverables
	Execute system test to validate critical specifications	System integration testing
	Scope is only “drop-in-replacement”	Drop-In replacement
Failure can be made during maintenance	Organize test session with inexperienced user, followed by process FMEA	Test with dummies
Failure can be made during production	Define/review factory acceptance test (FAT)	Factory acceptance test
	Plan system integration test to test performance	System integration testing
Failure can be made during use	Check validity of input values	Embed knowledge in system
Human safety is not considered	Review the design	Review deliverables
	System testing	System integration testing
Integrated system behavior is not considered	3D design check	Mock-up
	Bench testing	Hardware in the Loop
Maintenance is not efficient	Make upgrade plan	Clear plan
Project plan is not managed	Make clear plan	Clear plan
	Design review	Review deliverables
Requirement is not met	Extend factory acceptance test (FAT)	Factory acceptance test
Resources (hardware) are scarce	Use test bench	Hardware in the Loop
Resources (people) are scarce	Create resource allocation plan	Clear plan
System deteriorates over time	Execute lifetime tests of a component at the supplier	Module Testing
System is not reliable	Organize lifetime test	Module Testing

Appendix H Failure category statistics (based on FMEA and Retrospective reports)

This appendix provides a statistical overview of the data from FMEA worksheets and the retrospective reports. This overview relates the worksheet and report data based on the *failure categories*, and it can provide insight on how the two data sources relate to each other (see Table H-1).

The table is sorted on the first column in alphabetic order. The second column is an overview of the average severity ranking and the occurrences of the failure categories in the FMEA worksheets. These values were determined during FMEA analysis (section 3.4). This second column allows the possibility of relating the FMEA categories to the retrospective remarks. The third to sixth columns provide the amount of occurrences/assigned remarks and their ratio. With the ratio value, determining the focus of the FMEA and retrospective reports is easier. The *failure categories* with the highest average severity ranking (column 2) are marked in yellow. This column is used to sort the table. The three most mentioned *failure categories* (in columns 3, 4, 5, and 6) are also marked in yellow.

Table H-1: Failure category statistics

FMEA: Failure categories	FMEA: Average severity ranking	FMEA: Occurrences of failure category		Retrospective: Positive remarks		Retrospective: Negative remarks		Retrospective: Improvements	
Human safety is not considered	8.6	83	7.0%	0	0.0%	0	0.0%	0	0.0%
Customer expectation is not managed	6.7	12	1.0%	6	1.6%	7	1.2%	0	0.0%
Behavior of system is unknown	6.2	25	2.1%	0	0.0%	0	0.0%	0	0.0%
Integrated system behavior is not considered	6.2	153	12.9%	7	1.9%	23	4.1%	1	1.1%
Requirement is not defined	6.2	27	2.3%	10	2.7%	23	4.1%	3	3.3%
Test coverage is too low	6.1	10	0.8%	29	7.9%	54	9.6%	8	8.9%
System performance varies	5.9	23	1.9%	0	0.0%	0	0.0%	0	0.0%
Project plan is not managed	5.8	24	2.0%	50	13.6%	79	14.0%	18	20.0%
Project scope is not managed	5.8	8	0.7%	16	4.3%	35	6.2%	2	2.2%
Responsibility is not assigned	5.8	9	0.8%	2	0.5%	21	3.7%	1	1.1%
System deteriorates over time	5.8	107	9.0%	27	7.3%	14	2.5%	7	7.8%
Requirement is not met	5.7	92	7.8%	32	8.7%	21	3.7%	4	4.4%
Failure can occur during production	5.6	103	8.7%	11	3.0%	7	1.2%	0	0.0%
Status of system cannot be monitored	5.5	33	2.8%	2	0.5%	0	0.0%	0	0.0%
Vendor is locked-in	5.5	17	1.4%	0	0.0%	0	0.0%	0	0.0%
Design is not feasible	5.4	95	8.0%	4	1.1%	1	0.2%	0	0.0%
System is not reliable	5.3	125	10.5%	11	3.0%	0	0.0%	0	0.0%
Resources (hardware) are scarce	5.1	14	1.2%	10	2.7%	40	7.1%	7	7.8%
Configuration management is inaccurate	5.0	10	0.8%	0	0.0%	1	0.2%	2	2.2%
Failure can be made during use	5.0	23	1.9%	1	0.3%	0	0.0%	0	0.0%
Resources (people) are scarce	4.9	14	1.2%	14	3.8%	27	4.8%	7	7.8%
Maintenance is not efficient	4.8	53	4.5%	0	0.0%	5	0.9%	0	0.0%
Project intake is not managed	4.8	23	1.9%	7	1.9%	23	4.1%	4	4.4%
Maintainability of the system becomes difficult	4.5	15	1.3%	16	4.3%	23	4.1%	1	1.1%
Production is not efficient	4.4	24	2.0%	108	29.3%	151	26.8%	25	27.8%
Design documentation is not available	4.3	10	0.8%	4	1.1%	9	1.6%	0	0.0%
Utilization of system is not optimal	4.3	23	1.9%	1	0.3%	0	0.0%	0	0.0%
Failure can occur during maintenance	2.8	30	2.5%	0	0.0%	0	0.0%	0	0.0%

Appendix I Positive formulation of failure category

This appendix provides an overview of the translation from *failure category* to *success category*. The *success category* is a positive formulation of the *failure category*. Table I-1 translated all *failure categories* from Table G-1.

Table I-1: Translation of failure category to success category

Failure categories	Success category
Behavior of system is unknown	Behavior of system is known
Configuration management is inaccurate	Configuration management is accurate
Customer expectation is not managed	Customer expectation is managed
Design documentation is not available	Design documentation is available
Design is not feasible	Design is feasible
Failure can occur during maintenance	Maintenance action is improved to prevent failure
Failure can occur during production	Production is improved to prevent failure
Failure can occur during use	Usability is improved to prevent failure
Human safety is not considered	Human safety is considered
Integrated system behavior is not considered	Integrated system behavior is considered
Maintainability of the system becomes difficult	System remains maintainable
Maintenance is not efficient	Maintenance is efficient
Production is not efficient	Production is efficient
Project intake is not managed	Project intake is managed
Project plan is not managed	Project plan is managed
Project scope is not managed	Project scope is managed
Requirement is not defined	Requirement is defined
Requirement is not met	Requirement is met
Resources (hardware) are scarce	Resources (hardware) are available
Resources (people) are scarce	Resources (people) are available
Responsibility is not assigned	Responsibility is assigned
Status of system cannot be monitored	Status of system can be monitored
System deteriorates over time	System functionality improves
System is not reliable	System is reliable
System performance varies	System performance is constant
Test coverage is too low	Sufficient test coverage
Utilization of system is not optimal	Utilization of system is improved
Vendor is locked-in	Vendor is not locked-in

Appendix J Result of retrospective report analysis

This appendix summarizes the data from the retrospective reports in the context of the *failure categories* and FMEA worksheets. In some cases, a practice name is assigned to a *failure category*, which is inspired by the solution. The same name is used when solutions appears to be the same.

