Chapter

5

EVOLUTIONARY REQUIREMENTS

Ours is a world where people don't know what they want and are willing to go through hell to get it.

—Don Marquis

Objectives

- Motivate doing evolutionary requirements.
- Define the FURPS+ model.
- Define the UP requirements artifacts.

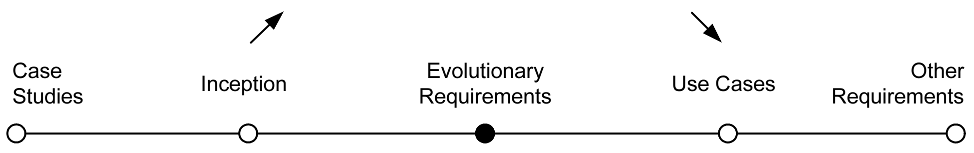
Introduction

other UP practices p. 33

This chapter briefly introduces iterative and evolutionary requirements, and describes specific UP requirement artifacts, to provide context for the coming requirements-oriented chapters.

In also explores some evidence illustrating the futility and unskillfulness of waterfall-oriented requirements analysis approaches, in which there is an attempt to define so-called "complete" specifications before starting development.

What's Next?



Having introduced inception, this chapter introduces requirements and their evolutionary refinement. The next covers use cases, the prime requirements practice in the UP and many modern methods.

5.1 Definition: Requirements

Requirements are capabilities and conditions to which the system—and more broadly, the project—must conform [JBR99].

The UP promotes a set of best practices, one of which is *manage requirements*. This does not mean the waterfall attitude of attempting to fully define and stabilize the requirements in the first phase of a project before programming, but rather—in the context of inevitably changing and unclear stakeholder's wishes, this means—"a systematic approach to finding, documenting, organizing, and tracking the *changing* requirements of a system" [RUP].

In short, doing it iteratively and skillfully, and not being sloppy.

A prime challenge of requirements analysis is to find, communicate, and remember (that usually means write down) what is really needed, in a form that clearly speaks to the client and development team members.

5.2 Evolutionary vs. Waterfall Requirements

Notice the word *changing* in the definition of what it means to manage requirements. The UP embraces change in requirements as a fundamental driver on projects. That's incredibly important and at the heart of waterfall versus iterative and evolutionary thinking.

In the UP and other evolutionary methods (Scrum, XP, FDD, and so on), we start production-quality programming and testing long before most of the requirements have been analyzed or specified—perhaps when only 10% or 20% of the most architecturally significant, risky, and high-business-value requirements have been specified.

What are the process details? How to do partial, evolutionary requirements analysis combined with early design and programming, in iterations? See "How to do Iterative and Evolutionary Analysis and Design?" on page 25. It provides a brief description and a picture to help explain the process. See "Process: How to Work With Use Cases in Iterative Methods?" on page 95. It has more detailed discussion.

Caution!

If you find yourself on a so-called UP or iterative project that attempts to specify most or all of the requirements (use cases, and so forth) before starting to program and test, there is a profound misunderstanding—it is not a healthy UP or iterative project.

In the 1960s and 1970s (when I started work as a developer) there was still a common speculative belief in the efficacy of full, early requirements analysis for software projects (i.e., the waterfall). Starting in the 1980s, there arose evidence this was unskillful and led to many failures; the old belief was rooted in the wrong paradigm of viewing a software project as similar to predictable mass manufacturing, with low change rates. But software is in the domain of new product development, with high change ranges and high degrees of novelty and discovery.

change research p. 24

Recall the key statistic that, on average, 25% of the requirements change on software projects. Any method that therefore attempts to freeze or fully define requirements at the start is fundamentally flawed, based on a false assumption, and fighting or denying the inevitable change.

