Incorporating Safety Requirements using Patterns in ArchWiz Tool

Zafar Ahmad Bhatti
sbio8002@student.mdh.se

Collaborated with:
ABB Corporate Research
Vasteras, Sweden

Supervisors:
ABB Corporate Research: Pia Stoll, pia.stoll@se.abb.com
Anton Janson, anton.jansen@se.abb.com
Malardalen University: Frank Lüders, frank.luders@mdh.se
Abstract

Usability Supporting Architectural Patterns (USAP) has already introduced a concept in software architecture for coping usability issues in better style and revealing its obscured dimensions. A support is also needed to develop the safety systems in such a way that they employ the same rules and get a better understanding of safety, its requirements and the architecture. A way to determine safety requirements from the patterns and working with the responsibilities of patterns was the aim for this thesis report. On the other hand, a useful tool with the name “ArchWiz” was to be developed further from its prototypical form—an assisting tool for architects to look for requirements and evaluation of their architecture. The mature development of ArchWiz tool, and incorporating the safety perspective with respect to USAP vogue was also the goal of the thesis.

In a development process, architecture designing is a crucial and vital part of software system. During architecture designing process very first decisions and information are gained to validate if the system has the potential to meet its requirements and intended behaviours. Along with other important quality attributes, safety architecture has played an important role in developing today’s critical software and automated systems. These safety issues especially in software architecture are to protect, recover, discover and mitigate the hazards, faults, failures and catastrophic perils. The deficiency and obscurity of these inherent dangers can be reduced by understanding the safety in general and analysing its requirements from unseen perspectives. Later, these requirements can be traced into the architecture of a similar system as a knowledge base or experience gained.

Architectural patterns and their investigation in safety provide a broad horizon for requirement and solution in various aspects. They help to bring out the requirements in refined way and in general manners too. The report, therefore, presents the suggestion to formalize the suggestions in safety with respect to requirement engineering in architectural context as well as reusable solution for these issues; alike in USAP style.
Contents

1. Introduction ........................................................................................................................................... 6

2. Usability Supporting Architectural Patterns ......................................................................................... 7
   2.1. USAP Research ................................................................................................................................. 7
   2.2. Patterns and USAP ............................................................................................................................ 8
   2.3. USAP Terminology ............................................................................................................................ 9
   2.4. USAP Pattern Language .................................................................................................................. 10
   2.5. The USAP Prototype ....................................................................................................................... 11

3. The ArchWiz Project ............................................................................................................................. 13
   3.1. Implementation Structure ................................................................................................................ 16
   3.2. Changes and Delimitations ............................................................................................................. 17
   3.3. Workflow .......................................................................................................................................... 18

4. Implementation Results ....................................................................................................................... 20
   4.1. Implementation Components and Modules ..................................................................................... 20

5. Introduction to Safety .......................................................................................................................... 23
   5.1. Dependencies for Safety .................................................................................................................. 24
   5.2. Relationship of Reliability, Availability and Safety ......................................................................... 25

6. Safety in Process Perspective .............................................................................................................. 26
   6.1. Safety Systems ................................................................................................................................. 26
   6.2. Engineering System Safety ............................................................................................................ 26
   6.3. Safety Risk Management ................................................................................................................. 28
      6.3.1. Risk Assessment ......................................................................................................................... 28
      6.3.2. Hazard and Failure Mode Identification ...................................................................................... 28
      6.3.3. Hazard Severity .......................................................................................................................... 28
      6.3.4. Hazard Probability ...................................................................................................................... 28
      6.3.5. Safety Order of Precedence ...................................................................................................... 29
      6.3.6. Risk Elimination ....................................................................................................................... 29
      6.3.7. Quantification of residual Safety Work ....................................................................................... 30
      6.3.8. Managing and Assuming Residual Safety Risk ........................................................................ 30
   6.4. Safety Standards and Their Role .................................................................................................... 30
      6.4.1. Process Assurance Based safety Standards .............................................................................. 30
      6.4.2. Product Evidence Based Safety Standards ............................................................................. 31
      6.4.3. IEEE1061 Model Framework ................................................................................................... 31

7. Safety in Technical Perspective .......................................................................................................... 32
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.1. Patterns’ Investigation Results</td>
<td>60</td>
</tr>
<tr>
<td>12.2. Safety Formulation Suggestion I</td>
<td>61</td>
</tr>
<tr>
<td>12.3. Safety Formulation Suggestion II</td>
<td>64</td>
</tr>
<tr>
<td>12.4. Safety Formulation Suggestion III</td>
<td>66</td>
</tr>
<tr>
<td>12.5. Analysis of safety formulization suggestions</td>
<td>68</td>
</tr>
<tr>
<td>13. Summary and Conclusion</td>
<td>70</td>
</tr>
<tr>
<td>14. Future Work</td>
<td>71</td>
</tr>
<tr>
<td>14.1. ArchWiz Tool</td>
<td>71</td>
</tr>
<tr>
<td>14.2. Safety Suggestions</td>
<td>71</td>
</tr>
<tr>
<td>15. References</td>
<td>72</td>
</tr>
</tbody>
</table>
1. Introduction

This thesis was done in ABB Corporate Research, Vasteras. The thesis work was divided in two tasks mainly including the implementation and research work. The implementation part of the thesis was the development of ArchWiz tool at ABB and the research area was determining the safety responsibilities and requirements in architectural context of a system. Therefore, the report mostly emphasises on the safety attribute of software quality and safety architecture to some extent.

ABB researchers had already carried out a comprehensive research on usability concerns and have developed a prototype of the ArchWiz tool. The purpose to develop ArchWiz tool is to provide software architects generally and within ABB especially for a comprehensive guidance and knowledge tool while designing their systems. ArchWiz tool can help architects of ABB to gather general reusable requirements and then to share this knowledge for other architects. Once the reusable requirements are listed then any architect can go through them and can match if the product on which he/she is working on has met those requirements in their architecture or not. The reusable solutions of those architectural issues are also available in ArchWiz tool which can assist an architect during the design of a particular product. ArchWiz is supposed to provide support for all quality concerns in a software system where user can add their general requirements but right now most of its focus is on safety, security and usability perspective.

ArchWiz project aims to utilize and extend the concepts in software architectural methodologies and software architecture techniques used in USAP (Usability Supporting Architectural Patterns). It focuses on the USAP technique to apply it likewise on safety and security quality attributes now, and others in later stages. On the other hand, it also intends to develop a modified and extended version of USAP prototype tool. The tool has to be developed based on the requirements and architectural issues faced by the architects so that they can contribute in their architectural knowledge in the form of architectural requirements and refined requirements. The design issues for developing the extended prototype tool also consider a distributed nature of environment, usability, portability and ease.

ArchWiz project was the implementation of such a tool to assist the architects and development of it was finished in December 2009. Though it was not finished and furnished completely at that time but a good foundation of GUI framework and structure to build different parts of it was established. Safety is one of the attribute which ArchWiz has to support. Therefore, the formalization of safety attribute in architecture according to the tool have also been suggested and analysed. These suggestions have not yet been incorporated in the tool but a proper user test and analysis would give it a place in ArchWiz. The user can also select or modify these suggestions according to an appropriate choice in the tool in future research.

In a nutshell, USAPs’ ideas, concepts and theory would be exploited to produce the reusable software quality requirements for safety attribute and then the feasible reusable solutions for those requirements. As a result of these hierarchical requirements and solutions, this tool can incorporate the requirements easily for the architects and developers in ABB.
2. Usability Supporting Architectural Patterns

This research work dispensed for the safety field and domain was to bring out the model, suggestion and the research thoughts on how the architects can mitigate the gap within the software architecture of a system. Moreover, the idea for utilizing the developed patterns for checking the predefined responsibilities by considering the requirements can solve many architectural issues. These responsibilities have been defined by digging down into the patterns which already give the solution for many of the reoccurring problems. The responsibilities help to realise that whether some specific requirement related to safety domain has been captured in the system or is relevant to the system under consideration. Thus to come up with the general solutions for the safety quality and defining the responsibilities for them is the basic challenge whereas the model has to fit in the safety domain as it already has been exercised in usability.

The prototypical tool (ArchWiz) provides the way for the architects in ABB to find out the general safety architectural solution based upon the requirements of the system they are developing. It should also help them to locate the solutions in different categories defined in the safety modelling and design, so that related problems can also be seen in the defined hierarchies.

Software quality attributes are important for designing a complete and better system design and the architecture. They play an important role with respect to the stakeholders and the developers throughout system analysis and design process. In most of the cases, during the design phases, many of the quality attributes of the system are overlooked in the software systems [1]. Studies show that now much importance is given to describe the quality requirements in software architecture to support the design right from the beginning which can save time, money and efforts in later stages of system development.

There are many common and important quality attributes such as modifiability, performance, usability, security, reliability, safety, testability.

Considering the importance and research interest on these attributes Software Engineering Institute (SEI)-Carnegie Mellon University in Pittsburgh, USA with collaboration of ABB corporate research centre started a project called the Usability-Supporting Architectural Patterns (USAP) Project. The project was intended to focus the usability quality attribute during the early design phase to help the architects to formalize the software architecture targeting the usability concerns particularly. This research actually provided an opportunity to investigate on usability in software architecture and developing the suitable software architecture related to usability consequently. The concrete result of the research was in fact a web based prototype for the architects in simplified form of a tool actually.

2.1. USAP Research

Usability Supporting Architectural Patterns are concerned with the usability quality attribute of the system which is considered one among the most important factors in software quality and architecture. Usability mostly outfits the ease of operation and the operation time limit in which that operation is supposed to be handled. If the usability concerns are not considered during the start of the design phase the risk that they will emerge later into the system becomes higher- because the modification of those user interface elements has strong relations with the functionalities running behind them. This problem makes us attentive to
incorporate usability during the beginning phase of the design to make the design more effective and flexible. Usability is usually thought a subset of the modifiability where the architects assume that separating the usability concerns and the functionality can modify the architecture locally [2]. But this perception proves to be wrong when system shows that the usability concerns are deeply rooted in the architectural design of the system and separation does not support all usability concerns which can add up to the cost and extensive re-architecting [2, 3]. Thus, the research conducted by ABB and CMU resulted into a set of patterns in the form of scenario and reusable requirements. These requirements stand common in most of the complex and large systems where usability is the focus. These scenarios produced the generic and common responsibilities for usability in system architecture which encompasses many of the decisive usability concerns.

These responsibilities and the structure were called patterns which resulted to utilize many of the techniques to incorporate usability into the beginning of the system design and architecture. The experiments were conducted to evaluate the practical assurance of these usability scenarios, a scenario is first developed with the responsibilities belonging to that scenario, and a sample solution is given. The results proved that the list of USAPs given to the users helped the users to remember the responsibilities when modification and complete solution was required in architecture. These scenarios provided the knowledge base system to the architects where the architecture is a vital implication for the system. These USAP scenarios were identified from the usability patterns in software architecture. Later, a prototype was built up by structuring of the scenarios and patterns.

2.2. Patterns and USAP

The patterns are the general solution for the reoccurring problems. This concept has been utilized for USAP where the architecture is the main concern to take into the account; they describe the solution at design and architectural level. Usability patterns mostly explain the look and the contents placing for users but USAP actually help to explain the problems in design from the usability perspective. The taxonomy for USAP has been mentioned in the USAP terminology part in section 2.3.

The information a USAP contains are defined into six parts as below [4]:

1. The first element is a scenario that is supposed to solve a situation. For example, `cancellations command` from user during the system execution to stop the previous command.
2. The second is the description of the condition under which a scenario is applied. For example, time constraints as of less than 5 seconds.
3. Then it shows the benefits which will be gained after implementing that USAP. For example, cancel can reduce the routine errors and give the boost the system speed.
4. The forces are mentioned that can have influence on the solution. For example, no prediction is available about when users will cancel the command.
5. There should be an implementation independent description for the software in the form of responsibilities. For example, system will always listen for the cancel command.
6. A sample solution in terms of UML diagrams where those diagrams are quite illustrative in overarching an architectural pattern.
USAP are actually the software responsibilities that can be applied to any structure in architecture, legacy systems and support the functionalities. USAP flows in the steps mentioned below in a hierarchy defined in USAP task flow model:

- Observe specific user interface to find out the relevant scenarios.
- Selecting the any activity mentioned in that scenario.
- Choose a task from an activity in the scenario.
- Pick the quality attribute and the responsibilities for that quality concern.
- Compare the responsibility implementation details with system architecture.
- Decide about the status of the responsibility in the architecture.

The above steps are repeated for all of the scenarios, activities, tasks and responsibilities against the specific quality concern. The role and the work products in fact help to choose the scenarios for the quality concerns.

2.3. USAP Terminology

USAP hierarchy starts with a scenario; the scenarios are then broken into the activities. The activities are resolved into the tasks and then the responsibilities are the elements to take the decision on and to set the status for the architecture. The purpose and the justification are also explained with the activities to make it easier to understand. The researchers have established a list of scenarios for usability in laboratory experiments to improve architecture design in system. These scenarios have been categorised in “User Profile”, “Alarms, Events and Alerts”, “Environment Configuration” evaluated to be the important ones. Below are the two (Authorization, Logging) of the activities defined with purpose, justification and the tasks to comprehend.

Authorization

Purpose: The purpose of these responsibilities is to identify and authenticate users (human or other systems) of the system.

Justification: Users must be authorised when security or personalisation is important.

Tasks:

1. Authentication
2. Permission
3. Log of

Logging

Purpose: The purpose of these responsibilities is to retain and examine selected information generated during execution.

Justification: Some information known only during execution needs to be retained either for debugging or audit-trail purpose.

Tasks:

1. Specify the items to be logged
2. Log items during execution
3. Post-processing
The responsibilities defined in the USAP also contain the rationale and the implementation details described in the text format ahead to the responsibility description. The rational contains the justification of the responsibility in the system while the implementation details are to define specifically what to do in the system architecture to comply with the requirement of the responsibility. The laboratory experiment suppressed to use the textural description for the implementation details rather than to use UML notation to express the pattern because the textural style was easy to describe the pattern which is used for sample solution. The responsibilities are the core and the gist in the USAP model where they are actually are considered as of requirements on the architecture of a system.

