Chapter 3

Software development processes

3.1 Introduction

The objective of software development is to efficiently and predictably deliver a software product that meets the needs of the community of its stakeholders. A *process* is a set of ordered steps intended to reach an objective. A *software development process (or model)* is an approach to building, deploying and maintaining software. The motivation behind defining a process for software development is to manage the size and complexity of software systems.

There is no single approach to such a process, and various models exist, some of which are discussed in the subsequent subsections. However, a key principle behind any process is the breakdown of a project into manageable components. This idea lies also at the heart of the differences between processes. On one hand, a breakdown can be performed in terms of activities. As an example, one may consider a 6-month project to break down into a 1-month requirements elicitation and analysis activities, a 1-month analysis, a 2-month architecture and design activities and a 2-month coding and testing activities. On the other hand, breakdown can be performed in terms of subsets of functionality, where one goes through all activities in successive cycles or ("iterations") where each iteration produces a subset of the overall functionality. For example, the first iteration may produce 20% of the overall

functionality, the second iteration may produce an additional 30%, etc. A common technique to iterative approaches is *timeboxing* which refers to forcing a fixed length to iterations. Should it be the case that not all planned functionality can be delivered in the current iteration, timeboxing implis that some functionality is slipped to a future iteration rather than having to slip the deadline of delivery.

The two different ways to project breakdown divides processes into two categories: *Linear process models* and *iterative process models*. The latter comes in many variations (some of which are very similar), but it is important to note that there is no such thing as a good or bad process model(s). In the subsequent subsections we will discuss some of the major models together with criteria for their deployment as well as possible advantages and disavdantages.

3.2 The waterfall software development process

The "waterfall" is a software development model in which development is seen as flowing steadily downwards (like a waterfall) through the activities (called "phases") of requirements, analysis, design, implementation, testing (verification), integration, and maintenance. The model follows a linear path of development (as opposed to "iterative" – see subsequent sections). When compared to its alternatives, it is referred to as the "pure" model (See Figure 3.1).

The waterfall model proceeds from one phase to the next in a sequential manner, and the model advocates that one should only move to a new phase once the preceding phase has been fully completed (and perfected). For example, one first completes and freezes requirements specification before proceeding to design. When the design is complete, one proceeds to implementation, etc. In the waterfall model, the software system is developed not in releases but as a whole and feedback is available once the system is delivered. The

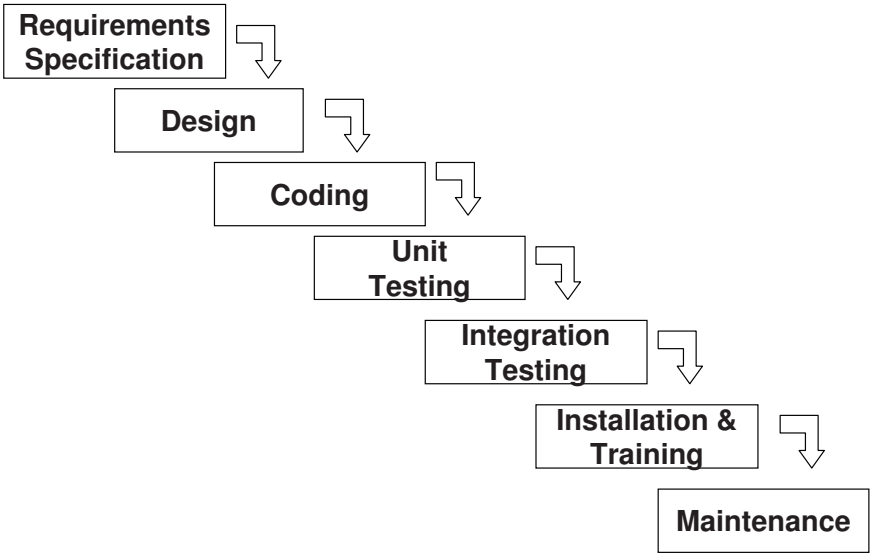

Figure 3.1: The pure linear ("waterfall") development process. Adapted from W. Royce, *Managing the development of large software systems*, IEEE Computer Society, 1970.

model provides limited (if not minimal) stakeholder feedback typically in the form of reviews and "sign-offs" (formal conclusion of a phase) between between adjacent phases of development. The model has its origins in the manufacturing and construction industries in which changes are either very costly or simply impossible. The model can be adopted in cases where both functional and nonfunctional requirements are well defined and the product is planned to be delivered in a single version.

