CHAPTER 2. SOFTWARE LIFE CYCLE MODEL FOR SAFETY SYSTEMS

2.1 Introduction

Software life cycle, is a structure imposed on the development of a software product. It is considered as a subset of systems development life cycle. There are several models for such processes, each describing the approach to a variety of tasks or activities that take place during the process. ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software [2.1]. Assessing various life cycle models and the identification of a suitable software life cycle model for safety systems, which ensures high reliability. software delivery is one of the prime tasks of the current research.

2.2 Available software life cycle models

The Planning, implementation, testing, documenting, deployment and maintenance are the steps in the software life cycle development process. Several models exist to streamline the development process. Each one has its pros and cons, and it's up to the development team to adopt the most appropriate one for the project.

2.2.1 Waterfall model

The waterfall model depicts a process, where developers are to adopt the following phases in order:

1. Requirement Specification (Requirement Analysis)
2. Software Design
3. Integration
4. Testing (or Validation)

5. Deployment (or Installation)

6. Maintenance

In this model, after each phase is completed, it proceeds to the next stage. Reviews may occur before moving to the next phase, which allow for the possibility of changes. Waterfall discourages revisiting any prior phase once it is complete. This "inflexibility" is the main limitation of this model [2.2].

2.2.2 Spiral model

The main characteristic of a Spiral model is risk management at regular stages in the development cycle. Barry Boehm [2.3] proposed a formal software system development "spiral model", with emphasis on a key area of risk analysis that is neglected by other methodologies. The spiral model is visualized as a process passing through a number of iterations. The first stage is to formulate a plan, and then strive to find and remove all potential risks through careful analysis and, if necessary, by constructing a prototype. If some risks cannot be ruled out, the user has to decide whether to terminate the project or to ignore the risks.

2.2.3 Iterative and incremental development

Iterative development model prescribes the construction of initially small but ever-larger portions of a software project to help all those involved to uncover important issues early before problems or faulty assumptions can lead to disaster. Iterative processes can assist with revealing and refined definition of design goal [2.1].
2.2.4. Agile development

Agile software development uses iterative development as a basis but advocates a lighter and more people-centric viewpoint [2.4]. Agile processes use feedback as primary control mechanism. First, one writes automated tests, to provide concrete goals for development. The next step is coding by a pair of programmers, which is complete when all the tests are successfully passed. The incomplete but functional system is demonstrated for the users. At this point, the practitioners start again on writing tests for the next most important part of the system.

2.2.5. Code and fix

"Code and fix" development is not a deliberate strategy and schedule pressure on software developers. With incomplete design, programmers begin producing code. At some point, testing begins (often late in the development cycle), to fix the bugs before shipment [2.1].

2.3. Model suggested for NPP

Since the software requirements are very well defined and finalized in the case of software for NPP, the waterfall model may be suitable [2.5]. But there is no checking mechanism in each stage of life cycle, so, by introducing verification at the end of every stage [2.6] with the respective artifact and enforcing validation of the product (code) with the specification it can fulfill the safety requirements of NPP [2.7]. The modified model suitable for software to be deployed in NPP is shown in Figure 2.1.
This model ensures verification at all the stages before moving on to the next phase and also the validation of the product. Each artifact is checked against the testing scheme, which ensures the completeness and correctness of the artifact. It also provides a complete control on the life cycle of the software, which in turn ensures well-matured and stable software product, which is the requirement for NPP software [2.8]. As the “test-cases” generated by an independent team in the presence of the developers is carrying out the testing, it ensures that the software product confirms to the specification of its indented functions. The software developed for safety system of fast reactor by following this “V” model resulted in better quality and phenomenal improvements in reliability. The only key parameter is that the testing team is an independent team but with the domain knowledge and its deployment with respect to the other systems and modes of operation. It should also be noted that an inherent assumption of the study is that the software described follows a waterfall life-cycle [2.9]. A waterfall life-cycle is typically characterized by the succession
of the phases from requirements to operation without too many backwards steps such as for instance the fact of going back from design to requirements. Other software development lifecycles exist such as for instance the spiral model [2.10], a lifecycle where development is driven by perceived risk areas and the resolution of these risks in an iterative fashion. Spiral development makes heavy use of prototyping and is typically used for software with a strong user interface component. Waterfall development on the other hand is recommended for programs with strong algorithmic component such as the software used in safety applications [2.11].

The Table 2-1 shows the mapping of the software development phases used in this study to the IEEE610 standard phases [2.12].

<table>
<thead>
<tr>
<th>Life-Cycle in this Study</th>
<th>Equivalent Life-Cycle in IEEE610 standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Concept</td>
<td>Requirements</td>
</tr>
<tr>
<td>Design</td>
<td>Design</td>
</tr>
<tr>
<td>Implementation</td>
<td>Implementation</td>
</tr>
<tr>
<td>Testing</td>
<td>Test, Installation &amp; Checkout Installation</td>
</tr>
<tr>
<td>Operation</td>
<td>Operation &amp; Maintenance, Retirement</td>
</tr>
</tbody>
</table>
REFERENCES