A worksheet has the descriptions *current design control* and *recommend action*. The positive remarks can be seen as examples of *current design control* proven to be successful. The retrospective improvement remarks can be seen as examples of *recommended action*. The negative remarks can be seen as examples of *failure categories* that did occur. For each *failure category*, a limited amount of retrospective examples is given. This limitation helps maintain the overview easy to read. When the reports do not have examples, the cell is marked with "-" (Table J-1). In the fifth column, practice names are provided. Refer to Table J-2 for details on the mapping between practice name and solution.

Table J-1: Failure category overview based on retrospective reports

Failure category	Negative remarks	Positive remarks	Improvement remarks	
Problem	Example of problem	Solution	Solution	Practice name
Behavior of system is unknown	-	-	-	
Configuration management is inaccurate	Configuration management is a mess	-	Create deliverables on a regular basis Create a configuration management plan	
Customer expectation is not managed	Communication to users regarding changes & new features is not sufficiently good Stakeholders are not involved Demo can provide wrong impression on current status Stakeholders expectations are too high	Obtain customer acceptance before delivering a feature Ask for feedback from the customers (e.g., Beta sites)	-	
Design documentation is not available	Missing top level overview Documentation is not up to date Insufficient time to do proper design	Create design incrementally Organize design sessions Focus on design	-	
Design is not feasible	Idea cannot be implemented	Obtain feedback from users Incremental development Organize design sessions	-	Design by team
Failure can occur during maintenance	-	-	-	
Failure can occur during production	Time pressure No attention for quality Problems are not solved	Introduce procedures Design together Review each other's work Use a template for functional requirements specification document	-	Design by team Review deliverables
Failure can occur during use	-	Encapsulate details by providing high level abstraction interface	-	
Human safety is not considered	-	-	-	

Integrated system behavior is not considered	Performance issues Calibration failures No time to concentrate on the big picture No correct error message is given	Arrange Beta sites that use Beta version and ask for feedback Verify feature on a real system Customer focus	Obtain clear requirements. Make decision on what to implement based on cost/benefit analysis	Early confrontation
Maintainability of the system becomes difficult	Deviation from standards Use of “over-complicated” & non-mature technologies Code difficult to read/obtain overview No time for refactoring	Select simple solution Refactor Fix issues (part of backlog management) Create stable codebase Increase quality of code by conducting reviews	Track design decisions that create future maintainability issues. This overview provides insight on the size of the maintenance problem that is being created.	Debt management Incremental improvement Keep it simple Review deliverables
Maintenance is not efficient	Poor description of problems Problem report not assigned to the right person Unclear how to manage problem report backlog Reentering issues	-	-	
Production is not efficient	Too many meetings Too many simultaneous activities Infrastructure is a bottleneck for development Engineers not located together	Prevent disruption (delivery of other teams) of tested baselines prior to delivery Focus on one feature Automate repetitive work Have continuous attention for process improvements Continuous integration and delivery	Organize knowledge transfer Develop software in pairs Developers will verify their own work on a system Create small work packages. This way it is easier to track the progress of the team.	Automate repetitive work Boundary involvement Constructive disagreement Co-ownership Design by team Document overview/interface Increase system knowledge Incremental architecture Incremental improvement Knowledge transfer Work in parallel Prioritize for focus Empower the team Short lines
Project intake is not managed	Cooperation with other project is far from optimal Commitments are not met Deliverables have a poor quality	System construction timing determines the delivery deadlines	Outsource team member to other team (from which the intake is received) Organize regular project meetings Organize team interaction (design and code review meetings) Discuss cooperation problems on management level	Boundary involvement Common plan
Project plan is not managed	Focus is only on solving issues Roadmap is not clear Project status is not communicated Project is poorly planned	Create roadmap (define features, define milestones) Create planning/work breakdown and track the plan Communicate project status Create focus on tasks	Organize regular meeting with stakeholder to discuss priorities and requirements of features Monitor and categorize all unplanned tasks Create short feedback loop to the team on the progress of the project Inform individuals on what is expected from them Improve estimation of the work	Clear plan Common plan Prioritize for focus

Project scope is not managed	Implemented features that were not requested No clear common view on project scope Conflicting priorities between projects Content of backlog not managed	Hold on to agreed milestones Make project goals clear Determine dependencies with others Decide to drop features with have lower priority when target cannot be met	Let management decide how to manage the delta between workload and available resources Clearly define the content of the work	Boundary involvement Clear plan Prioritize for focus Deliver or Delay
Requirement is not defined	Inconsistent wishes Last minute requirement changes Unexpected feature requests Unclear description of feature request	Discuss feature with customer and receive feedback Incremental delivery of feature. This allows early feedback on the feature. Proactively gather requirements (e.g., discuss with stakeholders, consult suppliers, execute investigation) Write requirements for each feature in a functional requirements specification document	Assign responsibilities for requirements definition Enforce the importance of requirement definition to all people involved Ask for requirements at management level	Boundary involvement Clear specification
Requirement is not met	No time reserved for extra fixes after testing by system engineer Shortcuts required because of limited time Much unforeseen work Specification is not implemented	Determine when feature is complete Determine content of software release	Execute code reviews Focus on solving Give support to clients that use the functionality	
Resources (hardware) are scarce	All systems are down Poor performance of prototypes No effective system test planning No system available with the correct configuration	Dedicated systems for testing Use factory systems (that are being produced for customers) for testing Use prototypes for testing Organize system reservation process	Use simulators Plan and reserve system time ahead Obtain reconfirmation of reserved system time prior to testing, because system can be down Bundle items that require system verification	Hardware in the Loop Simulator in the Loop
Resources (people) are scarce	Losing a team member Availability of team member reduced because of other priorities More ideas and activities than we have capacity to do	Add resources to the team Make balance between team capacity to project execution	Claim resources for a short period for a specific task (e.g., Testing, integration) Create overview of the burned effort on a feature. This insight can help understand how time is spent Create overview of the allocation claims on the resources. This insight can help set priorities Involve core knowledge efficiently (invite required external resource to join team meetings) Communicate to management the impact of a resource leaving the team	Tester in team
Responsibility is not assigned	Conflict with priorities of other team Product owner is not visible Unclear how to manage failing unit test Responsibilities of roles unclear	Assign one person to have the overall decision authority Define stakeholders with the responsibility to accept the deliverables	Revise the assignment of stakeholders during the project Create an overview of the stakeholders for each area of expertise	Product owner

