CE29x Team-Project Challenge

2020-2021

Gathering Requirements in Agile Projects

Professor Anthony Vickers

Room 1NW.3.17 email: vicka

(With acknowledgements to Keith Primrose/Iain Langdon/Mike Fairbank)

Requirements

- What follows is partly based on early chapters from the book:
  - Mastering the Requirements Process: Getting Requirements Right 3e (Robertson & Robertson, 2013, published by Addison Wesley). You do not need to purchase this book. There are some copies in the library but you do not need access.

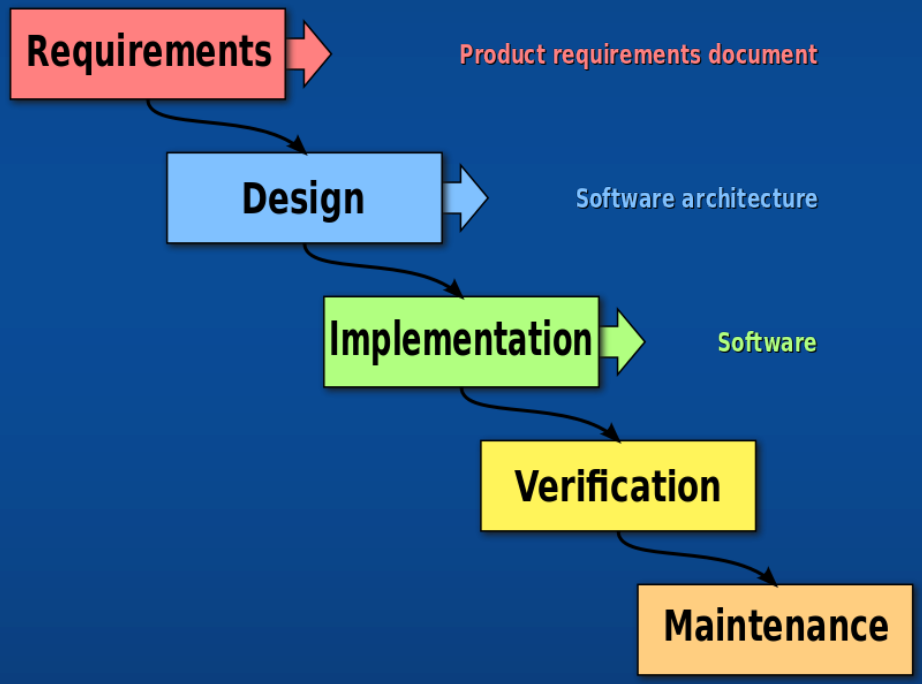
This lecture

1. Agile vs Waterfall
2. Requirements Introduction
    * Gathering Requirements
3. Requirements Documents (Waterfall)
    * SMART criterion
    * "7-Sins of a Specifier"
4. User Stories
5. Other Requirements Modelling Techniques
    * Functional / Non-functional requirements
    * Constraints

1. Agile vs Waterfall

Waterfall

Project is delivered in stages.



Each stage usually signed off by the client separately

Image source: https://commons.wikimedia.org/wiki/File:Waterfall_model.svg

Waterfall

Waterfall pros:

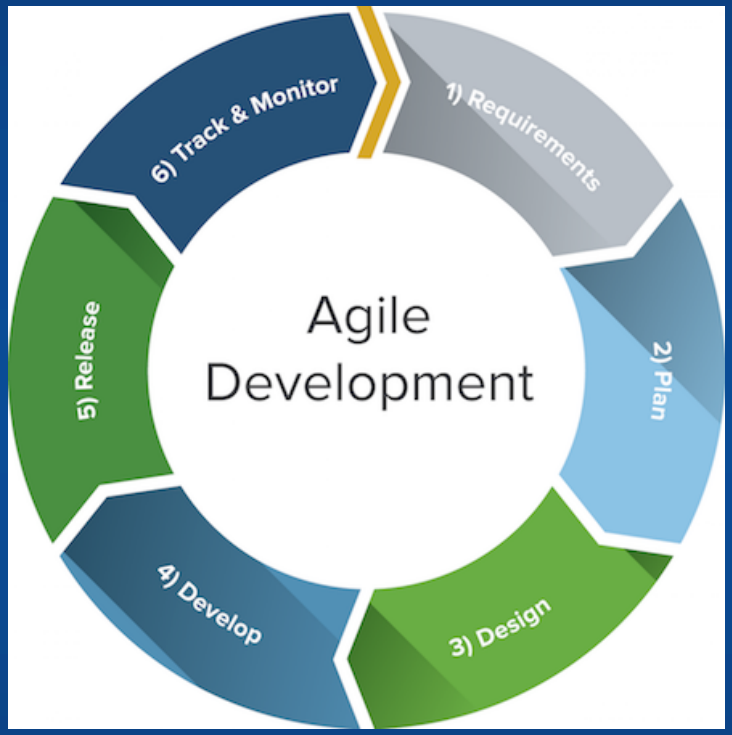
- Client knows what they are getting
  - Functionality
  - Price
- Gives clear milestones
- Clear documentation

Waterfall cons:

- Client might change mind when product arrives
  - Client doesn't know requirements
- Project scope and technology may change
- Very difficult for developers to accurately estimate what they can deliver on time
- Rigid structure not necessary for software (but maybe it is for building hardware / civil engineering)

Agile

Project is developed iteratively:



Client is involved in the process:

Deliver product early. Get feedback from client.

