# Contents

Contents 1

1 Introduction 2

2 UP 2

3 Inception 3

3.1 Requirements 3

3.1.1 Types of requirements 3

3.2 Glossary 4

3.3 Use Cases 4

3.3.1 Use Cases and Requirements 4

3.3.2 Black Box Use Cases 4

3.3.3 Types of use cases 4

3.3.4 Use Case Parts 5

3.3.5 Fully Dressed 5

3.3.6 Goals and Scope 6

3.3.7 Use Case Diagrams 6
1 Introduction

The goal of the material presented here is to learn skills in Object Oriented Analysis and Design (OOA/D). Into the fold, patterns and UML notation will be used.

The first question that needs to be asked is “What is OOA?”. Analysis is the investigation of the problem. It’s purpose is to uncover requirements. Requirements are what drives the eventual design.

OOD is a solution that fulfills the requirements. It is not the implementation. Ultimately, designs become implemented.

2 UP

The unified process (UP) is a basic set of guidelines for the creation of software. A basic tenent of UP is the idea of iterations.

While there have been many different software engineer methodologies over the years, the one aspect that is common to many of them is the idea of iterations.

Instead of trying to “get it right” in one shot. Many cycles are used to refine and expand the analysis, design, and implementation. UP takes the view that there are four basic parts in each iteration.

1. Requirements
2. Design
3. Implementation, Testing, Integration, More Design
4. Final Integration, System Test

After the end of step 4, a fully function system is the result. The entire process begins again. One should note that you may not have production ready system read for many iterations. The upside is that you have a somewhat functional system early on that can be used to better understand requirements and get feedback. This is what drives later iterations.

The length of an iteration is important as well. The idea is to keep it short but not too short. In the range of two to six weeks is good. For larger teams the duration may be longer due to more complex communications. For this course, we will be doing one iteration.

In addition to iterations, UP has four phases.

1. Inception
2. Elaboration
3. Construction
4. Transition

Within each phases, iterations occur. What differentiates these phases, is what is stressed during each iteration. The figure on page 22 of the book shows this idea.
3 Inception

The first phase in UP is the inception phase. This phase can be very short and is generally an exploratory phase. In this phases some basic bits of information are gathered to see whether the project is worth continuing. These bits may include: costs, need, competition, feasibility. A lot of this phase focuses more on the business side than on the software side.

3.1 Requirements

Requirements are capabilities and conditions that must be adhered to by the final system. The hardest part about requirements is the actual discovery and recording of them.

It is practically a rule of thumb that requirements will change as a project progresses. While it is very helpful for major requirements to be immutable during the course of a project, this is not always the case. As the end users get to try versions of the software and as the design team understands the problem in greater detail, requirements appear, disappear, or change. Therefore the ability to track the requirements is an important one.

Finding the requirements can be a difficult task. There are a variety of techniques for the elicitation of requirements. The two most common are use cases and workshops. Workshops are the more direct way of getting requirements. They are simply sitting down and listing them. Use cases are a more subtle means.

3.1.1 Types of requirements

Requirements fall into one of six loose categories, FURPS+. The FURPS+ categories are:

- Functional - features, capabilities, security
- Usability - Human factors, help, documentation
- Reliability - failure frequency, recoverability, predictability
- Performance - response times, throughput, accuracy, availability, etc
- Supportability - maintence, adaptability, configuration, internationalization
- + - Ancillary areas such as: languages, tools, external interfaces, system management, packaging, licensing

You can use these categories as a kind of checklist to make sure you have at least thought of different facets of the system. You can also categorize requirements by whether or not they are functional requirements.

Function requirements are requirements that are behavioral in nature. Generally these requirements need to be covered by use cases.
3.2 Glossary

A related document is that of the glossary or data dictionary. Often times non-domain experts work on projects. Words or phrases that may be have clear definitions to domain experts may be unknown to novices. The glossary is a place for definitions that may not be well known to all parties. This helps to make sure everyone understands what they are reading.