2.4. USAP Pattern Language

The teams constructing the USAP developed the USAP for each of the scenarios among User Profile, Alarm Events and Alerts and Environment Configuration but many of the responsibilities in these scenarios appeared to be redundant. The academic and industrial research teams grouped the responsibilities into similar categories and thus lead to define a pattern language [5] for that which can be seen in figure below. This pattern language defined the relationship between USAPs and the reusable set of responsibilities.

The language shows two types of USAPs. One is the End-User USAPS and the second is Foundational USAPs. The end-user USAPs are more and directly related to the end user benefit and can be thought of small scenarios and require the responsibilities to be fulfilled in the architecture. They also have the condition under which they are relevant. They follow the six steps defined in the section patterns and USAP. In the usability USAPs and tool, end-user USAPs is “User Profile”, “Alarms & Events” and “Environment Configuration”. On the other hand Foundational USAPs act as the support and framework to construct the end-user USAPs. They do not have scenarios, description of conditions and benefit for user directly. End-User USAPs are actually the specialization of the Foundational USAPs as can be seen that all of the three end-user USAPs have authoring portion and the execution portion. The foundational USAPs are actually the activity where the responsibilities are parameterized. These activities have been described as Authoring, Execution with authored parameters, Logging and Authentication. The relationship between the Foundational USAPs and End-
User USAPs has also been defined in the figure where these relationships have been expresses as generalization, use and depends on relationships [6].

**Generalization:** This relationship says that Foundational USAP is the generalization part of End-User USAPs or End-User USPAs are the specialization. The End-User USAPs pass the parameter to the Foundational USAPs and if Foundational USAPs also need any conditionals in the responsibilities then the End-User USAPs are also responsible to define the conditional values.

**User Relationship:** This relationship also passes the parameter but the communication remains at the same level i.e. between the activities (Foundational USAPs).

**Depends-On Relationship:** This is a temporal relationship. An example of such a relationship is that a system will not execute with authored parameters until those parameters have been authored before the execution.

### 2.5. The USAP Prototype

The responsibilities are listed according to activity in the prototype where against each responsibility architect has to make a decision. These decisions can be one among the option as whether the responsibility has not been considered for the architecture, or if the responsibility is not applicable to the system to be designed, or whether architectural design already addresses the responsibility or architectural design needs to be modified to address the responsibility. These listed responsibilities give architect the direction to ponder over the responsibility in the architecture to make system as much complete and perfect as possible. These decisions are required to be done for every responsibility with respect to the scenarios given in the relevant activity. Based on these decisions, a to-do-list is generated on the responsibilities which have been mentioned as must modify architecture, not yet considered or discuss status of responsibility. To elaborate the terminology and the USAP prototype a screen shot is given in the figure below.
This prototype and tool helped to remind the architects to cover the responsibilities in design and architectural level to support the quality concerns in the architecture of the system. The conclusion of the resulted work was that this efficient tool allows the usability engineers and software engineers to meet early during the design phase and discuss usability responsibilities and their impact on the architect with well formulated and understandable format of the tool [8].
3. The ArchWiz Project

The purpose of ArchWiz project is to develop a tool for the architects in ABB to map product requirements in the form of general requirements which can be set as a reusable; the architectural responsibilities. The mapping of product requirements to general requirements actually help to find out reoccurring requirements in a specific context. Therefore reusable architectural solution for those general requirements can be utilized. This tool is developed at ABB Corporate Research in Vasteras. There are two basic approaches to exploit this idea and use the concept; bottom-up approach and top-down. In bottom-up approach, architects gather the requirements which are related to the current system and have been used before for the similar systems. If the same set of requirements has been confronted for similar architectural solution, then a pattern can be distilled to describe those set of requirements in coherent way along with the application of general solution for them. On the other hand, in top-down approach, architectural patterns are used as a starting point and the responsibilities found in them are moulded according to the ABB specific responsibilities. The top-down approach was actually investigated by SECRC (ABB Corporate Research, Software Architecture) and Carnegie Mellon University/Software Engineering Institute, Pittsburgh, USA.

The scope and vision of ArchWiz tool includes supporting architecture evaluation, an architectural guidance and knowledge tool for software architectures within ABB, and domain specific information within that knowledge. The domains and the domain specific knowledge for ArchWiz tool in the form of general software requirements and responsibilities consists of many quality attributes such as usability, safety, security. The software products in ABB that have the quality concerns related to the quality requirements in ArchWiz can use this tool for their benefits and better architectural design. The ArchWiz tool has a process to follow for architecture evaluation. The process can be broken down into three phases where each helps to switch to next phase for better and specialized evaluation. In the beginning general quality requirements which have already been captured in ArchWiz tool can be used for requirements elicitation. The next phase requires that high level requirements are broken down into the refined requirements but making it sure that no important detail of the design and specific part of the system is overlooked. The last phase is in fact the evaluation phase where one of the quality attribute i.e. safety, security, usability is taken under consideration. This phase takes the preliminary architectural design as input and knowledge base together to generate a to-do list. This evaluation can indicate the important aspects of the problem, the potential risks and a clear understanding of viral attributes of architecture. It also helps to recognise that at least the basic responsibilities of the system related to the quality concerns have been captured and considered well.
The tool can adjust according to the product and domain specification. If a user wants to change or add more general requirements in the tool, it is also provided within the tool. If a user needs to save the requirements which are only related to a specific product, he/she can deselect the general requirements out of his/her scope. These products specific general requirements can be saved as a knowledge base and can be shared with other users too.

If seen from abstract terms, ArchWiz has simple system process transformation. Figure 3, shows the system process transformation of ArchWiz. This transformation mainly has three processes in ArchWiz. These processes are named as Pattern mining, Requirement Distillation and Architectural Evaluation. The main inputs are Pattern literature and Requirements coming from the Requirements Engineering process. In the bottom-up approach, the Requirements distillation process takes the requirements of a project and distils the reusable parts of them. The results of this process, i.e. reusable requirements, are stored in the knowledge base using the reusable requirements domain model. The reusable requirements are used for an Architectural evaluation of the Architecture design. In a similar way, the top-down approach uses concepts from pattern literature (e.g. responsibilities) using the Usability, Security, or Safety domain models to store the knowledge in the knowledge base, which in turn is used in the Architectural evaluation process. The Architecture evaluation results in a Todo list, which can be used in the Architectural Analysis and Synthesis processes to improve the Architecture design.

ArchWiz, in fact is a part of a big system used in architecting. ArchWiz actually tries to focus on small part of the architecting process and systems rather than capturing whole architecture systems’ process completely. It only concentrates on the architecture evaluation process now. Therefore, ArchWiz requires cooperating with a large variety of tools efficiently in architecting process. Mainly, ArchWiz interacts and work with the following different type of tools.

**Requirements tools:** These tools are used to keep, maintain and create requirements of a system. Examples of these tools are Doors, Word, and Rational RequisitePro.

**Architecture description tool:** They are used to create and maintain architecture description. Word, PowerPoint, Visio, Whiteboard drawing and Enterprise Architect are the typical examples of these tools.
Collaboration tools: They are used to share the created Todo-list in architecture evaluation to stakeholders of interest. Examples of tools are Lotus Notes, Windows Share-Folders and SharePoint.

Management tools: These tools are used in planning of the resulting actions from the Todo-list in the development process. Examples include Microsoft Project.

The primary usage of the ArchWiz can be understood by analysing the use cases of the tool. Below is the main use case diagram for the illustration of tool and its actors.
from the diagram so the detail description of them is omitted here. But, the detail of use cases “Take architecture decision using reusable architecture artifacts” and “Access reusable architecture artifacts” has been given below in the form of diagrams.

![Diagram of Make Architecture Decision](image)

**Figure 6: Make Architecture Decision. (Used with the permission of ABB)**

![Diagram of Navigate Knowledge Views](image)

**Figure 7: Navigate Knowledge Views. (Used with the permission of ABB)**

Some of the more details about state diagrams and detailed illustration can be found in the document of ABB called “ArchWiz System Description Revision 4”.

### 3.1. Implementation Structure

The implementation of ArchWiz project was carried out to develop a GUI framework and the underlying design support for the prototype. The thesis comprises of two parts, one of which was the implementation in developing the new ArchWiz tool while the other part was to
search and find the safety patterns and exploit them in the way USAP uses them. From the implementation perspective, it was supposed that tool may not be finished completely but at least a good foundation can be provided for it. The development of the tool was being done by me and another thesis worker along. One of the supervisor was helping us closely at ABB CRC and the other was the project manager and main supervisor of the thesis until project time finished in December 2009. One supervisor had some hours in a week to help and guide us through the implementation phases and development while project manager (also the main supervisor) was managing and planning the project for its appropriate results.

The tool is supposed to be developed in Java language and working environment chosen was Eclipse Environment which was a new and learning setup for both of the students. The user interfaces using Swing/JFace were quite novel and the working classes were also needed a good structure keeping in my complete flow and architecture of the tool. The time required to learning new language, environment, tools and technologies was also an important factor to plan. Another important issue was to develop the tools according to the USAP requirements and responsibility structure to map the research consequently.

The second part of the thesis required to come up with patterns, domain model for safety and suggestions. Using those suggestions, software architects can take the architectural decision whether to cope with the changes in software architecture from safety aspect or not to accommodate the responsibilities in a specific context. This part includes the duties to learn and study the thoroughly software literature on architecture and safety along with patterns. Later, with the gained knowledge through literature along with the understanding of how ABB today uses these safety requirements and responsibilities and the way ArchWiz is supposed to work, some suggestions on formalization were to be delivered.

3.2. Changes and Delimitations

ArchWiz tool corroborated some changes along the way during its implementation. The scope was changed a little to incorporate the general requirements knowledge base where all software qualities can be represented. Knowledge bases where the reusable requirements for different quality attribute, i.e. usability, safety, security are present. The configuration and modifiability of the ArchWiz tool can also be done according to the better suit of the project one is working on. A user can edit the contents of tool; a to-do list can be generated with the information about the decisions on different requirements. A user can also comment on the decisions made, an import and export were to be catered into the project for other knowledge bases. An additional feature requested from ABB was that a user should be able to split the screen in the tool so that he/she can take the decisions meanwhile looking at the history of decisions made before.

The lack of resources and the difficulty level along with strength of the tasks were pretty obvious from the beginning of the project. Thus, the goal was to develop the concept, architectural design of the tool and the basic implementation should be seen by the close date of project in the middle of December 2009. The research should be also carried out on the safety quality and not the other quality factors are to be addressed. The formalization should also be appropriate according to the way this tool is supposed to be used. Though, in the beginning of the project it was an evaluation tool but in the later stages it was decided that the tool is supposed to work purely for system requirements. This may have the implications on the way security suggestion would be formalized.
3.3. Workflow

The workflow of the thesis would help to understand the implementation details and the formalization of the suggestions given in the end of the report. It shows the main activities done during the time of its implementation and results. First of all, a better understanding of ArchWiz project was to be established that included the study of previous work done of USAP project. ArchWiz is actually a continuation of USAP and is a project made in this thesis. A deeper and thorough understanding of this project actually requires knowledge about software architecture in common and specifically software architectural safety patterns which open up a horizon for different design aspects of the architectural solutions.

GUI framework was chosen as the first activity to start with. Both of the project/thesis members along with their supervisors were involved in this task. The preliminary mock-ups of GUIs were made and then the iterative versions and improvements were built. Jigloo was chosen to build the first implementation of the GUI framework. The learning of different widgets and graphical elements was also involved which consumed some time how to relate the widgets and which of the elements has to be kept in relation to other graphical constraints. Changing in design also caused many times GUIs to be altered or adjusted during the mock-up creation. On the other hand, class structure and other various design issues were taken in parallel and sometimes one after another. Most of the classes and their implementation were my main tasks in the project implementation. My responsibilities also included some new tools and technologies for file structure and data handling and mapping of the elements with classes.

In addition to this, research part of the thesis made me to focus on reading different articles, books, publications and research papers. Many of the papers were gathered from ACM, IEEE and Springer. The subject of reading was the safety in architecture of computer systems and particularly safety patterns which help to solve many of the problems encountered in software architecture and implementation of its design. The knowledge gained with this study helped me to formalize the safety strategies and suggestions for safety requirements and responsibilities in terms of USAP model. These suggestions can be seen in the last section of the report.
Figure 8: Work Flow Model for Thesis

During the whole period of project/thesis, meetings were scheduled and held to understand the direction and progress of the work. These group meetings also helped to discuss the problems, ideas and division of work for the next week. In the beginning, it was planned that thesis report will be written during the whole thesis period but design, implementation and research caused a delay for it until the end of the thesis time. Also some of the registration problems granted me an extra time to submit my thesis report later.
4. Implementation Results

The implementation of the ArchWiz tool and results to be produced were already realized to be given very short time and the resources employed on the project were only two thesis members. Most of the GUIs work was assigned to another thesis mate and I was working on designing classes’ structure, data handling, mapping the classes’ data and business logic behind the tool. Though the tool was halfway finished when the project was closed in December 2009 because of the lack of time but a foundational structure and basic implementation was developed for future if ABB management decides to proceed the project. The project supervisor Pia Stoll and Anton Janson (CRC ABB) helped us to follow the directions and strategies for implementation and design strategies.

The implementation of classes and the design structure is the part of ABB propriety so I cannot reveal the details of them but they can be found in the design and implementation documents of ABB. For the understanding of my work and implementation, an architectural level detail can be presented to understand the modules I worked in. The domain model contained the classes for implementation and business logic but other components were also implemented to support the classes and the data mapped and stored. It also paved the path for GUIs to access the underlying logic and data to display and for user interaction.

