Department of Computer Science

CS411 Requirements Analysis

@perrydBUCS

Recap of SDLC processes

- Every company uses some form of a Software Development Lifecycle (SDLC) process
- There are two general forms
  - Waterfall or linear
  - Iterative or spiral
- We use repeatable processes so that we can
  - Fine tune the process itself (known as CPI or Continuous Process Improvement
  - Leverage what we've learned on previous projects to plan new ones
  - Be assured of a level of quality in the product

important

Cargo cult Adult

Waterfall

4 steps

also 4 steps

- Waterfall processes are linear, with each phase being entered when the previous one is finished
- Risk is managed by rigorous planning in early phases
- Integration testing happens relatively late in the project
- Waterfall projects tend to have explicit start-stop dates for each task based on early estimates
- It's not a bad way to run a project as long as the risks are understood

Iterative processes

- "Iterative" just means that we do the same small number of phases over and over…often called a "spin"
- Risk is managed more at the spin level
- Testing (even integration testing) happens continuously during each spin
- A spin is a mini-project with a start and a finish, a specific objective, and a usable output
- Iterative projects are more difficult to plan and manage but make up for that in flexibility

Four phases of the SDLC

- Most SDLC projects, whether waterfall or spiral, go through four distinct phases
  - Define
  - Design
  - Develop
  - Deliver
- IBM's Rational process (and CS411) calls these
  - Inception
  - Elaboration
  - Construction
  - Transition

RUP Phase 1: Inception

Getting a hand over the project
1) what's the goal of the project?
2) what the hell are we doing?

- The big goal of the Inception phase is to get all stakeholders to agree on what is going to be accomplished
- To do this we must
  - Gather as many high-level requirements as possible so that we start to get a picture of the end goal
  - Get some sense of whether it is even possible
  - Figure out how big the thing is
  - Develop a rough idea of how long the project will take
  - Come up with at least one valid architecture (probably more)

Requirements gathering

- This is the most important part of any project!
- It can also be the most annoying
- You basically are playing the role of a four-year-old software engineer:
  - User: “We’d like for this screen to have a blue background.”
  - You: “Why?”
- At this stage we do not have any specific approach in mind
- Requirements are all about “doing”…what is it you want to do? What should the system do now?
- WHAT not *how*

Why? Gold for us

- This word is your most powerful tool
- Users, analysts, stakeholders, developers all want to tell you HOW
- HOW predisposes you toward a particular solution
- In the early part of the inception phase, your view should be technology agnostic
- Otherwise you will get sidetracked into implementation details way too early

1) What's the business problem.
2) How to solve the business problem

What is a requirement?

unit of work

- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.
- This is inevitable as requirements may serve a dual function
  - May be the basis for a bid for a contract - therefore must be open to interpretation;
  - May be the basis for the contract itself - therefore must be defined in detail;
  - Both these statements may be called requirements.
- Requirements can be viewed as *constraints*

Types of requirements

- **User** requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.
- **System** requirements
  - A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