Status of system cannot be monitored	-	Create ability to execute diagnostics on system	-	System monitor
System deteriorates over time	Software is unstable (crashes/patches) Features do not work anymore because of modification Deliverables are not stabilizing	Improve quality of deliveries Improve codebase stability Solve (blocking) issues	At the end of the development phase, do not implement new requirements, but only concentrate on software crashes Increase priority of fixing issues through the defect handling process. This process is controlled by the change control board (CCB). Escalate issues to management Team members should follow way of working (review code, offline and online testing, etc.)	
System is not reliable	-	Create awareness on reliability Introduce tooling for tracking stability	-	
System performance varies	-	-	-	
Test coverage is too low	Not all high risk issues have been verified No testing is done on a real system No test plan available Too many new bugs found in the end phase	Create Unit tests Execute automated tests (e.g., weekend runs, nightly runs, smoke tests) Remote testing when specific hardware is on another location Execute system testing (e.g., verify deliverables, regression test)	Write unit tests Use simulation for (offline) testing Test deliverables Improve facilities for local testing	Duration runs Hardware in the Loop Simulator in the Loop Tester in team Unit testing
Utilization of system is not optimal	-	Made usage simpler	-	Customer centric development
Vendor is locked-in	-	-	-	

The first column of Table J-2 describes the problem. Such problems are the *failure categories* determined with FMEA Analysis (section 3.4). The second column contains the solution to the problem. These solutions are retrieved from the reports (positive remarks and improvement remarks). Most can be found in Table J-1, the others are copied from the reports. The third column contains the practice name, which is inspired by the solution. The same name is used when solutions appear to be the equal.

Table J-2 Mapping of practice name to solution

Problem (Failure category)	Solution (example of retrospective reports)	Practice Name
Design is not feasible	Organize design sessions	Design by team
Failure can be made during production	Design together	Design by team
	Review each other's work	Review deliverables
Integrated system behavior is not considered	Arrange Beta sites that use Beta version and ask for feedback	Early confrontation
Maintainability of the system becomes difficult	Fix issues (part of backlog management)	Debt management
	Create stable codebase	Incremental improvement
	Select simple solution	Keep it simple
	Increase quality of code by conducting reviews	Review deliverables
Production is not efficient	Automate repetitive work	Automate repetitive work
	A regular meeting with people on whom the team relies is arranged	Boundary involvement
	In order to make meetings more efficient, the team members follow "constructive disagreement" approach	Constructive disagreement
	Share knowledge. No islands: general knowledge is owned by the complete team	Co-ownership
	Think of solutions together	Design by team
	The "old" team members write the high-level overview of the architecture	Document overview / interface
	Developers verify their own work on a system	Increase system knowledge
	Step-by-step development of architecture	Incremental architecture
	Have continuous attention for process improvements	Incremental improvement
	Organize knowledge transfer	Knowledge transfer
	Do parallel development	Work in parallel
	Focus on one feature	Prioritize for focus
	Self empowerment of the Team	Empower the team
	Put team members together	Short lines
Project intake is not managed	Outsource team member to other teams (from which the intake is received) Organize team interaction (design and code review meetings)	Boundary involvement
	System construction timing determines the delivery deadlines	Common plan
Project plan is not managed	Create planning/work breakdown and track the plan	Clear plan
	Create roadmap (define features, define milestones)	Common plan
	Create focus on tasks	Prioritize for focus
Project scope is not managed	Determine dependencies with others	Boundary involvement

	Make project goals clear	Clear plan
	Decide to drop features with lower priority when target cannot be met	Prioritize for focus
	Plan additional release for missed features	Deliver or Delay
Requirement is not defined	Proactively gather requirements (e.g., discuss with stakeholders, consult suppliers, execute investigation)	Boundary involvement
	Write requirements for each feature in a functional requirements specification document	Clear specification
Resources (hardware) are scarce	Dedicated systems for testing	Hardware in the Loop
	Use simulators	Simulator in the Loop
Resources (people) are scarce	Claim resources for a short period for a specific task (e.g., Testing, integration)	Tester in team
Responsibility is not assigned	Assign one person with the overall decision authority	Product owner
Status of system cannot be monitored	Create ability to execute diagnostics on system	System monitor
Test coverage is too low	Execute automated tests (e.g., weekend runs, nightly runs, smoke tests)	Duration runs
	Execute system testing (e.g., verify deliverables, regression test)	Hardware in the Loop
	Use simulation for (offline) testing	Simulator in the Loop
	Adopt tester in team. This will increase the added value.	Tester in team
	Write unit tests	Unit testing
Utilization of system is not optimal	Made usage simpler	Customer centric development

Appendix K Practices with context and forces

This appendix provides an overview of the practices for which the context and forces are described. The problem description is summarized into a "How can .." question (see Table K-1). This helps to focus on the core of the problem.