3.2.1 Criticisms of the waterfall model

As requirements tend to change throughout development, it is usually not feasible to fully complete and "freeze" the requirements specification, then fully complete the design, then build, and only then proceed to test the software. Two associated problems with the linear model are as follows:

1. Risks are not explicitly addressed. As such, they can be encountered late in the development, and

2. Lack of flexibility of changing requirements, particularly in dynamic domains.

3.2.2 The modified waterfall software development process

In response to the criticisms of the "pure" waterfall model, the modified waterfall model realizes that feedback can lead from code testing to design (as testing of code may uncover flaws in the design) and from design back to requirements specification (as design problems may uncove conflicting or unsatisfiable requirements).

3.3 Spiral development model

Introduced by Boehm[1] in 1986, the *spiral development model* (see Figure 3.2) was the first to address the importance of iteration in software development. In this model the system requirements are defined in as much detail as possible. The notion of "iteration" (or "cycle") lies at the heart of this model. An iteration are an important concept as it is applicable in other development models (see later). An iteration follows the waterfall model and it is essentially a mini-project, containing the following development activities:

1. Determine objectives; Specify constraints: Define the problem addresses by the current iteration.

2. Generate alternatives: Define solution(s).

3. Identify and resolve risks. Risks are addressed in order of priority.

4. Develop and verify product: Work through requirements analysis, architecture, design, implementation and testing, producing a prorotype.

[1]Barry Boehm, A Spiral Model of Software Development and Enhancement, ACM SIGSOFT Software Engineering Notes, Volume 11 Issue 4, August 1986, ACM New York, NY, USA.

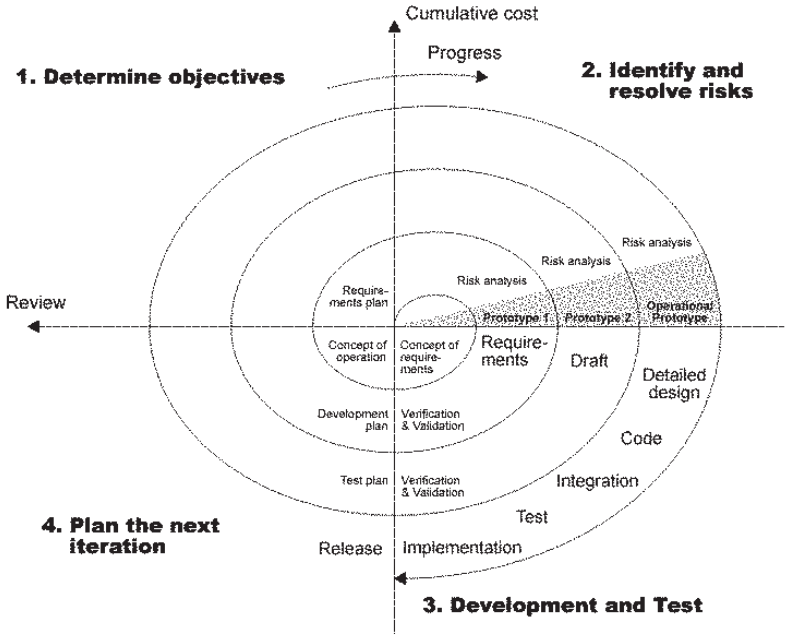

Figure 3.2: The spiral development model.