4.1. Implementation Components and Modules

The main models and components I worked in can be seen in component and connector view of ArchWiz application. To elaborate the detail and structure of it, below are the mentioned parts to consider. The figure shows the logical units (i.e. components) and their interactions (i.e. connectors)

ArchWiz Program: The black box view of the ArchWiz program is the main part in which many other units can be opened up as shown in figure.

Domain Model: It consists of domain objects that actually represent the different concepts used in ArchWiz tool. These concepts can be understood as activities, tasks, responsibilities, decisions etc. The domain objects can be implemented with annotated Java classes. These annotations are used to turn the Domain model into xml files and vice versa.

Figure 9: ArchWiz Application Implementation
**JAXB:** The JAXB library is used to convert the *Domain model* to *xml* files.

**Apache Derby:** The embedded database *Apache Derby* is used to provide convenience for the easy selection of data by programmer and to provide local transaction support.

**Hibernate Library:** *Domain model* and *Apache Derby* needs to have synchronization which is facilitated by a library. This library is called Hibernate which provides an object/relational persistence and query service. *Hibernate* connects to the *Apache Derby* database using a standard JDBC interface.

**SWT UI:** The user interfaces in ArchWiz program have been handled by SWT UI. It uses the Standard Widget Toolkit (SWT) to visualize the UI using the native widgets of the platform running the program.

**SVNKit:** The SVNKit is a Java library for achieving versioning and implementation of its features. It also provides a fully fledged API for communicating with a subversion server and using versioned files.

The implementation of above modules and units developed the basic structure and fundamental design when the project finished in December 2009. Many of the technologies during the implementation were quite new for me for which learning time was also consumed. The learning of these tools and technologies was achieved by doing many of the examples with small units and then implanting then in actual application. Later, near the finishing period, some of the changing requirements also influenced some part of the implementation redundant and little less fruitful than it was perceived in the beginning. Though it was not very easy to change some of the implementation issues but was enough to go almost half way for the implementation of the ArchWiz application.

The final outcome of the tool in working condition has been elaborated with the figure. The tools works in a node and tree structure as can be seen in the screen shot of it. Most of the details about the screenshots and GUIs have been implemented by my co-thesis worker Caroline and thus can be read in detail in her thesis report given to ABB. For a look how the tool was working, the below figure is provided.
This is the GUI framework which has been developed by the end of the project. Though some of the changes were incorporated in business logic of it in later stages but the effect of it on GUI was quite minor. The future work for the tool was also suggested for the improvement and the final version of it was still in good condition for further work on it.
5. Introduction to Safety

In easy words safety can be defined as

Safety is a property of the system that it will not endanger human life or the environment. [9]

Or

Safety of a system is absence of the catastrophic consequences on the user(s) and the environment. [10]

The above definitions actually imply that the safety is an attribute of the system in which we expect that nothing bad or un-expected happens. But when we dig down the detail of the safety as quality attribute in the software and computer systems then many of the other dimensions appear and they define the safety in the terms of risks like:

Safety is a judgment of the acceptability of risk, and risk, in turn, as a measure of the probability and severity of harm to human health. A thing is safe if its attendant risks are judged to be acceptable. [11]

These definition may seem very understandable and easy but in reality when it comes to the issue of safety in the systems and specially software critical systems, then safety becomes one among the most important and critical concern to be dealt with. Here comes one more important definition which is related to safety critical systems as the safety becomes the exigent factor in those systems and plays a vital role of the development in the software. To elaborate the concept of safety software system, the definition at below can serve.

Safety critical software is any software that can directly or indirectly contribute to the occurrence of the hazardous system state. [12]

Software Safety is actually a challenging research area where the problems are evolving and some are still to be searched on. Leveson has also mentioned that “software safety” term is also confusing and seems to be a misnomer because software itself is not harmful or dangerous for the people and the environment. But safety is more concerned with physical systems and software actually helps to cause the system hazards in the systems’ context.

The safety is counted as the system property in general and most of the research work shows that safety is not really related to software only in the system; but as a compound of many properties and components together. Thus it requires for the system engineering right from the beginning. To ensure the safety into the systems, other solid software engineering techniques also come into play. These engineering techniques include requirement engineering, software and system architecture, formal methods and rigorous testing.

Before understanding the phenomenon related to safety, we will first investigate what software safety means and what kind of attributes are referred to. Software dependability plays an important role for software safety. Because most concepts are common in both of them and they complement each other also many of their characteristics are derived mutually. Generally safety and dependability have been defined in terms of threat whereas threats are defined in terms of three basic notions fault, error and failure. We can define
them like as below to elicit further background and idea related to software safety and its dependability.

**Fault:** A fault is (the adjudged or hypothesized) cause of an error. When it produces and error, it is active, otherwise it is dormant. [10]

The IEEE also defines the fault in other words like:

*An incorrect step, process or data definition in a computer program* [13]

**Error:** An error is that part of the system state that may cause a subsequent failure. Before an error is detected by (the system), it is latent. The detection of an error is indicated at the service interface by an error message or error signal. [10]

**Failure:** A failure if a system is an event that corresponds to a transition from a correct service to incorrect service. It occurs when an error reaches its service interface. [10]

If we see the relation in fault, failure and error then a chain can be produced which explains how are they interconnected and which of them triggers what kind of actions. To illustrate the concepts, the below chains describes that the faults are the point where the problem starts to occur, which actually activates the error.

Once the error is faced into the system then it is propagated to cause the failure and then again fault. Thus this makes a chain reaction and goes on in circle to undermine the safety property of the computer systems.

### 5.1. Dependencies for Safety

When the safety becomes the concern in computer systems then mostly it means that those systems adhere to certain standards from the process perspective. From technical aspect safety relies on many of the other qualities of the computer systems. In computer systems, safety is in fact a quality of a system as it is supposed to be by the standards. One the other hand system takes the precautions so that nothing undesired happens into the system. These both perspectives lead to another quality of software named reliability.

![Dependability Tree](Figure 11: Dependability Tree [10])
The above tree shows that some of the authors consider the safety quality attribute within the security domain and they put many other qualities along with it. They classify safety in the same list of attributes of dependability and security. If we see one definition of the security, some researchers say it a composite of the attributes which refers that security is a big domain within which safety lies as one of its part. In relation to safety, there are many concepts in computer science that are very new and are mingled, but we will see that what kind of measures and attributes are taken by and large in safety domain.

5.2. Relationship of Reliability, Availability and Safety

Reliability and safety are very inter-related and close concepts. In fact reliability is the most intimated construct related to safety. Reliability is mostly defined as the ability of the system or component to perform its required functions and its specified level of performance under stated conditions for a specified period of time. Interestingly reliability on the other hand is similar to availability. They all have very common attributes and qualities and can be counted for their sub characteristics to understand the mutual relations. Reliability can be described into its further characteristics as fault tolerance, recoverability and predictability [14].

Though reliability and safety have a great influence on each other but they still remain at a distance when it comes to safety domain purely. Reliability and safety is often confused because of the very minor difference they hold. Safety is a freedom from accidents or losses while reliability can be described as the probability that a system performs its actions and intended functions satisfactory. The differences and their sharing regions can be illustrated by the figure below.

![Safety with Reliability Concept](image)

The figure shows that a system without fail-safe state actually falls into both categories - safety and reliability. But this mutual sharing part is not a little part which can be ignored. Therefore, our investigation will include reliability wherever safety requires it most and whenever a breach in safety of a system can be envisioned.
6. Safety in Process Perspective

The characteristic of the safety concerns in software actually consists of many perspectives. Most of the times, since being an overall quality attribute of a system, safety is thought to be a process in building and developing systems—may it be in environment, machines, instruments, computer programs, human interaction or any physical device. We can elaborate the concept of it on first stance as a general process and then later can investigate into the typical architecture of computer systems and design.

Safety is always considered with respect to the system but not only with software or the components in the system but it requires some special processes to build thorough safety measures. One of the processes that define safety in terms of eight different steps is called ROPES [15] which actually gives the guidelines to capture the safety earlier stages rather than when it gets more expensive and harder to tackle. The reason to discuss the general perspective and safety process is that it helps to understand the in-depth study of safety architecture and its general idea to apply in safety systems. Though, safety process perspective does not touch the very technical basis of safety view of the systems but at least manifests quite many common ideas and concepts emerging out for particular technical solutions. These understandings include safety system concepts, engineering safety systems, safety risk management, safety processes, standards and the limitation of them. Furthermore, the same risk and hazards which are found in general scheme of safety concerns are mitigated and analyzed in special case of computer architecture of safety. The other perspective has been detailed with respect to patterns which focuses mostly the general rules to apply safety in system architecture during safety design. We will first introduce some of the underlying and main concepts to understand the safety process and later the technical domain of safety.

6.1. Safety Systems

Safety is a quality concern more related to the critical and real-time systems where the failures become disastrous and harm caused by them are deemed necessary to be coped with. In those systems, safety has also been described as reliability regarding the critical failure modes. This failure mode has been categorized as malign in the literature. The malign failure mode is a mode of a system where the usage, utility and benefits of the system become less significant than its dangers, failures and loss. This kind of mode can occur in real-time critical systems where the reliability should be ultra high. Some examples of such systems are airplane flight control systems, intelligent brakes in automobile systems, medical equipments, nuclear power plants, train signaling control systems, and robots. In such systems, the computer controlled system should face average fewer failures than those of mechanical or conventional system should face. Here safety arises as the higher rate because the control over the system becomes less in human hands and the catastrophic results can be huge. The safety responsible systems are actually vulnerable not only for high cost, but can also lead to high material damage and personal causalities as well. Hence these systems require that faults can be avoided to maximum extent.

6.2. Engineering System Safety

To grasp the concept related to safety in software development, one has to know the details of the system safety and how it applies accordingly to the system development. The safety in the systems actually grows with systems engineering and system analysis whereas they evolve in combination with each other. Most of the investigation and research related to safety in these
The system engineering is very important to understand the safety in software, its design, architecture and implementation since the software engineer have very less understanding of system safety process and overall system behavior. Safety engineering investigates the factors to trace out the problems, deficiencies, operations, management process and issues in systems that should have been handled and corrected before using its services. The control places for safety engineering in a system are defined usually at concept design review, preliminary design review, critical design review, final acceptance review and during audit of operation and management [17]. Moreover, various techniques and approaches with experiences and experiments actually led to standards for safety and helped to minimize the hazards and to reduce the accidents in safety systems. Some of these safety engineering rules and standards are defined by SSP (System Safety Program), which actually guide the engineers and researchers to keep the following factors into their consideration when the system has to be adapted for safety:

i. The hazards should be obviated and minimize the associated risks in design even if material substitution is required.

ii. Isolate hazardous substances, components, and operations from other activities, areas, personnel, and incompatible materials.

iii. The equipment should be arranged properly during operations, servicing, maintenance, repair, or adjustment to minimize personnel exposure to hazards.

iv. Environment conditions should be analyzed according to their limits to reduce the risks. These conditions can be temperature, pressure, noise, toxicity, acceleration, and vibration etc.

v. The design should be prepared by keeping in mind the risks created by human error in the operation and provide support of the system.

vi. Alternate approaches for those hazards which are hard to eliminate can be a better idea. These approaches include interlocks; redundancy; fail-safe design; fire suppression; and protective clothing, equipment, devices, and procedures.

vii. Protect power sources, controls, and critical components of redundant subsystems by separation or shielding.

viii. Ensure personnel and equipment protection using warning and caution notes in assembly, operations, maintenance, and repair instructions as well as distinctive markings on hazardous components and materials, equipment, and facilities.

ix. Always try to reduce the severity of personnel injury or damage in any mishap.

x. Design software-controlled or monitored functions to minimize initiation of hazardous events or mishaps.

xi. Review design criteria for inadequate or overly restrictive requirements regarding safety. Design should always be based on criterion supported by study, analyses, or test data.

These objectives and factors actually guide engineers, managers and safety individuals to indentify, mitigate, track, eliminate, and control the hazards and failures in every cycle and
process of design, development, testing, implementing, production and integration of software and hardware.

6.3. Safety Risk Management

Risk is a highly important process in safety management, which actually requires extensive and critical tasks and work. This process is comprised of failure modes, failure analysis, determining and examining the hazards and their causes. Risk management also includes evaluating the probability and severity of the failures, verifying the requirements, risks which can be hidden in a system before its deployment or during the implementation. Safety risk management also focuses on technical safety aspects. Technical panorama in risk managements is prioritizing these hazards and risks in context of system design, implementation testing and overall configuration. The process is a well defined in safety domain for risk management. Risk management process has been mentioned from section 6.3.1 to 6.3.8.

6.3.1. Risk Assessment

Risk assessment is also one among the inaugural step to be taken when safety is the main concern of risk management.

Briefly it can be mentioned in some steps that are taken while determining the risk of the system and to evaluate how much a system is riskier and needs safety assessment.

- For each hazards, the potential of the severity is first analyzed and is determined how much dangerous or harmful a failure can be.
- The probability of the hazard is measured and is checked how much likelihood is there that a hazard will be faced in the system.
- It is evaluated that how long the user or potential victim will face that hazard and will be exposed to it.
- Determination that a risk can be removed and is possible to be eliminated from the system.

6.3.2. Hazard and Failure Mode Identification

The beginning of the risk managements is initial hazard analysis, and failure mode and effect analysis which actually paves the path for the engineers with the required information and base to perform initial safety risk assessment and hazards’ identification. Identification of the hazards and possible failure modes are the basis for the implementation and design of the system which in turn comply with the safety requirements.

6.3.3. Hazard Severity

The severity is defined in classification of hazards and categorizing them within the system context, user and the environment. The severity is classified on two things: 1) severity of damage 2) number of time the damage may happen. The severity can be classified into many classes like catastrophic, critical, marginal and negligible. These categorization are quite qualitative but some benchmarks and measures can help to estimate it and for proper procurement.