Table K-1: Pattern description with context, problem, and forces

Practice	Context	Problem	Forces
Common Plan	Stakeholder expectations, the vision of a particular stakeholder individual or group, result when they specify what is desired as an end state or as an item to be produced and put bounds upon the achievement of the goals. These bounds may encompass expenditures (resources), time to deliver, performance objectives, or other less obvious quantities, such as organizational needs or geopolitical goals (NASA, 2007, p. 34) The team should also ensure that the goals can be met and failure modes are considered, as is the entire system (NASA, 2007, p. 63) High-technology firms NPD teams work on projects that are inherently complex. Greater planning aids the effort, fostering integration, by providing the predictability and control required for progress on these uncertain and risky projects (Nakata & Im, 2010, p. 7)	How can a complex multidisciplinary project be predictable?	Projects that are inherently complex (Nakata & Im, 2010, p. 7)
Hardware in the Loop	Becomes as close to the actual concept of operation as possible to support verification and validation when the operational environment is difficult or expensive to recreate (NASA, 2007, p. 96) Development of mechatronic systems requires collaboration among experts from different design domains (Alvarez Cabrera et al., 2010) In practice, specific models are developed to perform tests at different stages of the design. Because of the use of domain-specific modeling tools, such models usually correspond to a specific system perspective, such as either the electrical or mechanical aspects, or continuous dynamics and discrete, sequential behavior (Bradley, 2010)	How can integration problems at the end of the project be prevented?	Fault detection and diagnoses can only be executed when the final system is available (Boucher & Houlihan, 2008) Because of the use of domain-specific modeling tools, such models usually correspond to a specific system perspective, such as either the electrical or mechanical aspects, or continuous dynamics and discrete, sequential behavior (Bradley, 2010) Models and simulations do not exactly represent the real system. Therefore, system verification cannot be executed. (Alvarez Cabrera et al., 2010) Resources (hardware) are scarce (FMEA Analysis, see 3.4) Inter disciplinary collaboration
Simulator in the Loop	Provide insight into trends and tendencies of system and subsystem performance that might not otherwise be possible because of hardware limitations (NASA, 2007, p. 96) Testing early to identify problems on system level (Boucher & Houlihan, 2008) Increase the ability to predict system level behavior prior to testing (Boucher & Houlihan, 2008)	How can integration start when not all disciplines have delivered their part of the product?	Resources (hardware) are scarce (Retrospective Analysis, section 3.5)

Appendix L Pattern sequences

This appendix describes the relationship between patterns.

Such new patterns are placed into sequence with the existing patterns of (Coplien & Harrison, 2005) based on the unresolved force(s). These are determined based on the resulting context of the solution provided by the first pattern and the problem statement of the second pattern (see Table L-1).

Table L-1: Pattern sequences based on unresolved forces

Common Plan	
Community of Trust → Common Plan	
Solution	"Do things that explicitly demonstrate trust. Managers, for example, should make it overtly obvious that they facilitate the achievement of organizational goals, rather than playing a central role to assert control over people. Take visible actions to give developers control over the process."
Unresolved forces	The deliverables between disciplines and the responsibilities need to be clear. Trust alone is insufficient for managing a project because the agreements can be influenced by external factors (e.g., third-party deliverables) and internal factors (e.g., resources allocation). These factors can be triggered from outside the Community of Trust.
Problem statement	"Orchestration of deliverables is required to manage the schedule." (see 4.2.3)
Common Plan→ Size The Schedule	
Solution	"Create a common plan of the deliverables that have dependencies between disciplines." (see 4.2.3)
Unresolved forces	A plan that considers all the deliverables between dependencies might result in an unrealistic schedule.
Problem statement	"Both overly ambitious schedules and overly generous schedules have their pains either for the developers or the customers."
Hardware in the Loop	
Incremental Integration → Hardware in the Loop	
Solution	"Provide a mechanism to allow developers to build all of the current software periodically. Developers should be discouraged from maintaining long intervals between check-ins. Developers should at any time also be able to build against any of the Named Stable Bases or the newest check-in software."
Unresolved forces	When developing a mechatronic system, the Named Stable Bases can be seen as a mechatronic system (hardware and software). This is only available at the end of the project because hardware is still under development, which means that frequent integration is not possible.
Problem statement	"It is important to identify multidisciplinary integration problems early in the development cycle." (see 4.2.4)
Get On With It → Hardware in the Loop	
Solution	"As soon as you have confidence about project direction, start developing area's in which you have high confidence."
Unresolved forces	When developing a mechatronic system, there might be high confidence of the software direction. This development is blocked when it requires hardware that still has a low confidence. For instance, it is clear which third-party Application Programming Interface (API) needs to be integrated, but the hardware requirements for that third-party are still unclear.
Problem statement	"It is important to identify multidisciplinary integration problems early in the development cycle." (see 4.2.4)
Hardware in the Loop → Private World	
Solution	"Build a hardware setup that can be used to verify the current state of the development." (see 4.2.4)
Unresolved forces	When developing with a multidisciplinary team, each discipline can have its own preferences on the setup. For instance, the software discipline wants to test their latest features and fixes, whereas service engineering wants to verify features against a previous Named Stable Bases (which can include other hardware).
Problem statement	"How can we balance the need for developers to use current revisions, based on periodically baselines, with the desire to prevent developers from experiencing undue grief by having development dependencies change underneath them?"
Simulator in the Loop	
Incremental integration → Simulator in the Loop	
Solution	"Provide a mechanism to allow developers to build all of the current software periodically. Developers should be

	discouraged from maintaining long intervals between check-ins. Developers should at any time also be able to build against any of the Named Stable Bases or the newest check-in software."
Unresolved forces	When developing a mechatronic system, the Named Stable Bases can be seen as a mechatronic system (hardware and software). This is only available at the end of the project because hardware is still under development, which means that frequent integration is not possible.
Problem statement	"It is important that system behavior can be verified before integration on a real system begins."(see 4.2.5)
Get On With It → Simulator in the Loop	
Solution	"As soon as you have confidence about project direction, start developing area's in which you have high confidence."
Unresolved forces	When developing a mechatronic system, there might be high confidence of the software direction. This development is blocked when hardware that still has a low confidence is required. For instance, it is clear which behavior modules should have, but building hardware setup is too expensive.
Problem statement	"It is important that system behavior can be verified before integration on a real system begins."(see 4.2.5)
Hardware in the Loop → Simulator in the Loop	
Solution	"Build a hardware setup that can be used to verify the current state of the development." (see 4.2.4)
Unresolved forces	Multiple teams can develop a mechatronic system. These teams can be on different locations and time zones. Supporting these teams with hardware setups can become expensive. The creation of Named Stable Bases can be done by executing unit in module tests. For a large software base, this testing can take hours. Therefore, this is usually done at night. In one night, multiple Named Stable Bases can be created (different releases, configurations, etc.). Supporting this by hardware setups is expensive.
Problem statement	"It is important that system behavior can be verified before integration on a real system begins."(see 4.2.5)
Simulator in the Loop → Private World	
Solution	"Build simulator(s) that can simulate (sub-) system behavior." (see 4.2.5)
Unresolved forces	During development of a simulator, its behavior might change over time. For instance, when a simulator is implemented incrementally, the initial behavior only supports happy flow. When the simulator matures, it can verify parameters, autonomously send events, or have a completely different start-up procedure.
Problem statement	"How can we balance the need for developers to use current revisions, based on periodically baselines, with the desire to prevent developers from experiencing undue grief by having development dependencies change underneath them?"