5. Plan: Prepare for the next iteration.

An iteration addresses only a subset of the initial set of requirements. Development involves a number of iterations whereby each one addresses a subset of requirements and producing a prototype (whose strengths and weaknesses are evaluated) until eventually an operational prototype is delivered. Interim prototypes may or may not be deliverables.

Additionally, each iteration includes four phases: During the first phase (upper left quadrant) developers determine objectives, constraints and alternatives. During the second phase (upper right quadrant), developers identify and resolve risks associated with the solutions defined during the first phase. We should note that (together with the cleanroom model: see later), the spiral model is the only one that explicitly includes risk management. Risk identification and analysis is performed during each iteration of the spiral. During the third phase (lower right quadrant) developers implement a prorotype associated with the risks identified in the previous phase. During the fourth phase

(lower left quadrant) developers plan the next iteration based on the results of the current iteration. The distance from the origin represents the cost accumulated by the project and the angular coordinate indicates the accomplished progress within each phase.

In its purest form, the spiral model bears a similarity to the waterfall model in that the system is available only when it is complete, not in phased releases (even though it could be adopted to provide releases). However, unlike the spiral model, the waterfall model makes no attempt to identify and tackle riskiest requirements first.

3.4 Evolutionary development model

In evolutionary development, the software requirements for the first version of the product are established, and the product is developed and delivered to its stakeholders. During development or after the first product delivery, the requirements for a second or third etc., product release are defined and implemented for the next release. These evolutions of the product are referred to as evolutionary spirals. Each evolution has the characteristics of a waterfall process. Ideally, there should be no overlaps between developments. However, if overlaps must exist, then it is vital that the development of the second release does not start until the design of the first release is or has become stable. Microsoft Windows operating system and Microsoft Word wordprocessor are examples of evolving products.

3.5 Iterative and incremental development model

An important point in this model is that the complete set of requirements is elicited, analyzed and allocated to individual small scale projects (called increments, or iterations) before full-scale development begins. The main idea behind the iterative and incremental development model is to build a system

through repeated development cycles and in relative small portions. An iteration represents a complete development cycle and it includes its own treatment of all activities (normally in varying workloads): requirements, analysis, architecture, design, implementation and testing activities. Each activity produces a set of artifacts, and the outcome of each iteration is a tested, integrated and executable system.

In the iterative and incremental development model, only partial testing is possible at the completion of each increment. The full functionality and the ability to test all requirements cannot be completed until all increments are complete by which time the system is ready for production.

It is important to note that the executable of each iteration is not some experimental throw-away prototype but a production subset of the final system. The executable of each iteration is made available to stakeholders who are expected to provide feedback. This creates an opportunity for adaptation. As a result, the system is successively enlarged and refined though multiple iterations.

Which criteria (if any) can determine the choice of requirements for a particular iteration? We can identify two possible criteria (even though these are neither explicitly addressed nor in any way enforced by the model):

1. Based on criticality of requirements (a combination of risk and value). This implies that high-risk/high-value requirements constitute critical requirements and are implemented during initial iterations. Low-risk/low-value requirements are left for last.

2. Based on stakehoders' opinions. This implies that stakeholders determine which requirements should be implemented initially and which should be left for last.

3.6 Rapid development model

The rapid application development (RAD) model is a variation of the iterative and incremental development model that emphasizes very short development cycles. The model assumes that requirements are well understood.

3.7 Agile development model

Development methods tend to become "heavy-weight" when certain characteristics are present, such as heavy regulations, delay for the production of an executable, lack or low degree of adaptation, and infrequent stakeholder feedback. As an alternative to this, "light-weight" (or agile) methods have been proposed.

Agile methods are a variation of the iterative and incremental development model, advocating short iterations. In agile development, tests are written before the functionality is coded. Further, an agile team will contain a customer representative (appointed by stakeholders) to participate during iterations in order to answer questions. At the end of each iteration, stakeholders and the customer representative review progress. Several instances of the agile model exist, perhaps the most popular one being *extreme programming* (XP). Extreme programming advocates programming in pairs, extensive code review, unit testing of all code, avoiding programming of features until they are actually needed, a flat management structure, simplicity and clarity in code, and frequent communication with stakeholders.