6.3.4. Hazard Probability

The determining of safety risk also requires performing the process of identification of the probability of the occurrence of the hazards. There are many statistical techniques to evaluate that in un-controlled environment how many chances are there that a failure may occur. An
example is of these technique is MRI (Hazards Risk Index) matrix, which helps to realize and analyze the engineers to put the different requirements accordingly to the specific cell. These matrixes help the safety engineers to prioritize and provide them the flexibility to adjust the hazards in relation to their probability and severity.

![HRI Matrix](image)

Figure 13: HRI Matrix [18]

There are many kinds and tools to use this matrix it can evaluate the system safety and can guide the engineers in design, analysis, test and evaluation to allocate the resources but it depends on the system context, software inputs and information known about software.

There is one more such a risk measurement process that is expressed in the form of charts. It can also be used to determine the level or severity of the risks.

![Safety Order of Precedence](image)

Figure 14: Taken from DIN V-19250

6.3.5. Safety Order of Precedence

Safety order of precedence is the process which actually eliminates this risk and it has some defined tasks to assure it. These tasks include design for minimum risk, incorporate safety devices, provide warning devices and develop procedures and training [18].

6.3.6. Risk Elimination

Once the analysis of the safety hazards and failures has been occurred, the proper solution and reduction techniques are identified. The hazards may contain several specific design requirements which are supposed to be incorporated into the system. Along with it, other supplementary requirements are jotted down for safety devices, warning devices and training.
etc. The safety is not eliminated just by taking out the safety requirements but the engineers actually supervise and verify that the safety requirements are implemented into the system as they are supposed to be there.

6.3.7. Quantification of residual Safety Work
Through the hazard analysis and risk elimination, the hazards are not completely eliminated but only reduced and thus there remains the factor of residual safety risk. Therefore, another risk assessment process is carried out by engineers when the requirements are implemented to some extent. They identify and verify the residual risks during the operations and support activities. Here the amount of design and test data is changed and hazards are once again checked for severity and probability of occurrence.

6.3.8. Managing and Assuming Residual Safety Risk
Managing the residual safety risk is a simple process but needs much more efforts, time and resources. At this level the safety manager establish a system of accountability where the assumption of residual risk is based on contractual obligation, negotiation and user inputs.

6.4. Safety Standards and Their Role
There are a number of safety standards which actually vary according to the safety requirements. Safety standards define the practices which are required to assure safety. Although they help to support the compliance with other components and assure the better practice but still the difference in their details depends on the philosophy of their emergence. The detail about the standards is out of scope of the report but still we can see the famous standards and their basics so that we can see that how to comply with them can we solve and assure the safety in the systems. The most of the common of those standards are Process Assurance Based Safety Standards and Product Evidence Based Safety Standards.

6.4.1. Process Assurance Based safety Standards
The process based standards are the set of the practices which adhere to development, verification and validation of the software practices. The examples of such standards are DO-178B[19] and IEC-61508[20].They mention the guidelines for the objectives for software life cycle processes, the description of activities and design consideration, and the description of evidence that proves that the objectives have been satisfied. Just as Do-178B focuses mainly on traceability and human review of the artifacts produced in each state. It also manages the compliance of the artifacts in different stages e.g. artifact from source code to low level requirements. Moreover, they strongly believe testing as the primary way of verification.

Limitations

The process based safety standards can be apply and implemented in adaptive and safety critical systems but there are some pitfalls which occur during this standards process implementation. In conventional system the requirements are decomposed and refined at the point where a deterministic solution is produced but in the safety and adaptive system the refinement of the requirements are operational behavior, which is performed at run time. In these systems operations within a system are supposed to be developed where the system can alter its behavior and response is based on stimuli. Thus by adapting these standards even the assurance of safety in the system cannot be guaranteed, it is hard to satisfy the safety with purely defined positively expressed requirements .The safety requirements often have negative focus because they define those requirement which software should not exhibit during the operation [21].Furthermore in such systems, the development may start from the
incomplete specification which are supposed to be developed through training. Also when the behavior of the system needs to determine the requirements inclemently, these standards cannot be used properly. Testing as the verification process here also makes it hard to provide adequate test and is likely to be impractical if the input space is large [22].

6.4.2. Product Evidence Based Safety Standards
These standards focus on the well structured and reasoned argument, based on the demonstration of the satisfaction of product specific safety objectives. These objectives are taken usually from hazard analysis which concerns the product specific targeted evidence and justify the acceptability of the safety. The product based evidence requires that the safety claims are hazard based; failure of software that can lead system level hazards. These standards actually focus on “demonstrating the safety of”, instead of “demonstrating the development of” the system. One of such standard example is ALARP (As Low as Reasonably Practicable).

Limitations
The product evidence standards focus mostly on the evidence of hazards, risks and the identification of the hazards that can lead to the system level failures. This approach is not concerned how the system has been developed but it has more interest in operational behavior of the system. Thus it engages more on the arguments of safety for details of stimuli and mechanisms that can lead the system failures with potentially unsafe behavior. There are some cases where the testing of these standards has shown the inadequate evidence of safety system. Analysis is therefore needed in order to provide adequate guarantees of their real-time performance [23].

6.4.3. IEEE1061 Model Framework
IEEE 1060 also defines the models for a quality attribute in terms of hierarchal decomposition into the characteristics of quality, sub characteristics and their metrics. They can be categorized as stimuli, responses and architectural parameters. Stimuli and responses together form scenarios that describe how architecture should react on a certain stimulus.

The safety standards usually have some criticism because they often lack the guidance, poor integration of software and the issues in certifications [24]. Furthermore, experiences and history of the standards shows that they are not aiming completely for software safety. They also lack the modifications and advancements - sometimes proved to be insufficient when a pragmatic approach about the standards is taken. As mentioned in one of the article that “In the current safety standards we find that the argument in and of itself is logically weak and represents a poor basis with which to demonstrate the safety of a system” [25].
7. Safety in Technical Perspective

The process perspective of safety has already been mentioned in detail. Now the technical aspect of safety will be discussed. These technical perspectives include safety requirement, safety properties, components, safety architecture, patterns and their relation to ArchWiz and USAP suggestions. This part of the report is main research areas. But as already mentioned that safety is not only a technical aspect of a system but a process based aspect of safety is also necessary to conceive the concept in broader perspectives.

7.1. Safety Requirements

The safety in software starts right from the very beginning of the system when the requirements of the systems are being captured. The requirements mostly remain different form system to systems and thus need a real attention to be defined in the systems where safety is the main concern. Some of the researchers also state that safety solutions are dependent on the formal specification of the requirements in proper form. There are many methods devised which actually prove that a system fulfills the safety requirements. These techniques are called Temporal logic, Computation Tree logic, Fault tree analysis, Failure Mode and Effective Analysis. In these methods, safety requirements are typically defined in terms of hazards. The hazard probability is determined in risk analysis where all the hazards in a system are combined together and are evaluated for acceptable risk. The hazards can be categorized in safety context as [26, 27, 28] where the safety is mostly breached because of incomprehensive requirements.

The requirements and their completeness is a very common trace to find any flaw in software. All of the faults can be traced somehow in the problems of understanding the requirement and their incompleteness. The incomplete system understanding, wrong assumptions, undefined system states, unhandled control of the system and environmental condition all have their roots in the requirements understating. Generally, the requirements in safety systems are defined in four types as below [29]:

- Safety Requirement: They are mostly defined as quality factor within a quality model and are dependent on quality criterion as minimum or maximum threshold with a quality measure. They define how safe a system must be.

- Safety Significant Requirements: These requirements belong to functional, data, interface requirements in terms of safety. These requirements are supposed to lead to harm and dangerous accidents if not implemented into the system.

- Safety System Requirements: These requirements are actually related to the components of the system

- Safety Constraints: These requirements include the design and the architecture of the system where the constraints make them bound to stick with specific safeguards mechanism and measures.
7.2. Safety Properties

Safety properties are those factors that are directly involved in measuring the safety. The top level classification of the safety properties can actually be seen in figure 15. But the subdivisions of these properties are also possible which determine the failure modes based on these properties too. One of the possible sub classifications can be mentioned as detectable and undetectable value domain failures.

Hazards are also determined based on the safety properties and have the same type of classification and are described almost in the same terms as of the properties:

**Actions:**

These types of hazards are caused because of the actions. They have been introduced by the system if an inappropriate action is taken into the system. It can also be referred if an appropriate action was required by the system and it was not taken.

**Timing:**

These kinds of hazards are dependent on the timing of the system action, input, duration or output. The hazards in this category happen if some action is taken too early or too late into the system.

**Sequence:**

This form of hazard is based on the sequence of the actions, functions, data, input, process or any output. It also includes if some actions has been skipped while the system needed that for proper execution or is if the action taken was out of order.

**Amount:**

The amount or quantity of the data or any input or output also matter for hazards to appear into the safety systems. It can be caused by too much amount of something or too little quantity of anything required by the safe system.

7.3. Safety Components

Safety now can be seen in different perspective and will include many of the attributes and components. The components of safety are actually the measures and parts which are crucial to handle failures. The reasons for failures primarily drive the components of safety.
Dependable and critical systems which are thought to be primarily safe actually face the failure because of the three reasons as [30]:

a. The systems is not to be understood properly and the incomplete understandings which make a system safe.

b. The system is embedded into an already larger system which has not been considered well and implemented concept is seemed to be relevant.

c. A single point of failure can make a whole system unsafe, thus ignoring these failure points will cause breach when taken into practice.

From the above reasons, safety can have several components to stop the failures. These components are the primary base to focus whenever designing a safety system. They are also sometime called the techniques to avoid failures.

**Fault Detection:** The ability to recognize the functional ability of device is called fault detection. Fault detection is a concept related to the monitoring of the system and identification of the faults occurred in system and the location of those faults. The fault detection can be defined also as the detection of the failures in hardware or in software. The detection can be used in many of the ways like logging, alarm triggering, manual intervention and some other ways. Fault tolerant systems can tolerate only limited and certain types of faults which are usually distinguished in three categories; normal, exceptional and catastrophic [31]. Fault detection should also diagnose and identify the components which are causing the faults and also whether the fault is temporary or permanent.

**Fault Avoidance:** Fault avoidance is the capability of the system to prevent faults from occurring during the operations. It also reduces the introduction of faults during the system design and construction. Fault avoidance also includes fault prevention, fault removal and fault forecasting. Fault prevention is the mechanism to stop any fault creeping into the system before it goes on operational mode while fault removal attempts to find and remove the causes of errors. Fault forecasting actually predicts or alarms the system if a fault is imminent to be occurred. Forecasting also includes the estimation of presence, creation and the consequences of faults. Fault avoidance helps the safety system to improve quality of the system and components both.

**Fault Tolerance:** The ability of a system or component to continue normal operation and to maintain a specified level of performance in cases of failures of portions of a system and unplanned situations such as unexpected inputs, hardware or software faults, or stressful environmental conditions. One important thing to notice is that fault tolerance is sometimes also referred to as robustness.

**Recoverability:** The ability of a system or a component to restore itself to a working state after an interruption, such as in case of a failure. This includes recovering from certain classes of errors to a predefined state, to re-establish a specified level of performance, and to recover data directly affected by the interruption. Recoverability is thought most of the time the most vital characteristic for ensuring high availability is software systems [32].

Rebuilding a software system or the component after the disasters or faults in the system such that system can be brought again into the operational mode is called recovery process. The recovery for the software can be in terms of data, process, function or information. In safety critical system the importance of recovery is essential to avoid the hazards and failures.
because one the system face them and cannot handle it to turn to the proper way, the hazards can lead to disastrous states.

**Predictability:** For safety system, predictable behavior is a key aspect and especially for mission-critical and life-critical systems, predictability with respect to schedule ability and memory is crucial. Predictability, in general, is the quality of being predictable and the predictability of a system is the extent to which its response characteristics to all possible events can be known in advance. The more predictable a system is, the more reliable it will generally be.

**Redundancy:** Redundancy is the duplication of critical components of a system with the intention of increasing reliability of the system, usually in the case of a backup or safe. The redundancy can be applied to hardware or software both. Redundancy is the high demand for the systems which demand high dependability in safety critical application. The standard approach to achieve fault tolerance is redundancy.

**Failover:** Failover is the capability to switch over automatically to a redundant or standby computer server, system, or network upon the failure or abnormal termination of the previously active application. Failover mostly are not caused by human interaction and without any warning. Failover is considered as a backup mode of the system where if the primary system becomes unavailable the secondary takes the place. Failover is an important fault tolerance function of mission-critical systems that rely on constant accessibility.

![Diagram of Safety Concepts](image-url)
Software Architecture

Software architecture is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them [33]. As like other quality factors, safety is also an important part of architecture.

Safety case [34] architecture can be defined in very similar terms:

“Safety architecture is the high level organisation of the safety case into components of arguments and evidence, the externally visible properties of these components, and the interdependencies that exist between them”. [35]

A main and key aspect of a software system design belongs to its architecture where the organisation of the component embodied by the system, their relationship and environment is observed. The architecture is mostly considered the abstract view of the system which is more concerned about the how the elements work in together and their interaction during the running system. It also defines the behavior of the elements and structural constraints.

There are two viewpoints which are related to the architecture of the software system; structural and the behavioural [36]. The structural view point consists of the following units:

- Components: They are the units of the computation of the system.
- Connector: Interconnections among the components to support their interactions.
- Configurations: is the set up of the components and the connectors.

The behavioural perspective elucidates the actions a system executes and participates in, the relations among actions to specify the behaviours as well as behaviour of the component and connectors to observe how they change and interact and the state of active system.

Is it Important?

Architecture of the system is important from business, process and the technical point of view collectively. The stakeholders are mostly interested in it to have an abstract overview and understand negotiation, and communication. It also plays an important role in early design of the system though development, deployment and maintenance eventually are dependent on it. The architecture also guides to structure the elements in the similar fashion if the attributes and requirements seem to be similar in previous systems. Software architecture also defines the earliest design decision with constraints and limits posed on the system, which help in the separation of concern. It also makes the organizational structure and processes easy to stick together, and drives the quality attributes of the system with assurance that it will work accordingly. [33]

Software architecture also makes sure that it meets the requirements and reduces the risk by understanding the information necessary for the system. The importance of architecture makes the whole system very dependent it because if the basis of are not solid and understood very well, the whole system remains vulnerable. Critical system’s software engineering direly needs the understanding of roles of the components of software and the interactions which they have with the whole system [37, 38].