Another purpose is to provide clear definitions for ambiguous words and phrases. Often times the same or similar words are used in different disciplines. If the use or definition are differently this can lead to break-downs in understanding. The glossary acts as a referee for these situations.

3.3 Use Cases

The use case is a tool of communication between developers and users. In simple terms they are stories of users using the software. While there are many different formats they all serve the same purpose, to capture how a user might use the software.

There are two basic parts to a use case. Actors and scenarios. Actors are they users of system. They can be real persons, other software, or even time. Scenarios are the actual story. A scenario tells a story of how the actor interacts with the system and how the system responds. Scenarios can vary from being rather brief (maybe a paragraph) to extremely detailed. Generally, as you pass through more iterations use cases become more elaborate.

3.3.1 Use Cases and Requirements

Use cases are related to requirements in a number of fashion. First, they can be used to discover requirements. If a use case mentions an error, then the ability to handle certain errors gracefully becomes part of the requirements. Second, they can capture a requirement. If a requirement exists for response times, then atleast one use case should mention it. And last, the use case and be a requirement. This is the idea that a system must be able to accomplish a specific task in a specified way.

3.3.2 Black Box Use Cases

In almost all cases use cases are written from the standpoint of a black box. This means that the use case should not include any software development related jargon. If a system needs to record a bit of information, it should merely say that it records it. The manner in how or where it records it should not be mentioned. I.e. No SQL syntax or any other ilk.

3.3.3 Types of use cases

There are three main types of use cases: brief, casual, and fully dressed. Brief use cases are generally a paragraph long and cover only the main success of a scenario. Casual use cases are more detailed and generally includes both the main success scenario and other either error scenarios or alternate scenarios. The final type, fully dressed, is a very
formal type. It includes information such as pre and post-conditions, actors involved, enumerated steps with alternate steps for similar or error scenarios. Guarantees and non-functional requirements and issues might also be included. In all cases there are some common parts.

### 3.3.4 Use Case Parts

For each use case, no matter what they type, there are some common attributes.

- **ID** - each use case has a unique ID
- **Name** - each use cases contains a name
- **Actors** - at least some indication of who would use the use case
- **Scenario** - the actual story

At the very least, these parts must be present.

### 3.3.5 Fully Dressed

For the fully dressed use case, there are a variety of sections that should or could be present. The more common section are as follows, section appear in the order they would appear in a use case:

1. ID
2. Name
3. Primary Actor(s)
4. Stakeholders and Interests - These are people interested in seeing the scenario complete successfully.
5. Preconditions - what needs to be in place before the scenario can start
6. Postconditions - In what state is the system after a successful completion
7. Scenario - the main scenario, a numbered list of the events that take place during the use case
8. Alternate Scenarios - Events that might occur given certain conditions
9. Special Requirements - non-function requirements for the scenario to take place
10. Technology and Data Variations list - how something must be done
3.3.6 Goals and Scope

With use cases, there are varying levels of use cases. The book mentions EBPs or Elementary Business Processes which are processes that are accomplished by one person in one place at one time that benefit the business and leaves data in a consistent state.

Generally, a use case should be something done in anywhere from 5 minutes to an hour. The more important thing to keep in mind is that the purpose of a use case is to accomplish a goal for the actor. So a good way to identify use case is to define what the goals of the users in the system are. This leads use to a second problem, Identifying actors (users). This can sometimes be easy, or sometimes hard. To define that actors, you need to ask yourself, sometimes silly, questions about the system. Who is going to start/stop it? Who manages the users? Is time an actor? Does anyone monitor the system? Who maintains the system?

After identifying the actors, the next step is finding what the goals for each actor is. The best method is a two column system with one column for actors and the other for goals. Each actor has their own row.

3.3.7 Use Case Diagrams

This is a simple UML Diagram made up of three parts: Actors, Use cases, Relationships. Actors are generally stick figures, Use Cases are named ovals. If an actor participates in a use case a line is drawn between the two. Its that simple. Use case diagrams can get more elaborate but this isn’t really necessary.