3.8 Cleanroom software engineering techniques

Borrowing its name from the semiconductor manufacturing industry, "cleanroom" software engineering focuses on the prevention of defects (as opposed to defect removal). To achieve this, one aims to write code correctly the first time rather than identifying and remove bugs once the code is written. Cleanroom

development adopts the iterative and incremental development model deploying formal methods and quality control and it follows a top-down design with stepwise refinement, where the entire program is viewed as a black-box. Functional decomposition produces formal descriptions of modules as black-boxes. These descriptions are normally mathematical functions. Once code is written for a module, it is verified against its intended behavior. The cleanroom model explicitly addresses risks by deploying formal methods to eliminate the possible introduction of errors.

3.9 Code-and-fix

As its name suggests, code-and-fix involves writing code and correct possible errors. Strictly speaking this is not a development process model, but rather an infomal description on how software can be written. The model does provide feedback but only in an informal manner.

On one hand this model has no overhead as it does not entail any activities other than implementation. This can perhaps be advantageous for very small projects. However, for any project of substantial size, the model is not viable as it does not include solid engineering principles (such as planning, analysis, design, quality control).

3.10 A framework for software development: The Unified Software Development Process (UP)

Strictly speaking, the Unified Software Development Process (or Unified Process, or just UP) is not a development model, but rather a framework for software development, based on the iterative and incremental development model, where iterations are grouped into the following four *phases*:

1. Inception. The goals of this phase include establishing a justification (business case) and the scope for the project, outlining the use cases and key requirements, outlining candidate architectures, define risks, and prepare a preliminary schedule and cost estimate. The inception phase should provide a justification on the feasibility of the project.

2. Elaboration. Implements the core architecture through the implementation of critical use-cases. Critical use cases contain high-risk and high-value requirements and the UP addresses them during the Elaboration phase. The motivation is to drive down the risk of the entire development so that if the project is going to fail it will do so early, not late.

3. Construction. Implements secondary (non-critical) use cases. Testing during Construction is referred to as *alpha testing*. Construction is complete when the system is ready for operational deployment and all supporting artifacts are complete (user guides, etc.)

4. Transition. Pleases the system into production use. Testing during Transition is referred to as *beta testing*.

With the possible exception of the inception phase, all other phases contain possibly several iterations. Even though an iteration includes work in most (if not all) disciplines, the relative effort and emphasis change over time. Early iterations tend to apply greater emphasis to requirements and design, and later ones tend to put more emphasis on implementation. A picture that illustrates the four phases (broken down into several iterations) and the activities involved in each iteration together with an indicative workload in each activity is shown in Figure 3.3.

Two notions are vital to the whole concept of UP: Timeboxing and milestones.