Appendix M Pattern feedback

This appendix contains a summary of the feedback retrieved from the reviewers on all patterns. This feedback was gathered based on a review form (Table 3.7-1). The result is presented with the same form. The average time required by the reviewers to review, and the average rating of the description, is presented between square brackets ("[..]").

Common Plan	
Review feedback on pattern	**Reviewer's feedback**
How long did the review take? (e.g., 15 min)	[28 min]
Is the pattern description clear? *Rating 1...10 (1 is bad; 10 is excellent)*	
Context (Prologue)	[7.6]
Problem	[7.4]
Forces/Trade-offs	[7.6] From here I understood the context, not from the beginning
Solution	[7.9]
Discussion on why pattern works	[7.8]
Pattern being used	
Are there other forces/trade-offs? (Yes/No) (Please explain your reasoning and rational for your view)	Can contain constraints Could also be influenced by: standards and policy, regulatory requirements, QA, testing Different mindset/approach perception, quality about the deliverable and/or goal, between departments/disciplines.
Do you think that the proposed solution will solve the problem? (Yes/No) (Please explain your reasoning and rational for your view)	We always use a action plan (*Dutch: "plan van aanpak"*) in multidisciplinary projects this way Maybe strengthened by use of a stakeholder map You hint at standup meetings, but it also uses other scrum methods, such as poker planning/sprints
Reviewer comments	
Do you know another pattern name (alias) that also covers the pattern description?	Scrum sprint and backlog meeting Shared Plan (It is a plan with shared interest)
Additional feedback (if applicable)	Not all the *tradeoffs of forces* seem to completely match with the problem of *common plan*. *Throw over wall* and *status* are more like communication issues than the lack of a common plan.

Hardware in the Loop	
Review feedback on pattern	**Reviewer's feedback**
How long did the review take? (e.g. 15 min)	[22 min]
Is the pattern description clear? *Rating 1...10 (1 is bad; 10 is excellent)*	
Context (Prologue)	[8.2]
Problem	[7.1] Description can be more concrete
Forces/Trade-offs	[8.0]
Solution	[7.9]
Discussion on why pattern works	[7.6]
Pattern being used	
Are there other forces/trade-offs? (Yes/No) (Please explain your reasoning and rational for your view)	For the setup itself a skilled human resource might be needed, which is not related to cost but a scarce resource. Hardware deliverables typically have a high dependency on (multiple) external suppliers and relatively long lead times (LLI = long lead items), causing lack of availability of the hardware. Missing research; testing multiple components, which is best. Allows reliability on component level to be partially tested/estimated Allows validation and verification of use cases and test scripts. New insights during development of both hard- and software might not be communicated because of lack of right communication channels.
Do you think that the proposed solution will solve the problem? (Yes/No) (Please explain your reasoning and rational for your view)	No, depends on the product. To use cheaper or other parts/components for testing can also increase error rate when implementing the correct part. or in the worst case, if the correct part does not full fill your requirements, it will be noticed in a late stage of your development. No, not always possible No, specification and design issues do not usually appear at this phase.
Reviewer comments	
Do you know another pattern name (alias) that also covers the pattern description?	No, although from a hardware point of view the pattern could be called "Software in the loop." HW integration testing
Additional feedback (if applicable)	What should be the balance between investing in the hardware twice and/or a surrogate hardware solution against investing more time in the final integration phase with just one final piece of hardware? The need for a temporary hardware setup (FUMO, PROTO) is not only dictated by integration needs with software, but also for early verification purposes of the hardware itself (e.g., product handling on transport system). Regarding threats to the setup, I've seen prototypes created that were designed to be commercially useless to prevent losing it prematurely. Also seen in experimentation/prototype projects

Simulator in the Loop	
Review feedback on pattern	**Reviewer's feedback**
How long did the review take? (e.g., 15 min)	[23 min]
Is the pattern description clear? *Rating 1...10 (1 is bad; 10 is excellent)*	
Context (Prologue)	[7.1] A little lightweight description
Problem	[7.2] Not formulated as problem
Forces/Trade-offs	[7.1]
Solution	[8.1]
Discussion on why pattern works	[7.7] I would expect that restrictions are also mentioned. Simulator is also is complementary to model based development and model based testing
Pattern being used	
Are there other forces/trade-offs? (Yes/No) (Please explain your reasoning and rational for your view)	When and why is this pattern not a solution? Building a simulator is an investment as well Simulators can also be used as production test tooling. - System also includes the environment which can also be simulated - Simulation is never 100%; so is always an abstraction of the sum of part of the environment and the sum of the sub systems simulated [gives a coverage of 40%] You could also add a caveat that the simulation can also have its own design flaws and bugs. While hardware is still in development the actual behavior might be still unclear. Risk is that wrong assumptions are made. Maintainability & development costs can be very high compared to the product, which might lead to the decision not to invest in *simulator*. Yes, with the remark that building a simulator must been seen as serious business. Many hours of development work and communication are needed to have a good usable simulator. Ideally one could think of a model driven behavior description that is used in hw development. Any updates might be taken over in sw sim. Also there is the risk of false feeling of correct working of the product. It does not reduce the late found integration errors in this case.
Do you think that the proposed solution will solve the problem?	Yes, it is more a tool to speed up the development. Yes, expensive solution although
Reviewer comments	
Do you know another pattern name (alias) that also covers the pattern description?	SW system under test [SUT]
Additional feedback (if applicable)	The problem statement is more or less the same as "Hardware in the loop," yet the solution is different depending on the need of a temporary hardware setup and simulation needs. Simulators are handy even when the hardware is available. It takes no physical space. It can also help in situations where the hardware is located in the far distance of the developer site. Using keywords like *mechatronics* in the pattern descriptions makes the patterns less general. What is the definition of a system? I assume it is meant the lack of a proper hardware environment? The way how the *problem statement* is formulated, does not actually read like a problem. Often used in safety critical systems

Appendix N Mapping literature practices to failure categories

This appendix shows how the practices found in the publications (section 3.3) relate to the failure categories determined during FMEA worksheet analysis (section 3.4). This relationship is made based on the problem description presented in Table D-1.