User and system requirements

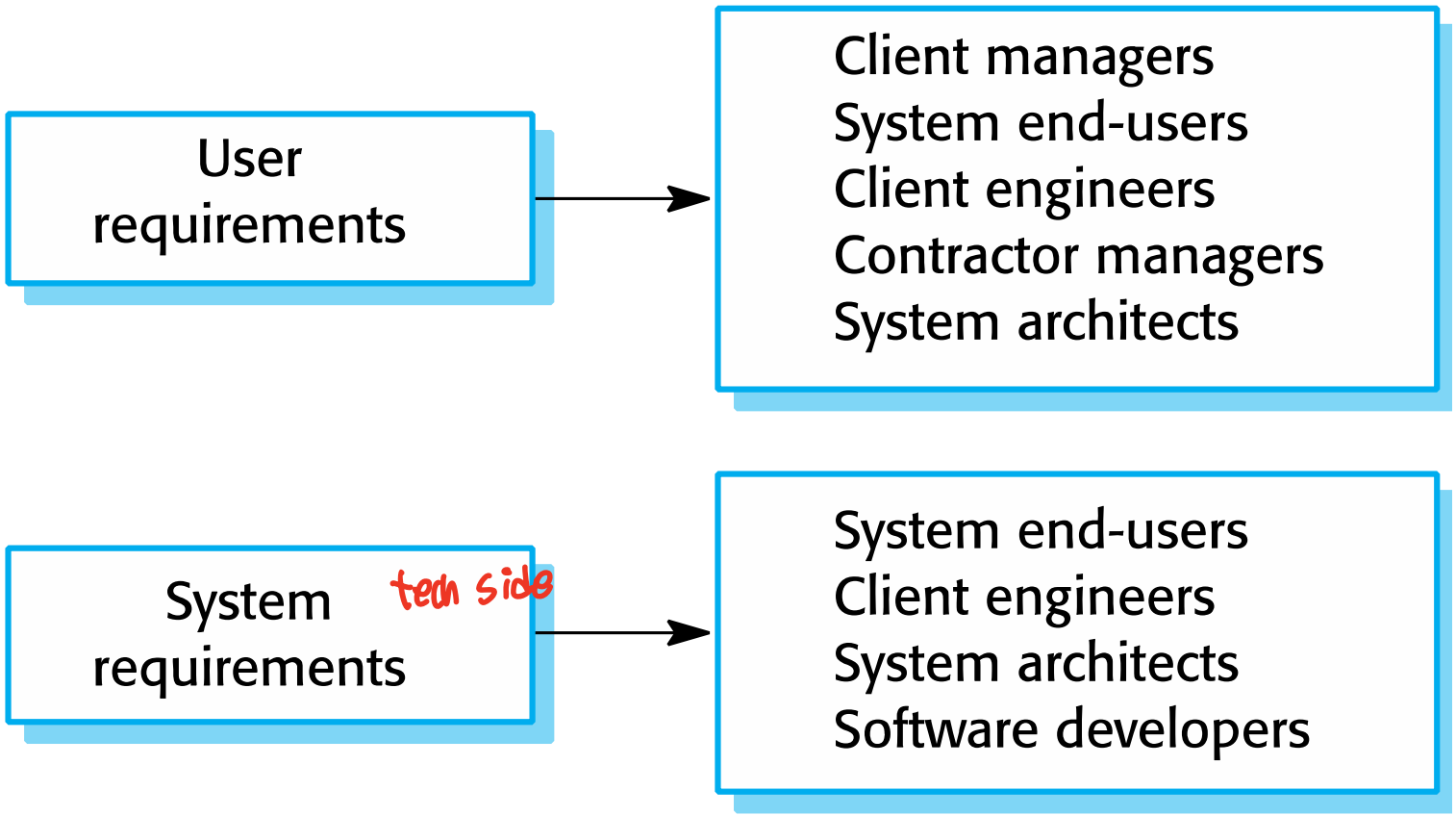
User requirements definition

1. The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20mg, etc) separate reports shall be created for each dose unit.
1.5 Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.

Readers of different types of requirements specification



Flavor of requirement

Functional requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.
- Most user requirements are functional requirements

Requirements imprecision

- Problems arise when functional requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
- Consider the term ‘search’
  - User intention – search for a patient name across all appointments in all clinics;
  - Developer interpretation – search for a patient name in an individual clinic. User chooses clinic then searches.

Requirements completeness and consistency

- In principle, requirements should be both complete and consistent.
- Complete
  - They should include descriptions of all facilities required.
- Consistent
  - There should be no conflicts or contradictions in the descriptions of the system facilities.
- In practice, because of system and environmental complexity, it is impossible to produce a complete and consistent requirements document.

* -lLities

Non-functional requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc. (speeds and feeds)
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.
- We often call these the '-ilities' since they describe things like reliability, scalability, and so on

Non-functional requirements implementation

- Non-functional requirements may affect the overall architecture of a system rather than the individual components.
  - For example, to ensure that performance requirements are met, you may have to organize the system to minimize communications between components.
- A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements that define system services that are required.
  - It may also generate requirements that restrict existing requirements.

MQ: message queue

Non-functional classifications

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Examples of nonfunctional requirements in the Mentcare system

Product requirement
The Mentcare system shall be available to all clinics during normal working hours (Mon-Fri, 0830–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

Organizational requirement
Users of the Mentcare system shall authenticate themselves using their health authority identity card.

External requirement
The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Goals vs requirements

- Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.
- Goal
  - A general intention of the user such as ease of use.
- Verifiable non-functional requirement
  - A statement using some measure that can be objectively tested.
- Goals are helpful to developers as they convey the intentions of the system users.