Safety Software Architecture

The term software architecture with regards to safety is an important concept because it actually separates and defines the components of the system regarding to the safety. It
provides the decisions about the architecture of a software artifact and the implementation of it. The nature of the architecture being very abstract in fact also helps in reusing the well known, proven and understood solution and attracts to search the patterns into this domain. Partitioning the design space in this way supports the development of safety arguments in a modular and hierarchical fashion [39].

The relationships between architecture and safety are still undefined in many of the aspects and have been described by many researchers. Some problems of particular interest can be seen in safety consequences of flexible and adaptable architectures (e.g., using integrated systems for in-flight reconfiguration) [40]; evaluation of architectures for safety-critical product families [41]; partitioning to control hazards enabled by shared resources [42]; architectural solutions to the need for techniques that augment the robustness of less robust components[43].

7.7. Architectural Patterns

Architectural patterns characterize and specify structural and behavioral properties of the system that in turns allow the provision of the solutions for classes of problems. Patterns also introduce the enhancement since they utilize reusable expert knowledge of software development.

Patterns can describe the classes, components, relationship and details of the components along with their responsibilities. Therefore patterns are at the level of abstraction where the problems and their solutions are efficaciously explained. As the abstraction and determining the structure of the design is common in patterns and architecture; they can provide a helping hand in building complex architecture in a predefined and experienced style.

The patterns that have architectural focus explicitly address the operational and development requirements of software systems with their quality measures in safety, security, performance, scalability and extensibility. The systems that have special evolving requirements and where the run time behaviour is more of concern, architectural styles and pattern become more important for them.

Architectural patterns can be divided into two subsets; patterns and reference models. Patterns describe the global organizational structure such as layered systems, client server structure, event based systems etc while the reference models consist of specific and parameterised configurations of the components and connectors for specific application areas. [44]

The architectural patterns are mainly defined in different categories but are used together to serve for the reoccurring solutions. On the other hand tactics are different for the scope but solve some smaller problems and do not affect directly the overall structure. These both are named as architectural styles or patterns .The knowledge of patterns in architecture is comprised of architectural tactics and reference architectures as well.
8. Safety Patterns

When we consider the patterns in the safety domain then all of the safety characteristics come into the play to have their proper role. These characteristics have a greater direct influence in the systems that require safety. While finding the patterns related to safety we have to see reliability, availability mainly and the patterns related to it. Since, these patterns serve for these different qualities attributes and their sub attributes at the same time, so can be listed in the safety domain. Like, to improve the reliability into the system we need redundancy and to fulfill the system redundancy the literature has found many patterns that are applied. When these patterns are defined in architectural form that involves redundancy, they decrease the risk of the system failure due to software or hardware fault, and in turn improve the safety of the system.

8.1. Formal Specification Patterns

The formal specification patterns help to formulate the safety requirements in formal notations and some available approaches help to jot those requirements in natural language. Furthermore they actually transfer expert knowledge because of the patterns has been devised by the expert of formal methods and formal notations. These patterns actually serve as a recipe book which contains all of the safety requirement in formal notations and a user can pick the relevant requirement form these as a check list on which the safety requirements can be applied. In that check list a safety engineer can watch that which safety requirements are relevant or the specific respective system development.

Formal specification patterns are used in the following way [45]:

1. The user selects and determines the suitable formal formula in a list with all kinds of common formal specification patterns for safety requirements.

2. In the second step the user has to adapt the pattern to the respective safety requirement in context to the operational model. As a result we get formal specified safety requirements, which are instances of the patterns in the list.

These patterns have been put in order and given a classification based on which distinct characteristics of the different specification patterns can be known.

<table>
<thead>
<tr>
<th>Static Safety Requirements (Invariants)</th>
<th>Safety Requirements about Chronological Succession</th>
<th>Safety Requirements about General Access Guarantee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Safety Requirements</td>
<td>Beginning of Validity</td>
<td>Beginning &amp; Duration</td>
</tr>
<tr>
<td>Safety Requirements with Temporal Dependencies</td>
<td>Duration of Validity</td>
<td>Time Triggering</td>
</tr>
<tr>
<td>Safety Requirements with Explicit Time</td>
<td>Beginning &amp; Duration</td>
<td>Event Triggering</td>
</tr>
</tbody>
</table>

The benefits of using formal method are apparent and quite obvious but at the same time it needs a great effort and economically pretty expensive with respect to resources and man power. Furthermore, the common perception of the engineers is that the formal methods are difficult to understand and hard to apply for the software development. Therefore it is used.
only for safety parts of the automations and critical parts of the system. More to this, some of the attributes and properties are very complex to express in temporal logics and sometime error prone as well. Most of the practitioners and verification users are also not logicians although they know very well about the properties which they want to verify but cannot expressed it in good logic and formal language.

The detail and the example of these patterns classification can be described in below:

**Static safety requirements**

It includes the property that an operation should hold that property as true in the whole operational model. Traffic light crossing requirement can be applied here when it is not permitted that a green light would be turned on for main road and at the same time for side roads.

**Dynamic safety requirements**

These requirements include that a property must be true in some of the states while in other states it should be false. The permission to open the traffic gates is given only when the train has passed.

**Safety requirement about general access guarantee**

These requirements are related to all model states. Emergency break is one of these requirements where the emergency break should be actuated or accessible in all situations.

**Safety requirements with Temporal Dependencies**

In these requirements the temporal dependencies exist between the event of detected defect and the action against it. It requires when a specific statement should begin and for how long it should remain valid. Pneumatic break is one of such example when the defect is detected software control system has to be switched off after a certain delay.

**Safety requirements about chronological succession**

In these requirements the property is dependent on the occurrence of other properties. It has the proposition of “if then” and it tells how long exactly the predecessor and from when exactly the successor event is permitted to be valid.

**Safety requirement about duration of validity**

These requirements refer to the duration respectively ending of the validity of a property which is actually dependent on the other properties. Like, the flow of water is only possible when the temperature reaches to a certain level in the pipeline.

**Safety requirement about beginning of validity**

These requirements are about the beginning of validity of a property, which is dependent on the other properties too. An example can be “The safeguard of a level crossing is only permitted to be terminated, strictly after the railroad crossing has been completely vacated if the train had passed.”

**Safety requirement about beginning and duration of validity**
These requirements deal with both of the beginning and duration of the property which is in a certain sequential dependence on other properties. Such as only after the temperature sensor has relayed to some certain value, from then on it is permitted that the inflow is opened but only as long as the level of the tank has not reached the minimum value.

**Safety requirements with explicit time**

In these requirements the beginning and duration of property is dependent on certain time duration or certain point in time of the timer. The gates must be in the closed state for 6 seconds before the railroad crossing has the status safeguarded. These requirements are further classified into time triggered and event triggered operational modes.

**8.2. Safety Tactics**

Architectural tactics can also be attributed as the underlying primitives of patterns to solve the problem. These tactics actually help to understand the safety into its smaller domain but can be used to form the patterns.

![Figure 18: Hierarchy for Safety Tactics](image)

These safety tactics define the architectural parameters that can be controlled and specified in terms of failure detection, failure elimination and failure tolerance. Thus these tactics can be applied in architectural patterns definition and software architecture. Some of the tactics are directly related to safety while some actually have the reference to other quality attributes as well but can be applicable for safety too. The main advantages for using these tactics and setting them together are to evaluate the problem domain as well as an idea to solve those problems. These tactics have some special properties which make them very useful when defining the design and architecture of the system. Some of these have been defined below.

**Refinement**

These tactics can be further divided and refined. Thus, they stand at various levels of abstraction and details. As an example Redundancy can be refined and grouped into more details of replication, functional redundancy or analytical redundancy.
**Dependency**

The tactics are some time not entirely independent from each other and one application implementing one of the tactics can assume that another tactics has already been applied. Just like as in voting some kind of redundancy already exists in the application.

**Combination**

Mostly each tactics has some specific issue to discuss and addresses special aspect of the safety but they can be used in combination with other tactics to address multiple aspect of the safety. In voting to avoid single point failure and faulty inputs, it can be advised to apply redundancy by additional voter elements.

**Hardware/Software Implementation**

The tactics actually give the choice to architect to implement them on software level or hardware. Using the same voter example, it can be implemented by software or hardware. The choice and decision to apply these tactics on hardware or software depends on the trade offs, cost, performance usually the complexity.

### 8.2.1. Safety Tactics and Patterns

Safety tactics can be manipulated to construct and establish the patterns in software safety architectures. These patterns are written in the context of scenarios which are combinations and collection of tactics in fact. The combinations of these tactics produce number of scenarios and thus can derive many forms of different patterns or different forms of one pattern. We can have one example to show how the different scenarios can be applied to patterns and how the tactics help to produce them.

Scenario: There is a basic safety requirement that the software must monitor any potentially hazardous conditions so that proper protection mechanisms can be executed.

Tactics: Detection, Barrier, Redundancy, and Masking. These tactics can be further decomposed into more concrete forms.

We can define the responsibilities for the separate architectural elements. The tactics for, we will use in this example, are condition monitoring, functional redundancy, comparison and interlock.

*Condition Monitoring* is used to detect any malfunction of the actuator or component by monitoring it or monitoring the performance based on different condition.

*Functional Redundancy* is used for the additional independent version of the control function and eliminates the risk of the failure.

*Comparison* is used to detect any discrepancy or conflict between the outputs of the two independent version of control function.

*Interlock* is responsible to enforce the correct sequence of the events. It ensures that the next result value is not produced until the previous has been consumed or given acknowledgement.
Pattern-1

Now using the same tactics above we can have many different scenarios and patterns which can be derived out from them according to the requirements and scenarios. The first pattern [46] can be where we need ‘Monitor’ component to detect the failures solely but no protection is taken against the failure which can lead to single point failure into the system. Although there is a strong dependency between monitoring and control function but they both are implemented in ‘Monitor.’

![Pattern-1 Diagram](image)

Figure 19: Pattern-1

Pattern-2

In the other version of the tactics a new pattern can be formed when we allocate safety responsibility to both ‘Monitor’ and ‘Control’ channel where the tactic of comparison is applied for each channel by cross checking. If the two channels do not agree with each other then the action would be taken by ‘Output’ component to stop the actuator at proper time.

![Pattern-2 Diagram](image)

Figure 20: Pattern-2

8.2.2. Tactics Identification and Selection

The requirements in the beginning phase can elicit and help extensively for the architecture and to choose the good design. The high level requirements can be refined into further low level requirements which can be functional and timing as well.

If the proper scenario for the specific system has been chosen then it is always easier to work out for relevant tactics and the patterns produced by them.

The functions where the failure has the importance, although failures cannot be avoided but the tactics can be selected from the categories of failure detection and failure containment. Some of the examples like:

- If there is something to act on the validity signals then sanity checking can be a possible option for it.
- If there is a need which deems necessary for a recovery action taken immediately can stop the failure then the recovery tactic can be selected. The barrier tactics can be used where the passive protection mechanism is required.
The same way redundancy can be applied with combination of many other useful tactics like in cross checking comparison and functional redundancy can be used. In monitoring, analytical redundancy can be a better option.

The selection of the tactics is actually based on the arguments for each candidate in the tactics and these arguments in result contribute the decisions.
9. Safety Architectural Patterns

These are the patterns that actually make a knowledge centered framework for critical and safety systems for effective detection and prevention of software defects and failures in design.

These patterns divide the architecture into three abstract and refined layers where the first layer defines architectural design, the second projects the details of the internal components of the architectural elements and the third layer is responsible for the functional design. The whole framework is consisted of assessment patterns and their associated metrics. Each of these patterns is designed to describe a particular concern in safety. These patterns also include the strategy, general solution and techniques to implement those strategies. Each technique is broken into the building blocks and these blocks actually develop a check list for assessment. [47]

To elucidate the safety concerns, these patterns have been defined in the pattern style format which also helps to see the component and the interaction of them along with a check list for their solution components. They are listed to illustrate the strategy, techniques used to implement those strategies and the overall consequences.

9.1. Automatic Failure Detection

Context
The identification of the approaching failures in the system is very critical. It has been achieved in most of the systems by automatic failure detection for minimizing, focalizing and fixing the failures and the influencing part of the failures.

Problem
A detection method, where the fault or failure has a chance to occur is necessary, to get rid of it. If a true result has to be gained then a comparison has to be made with an already or existing result of the system. Otherwise the false or wrong result can be passed to the system for a computation. A true result is deemed required for addressing this issue.

Solution
A solution to the problem of detecting fault can be found in comparison. Comparison is usually achieved with redundancy—may it be physical, logical or any other type. The techniques in this strategy use the comparison method though these techniques differ in the way they obtain the true result.

In order to check for logical errors and others sources of failures, two separate units of functionality are developed. The key factor here is to use the same input for both of these units which actually should produce the same output. The same CPU is used in both of the execution cases but these units run in a sequence. A comparator is used to identify the failures in a way that even if both of the units produce errors, the results are not affected in the same way. Logical redundancy is also the part of this solution where an additional CPU can also be used and different constraints can be applied. Oracle can also be used for the detection of failures automatically. The oracle knows about everything, the computations are specific. This pattern actually focuses on the errors which can put the system in unsafe state while the others are left unhandled usually. This can greatly simplify the oracle because for some problems this method is sufficient.
Consequences

- The results produced in these techniques and solution strategy can be imperfect because they are obtained at run time and hardware failures can yield their affect on them.
- These techniques will detect failures, but how effective they are depends on the implementation. Thus, implementation of the technique of using this pattern is also an influencing factor here.

Building Blocks
The building blocks or the vital components of automatic failure detection include computation, number of CPUs, version programming, comparator and oracle. The details of them can be found in [47] material.