Table N-1: Mapping practices to failure category

Practice	Problem	Failure category
Common Plan	Projects suffer from overlapping responsibilities and ambiguous command chains, generating psychological distance and intense conflict among team members (Nakata & Im, 2010, p. 7)	Responsibility is not assigned
Hardware in the Loop	Models and simulations do not exactly represent the real system. Therefore, system verification cannot be executed. (Alvarez Cabrera et al., 2010)	Integrated system behavior is not considered
Hardware in the Loop	Many integration problems at the end of project. (Alvarez Cabrera et al., 2010), (David Bradley, 2010)	Production is not efficient
Hardware in the Loop	Fault detection and diagnoses can only be executed when the final system is available. (Boucher & Houlihan, 2008)	Production is not efficient
Simulator in the Loop	System level issues are discovered late in the design process. As a consequence, design options are reduced because critical decisions are already made. (Boucher & Houlihan, 2008)	Integrated system behavior is not considered

Appendix O Referenced patterns

This appendix provides a summary of the patterns (also known as patlets) referred to in the pattern descriptions (see section 4.2). The summary of these patterns consists of the problem and the solution (*"If…Then…"*). The descriptions are retrieved from (Coplien & Harrison, 2005). The pattern names are sorted alphabetically.

Table O-1: Patlets of referred patterns

Pattern name	Summary
Application Design is Bounded by Test Design	If you want to organize the interworking between test and software developers, Then: organize the process so that "Application Design Is Bounded By Test Design."
Architect controls Product	If a project has a long life, Then: use the architect to carry the vision forward and serve as the long-term keeper of architectural style.
Architecture Team	If you are building a system too large or complex to be thoroughly understood by a single individual, Then: build a team with both the responsibility and the power to create the architecture.
Build Prototypes	If early acquired requirements are difficult to validate without testing, Then: build a prototype whose purpose is to understand requirements.
Code Ownership	If you need responsibility for code and want to build on "Domain Expertise In Roles," Then: grant various individuals responsibility for the overall quality of the code.
Conway's Law	If organization structuring concerns are torn between geography, expertise, politics, and other factors, Then: align the primary organizational structuring with the structure of the business domains and the structure that will be reflected in the product architecture.
Developer controls Process	If you need to orchestrate the activities of a given location or feature, Then: put the Developer role in control of the succession of activities.
Engage Customer	If you want to manage an incremental process that accommodates customer input, and if you want the customer to feel loved, Then: engage customers after Quality Assurance and project management are prepared to serve them.
Engage Quality Assurance	If developers cannot be counted on to test beyond what they already anticipated going wrong, Then: engage QA as an important function.
Face-to-Face before Working Remotely	If a project is divided geographically, Then: begin the project with a meeting of everyone in a single place.
Get on With It	If you are starting a project and have sufficient information to start parts of it, Then: do not wait until you have a complete schedule before starting to do parts of the project.
Group Validation	If you want to avoid being blindsided in quality assurance, Then: engage Customers and Developing In Pairs and others to validate the system.
Holistic Diversity	If Development of a subsystem requires many skills, but people specialize, Then: create a single team from multiple specialties.
Incremental Integration	If you want developers to be able to test changes before publishing them, Then: allow developers to build the entire product code independently to allow testing with the latest base (not with the latest *Named Stable Bases*).
Named Stable Bases	If you want to balance stability with progress, Then: have a hierarchy of named stable bases against which people can work.
Organization follows Location	If you need to distribute work geographically, communications suffer, but you can limit the damage if work is partitionable. Therefore: organize work at locations so that groups of people that work together are at the same location.
Owner per	Ensure every deliverable has one and only one owner.

Deliverable	
Patron Role	If you need to insulate Developers so that Developer Controls Process and provide some organizational inertia at the strategic level, Then: identify a patron to whom the project has access, who can champion the cause of the project.
Private Worlds	If you want to isolate developers from the effects of changes, Then: allow developers to have private workspaces that contain the entire build environment.
Programming Episode	If you need to divide work across time, Then: do the work in discrete episodes with mind share to commit to concrete deliverables.
Shaping Circulation Realms	If you require mechanisms to facilitate the communication structures necessary for good group formation, Then: shape circulation realms.
Size the Schedule	If the schedule is too long, developers become complacent; however, if it is too short, they become overtaxed. Therefore: reward meeting the schedule, and maintain two sets of books.
Stand-up Meeting	If there are pockets of misinformation or people out of the loop, Then: hold short daily meetings to socialize emerging developments.
Surrogate Customer	If you require answers from your customer, but no customer is available to answer your questions, Then: create a surrogate customer role in your organization to play advocate for the customer.
Unity of Purpose	If a team is beginning to work together, Then: ensure all members agree on the purpose of the team.

Appendix P Pattern Language

This appendix presents two adapted pattern languages from (Coplien & Harrison, 2005).

In Figure P-1 and Figure P-2, the patterns of (Coplien & Harrison, 2005) are presented. The new patterns (4.2) are integrated within this language. The new patterns can be identified by their gray squares. To understand the relationship between the patterns, please refer to section 4.2.2.