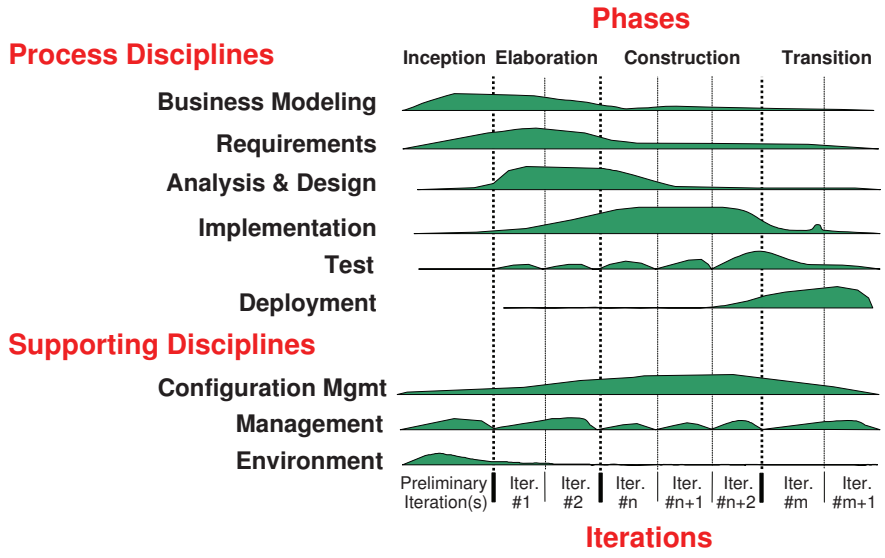

Figure 3.3: The Unified Software Development Process (UP).

Timeboxing

The notion of *timeboxing* is central to the UP. The UP recommends short and fixed ("timeboxed") iteration lengths to allow for rapid feedback and adaptation. The idea is that long iterations increase project risk. Moreover, if meeting a deadline seems to be difficult, UP recommends to remove some tasks (or requirements) from the current iteration and include them in a future iteration.

The main motivation behind timeboxing is that the development team is forced to focus and produce a tested executable partial system on-time. The team must prioritize requirements and tasks based on business and technical value. Furthermore, complete mini projects can build confidence both for the team as well as for the stakeholders.

Milestones

Each iteration concludes with a well-defined milestone, a point at which a review provides an assessment of how well the objectives of the iteration have been met, and if not, how the project can be brought back to track. The end

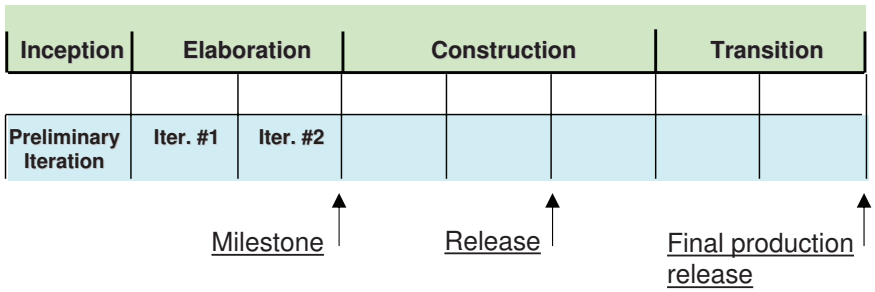

Figure 3.4: Milestones in the Unified process.

of each iteration produces a minor release, which is a stable executable subset of the final product (see Figure 3.4). We can also distinguish between minor and major milestones. The latter would signify the end of a phase.

It is also important to note that occasionally an iteration may not necessarily add new requirements, but it may modify existing requirements (or even drop existing requirements). Naturally, feedback from stakeholders in the form of requests (for additions, modifications or deletions) cannot be unconditionally implemented but it is taken into consideration by the development team which now has to decide on the feasibility of the request.

The concept of iterative development can be applicable to maintenance as well. Any form of maintenance can be addressed iteratively.

Advantages of the UP

We can identify three main advantages of the Unified Process:

1. Risks are identified early.
2. Handle evolving requirements (through feedback and adaptation).
3. Attain early understanding of the system.

How to get it wrong in UP

Consider the following statement: "During the Inception phase all requirements are defined and frozen. The purpose of the Elaboration phase is to

produce a design of the system. Construction, as its name suggests, builds the system." What is wrong with this statement? The answer is that the statement misinterprets the Unified Process and superimposes it over the linear model. Recall that the Inception phase defines the scope of the project, the Elaboration phase builds the core architecture of the system and the Construction phase builds the remaining features. Iterations in all phases contain work in most (if not all) disciplines (activities) in varying degrees of workload.

3.11 Examples

Example 3.1. In the literature it is claimed that for medium- to large-scale systems, a linear software development process "is not realistic." Briefly explain what it meant by "not realistic" and describe two problems usually associated with the linear ("waterfall") process model.