9.2. Managing Component Interactions

Context
Improper interactions between the components are the most common sources of failures in safety critical systems now a day. The sequence in which different components collaborate and co-operate is as important as the individual functionalities of the components. Individual components operate as are made for, but the sequence in which they interact breaks down mostly. Consequently it is imperative for safety-critical systems to handle the component and their interactions.

Solution
Separation of concern is the appropriate design for solving the problems of interactions and responsibilities of the components/units. During software development there are many concerns, which have to be addressed system wise and also component wise. At the component level these concerns include functionality, behavior and others. For safeguarding interactions certain concerns are more important than others are. Usually concerns appear all over architecture and are intertwined with each other. To focus on one separate concern is very difficult. The strategy of separation of concern promotes the idea to separate concerns as best as possible. When a single concern is isolated it can be managed much more effectively. It also requires isolating and capturing the concern of system control from the rest of the system, since it is the origin of all component interactions.

Interactions of the component can be made safe by analyzing the behavior of all the components. Controller and worker components are the part of encapsulating the behavior for separation of concerns. A controller component takes the responsibility to control the actions, sequence and correction of interaction. It makes the responsibilities of worker components easy and straightforward by reducing the functionalities and interactions between the components. This architecture gives the benefits of encapsulated, effective and isolated changes to behavior. In the case of behavioral changes in a system only the controller part has to be changes while the worker remains intact.

Breakpoint and monitoring the interactions is also one part of this pattern in architectural domain. Monitoring is mostly handled by sending of data and components interactions while breakpoint acts as an access point to the data between two components. Thus, a middle layer for interaction is introduced in between the components.
Consequences:

- Low coupling and high cohesion remains the gist of separation of concern in this pattern, otherwise the architectural problems would remain same in a system.
- The components sometime are direly dependent on each other and almost impossible to separate. In that case, an efficient measure and capability has to be taken into design issue or this scheme would not work properly.

9.3. Failure Isolation

Context
The struggle to minimize the effect produced by a fault in a system is necessary whenever failure happens. Isolation of failure can reduce the amount of impact only the place where fault has been occurred. Failure isolation, if applied correctly can minimize the problem and let the other part of the system act normally and properly.

Solution
One of the first steps during software architecture design is to partition the system into smaller parts. These parts can be sub-systems, layers, components, modules and others. System partitioning influences the capability of the architecture to isolate error. The focus of the solution in this pattern remains to localize the failure and keep the architecture better.

Layering in the architecture is one of the probable solutions. The golden rule of layering applies here too that only the adjacent layers can call each other and it helps to protect the failures to pass through to other layers. Decomposing the system into physical subsystems is also one option from the architectural perspective of failure isolation. This pattern also uses the separation of concern in the form of controller worker components which has been mentioned before.

Building Blocks
The vital building blocks of this architectural pattern include modules with low cohesion, high cohesion on sub-units, low coupling and worker object.

9.4. Automatic Failure Recovery

Context
Once a failure is detected, the next logical step is to recover from the failure. The different ways can be utilized for recovering from failures that includes re-calculation, aborting the calculation and moving system to safe-state.

Solution
State based recovery is one of the solutions for the problem where a system tries to recover from the failure. It does not recover directly though but moves itself to a predefined system state for masking the occurrence of failure. Continuous snap shot of the system during it normal execution can help. These state snapshots are stored in repository and in the event of failure, system can resume from archived states.

Whenever a failure occurs a known action is taken to remove the failure and any effects resulting from it from the system. Total recovery is the preferred method for recovering from failures because the operation the failure happened in still finishes with a true result.
Error handler is also one of the parts used in solution. Architecture and financial wise, error handlers are very cheap to produce. They can be included in the code of the module they safeguard in the form of try-catch blocks. Error handlers are also beneficial because they try to handle a vast amount of failures. To summarize, error handlers are an easy and inexpensive way to recover from basic failures, leaving the more complicated failures to the more powerful techniques.

**Consequences**
- All of the techniques described below have to be supported by some error detection method.
- If a failure happens, which the recovery mechanism was not designed for, it will not be caught or handled.
- It needs to be supported by some error detection method. A system can only recover from a failure, if it knows about its presence.

**Building Blocks**
Error detector, error handler, state manager, state repository, safe state and capability protocol constitute the basic building blocks of this architectural solution and pattern [47].

**9.5. Reconfiguration**

**Context**
At any given moment in time, one module of a system can have an internal failure. If this failure is not handled by any safety technique, then the corresponding module is broken beyond repair. When one module of the system does not function, usually the entire system does not function. Reconfiguration is a method used to handle permanent failures within a module of a system. During reconfiguration, the module in question is removed from the system. Thus most of the time system get rids of faulty element or unit during execution.

**Solution**
The replacement of the faulty component is required to compensate the failure of one component. Just like many other techniques, redundancy servers for the very purpose. Static redundancy, dynamic redundancy, graceful degradation is all those kind of patterns and techniques used for replacement. Static redundancy uses redundant modules, which are active at all times during execution. In other words all of the redundant modules will receive the information to produce results, until one of them gets faulty and can be removed then.

On the other hand dynamic redundancy uses spares modules to replace defective module. During normal operation, only one or a selection of modules is actively involved in producing results but when the fault occurs, spare module gets its place and continue working. To avoid the crash in a system graceful degradation is applied because if a failure occurs, it will only lose functionality. In other words, the system reconfigures itself, but it just does not replace the defective part.

**Consequences**
- Reconfiguration will only be successful if a fault can be detected and its origin is determined. So, fault detection has an important role for the application of this pattern.
Reconfiguration only works well with system made up of components, because only components as a whole can be removed /added. A tightly coupled architecture can make the implementation of reconfiguration difficult and sometimes even impossible.

Building Blocks
The building blocks of this architectural pattern include computation, configuration control, error detector, spare module, version programming and capability protocol.

9.6. Testability

Context
A method used to increase the amount of errors caught in the testing phase by providing interfaces for testing tools.

Solution
The interface for test tool into the code of the system is the focus of testability. When the system is completed this test interface is used to insert test data into the system and gather the test results. This allows for automated testing through use of sophisticated test tools.

The internal and external interfaces for testing tool in all of the components are designed in test layer. This layer actually links the system to test layer and mostly handles only simple problems. The test layer is the interface between the system and the test tools and it is essential for all testability techniques. This layer itself does not allow any testing capability but an external process is required to accomplish testing. The external process uses the test layer to gain access to the system where the layers extracts data from the system and runs an analysis on it. External process can also insert data into the system using test layer.

Capturing the snap shot of the states is also a part of the patterns where an external process can analysed the system with the available states in certain occasions. The concept of state manager with some variation can also be used here as have been described before. Additionally, fault injection into the system can measure if the critical system deals with them or not and those states are marked as invalid. Break points are also used in this pattern through interfaces to give access to the data inside communication.

Consequences
- The techniques and designed used in this pattern can contribute to a large amount of code. Hence, the system can go slower and bigger if they are applied without proper attention and measures.
- The technique used in this pattern is very hard to apply on hard real-time systems or a system where the constraints of hardware resources are rigid.

Building Blocks
The building components of this pattern are test interface, breakpoints, state manager and snapshot repository.
10. Safety Architecture Transformational Patterns

Transformational patterns describe the structural formation and changes in the software architecture in an abstract way. These changes may incorporate the components and elements in architecture which should be added or removed or take part in direction or redirection of the elements in architecture. These patterns also contain the detail of the strategies and techniques mentioned in Knowledge Centred Assessment patterns. Therefore these patterns have the granularity level of the previous described patterns and show the components in their own detail designs. They also serve as the specialized form of the strategies and techniques in safety architectural patterns. These patterns give an overview to utilize the safety properties in the system and reduce the probability of hazards in the system. These patterns have been derived from many of the researchers to evaluate the architecture in the safety domain and the detailed description and the selection where these patterns can be applied, is found in these papers [48, 49, 50, 51, 52].

10.1. Protected Single Channel

Context
If the failure of a hazard is precisely identified and is of an architectural element—an element generating an incorrect output of architectural element and misses a hard deadline, then this pattern is applicable. The detection of failure during run time is required to apply this pattern.

Problem
To reduce the probability of the hazards that is caused by an erroneous behaviour of a software component in the deployed system.

Solution
Failure error detection in architecture is carried out by a component dedicated for it. This component is responsible to detect failures or errors that lead to a safety critical behaviour. On any detection of specific error or failure, a message is sent to the mitigation component that removes the error.

![Protected Single Channel](image)

The spoofing ports are utilized to access the relevant output of the component under consideration. The test ports of the failure/error detection component are used to detect errors in the component which can cause a safety critical failure. Moreover the connection between error detection and mitigation component is helpful to inform mitigation component when an error is occurred. This pattern is also known as Single Channel Protected Design.

Rationales
Economically no further hardware is needed. Technically, only single point failures are detected and failures must be understood and identified.

**Consequences**

**Advantages:**
- It provides some level of reliability and safety for systematic and random faults both.
- The cost in developing the recurring is inexpensive in this pattern.

**Disadvantages:**
- This pattern is able to handle only transient faults—the faults that are no longer present if power is disconnected and do not permanently affect the system.
- The pattern is appropriate for the system which can bear fail-safe state or are not meant to function continuously.

### 10.2. Recovery Block

**Context**

The availability and correctness of the life protection system is vital and crucial for the safety properties. It is possible to apply this pattern if the erroneous behaviour (failure) that causes the problematic hazard is precisely identified and can be detected at the runtime of the system.

**Problem**

To reduce the probability of hazards those are caused by a systematic failure of software components in the deployed system, if the function of these components is critical for the safety properties.

**Solution**

The basic idea of this pattern is to use multiple heterogeneous developed components operating in parallel on a single hardware platform. All components receive the information from the environment, but one of them acts as primary component while other remain as backup components.

![Figure 22: Recovery Block. After [55]](image)

The primary components perform the desired operation that is checked by an acceptance test component. This acceptance test component can check the primary component itself to detect errors. On the occasion of an error or failure, primary component goes to backup while the next backup component takes its place as primary. To get a protection against failures caused by systematic errors, the heterogeneous components must fail independently. For this, these components should be heterogeneous.

**Rationales**

To use this pattern there is no necessity for additional hardware but additional and diverse implementations of the software-components are needed. The correctness of the acceptance test component is essential after the application of this pattern.

**Consequences**
**Advantages:**
- The functionalities can be developed independently and these equivalent versions of functionalities can be executed in parallel or sequence.

**Disadvantages:**
- The complexity of acceptance test can vary widely.

### 10.3. Process Fusion

**Context**
In architectures with a large number of active software components (tasks or processes), which are processed and scheduled on a single hardware platform, the time to switch between these processes is high. This can have a negative effect on the temporal correctness. Therefore, the pattern application is possible, in case a safety critical situation occurs, if an architectural element misses a hard deadline

**Problem:**
If the system has to meet a hard deadline then the process has to schedule properly for switching problem.

**Solution**
To reduce the number of the small active processes, one can compose two processes to one new process, which acts as the two processes did before. This will lead to a scheduling plan, where a task switching does not occur as often as before the application of this pattern. As an effect, the resulting components will have a better chance to meet their deadlines.

**Rationales**
Reducing the coupling between the processes and the binding of the processes will influence the development and maintainability effort. It also reduces the encapsulation, because one component has now access to the data of the other component.

**Consequences**
- The process will be scheduled properly and will meet their deadlines.

### 10.4. Hardware Platform Substitution

**Context**
The architecture misses its safety requirement due to a set of software components executed on a hardware platform with a low quality. This low quality may result in reliability and/or in performance problems. In case of reliability problems, where a software component is not available, generates an incorrect result or misses a hard deadline, this pattern can be useful.

**Problem**
To reduce the probability if hazards that is caused by an improper hardware platform.

**Forces**
The performance of the hardware platform influences the performance of software components, which are executed on this hardware platform.

**Solution**
To solve the problems, low-quality hardware platform is substituted by a newer and better hardware platform and all software components are now processed on the new hardware platform. This implies that if the new hardware platforms have better reliability properties, the probability of the safety critical failures due to unavailability and incorrectness are reduced. The new hardware can have higher execution speed and therefore the components in resulting architecture can meet their hard deadlines in a better way.
Rationales
For applying this pattern a hardware platform with a better quality may be more expensive. Existing software components which are developed for the old hardware platform can be incompatible to the new one. The safety cases must be generated for newer hardware platforms.

10.5. Hardware Platform Reassignment

Context
Sometimes the same hardware platform processes too many processes and it becomes impossible that this platform schedules a software element with respect to its deadline.

Problem
To reduce the probability of the hazards that is caused by a component that is executed on an improper hardware platform

Solution
As a solution the software element can be reassigned to another hardware platform in the system, where the software element can be scheduled without any problem or the probability of common because failures are reduced.

Consequences
Advantages:
- The communication overhead must be considered between the reassigned software component and software components on the former hardware platform
- The influence of common cause failures will be changed

Disadvantages:
- The software component and the new hardware must be compatible.

10.6. Integrity Check

Context
It is possible to implement this pattern as a refinement to the failure/error detection component of the Protected-Single-Channel, the Two-Channel-Redundancy or the Recovery Block pattern, to detect a critical incorrect behaviour, because an architectural element generates an incorrect output.

Problem
To identify the failure or error those are caused by an alteration of the software component. The safety critical systems must be permanently available over large time period. Thus, the probability that the component is altered is very high and it is necessary to detect the failures.

Solution

![Figure 23: Integrity Test. After [54]](image-url)
The component of the Protected-Single-Channel, the Two-Channel-Redundancy or the Recovery Block pattern by a component that checks the integrity of the safety sensitive component called integrity test. The basic idea behind this pattern is to add redundant (CRC, checksums or parity) information to the data and code to determine, if something or someone has injected errors. In the implementation, it is necessary to distinguish between the code and data segment, because the code segment is static and the data segment is changing by an operation of the component.

Consequences

Advantage:
> Information integrity can be assured by the pattern.