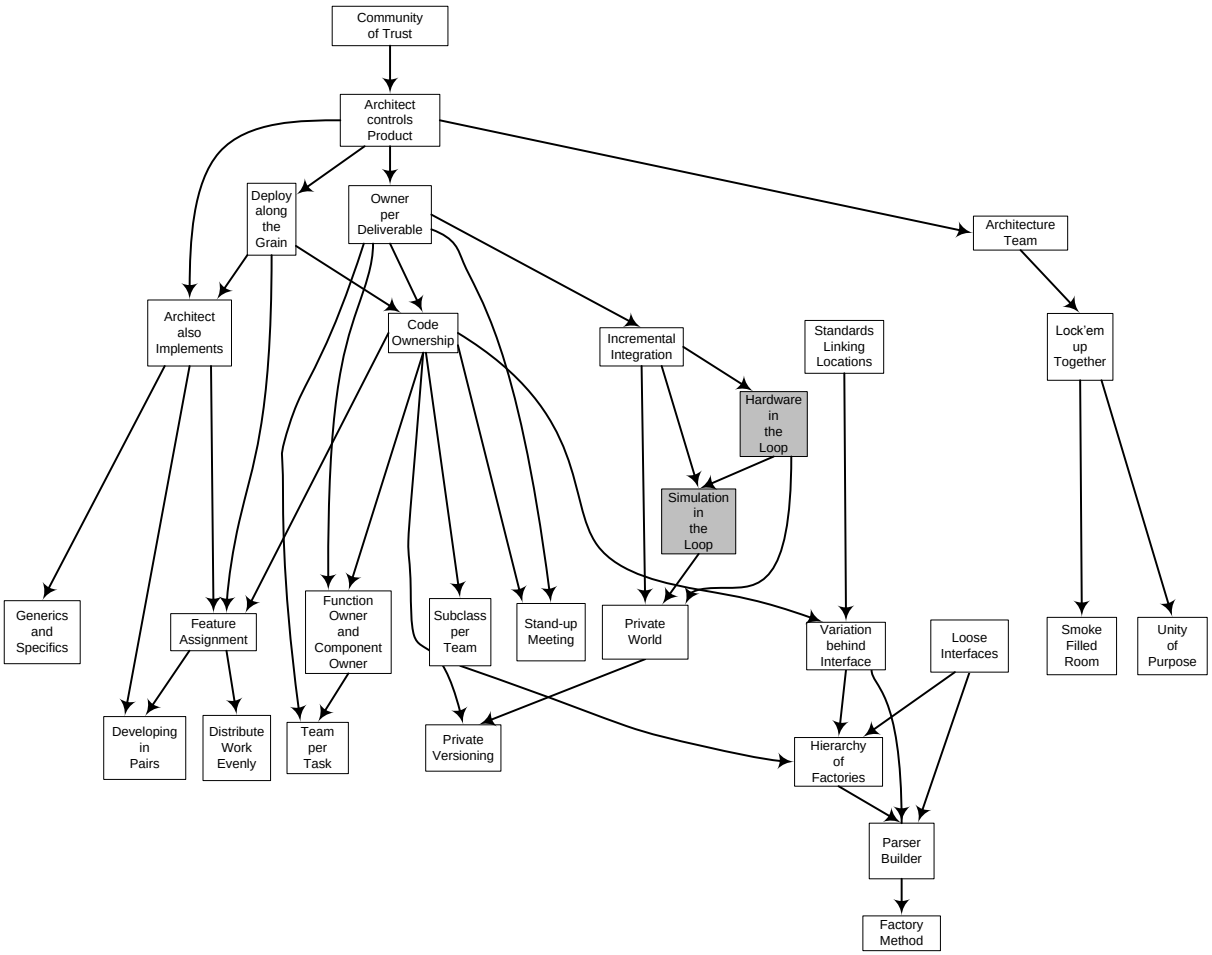


Figure P-1: Integration of patterns (gray squares) in people and code pattern language

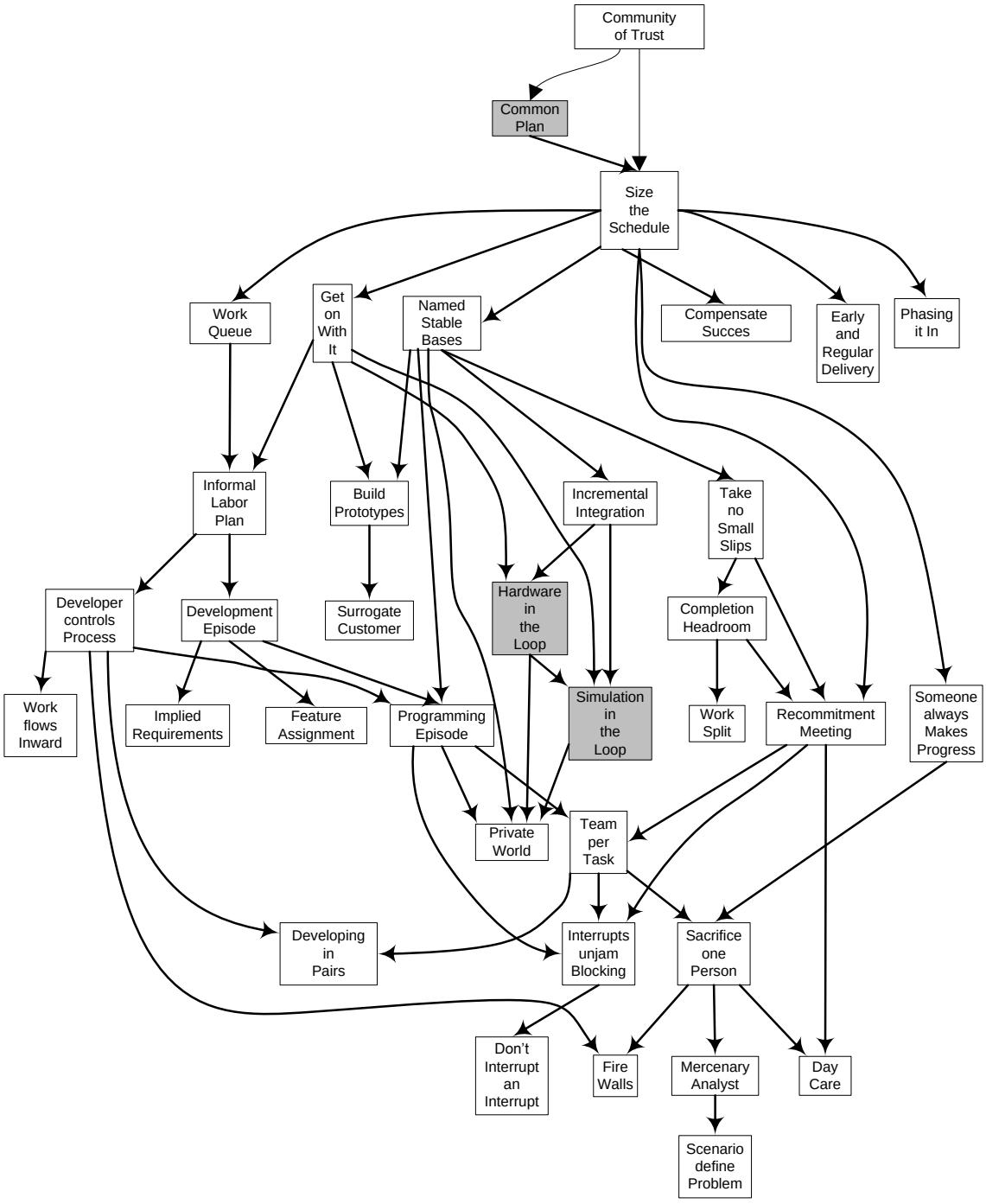
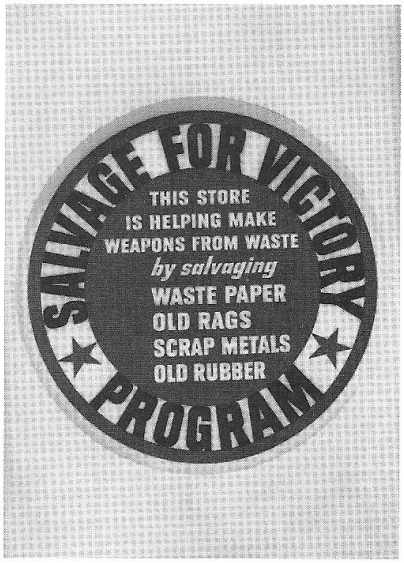