Solution:

As requirements tend to change, it is usually not feasible to fully complete and "freeze" the requirements specification, then fully complete the design, build and then test the software. Two associated problems with the linear model are:

- Risks are not addressed early but can be encountered late in the development.
- Lack of flexibility of changing requirements.

Example 3.2. Briefly describe what is meant by "timeboxing" and what is the motivation behind short iterations.

Solution:

Timeboxing is a term that refers to the fixed length iterations. Long iterations increase project risk. On the other hand, short length iterations allow the rapid feedback and adaptation.

Example 3.3. Consider and comment on the following quote: "During the Inception phase all requirements are defined. The purpose of the Elaboration phase is to produce a design of the system. Construction, as its name suggests, builds the system."

Solution:

The above quote misinterprets the Unified Process. The Inception phase defines the scope of the project, the Elaboration phase builds the core architecture of the system and the Construction phase builds the remaining features. Iterations in all phases contain work in most disciplines.

Example 3.4. Provide brief definitions for the two broad categories of requirements. In the context of the Unified Software Development Process (UP), briefly describe when requirements are captured, and which artifacts are used to capture them.

Solution:

Functional requirements deal with the observable behavior of the system. In the UP functional requirements can be captured in use case scenarios as they describe the interactions between end-users (actors) and the system. Nonfunctional requirements describe constraints and qualities of the system. NFRs are normally captured in the Use Case Model and in the Supplementary Specification. Both artifacts start during the preliminary iteration (of the Inception phase) and can be refined in subsequent iterations.

Example 3.5. Briefly explain what is meant by "critical use cases" within the context of the Unified Software Development Process (UP) and what is the motivation behind this notion.

Solution:

Critical use cases contain high-risk and high-value requirements and the UP addresses them during the Elaboration phase. The motivation is to drive down the risk of the entire development so that if the project is going to fail it will do so early, not late.

Example 3.6. Consider the Unified Software Development Process (UP). Briefly explain what is meant by the term "artifacts" and describe the two dimensions over which we can view artifacts. What is the relation between artifacts and deliverables.

Solution:

The term "artifact" is used to describe the documentation of an activity. We have two dimensions of artifacts. Horizontal dimension: the set of artifacts (across all disciplines) over a single iteration, and Vertical dimension: the set of artifacts of one discipline (activity) throughout the entire development (throughout all phases). A deliverable is a document that describes sets of artifacts by the end of an iteration.

Example 3.7. List the phases of the linear model. Where do these phases get their names from? How does this model relate to the phases of the Unified Software Development Process (UP)?

Solution:

The names of the phases of the linear model are: Requirements, Analysis, Design, Implementation, and Testing. The names refer to the activities entailed in each phase. On the contrary, the phases of the UP contain all of the activities (within iterations) in various workloads.

Example 3.8. Briefly describe three advantages of an iterative process.

Solution:

Three advantages of an iterative process can be the following:

- Risks are identified early.
- Handling evolving requirements (through feedback and adaptation).
- Attaining early understanding of the system.

Example 3.9. Briefly describe the primary objectives of the Elaboration phase of the Unified Software Development process. In your answer take into consideration the outcome of the previous phase.

Solution:

The primary objectives of the Elaboration phase are:

- To produce an executable core of the system
- To address the critical use cases identified in the Inception phase.

Example 3.10. Consider the case where as a project manager you need to make a decision on the type of process your team should follow for the development of a large-scale distributed and concurrent software system (to handle marketing and sales) which is expected to aid a company to become competitive in a dynamic environment. You may assume that stakeholders are confident about all their goals well in advance. Briefly describe what your decision will be and what factors have determined your decision.

Solution:

Despite the fact that all requirements are currently known and you may be tempted to consider them as frozen, there is still no guarantee that they will remain frozen throughout development. This is due to the dynamic nature of the operating environment. We will follow an iterative development process and factors which determine this decision are as follows:

- Complexity of the system.
- Risks involved.
- (Potentially) changing requirements.

Example 3.11. Briefly describe two common factors between the Unified Software Development Process (UP) and Extreme Programming (XP) as well as two factors which are unique to each one.

