

Chapter 2

Analysis

A hypothetical conversation:

Customer: I am having trouble with text formatting.

Analyst: What is the problem?

Customer: Oh well, I have this text that I would like to format in this nice format and I don't know how to do it.

Analyst (sympathetic): That is a problem. What would you like me to do?

Customer: Solve the problem.

Analyst: Please be more specific.

Customer: I am being specific. I will have some text and formatting directions and I would like the text to be formatted using the directions.

The analyst comes back next day with a secretary.

Customer (surprised): That was quick. Did you solve my problem?

Analyst(excited): Yes, I did.

Customer: Where is the solution?

Analyst(pointing to the secretary): Here. She will format your text for you.

Customer (worried): but, I don't want to pay her every month. I want to do it myself.

Analyst(angry): You didn't say that yesterday. I asked you to be specific.

Customer(angry): but you didn't ask whether I want to pay monthly wages or not. I thought it was implicit that I don't.

In this chapter, we will learn about analysis aka requirements analysis. We will learn about the importance of this phase in software development process. We

will also learn about the possible errors in this phase, and finally we will briefly review two key type of software development processes.

2.1 Do the right thing

Decades of experience with failed projects, budget-overrun projects, and livid stakeholders has taught software practitioners some hard lessons. One of the most important lessons that we have learned is that we have to build the right system. What is the right system? The right system is the one that solves the right problem and satisfies the expectations of most stakeholders in the software project. The requirement analysis or *analysis* is a component of the software development life cycle in which we study the problem that the system is going to solve, the environment in which it is going to operate, the people or other software systems that it is going to interact, expectations of various stakeholders, budget, etc. These factors constitute the domain of the software system.

Analysis is *the* crucial activity and an essential component of any software development life cycle. It is critical to the success of any software project. The output of this process is the set of requirements. It might not be necessary to write down the requirements for a minuscule software system. For example, may be you can skip writing the requirements of a software system that computes the sum of two positive integers less than 10. For most other systems, however, it is often a good idea to write them down in the form of a document, called requirements document. It is also essential that the analysts and the stakeholders agree on a common set of terminology to be used in the requirements document.

One may ask why we should worry so much about the analysis and the requirements. If we find an error in the requirement later, we can always go and fix it. The problem is that fixing an error in the requirement becomes costlier later on in the software development life cycle. The other components of a software development life cycle are typically design, implementation, verification, and maintenance. Each of these activities produces some artifact such as design of the system, implementation of the system, verification results, etc. These artifacts are directly or indirectly based on the requirements. Therefore, the correctness of these artifacts is dependent upon the correctness of requirements. If an error in requirement is discovered, one has to not only correct the requirement but also modify these other software artifacts that use the requirement as the basis. The Figure 2.1 shows that the cost to fix an error detected in the later phases increases almost exponentially. If a significant error is detected in the deployment phase, such as incorrect security requirements for the automatic teller machine (ATM), entire software projects are often scrapped.

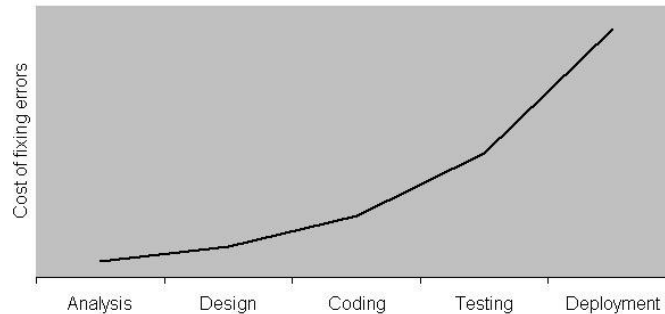


Figure 2.1: An error detected in later phases is more expensive to fix

2.2 Analysis Errors

There are two common types of errors in requirements. Either some requirement in the domain of the software system is not analyzed and documented. For example, the analyst may miss the requirement that the text formatting has to be done without paying monthly wages. Such errors are often referred as missed requirements. The second type of error is when there is an inconsistency between the documented requirement and the actual requirement in the domain. Such errors are called misrepresented requirements.