Disadvantage:
> The memory usage of the component increases due to redundant information that must be saved.
> The performance of the components is influenced negatively due to the overhead to store and to check the additional redundant information.
11. **General Safety and Reliability Patterns**

The following patterns also apply to the architecture of the system to solve different problems in safety and reliability quality factors and have been taken from literature. Though these patterns are also used in design of the system and address the design perspective but many of the architectural issues are covered by applying these general safety patterns. These patterns have been given with the problem they address, description of the solution and the rationale behind them which justify the solution domain. They are applied according to the following scenarios (combination of stimuli and responses against them) in the system and are chosen with respect to the context and the description of the system requirements [general patterns-53].

**Stimuli and Responses**

The stimuli of safety and reliability issues mostly arise in software faults, hardware faults, and unexpected behaviours of events or invalid inputs of the system which procures a wrong output. The responses to these stimuli involve the high availability of the system in most of its time. The other responses are tolerance, detection, high performance and high level of services in the presence of stimulus. Below are those patterns which apply these responses to keep the safety and reliability integral into the system.

11.1. **Homogenous Redundancy Pattern**

**Context**

Most of the faults in a safety system happen randomly which actually turn the system weaker and uncertain. The capability potential of a system is required to the level where a continuation of a function would not be compromises in the presence of failure.

**Problem**

Increase reliability and safety in the presence of single point hardware failures with only small efforts.

**Solution**

The homogeneous redundancy pattern is to provide reliability by offering multiple channels. This pattern uses a number of identical channels running in parallel and whose output is compared by a controller to increase reliability. The channels are a set of devices including processors and software running on processors that handle a related cohesive set of data or control flows from incoming event to ultimate system response. When using an odd number of channels, a majority-wins policy can be implemented that can detect and correct failures in the minority channels.

![Homogeneous Redundancy Pattern](image)

**Rationale**

The channels are identical and save the redevelopments but it only detects the failures rather than the errors. It mostly protects against the hardware failures, but not software errors. The
cost can be high and more power consumption is required in this pattern if the redundant hardware devices are available.

Consequences

Advantages:
- Homogeneous redundancy pattern improves the robustness in the presence of failures in the channel without large development overhead. It provides good coverage for random faults.
- The pattern is easy and simple to design especially with respect to hardware equipments.

Disadvantages:
- It can detect only failures but not the errors in the system while keeping in view that failure is the event that happens in particular time but the error is the systematic fault which is a flawed system condition.
- The cost can be higher because of the duplication of the channels in the system.
- It does not detect error in code and if the channel is flawed then all the channels are flawed.

11.2. Diverse Redundancy Pattern

Context
If the error is intrinsic in the system design or implementation then it is required to save the system through run time checking to avoid the errors and failures.

Problem
Increase reliability and safety in the presence of single point failures and systematic errors. When a primary channel detects a fault, a secondary channel takes over.

Solution
The diverse redundancy pattern mitigates the primary disadvantages of homogenous redundancy by providing redundant channels that are implemented using different means. Depending on how the redundant channels are implemented, diverse redundancy patterns possess different characteristics.

The most expensive type of diverse redundancy is when all the channels are approximately equal in terms of accuracy, throughput, and reliability and each channel is implemented differently using different software and hardware components. Two special types of this pattern are the Monitor-Actuator pattern and the Sanity-Check pattern that are much less expensive.

![Figure 25: Diverse Redundancy Pattern. After [56]](image-url)
Since all channels are implemented differently using different software and hardware components, a system implemented using this pattern cannot only detect failures and errors, but can also continue to execute correctly in the presence of such faults.

**Consequences**

**Advantages:**
- It provides the improved tolerance of faults by detecting the systematic faults.
- This pattern is better in fault detection than fault tolerance and can continue to work properly in presence of failures and errors.

**Disadvantages:**
- This is the most expensive kind of redundancy because not only the recurring cost is increased but development cost is increased as well intensive design efforts are required.

### 11.3. Monitor-Actuator Pattern

**Context**

When a system does not have extra ordinary high availability requirements, it can be put in fail-safe condition-condition of the system known to be always safe then the lower cost redundancy is applied.

**Problem**

Increase reliability and safety in the presence of single point failures and systematic errors at reasonable costs.

**Solution**

The monitor-actuator pattern is a special type of diverse redundancy and separates the channels into monitor and actuator channels. While actuator channels perform the actions of controlling or moving mechanical devices, the monitor channel keeps track of what the actuation is supposed to be doing. It also monitors the physical environment to ensure that the results of the actuator are appropriate. The basic concept behind this pattern is that the monitor channel identifies actuation channel failures so that appropriate fault-handling mechanisms can be executed.

![Figure 26: Monitor Actuator Pattern after [53]](image)

**Rationale**

- Although the additional monitoring channel increases the recurring cost, but it increases reliability and safety of a system in a more cost-effective manner diverse redundancy pattern.
- The system continues to be correct if the monitor channel fails and the actuator did not fail. Since the monitor and actuator channels often exchange messages
periodically to ensure both are operating properly, performance might be influenced negatively in some cases.

**Consequences**

**Advantages:**
- If the monitor channel fails, it is not necessary that actuator also fails so it continues to be correct under the single fault isolation assumption.
- Monitor actuator actually provides relatively inexpensive safety solution and have the ability to function in systematic and random faults.

**Disadvantages:**
- It can only be used for those systems which are required to work under fail-safe state.
- Monitor actuator pattern is not suitable for the systems which require high availability requirements.
- A system cannot function under this pattern if the fault is identified.

11.4. **Sanity-Check Pattern**

**Context**
Sometimes the purpose of the system is to ensure that system is doing more or less something reasonable if not absolutely correct. It happens when actuation is not as critical but is able to harm the system if it goes wrong.

**Problem**
Provide a simple and cost-effective means for ensuring correctness and detecting faults rather than tolerating faults particularly when there is a known fail-safe state.

**Solution**
As a type of diverse redundancy, the sanity-check pattern includes two channels: a primary channel that realizes the functionality and a lightweight secondary channel that does not have same accuracy and/or range. The secondary channel only ensures the correctness of the primary channel by performing a reasonableness or sanity check on the results from the primary channel. It is a variant of the Monitor-Actuator Pattern and Diverse Redundancy Pattern.

**Rationale**
Since the secondary channel does not have the same accuracy and/or range of the primary channel, the sanity check pattern is much better at fault detection than fault tolerance. In fact, the secondary channel can identify when the primary channel is broken but typically cannot take over all its duties when this occurs. Thus, other fault tolerance means must be present if the system must remain operational in the presence of errors. However, the sanity check pattern is not very expensive to implement and manufacture.

11.5. **Watchdog Pattern**

**Context**
The actual output of the system is sometimes not concerned but the internal computation processing is sometime monitored for the expected results.

**Problem**
Provide a very simplistic and low-cost solution to detect and handle failures and faults during the execution of safety-critical systems.

**Solution**
A watchdog is a component that observes the behaviour of other components and that acts when they appear to misbehave. In general, a watchdog is implemented as a passive receiver
of liveness messages (or strokes) from the observed components on a periodic or sequence-keyed basis. If an event occurs too late or out of sequence, then the watchdog initiates a corrective action. Watchdog Pattern merely checks that the internal computational processing is proceeding as expected. This means that its coverage is minimal, and a broad set of faults will not be detected. On the other hand, it is a pattern that can add additional safety when combined with other heavier-weight patterns.

![Figure 27: Watch dog Pattern. After [55]](image)

**Rationale**
The watchdog is a low-cost solution for ensuring the liveness of a component but does not provide very complete coverage and the normal action to be taken when the watchdog time out is reset.

**Consequences**

**Advantages:**
- Watchdog is simple and mostly implemented in hardware to protect them from software faults.
- If a component does not send a message within its defined period, the watchdog takes corrective actions (close, reset, alarm or signal a more complex component).

**Disadvantages:**
- Watchdog can be too simple to support error handling and fault recovery.
- Watchdog is very hard to be integrated with main system.
- Watchdog is not very much suitable for the very critical safety systems and requires to be used in combination with heavier design patterns.

**11.6. Safety Executive Pattern**

**Context**
Sometimes the control of the safety measures of the system is very complex because the system cannot be shut down and is needed to follow some sequence of actions to achieve the fail safe state.

**Problem**
Provide a consistent and centralized point for managing potentially complex safety processes in the presence of faults. The pattern is particularly useful when many multiple safety concerns have to be addressed like when fault detection is complex, or the fault recovery mechanisms are elaborated.

**Solution**
The Safety Executive Pattern uses a centralized coordinator for safety monitoring and control of system recovery from faults. The safety executive tracks and coordinates all safety monitoring and ensures the execution of safety actions when appropriate. Typically, it captures watchdog timeouts, software error assertions, or faults identified by monitors in the Monitor-Actuation pattern.
Rationale
Because in the safety executive pattern the safety mechanisms are centralized and isolated from the other system components, complex fault isolation and recovery can be handled. It also makes safety measures can easily verified and validated.

Consequences
Advantages:
- It isolates the higher level application from the safety issues and prevents extra checks and recovery actions.
- The advantage of the safety executive pattern is that the safety mechanisms for complex systems are centralized and isolated away from other system components.
- This pattern also has the ability to handle complex fault isolation and recovery.

Disadvantages:
- It has sometimes very complex design and difficult to establish it.
- Safety executive pattern is also a little expensive to implement.
12. Safety Formulation Suggestion

Safety is not an easy concept to absorb especially when it comes to the technical side of software. The process aspect of safety may contain some descriptive elements in the form of process evaluation and standards but when the concept related to technical issues are addressed, they become mingled up exhaustively. The relation of these concepts have been tried to put in a figure to conceive the relation they have.

These concepts show that the relation between them is quite complex and if we try to bring out any of these areas, the safety can be compromised. There are still many other entities to strengthen the safety but as many as understood from literature have been described above. The concept of safety is neither too much concrete nor does it rely on only one quality aspect of the system. Also research on this area is quite interesting and also pretty new.

12.1. Patterns’ Investigation Results

The patterns given above have been scrutinized for their building blocks, requirements, solution domain and distillation. They have the strategies and techniques to fulfil those issues in specific conditions. The scenarios then can be generated from these techniques and scenarios in which they are probable to be applied. The investigation of the strategies and technique in the form of below table is not complete but gives a broader list of the common patterns used in safety domain and thus a further study can be taken up for more results.
These listed scenarios have been derived from the strategies and the techniques available in the safety patterns. They help to understand how the solution domain of patterns actually can determine the requirements and responsibilities from them. The can be broken down into more refined scenarios and then can be utilized the same was as USAP has introduced before. The techniques and the mechanism shown above are common in different patterns and are used in for defining the scenarios. Some of the safety formulizations are also being evaluated on the basis of these scenarios and techniques. They also help to dig down into the requirements and refined requirements during the safety architecture evaluation.

12.2. Safety Formulation Suggestion I

The first suggestion to incorporate is a hierarchal model that has been used in some patterns and can be implemented with the help of already existing patterns. The first and the foremost place to start with are always, from the requirements because the architectural decision taken on that level always saves the efforts and time. The requirements are gathered and specified in terms of goal here. These goals can be categorised in different formulation of fault and
failure classifications. Then theses requirements in the form of goals are further broken down to elaborate the context and the technique or mechanism to satisfy the goals in broader context. The classification of the failures actually formulates the mechanism to determine and adapt for the special type of requirement goals and thus uses different techniques to eliminate them.

![Diagram of safety formalization suggestion](image)

**Example:**

- **FD-Failure Detection (Goal - Objective)**
  Detection of the safety critical faults can protect the system to fall into the havoc. The faults should be detected as early as possible into the system and the proper action need to be taken. Fault detection is a mechanism to recover them sooner in a system or even before putting a system into its real environment. The detection of faults can prevent a system to introduce more faults which may lead to fail other components in a system.
  - **FD1 - Comparison (Strategy and Mechanism)**
    - **Mechanism Based Refined Requirements**
      **FD1.1**-The system must have a unit of a functionality which has defined input and output and can access them. (Computation)
      **FD1.2**-There must be a model available in a system as like algorithm, protocol, network, and topology to follow.
      **FD1.3**-There should be a functionality or unit of a function to compare the output from various entities. The output should be the one result to evaluate whether it matches or not with the other units. (Comparator)

- **FD2- Breakpoints (Strategy and Mechanism)**
  - **Mechanism Based Refined Requirements**
    **FD2.1**-The system must have a unit or a component that checks the validity of the interaction and the data sent through it at the time the interaction is active.
    **FD2.2**-There must be a technique in a system which can allow the redirection of program flow.
There should be a functionality or unit of a function/module or component, which detects the presence of a failure in the output of another module. (Error Detector)

There should be a functionality in a system which can inject error intentionally in a system to detect and separate the actual data from the errors.

- **FD2- Snapshot (Strategy and Mechanism)**
  - **Mechanism Based Refined Requirements**
  - **FD3.1**-The system must have state manager that records the current state of the system.
  - **FD3.2**-A repository should be available to keep the state records in a system.
  - **FD2.3**-System must have a test interface to have an access to recorded states to take further actions.
  - **FD2.3**-Test ports should be available in a component which is highly likely to produce errors. These ports help to detect the errors during program and component execution.

**FP-Failure Prevention (Goal-Objective)**
Failure prevention is a defensive objective of the system. It means that the system is made strong right from the beginning so that the chances of failure to happen later into a system are minimized. The goal includes that even in the case of any fault; the system is still safe to work according to its desires situation.