Underlining this point, for example, was a study of failure factors on 1,027 software projects [Thomas01]. The findings? Attempting waterfall practices (including detailed up-front requirements) was the single largest contributing factor for failure, being cited in 82% of the projects as the number one problem. To quote the conclusion:

> *… the approach of full requirements definition followed by a long gap before those requirements are delivered is no longer appropriate.*
>
> *The high ranking of changing business requirements suggests that any assumption that there will be little significant change to requirements once they have been documented is fundamentally flawed, and that spending significant time and effort defining them to the maximum level is inappropriate.*

Another relevant research result answers this question: When waterfall requirements analysis is attempted, how many of the prematurely early specified features are actually useful in the final software product? In a study [Johnson02] of thousands of projects, the results are quite revealing—45% of such features were never used, and an additional 19% were “rarely” used. See Figure 5.1. Almost 65% of the waterfall-specified features were of little or no value!

These results don’t imply that the solution is to start pounding away at the code near Day One of the project, and forget about requirements analysis or recording requirements. There is a middle way: iterative and evolutionary requirements analysis combined with early timeboxed iterative development and frequent stakeholder participation, evaluation, and feedback on partial results.

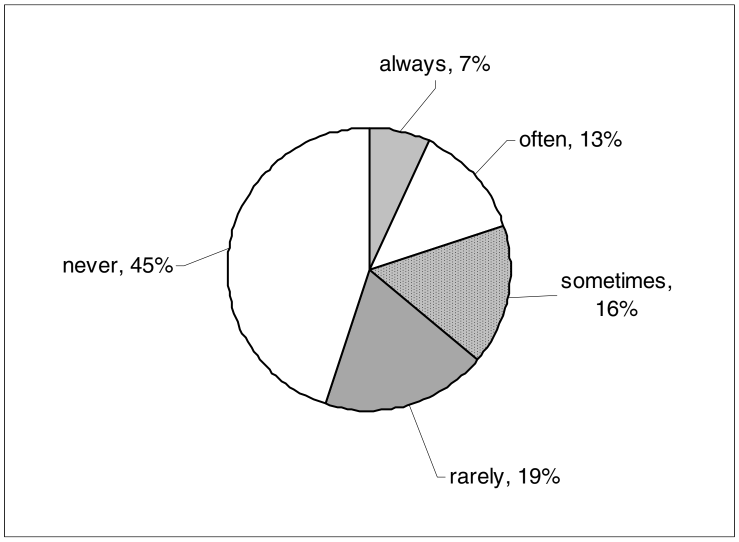


Figure 5.1 Actual use of waterfall-specified features.

5.3 What are Skillful Means to Find Requirements?

To review the UP best practice *manage requirements*:

> *…a systematic approach to finding, documenting, organizing, and tracking the changing requirements of a system. [RUP]*

Besides *changing*, the word *finding* is important; that is, the UP encourages skillful elicitation via techniques such as writing use cases with customers, requirements workshops that include both developers and customers, focus groups with proxy customers, and a demo of the results of each iteration to the customers, to solicit feedback.

The UP welcomes any requirements elicitation method that can add value and that increases user participation. Even the simple XP "story card" practice is acceptable on a UP project, if it can be made to work effectively (it requires the presence of a full-time customer-expert in the project room—an excellent practice but often difficult to achieve).

5.4 What are the Types and Categories of Requirements?

In the UP, requirements are categorized according to the FURPS+ model

[Grady92], a useful mnemonic with the following meaning:[1]

- **Functional**—features, capabilities, security.
- **Usability**—human factors, help, documentation.
- **Reliability**—frequency of failure, recoverability, predictability.
- **Performance**—response times, throughput, accuracy, availability, resource usage.
- **Supportability**—adaptability, maintainability, internationalization, configurability.

The "+" in FURPS+ indicates ancillary and sub-factors, such as:

- **Implementation**—resource limitations, languages and tools, hardware, ...
- **Interface**—constraints imposed by interfacing with external systems.
- **Operations**—system management in its operational setting.
- **Packaging**—for example, a physical box.
- **Legal**—licensing and so forth.

It is helpful to use FURPS+ categories (or some categorization scheme) as a checklist for requirements coverage, to reduce the risk of not considering some important facet of the system.