The errors in the requirements such as misunderstood requirement or missed requirement are especially difficult to deal with in software development life cycle where validation of the requirements does not happen until very late in the software process. By validation we mean *examining whether we have built the right thing*. Verification on the other hand is the process where we examine whether *we have built the thing right*. We will talk more about it in the next chapter. To examine a subject, we need a point of reference. For example, your homework answers will be evaluated using the homework key as a point of reference. For validation, the subject is the software system, and the point of reference is the requirement for that software.

One such software development life cycle follows the waterfall model (See Figure 2.2). The typical phases of this life cycle are analysis, design, implementation, verification, and maintenance. In a software project that is following the waterfall model, a phase of the life cycle does not start until the previous phase is finished. For example, design of the software system will not start until the requirements of the system are well understood. The implementation does not start until design is done, and verification and validation does not start until implementation is over. As a result, we do not validate the requirement until the design is made, the code is written, etc. If the validation phase reports an error, the effect of the error ripples

through the design and the code making it very expensive to fix it.

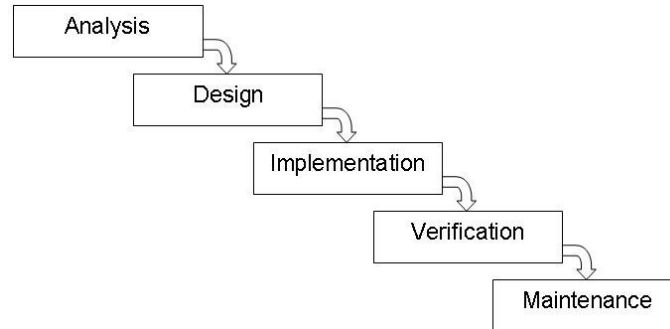


Figure 2.2: The original waterfall model - Like a waterfall, progress in software process flows from the top to the bottom

An alternative software development life cycle follows the iterative model (See Figure 2.3). In this model the software system is built incrementally in several iteration, hence the name. A small subset of requirement is handled in every iteration. We analyze this subset, design the relevant parts of the system, implement it, verify it, and validate it. The subset is intentionally kept small so that there is little time spent between analysis and validation. The benefit is that we do not have to wait for the entire system to be built to validate a given requirement; therefore the cost of fixing errors in a requirement is also lower. Like most things in the world, requirements for software systems also change. In fact, they are rarely stable. The iterative model is better in dealing with changing requirements.

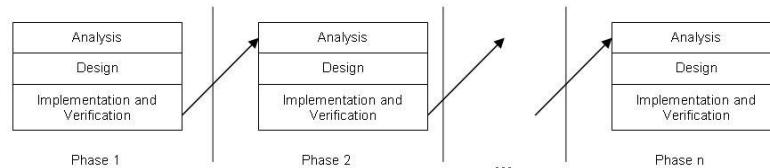


Figure 2.3: Iterative Software Development - Building software in increments

Sometimes we absolutely cannot afford to have errors in the requirements of a software system. In these cases, the requirements are written using a formal notation so that they can be mathematically verified either manually or automatically using programs that can verify mathematical theorems. Such programs are called automatic theorem provers. In case of these systems, the software process may not continue before the requirements are proven to be correct. An example of such system is an industrial aircraft collision avoidance system (TACAS II). Considering the critical nature of this system and the severe consequences on its failure, the Federal Aviation Administration imposed stringent guidelines on building this system.

The requirements of this system were verified using Requirements State Machine Language (RSML) [23] [14].

2.3 Exercises

1. Why is analysis important? How can it save money or make programming more efficient?
2. Let us assume that you are analyzing the requirements of the software component for an artificial pacemaker. A pacemaker is a cellular structure that by contracting and expanding causes heart beating. Artificial pacemakers are implanted to substitute a faulty pacemaker.
 - (a) Will you write an informal or a formal requirements document for this system?
 - (b) What are the factors influencing your decision?
3. Let us assume that you have the authority to decide which software process model will be used for a project. The two options are waterfall model and iterative model discussed in this chapter.
 - (a) Which software process model will you use to develop a media player? Why? (1-2 line)
 - (b) Which software process model will you use to develop the pacemaker? Why? (1-2 line)
4. Please pick an open-source project of your choice.
 - (a) Briefly describe the software process model that they use (2-3 lines).
 - (b) Is it a variation of the waterfall model or the iterative model?
 - (c) What are the key differences? (1-2 lines)
5. Scrum is a very popular software process model. Please read about Scrum on the World Wide Web and briefly describe the key features of Scrum. (4-5 lines)