Figure P-2: Integration of patterns (gray squares) in project management pattern language

Appendix Q Examples of form layout

This appendix provides two pattern descriptions of (Coplien & Harrison, 2005). These illustrate the form layout used in (Coplien & Harrison, 2005), which describe a total of 93 patterns.

4.2.12 Unity of Purpose **



... the team is beginning to come together. Team members may come from different backgrounds and may bring with them many different experiences.

✣ ✣ ✣

Many projects have rocky beginnings as people struggle to work together.

Often, people have different ideas about what the final product should be. In fact, the final product may well be a pretty fuzzy concept. Yet people must have a consistent view of the product if there is any hope of it getting done.

Each person is different and has unique views and opinions. They come with different backgrounds and experiences, and they must learn to work together.

It is important to get off to a good start—initial impressions, good or bad, tend to be lasting.

Therefore:

The leader of the project must instill a common vision and purpose in all of the members of the team. This "leader," whether a manager, the PATRON ROLE (4.2.15), or a customer advocate, should be someone who holds the team's respect and who has influence over the team's thinking. Gaining respect and influence requires overt action; you can't count on it

happening automatically. The leader should make sure everyone agrees on the answers to the following questions: What is the product supposed to do? Who are the customers, and how will the product help them? What is the schedule? Does everyone feel personally committed to the schedule? Who is the competition?

An important component of these team unification exercises is to identify the strengths of the team and to use these strengths as rallying points. The team thus identifies the challenges and competition and unites to overcome and surpass them, respectively.

As time goes on, the UNITY OF PURPOSE continues to emerge from ongoing dialogue within the team and with customers and other stakeholders. While the team leader primes the pump, team dynamics take over and keep things going.

The obvious result is that the team is working together rather than working at cross purposes. But a more subtle yet probably more powerful effect is what healthy team dynamics can do for the morale of the team. The best teams tend to feel that they are somehow better than others—and they work to prove it!

This pattern relates to some deep-seated principles and values of organizational health. There may be no more important single property of an organization than that its members have a shared vision that they are motivated to achieve. Communication—which receives the bulk of the attention in this book—is just a means to achieving that shared vision. UNITY OF PURPOSE, thus, is a deeper principle than even effective communication; communication is just a means to UNITY OF PURPOSE.

RELATED PATTERNS:

SHARED CLEAR VISION (A.5.25) ([Bramble 2002], p. 80) notes the importance of a clear vision in creating unity, from the point of view of writing use cases. SELF-SELECTING TEAM (4.2.11) outlines how a team should come together, though this guidance alone is insufficient to achieve UNITY OF PURPOSE. LOCK 'EM UP TOGETHER (5.2.5) helps achieve unity, particularly unity of architecture. A GATEKEEPER (4.2.10) also can help the team become more unified in establishing the requirements needed to ENGAGE CUSTOMERS (4.2.6). This pattern sets up COMPENSATE SUCCESS (4.2.25): it's much easier to compensate success when everyone knows what success means. And, while UNITY OF PURPOSE is important in galvanizing the team, effective team dynamics can develop only if every team member is also valued as an individual. HOLISTIC DIVERSITY (4.2.19) comes to play here.

4.1.26 Don't Interrupt an Interrupt *

The original interruption device.

... you've applied INTERRUPTS UNJAM BLOCKING (4.1.25), but you notice that the organization is now thrashing, particularly in the end game or under heavy churn.

❖ ❖ ❖

It's important to balance a desire that SOMEONE ALWAYS MAKES PROGRESS (4.1.20) with the thrashing that can accompany short-term priority calls. One worker will inevitably be blocked on you—you can't do both things at once. Complete foresight and perfect scheduling are unreasonable to expect.

Therefore:

If a developer is already working in "interrupt mode" on a critical issue, don't put that work aside until it is complete or until that issue itself becomes hopelessly tangled.

❖ ❖ ❖

This pattern prevents the endless churn that can result from too much context switching. It also helps to ensure that SOMEONE ALWAYS MAKES PROGRESS (4.1.20). And it provides some "back pressure" in the process that can help temper irresponsibly quick reversals of position in the front end.

This is a simple, though somewhat arbitrary, rule that keeps scheduling from becoming an elaborate ceremony.

This concept relates to the "red zone" from Linda McLyman's analysis of the Satir change model [Satir 1991], which suggests that if a foreign element (problem) arrives before the organization starts to learn its way out of the last foreign element, recovery is difficult.