Solution:

Common factors are:

- Both of them has an iterative and incremental development process.
- They both are getting feedbacks from developers and stakeholders.

Differences are:

In UP:

- We have time-boxed iterations.
- Critical use cases are handled first (in Elaboration phase).

In XP:

- Stakeholders define stories to be addressed.
- Encourages test-first development.

Example 3.12. Various iterative development processes like the Unified Software Development process (UP) recommend that critical requirements should be addressed in early iterations. One may argue that this does not make much sense since all requirements must be implemented anyhow and all requirements are important. Respond to this argument by briefly describing what is meant by the term "critical requirements" and provide a rationale for their early implementation.

Solution:

The term "critical" (in the context of software requirements) is not a synonym to "important." In this context, "critical" refers to those requirements which provide high-value and pose high risk. One reason they have to be implemented in early iterations is to not delay risk and is to drive down the risk of the entire development so that if the project is going to fail it will do so early, not late.

Example 3.13. Briefly describe what is meant by "iteration", by including any and all (engineering) activities. What management activities would you include in an iteration, and how do these relate to the engineering activities? What is the end result of an iteration? What are the benefits and obligations of stakeholders at the end of an iteration?

Solution:

An iteration is a mini-project: it includes its own set of requirements elicitation and analysis, (object-oriented) analysis and design, mapping from design to implementation, testing (and integration).

Further, an iteration would include activity planning and risk assessment in the beginning and risk control throughout all the activities. The end result is a deliverable which consists of a set of artifacts (such as design model, source code etc.) and a tested executable.

Stakeholders would be interested as they would want to attain an understanding over the project, and assess the executable against their requirements. These constitute their benefits. Their obligations are implied here through their engagement in the provision of feedback.

Example 3.14. Consider use cases:

1. In the context of iterative development, what makes a use case *critical*?

2. In the context of the unified software development process (UP), during which phase are critical requirements addressed and what is the rationale behind this decision?

3. Who are the parties involved in a use case?

4. How are each party's benefits guaranteed and how are its obligations specified?

5. Regardless of the development process model followed, how can use cases be deployed to participate in requirements verification?

Solution:

1. Critical use cases contain high-risk and high-value requirements.

2. In the UP critical use cases are addressed during the Elaboration phase. The motivation is to drive down the risk of the entire development so that if the project is going to fail it will do so early, not late.

3. The parties in a use case are actor(s) and the system seen as a black box.

4. The actor must guarantee preconditions; benefits from postconditions, and the system must guarantee postconditions and benefits from preconditions.

5. The executable can be demonstrated against the use case scenarios in order to answer "have we built the product right."

Example 3.15. In the literature we frequently read of *iterative development*.

1. Provide a brief description of what is meant by "iteration" and briefly describe how one can deploy an iterative model to achieve incremental development.

2. What is the outcome of an iteration (in terms of a deliverable) and how does this outcome relate to the overall system under development.

3. What are the benefits and obligations of stakeholders at the end of an iteration?

Solution:

1. An iteration is a short fixed-length mini-project: it includes its own set of requirements elicitation and analysis, (object-oriented) analysis and design, mapping from design to implementation, testing (and integration). One can adopt iterative development by organizing a series of (normally

short and fixed-length) iterations, each one addressing new requirements. As a result, the system can grows incrementally over time, iteration by iteration.

2. The outcome of an iteration is a deliverable which consists of a set of artifacts (such as design model, source code etc.) together with a tested, integrated and executable system which is a subset of the overall system under development.

3. Benefits of stakeholders include an understanding over the project, and an assessment of the executable against their requirements. Their obligations are implied here through their engagement in the provision of feedback.

3.12 References

1. Kruchten, P., The Rational Unified Process: An Introduction, 3rd edition, Addison-Wesley Professional, 2003.

2. Royce, W. Software Project Management: A Unified Framework, 1st edition, Addison-Wesley Professional, 1998.