Some of these requirements are collectively called the **quality attributes**, **quality requirements**, or the "-ilities" of a system. These include usability, reliability, performance, and supportability. In common usage, requirements are categorized as **functional** (behavioral) or **non-functional** (everything else); some dislike this broad generalization [BCK98], but it is very widely used.

architectural analysis p. 541

As we shall see when exploring architectural analysis, the quality attributes have a strong influence on the architecture of a system. For example, a high-performance, high-reliability requirement will influence the choice of software and hardware components, and their configuration.

1. There are several systems of requirements categorization and quality attributes published in books and by standards organizations, such as ISO 9126 (which is similar to the FURPS+ list), and several from the Software Engineering Institute (SEI); any can be used on a UP project.

5.5 How are Requirements Organized in UP Artifacts?

The UP offers several requirements artifacts. As with all UP artifacts, they are optional. Key ones include:

- **Use-Case Model**—A set of typical scenarios of using a system. There are primarily for functional (behavioral) requirements.
- **Supplementary Specification**—Basically, everything not in the use cases. This artifact is primarily for all non-functional requirements, such as performance or licensing. It is also the place to record functional **features** not expressed (or expressible) as use cases; for example, a report generation.
- **Glossary**—In its simplest form, the Glossary defines noteworthy terms. It also encompasses the concept of the **data dictionary**, which records requirements related to data, such as validation rules, acceptable values, and so forth. The Glossary can detail any element: an attribute of an object, a parameter of an operation call, a report layout, and so forth.
- **Vision**—Summarizes high-level requirements that are elaborated in the Use-Case Model and Supplementary Specification, and summarizes the business case for the project. A short executive overview document for quickly learning the project's big ideas.
- **Business Rules**—Business rules (also called Domain Rules) typically describe requirements or policies that transcend one software project—they are required in the domain or business, and many applications may need to conform to them. An excellent example is government tax laws. Domain rule details *may* be recorded in the Supplementary Specification, but because they are usually more enduring and applicable than for one software project, placing them in a central Business Rules artifact (shared by all analysts of the company) makes for better reuse of the analysis effort.

What is the Correct Format for these Artifacts?

In the UP, all artifacts are information abstractions; they could be stored on Web pages (such as in a Wiki Web), wall posters, or any variation imaginable. The online RUP documentation product contains templates for the artifacts, but these are an optional aid, and can be ignored.

5.6 Does the Book Contain Examples of These Artifacts?

Yes! This book is primarily an introduction to OOA/D in an iterative process rather than requirements analysis, but exploring OOA/D without some example or context of the requirements gives an incomplete picture—it ignores the influence of requirements on OOA/D. And it's simply useful to have a larger example of key UP requirements artifacts. Where to find the examples:

Requirement Artifact	Where?	Comment
Use-Case Model	Introduction p. 61 Intermediate p. 493	Use cases are common in the UP and an input to OOA/D, and thus described in detail in an early chapter.
Supplementary Specification, Glossary, Vision, Business Rules	Case study examples p. 101	These are provided for consistency, but can be skipped—not an OOA/D topic.

5.7 Recommended Resources

References related to requirements with use cases are covered in a subsequent chapter. Use-case-oriented requirements texts, such as *Writing Effective Use Cases* [Cockburn01] are the recommended starting point in requirements study, rather than more general (and usually, traditional) requirements texts.

There is a broad effort to discuss requirements—and a wide variety of software engineering topics—under the umbrella of the Software Engineering Body of Knowledge (**SWEBOK**), available at www.swebok.org.

The SEI (www.sei.cmu.edu) has several proposals related to quality requirements. The ISO 9126, IEEE Std 830, and IEEE Std 1061 are standards related to requirements and quality attributes, and available on the Web at various sites.

A caution regarding general requirements books, even those that claim to cover use cases, iterative development, or indeed even requirements in the UP:

> Most are written with a waterfall bias of significant or "thorough" up-front requirements definition before moving on to design and implementation. Those books that also mention iterative development may do so superficially, perhaps with "iterative" material recently added to appeal to modern trends. They may have good requirements elicitation and organization tips, but don't represent an accurate view of iterative and evolutionary analysis.

Any variant of advice that suggests "try to define most of the requirements, and then move forward to design and implementation" is inconsistent with iterative evolutionary development and the UP.