- **FP1 - Logical Redundancy (Mechanism)**
  - **Mechanism Based Refined Requirements**
  - **FP1.1**-The system should have the capability to implement single functionality in different ways e.g. different algorithms for the same calculation. This unit is used to be called the redundancy. (N-Version Programming for Second Computation)

- **FP2 - Physical Redundancy (Mechanism)**
  - **Mechanism Based Refined Requirements**
  - **FP2.1**-The system should have the capability to implement single functionality in different ways e.g. different algorithms for the same calculation. This unit is used to call the redundancy. (N-Version Programming for Second Computation)

  - **FP2.2**-The system should be able to execute more than one task at one moment of time. It needs additional hardware mostly or CPU. (Processing Hardware)

- **FP3 - Acceptance Test**
  - **Mechanism Based Refined Requirements**
**FP3.1**—There should be a component or an entity, which validates an input as correct in a given domain. (Oracle)

The same way a hierarchy can be defined for failure prevention, failure recovery and mitigation depending on the techniques and mechanism related to them. These refined requirements are not complete yet for each of these mechanisms but at least they give a better and understandable view of the suggested hierarchy.

### 12.3. Safety Formulation Suggestion II

There can be another formulation to help out the architectures to cater the safety requirements and aspects of system in different and various disciplines of software safety products. This formulation is actually a hybrid approach which covers the requirements as reoccurring issues and the tactics along with them together. The tactics make a wider decomposition of the requirements and the reoccurring requirements cover the detail of them. Thus a mixture of both narrows down the scope of the problem to be solved and provides the components of the patterns in the form of requirements. These requirements and tactics can be observed during the decision making about the architecture and the requirement specifications.

This approach actually works in opposite direction in contrast to other solutions where requirements and problem leads to solution. But in this solution we go from solution to requirements and analyse whether all the requirements related to that solution have been captured into the system architecture or not. It also gives a wider and broad view of the requirements to the architecture and he/she can think of the system design by keeping in mind other aspects in that context efficiently and completely.

![Safety Formulation Suggestion 2](image)

Figure 31: Safety Formulation Suggestion 2

### Scenario Based Requirements
If the requirement is time related and result is dependent on the timing behaviour and response of the system. These requirements can be taken by the defined requirements patterns of Friedemann Bitsch [45].

❖ Timing Scenarios(Goal-Objective)

➢ Scenario Based General Safety Requirements

➢ Time Stamp (Tactic)

TST1-The system should check time to time for the state and be able to store the state somewhere to recall it.
TST2-The system should have mechanism to recall the required state.

➢ TS - Time Stamp Scenario Based Refined Requirements

TS1-The safety requirement only with implicit time specification
TS2-The safety requirements with explicit time
TS3-The requirements with duration of validity
TS4-The requirements with the beginning of validity or points in time of validity
TS5-The safety requirement of duration and beginning of validity

➢ Time Out (Tactic)

TOT1-The system checks the functionality for a specific time period and after a defined limit of that time period.

➢ TO - Time Out Scenario Based Refined Requirements

TO1-A requirement with validity/reachability from a certain point in time
TO2-A requirement validity/reachability strict after a point in time
TO3-A requirement validity/reachability exact/immediately after a point in time
TO4-A requirement validity/reachability until a certain point in time
TO5-A requirement validity/reachability before a certain point in time

➢ Barrier (Tactic)

BRT1-The system should have a mechanism to stop the functionality in the system on some conditions to avoid the faults at initial levels.

➢ BR - Barrier Scenario Based Refined Requirements

BR1-A requirement validity prohibition before point in time / conditional
BR2-A requirement reachability prohibition inclusive the time before the point in time
BR3-A requirement validity prohibition until point in time / conditional
BR4-A requirement reachability prohibition inclusive the time until the point in time
**BR5**-Requirement validity/reachability permission before point in time  
**BR6**-Requirement validity/reachability permission until point in time  
**BR7**-A requirement validity prohibition from point in time on/conditional reachability  
**BR8**-A requirement prohibition inclusive the time from the point in time on  
**BR9**-A requirement validity prohibition strict after point in time /conditional reachability  
**BR10**-A requirement prohibition inclusive the time strict before the point in time  
**BR11**-Requirement validity/reachability permission from point in time on  
**BR12**-Requirement validity/reachability permission strict after point in time on

The same hierarchy can be found for the sequence, value, actions and amount domain for the safety failures. These tactics can be taken alone for the mitigation of the failures or a combination of them in appropriate situations can also be a good example to employ.

### 12.4. Safety Formulation Suggestion III

This suggestion is based on different safety perspectives, dangers and threat types. The danger and the extent to which a danger can harm is a natural way of safety thinking. Security and safety requirements are different than other type of requirements in a system. Since the general requirements are defined in terms of what should be there in a system but in safety context it requires that what must not be allowed to be existent in a system. The natural idea of thinking about the severity of danger and type of threat actually thus determines this type of suggestion. The dangers and threats type affecting the safety measures of a system can be reduced by listing down the counter measures in a number of ways. These counter measures can be seen in terms of refined requirements for the evaluation of the system. Each of these counter measures are broken into many granular requirements in designing a safety system.

The architect goes through these dangers and threats of safety which a system is supposed to handle and then takes a glimpse of the refined requirements of the counter measures for those threats. The rationale, reason and the context in which these counter measures have to be taken are also part of the formulation suggestions.
 **Free access:** If information or data can be manipulated, accessed and changed with no rule then it is called a free access. Free access is actually an easy way to exploit the environment, communication, data or any concept where the restriction has to be applied.

**Restriction/Authorization/Authentication (Countermeasure)**

- **RE - Restriction Based Refined Requirements**
  
  **RE1** - There must be a way that the role of a user is known in the system to grant access.
  
  **RE2** - A system should only provide access to those parts to the user for which a role has been allowed to take privileges.
  
  **RE3** - The system should not show the areas where the user is not allowed to enter to minimize the exploitations and risks.
  
  Etc.

- **AU - Authorization Based Refined Requirements**
  
  **AU1** - A user can use only those commands for which he/she is allowed to change data or communication in a system.
  
  **AU2** - Only administrator can change the privileges in the system.
  
  Etc.

- **AT - Authentication Based Refined Requirements**
  
  **AT1** - A way to identify the user must be handled and the user should be given a way to enter his/her credentials.
  
  **AT2** - There should be a mechanism in the system for hiding the credential of a user during authentication.
  
  Etc.

- **Data/Communication Loss:** The loss of data means unforeseen or missing of information in a system. Loss is one of the factors that breach the safety of a system. It can be in the form of data, communication, instrument, action or command.

**Redundancy/Backups (Countermeasure)**
RE - Redundancy Based Refined Requirements  
RE1-System must provide a different channel in a case of redundancy channel so that the original channel is not dependant on it.  
RE2-The system must provide a way to check the redundant channel/data for its correction in a specified time.  
Etc.

BU- Backup Based Refined Requirements  
BU1-The storage of the data in the case of backup should not be places in the same location.  
BU2-System should provide a way to keep the log of backup to check it later.  
Etc.

Endangered Life: The most important and prominent factor in any safety place is the life or lives that can be protected. The foremost threat to any safety system is about the life as a greatest risk.

Alarm/Instrument/Controlled Environment (Countermeasure)  
AL - Alarm Based Refined Requirements  
AL1-The system must have a mechanism to inform before any dangerous event occurs or prevail in an environment.  
AL2-A warning must be made keeping in mind about the lives, mode and environment of a system.  
Etc.

IN - Instrument Based Refined Requirements  
IN1-Reserved instruments must be available to utilize during the time of danger.  
IN2-There must be a way to access the reserved instruments before any mishap happens.  
Etc.

CE - Controlled Environment Based Refined Requirements  
CE1-There must be a way to control the environment before it goes to havoc.  
CE2-A mechanism should be provided to control the environment outside of its boundary in a case of internal control goes unhandled.

Alike the hierarchy above, other threats and their counter measures can be arranged. These threats may include breaking, hindrance, deterioration, negligence, communication barriers, vagueness, diversity.

12.5. Analysis of safety formulization suggestions  
The suggestions for formulization of safety from the different perspectives are a better way to understand the safety completely and comprehensively. These suggestions have been mentioned in different ways which in fact serve to different purposes in different conditions. Furthermore, as ArchWiz tool is designed to serve as a requirement tool on first priority but if the safety quality attribute has to be kept in mind then these suggestions also suite for
architectural evaluation. The most appropriate suggestion for the requirement tool perspective is suggestion 1 because it directly involves safety objectives and the mechanisms to extract out the requirements as well as the hint of their solutions. Suggestion 1 is also well formed and aligned with the USAP model and ArchWiz tool in the sense of clear requirements specification. Suggestion 2 mostly uses the tactics involved in safety measures but those tactics have been used in conjunction with the requirements they serve for. Most of those requirements mentioned along with tactics have been taken from Friedemann Bitsch[45] research papers and writings because they are quite practical and have been classified in clear manners. These requirements have been listed on a large number in his literature from where it is easy to develop them according to the suggestion 2 model. The third suggestion is a reverse approach to attain safety in a system or environment. The strong point of suggestion 3 is that it gives a view to an architect to look upon the system from the perspective of loss and severity of it. It also extends the common perception of safety issues because the different threats and their counter measures relate to each other which in turn bring upon new perspectives. They can also be used in threat model and if evaluation has to be taken as a preferred element in ArchWiz tool.

The suggestions have been supported with UML figures to show the proper hierarchy in which there requirements have been formed. These UML elements can have the contents in the form of rational, implementation details, node type and details of them. This also depends on how the ArchWiz tool has been used—whether for requirements specifications of the evaluation of architecture in safety context. UML diagrams in these suggestions also demonstrate the level of requirements on which they happen to exit in hierarchy. The structure of the requirements hierarchy is also flexible since it suits to the ArchWiz tool usage. These suggestions can also be used in collaboration with each other if the future development intends ArchWiz to focus for requirements specification purposes.

The common depiction of the suggestions can be elaborated in future work also. In all of these suggestions safety has been broken down in requirements and then refined requirements. The decision is finally taken on these refined requirements which can be further refined and is a work of future studies.
13. Summary and Conclusion

During the development process of software systems, software architecture plays an important role and remains a key factor to decide about the structure of better design. It determines in the very initial phases that whether the software will have the potential to meet its requirements or not. It makes the design decisions easy if kept in mind earlier according to the requirements level. Software architecture thus has to capture all side of the system specially the quality factors of the system.

Safety is an important and crucial quality of software that has a strong relation to the software architecture as well. The research and studies are growing in this field to capture the safety issues in designing the architecture well before time. Though safety is an inherent quality of the software and breach of it is not an intended play from the outside factors but the threats seemingly inside from the system actually turn into big destructions. Therefore, these failures, attack, threats, errors, faults and deficiencies have to be addressed and removed. Incorporating safety mechanisms in the system and knowing how to handle the safety measures in the system may give better architectural design decisions and safer systems than before. The requirements to make a system safe are not very clear though in every design but are taken from harms and dangers encircling the system with undesirable behaviours. Safety requirements are derived from the idea what must not be allowed to happen in a system unlike other requirements which are expressed in terms of higher goals and what must be within the systems. Safety requirements are based on the understanding of overall system and analysis of implicit or explicit dangers or harms that affects the assets of value. Therefore, it appears that safety requirements are supposed to be handled in slightly different manners than the others. Because they mostly are unseen and may not have occurred at all before in time, so a minor requirement can have vital influence on overall system. The other reason for safety requirements to be handled differently is that they act like a chain reaction-once a safety deficiency is there then most probably the others will be triggered sooner or later.

The ArchWiz tool has been developed to help architects in ABB to capture a big number of requirements in relation to the system for the important quality attributes such as usability, security, safety during architecture design. The efficient usage of this tool depends on the better formulization of quality requirements that can be provided generally for all system as a default way to check its quality concerns. Once a general list of quality requirements has been established and a good formulation of a quality factor has been introduced then the users can easily use them for their particular systems and according to their system specifications.

Designing a good system architecture and flexible structure has already been managed and guided by patterns. These patterns are architectural and some of them are more concerned with design issue. These patterns solve a wide range of problems in different areas or at least give guidance to their solutions. USAP researchers have already intrigued these patterns and have introduced some more in usability scenarios in special way and formation. Although there is quite large number of patterns in safety domain but they also have dependency on reliability and availability domain. They are very helpful and work as a guidance for developing safety software. Many of these patterns have already been used and implemented for their benefits and usefulness, so utilizing them in different safety issues would enhance the architectural problems and requirement specification details. These patterns have also helped to formulize the safety suggestions in quite similar way of USAP and thus can be introduced successfully in ArchWiz tool.
14. Future Work

14.1. ArchWiz Tool

Developing ArchWiz tool was a challenging task because the time period of implementation and resources were limited. These limitations made ArchWiz not to be finished completed by the end of the thesis implementation time. There were quite many implementation tasks on which development was going on. The management would decide on these tasks to carry on in future and further development. These tasks include some of the GUI information, cooperation between different modules and their extended functionalities. The intended future comprise of import/export functionality for loading knowledge base files, a to-do list to generate the used based decisions on architecture evaluation of a specific system, proper population of the data coming from project files, helping functionalities, tree node modifications and progress bar status.

The basic structure can thus be extended and also modified to make the ArchWiz tool in full working condition. There were some changes in the requirements during the ArchWiz implementation because it was also intended to make ArchWiz tool a pure requirement specification tool but those changes can be incorporated in future development.

14.2. Safety Suggestions

The construction of safety suggestions have been designed keeping in mind the structure of USAP and usage of ArchWiz tool. These suggestions use patterns, threats, requirements, refined requirements and tactics. In suggestion 2 and suggestion 3 a concept of scenario has also been introduced which can be extended further for suggestion 1. The scenario can help to visualize the architect to look for the requirements and refined requirements in their proper context. Also the list of refined requirement is not a complete list in each suggestion rather only a pointer to deduce them from patterns or techniques intended to solve those issues.

The combination of using tactic and pre-defined requirements (serving as pattern of requirements) can also be studied more. Breaking down the requirement into their refine form is not an easy task and consume quite much time. Hence a research can be carried on to intrigue these patterns and their distillation likewise USAP purposes.
15. References


[29] Donald G. Firesmith, “Engineering Safety-Related Requirements for Software-Intensive Systems,” (ICSE), St. Louis, Missouri, USA, 2005