Non-Goals

- It's just as important to state what the system will NOT do
- This helps to set both expectations and boundaries
- For example:
  This release will not send automated transaction records to the accounting system
- Since requirements documents often become a contract, non-goals save a lot of finger-pointing later on
- "Day two"

Precision in usability requirements

- The system **should** be easy to use by medical staff and should be organized in such a way that user errors are minimized. (Goal)
- Medical staff **shall** be able to use all the system functions after four hours of training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use. (Testable non-functional requirement)
- The use of the words *should* and *shall* here are very precise
- There's even an RFC related to precision: RFC2119

Use cases vs user stories

- In prescriptive processes like RUP the **use case** is a formal document; it contains a significant amount of information
- Most of the time we'll start with **user stories**, which are less formal
- The user story is a description, with props if needed, of a specific set of interactions with the app from either a user's or a subsystem's perspective
- Once we have a set of user stories we can flesh them out into more formal use cases
- For small projects the user story is often enough; larger, more complex projects go with use cases

- Many agile methodologies incorporate a user story into planning
- The story takes the form of “as a <specific actor> I want to <some action> in order to <benefit>”
  - Example: As a logged-in user I want to change my password to improve the security of my account
- RUP and other formal methods also use stories, but they tend to be broader and encompass more functionality.
  - Example: The RUP use case might be “Manage Account Security” and would include the user story along with other stories

Where do you start?

- Developing good use cases or user stories takes practice
- Brainstorming in groups can be a very effective way to get the process started
- Use white boards or stickies to provide flexibility as ideas will change rapidly
- Stay at a high level when doing use cases…they are about actors, actions, and results, not about constraints
- The overall number of use cases will be relatively small; in large systems the UCs or stories are grouped into areas to keep them manageable

- You don't need to get all the UCs nailed down before moving on
  - Iterative processes expect new UCs to be created during development phases
  - You'll always think of something new once you get started
  - It's more important to get started!
  - Analysis paralysis
    Just deserts. Just desserts

Happy paths

- Much of the use case or user story will describe actions along the intended path…the happy path
- If there are conditions that move the actor away from the happy path, document them as exceptions in the use case
- If you find that you have a lot of exceptions, it might be an indicator that you need to go back and examine the use case to see if it can be simplified

Exceptions

- Describe situations (failures or user choices) that cause the system to exhibit unusual behavior
- Brainstorming should be used to derive a reasonably complete set of exceptions for each use case
- Are there cases where a validation function occurs for the use case?
  - Are there cases where a supporting function (actor) fails to respond appropriately?
  - Can poor system performance result in unexpected or improper use actions?
- Handling exceptions may require the creation of additional use cases

Use cases and requirements

- The use case is the collecting point for describing how a system will behave in broad areas.
- Once use cases have been developed, another pass is made in order to elicit requirements
  - Example: The use case for account management includes the user changing a password
  - The requirements would describe how many characters the password must be, what character set, duration, and so on

Requirements and testing

- The use case leads to requirements
- Requirements lead to tests
- In the example earlier, the use case is Account Management, and a specific requirement for password length and character set was stated
- The test would be against the requirement…both positive and negative tests
- This leads us to believe that requirements should be very specific!

Characteristics of good requirements

- Most of these are from the Systems Engineering Guide at MITRE
- CMU's Software Engineering Institute also covers these in great detail
- We'll look at several of their best practices

- **Traceable**: Each requirement must lead back to a specific business or system need which is owned and documented
- **Unambiguous**: Wording should be specific and clear and avoid jargon and ‘intended’ meaning
- **Singular**: One requirement at a time
- **Measurable and Testable**: It must be possible to develop a test to verify that the requirement has been met
- **Self-consistent**: Requirements should not contradict another

- **Feasible**: Someone must be able to actually meet the requirement; prototyping might be needed in order to show feasibility
- **Uniquely identified**: Each requirement must be identified and attached to a use case
- **Design agnostic**: Requirements gathering and documenting is part of the inception phase…no technology assumptions should be made at this stage
- Formal: Use words with specific meaning such as *shall*, *must*, *may*, *should*, and so on to specify intent

Bottom line

- It's important to spend as much time as needed to nail down requirements, since they will form the basis for the project
- Unfortunately humans have a lot of trouble being precise, and cultural, language, educational and other differences can cause misunderstanding
- We'll eventually group these requirements into use cases, which describe at a high level how a system functions
- We use stories (use cases) to start to flesh out the various interactions that the software will have with users and other systems