Encourages rapid and flexible response to change.

Iterative

Image source: https://number8.com/kanban-versus-scrum/

Agile

Agile pros:

- Flexibly adapts to changing technologies/desires of client
- Client has involvement in product
  - does not get a surprise at the end
- Planning and estimation of tasks is refined as the project develops

Cons:

- Client doesn't usually get fixed price for given feature list
- Work may set off down bad design route before realising pitfalls
- Not suitable for all engineering projects
  - Foundation can't be changed after tower is built

This lecture

1. Agile vs Waterfall
2. Requirements Introduction
    * Gathering Requirements
3. Requirements Documents (Waterfall)
    * SMART criterion
    * "7-Sins of a Specifier"

3. User Stories

4. Other Requirements Modelling Techniques
    * Functional / Non-functional requirements
    * Constraints

2. Requirements Introduction

Requirements

- Requirements define what 'a product' is meant to do and meant to be.
- They exist regardless of
  - Whether you discover them or not
  - Whether you write them down or not
- The product will never be right unless it conforms to the requirements.
  - This may seem obvious but you can probably think of any number of counter examples.*
- The art lies in discovering the "real problem"

Requirements

- You must come to a correct understanding of the requirements, and get the client's agreement, or your product will be deficient.
- Getting the agreement in writing protects both you and the client.
- Products should be created to solve problems, not just so we can say we have created a product.

Requirements

- You still need requirements regardless of the development lifecycle: Agile, Prototyping, Rational Unified Process, Spiral, Waterfall, whatever.
  - You can be as iterative as you like, but you still need to understand the needs of the client.
  - No matter how you develop the product, the need to understand the client's problem, and what the product has to do to address that problem, remains.

Requirements

- Ideally we should understand the problem well enough to deliver a solution that provides the best payback at the best price.
- To do this we need to understand what the 'owner' values.

Eliciting Requirements

- Ask questions
- Your job is to drag the necessary information out of the client
- For question gathering, consider the 5 W's (What, Why, When, How, Where and Who?) See: https://en.wikipedia.org/wiki/Five_Ws

When questioning different potential users:

- Expect and watch for contradictions
  - Different requirements from different stakeholders
  - Unfortunately, this nearly always happens
  - Must be resolved

Requirements Documents

- essential for waterfall delivery

Requirements Documents

- In waterfall, one stage is to produce a "Requirements Specification" document
  - often abbreviated to just "the specification", or "the spec", or SRS (Software Requirements Specification)
- If you need to write a SRS, consider basing it on "IEEE Software Requirements Specification" (SRS)
  - Template on Moodle CE291
- If your SRS is high quality then it's good for you and the client:
  - The client knows what she's getting.
  - You know you won't be sued afterwards for delivering something different

SMART

Individual Requirements should be SMART

* **S**pecific (what, why, where, when, who, which?)
* **M**easurable (How much, how many, how will I know when it is achieved, indicators for completion)
* **A**chievable (How? No constraint clashes)
* **R**ealistic (Can be done with the available resources)
* **T**ime-bound (When? Give date lines for progress)

* See (https://en.wikipedia.org/wiki/SMART_criteria)

Seven Sins of a specifier

Bertrand Meyer (1985) identified a list of common problems that occur in software specification documents. So a good spec. should avoid these:

1. Noise:
   * A element that does not carry information relevant to any feature of the problem.
2. Silence:
   * The existence of a feature of the problem that is not covered.

Seven Sins of a specifier

3. Over-specification:
   * An element that corresponds not to a feature of the problem but to features of a possible solution.
4. Contradiction:
   * Two or more elements that define a feature of the system in an incompatible way.
5. Ambiguity:
   * An element that makes it possible to interpret a feature of the problem in at least two different ways.

Seven Sins of a specifier

6. Forward Reference:
   * An element that uses features of the problem not defined until later in the document.
7. Wishful Thinking:
   * An element that defines a feature of the problem in such a way that a candidate solution cannot realistically be validated with respect to this feature
   * In other words, if a candidate solution cannot ever be "validated" against the specification, then the specification document is a failure.

See Meyer, B., "On Formalism in Specifications," *Software, IEEE* , vol.2, no.1, pp.6-26, Jan. 1985
doi: 10.1109/MS.1985.229776

Grey Areas

We've seen that:

1. The SRS document should be specific
2. The SRS should not usually give the final design (see 7-sins "over-specification")

So how do we balance these seemingly contradictory aims?

Guiding principles:

- Don't leave the requirements so vague so that your client and you can disagree on whether the objective has been met
- Try not to take options away from your designers
- If something is agreed with the client as a requirement…
  - then put it into the Requirements document

This lecture

1. Agile vs Waterfall
2. Requirements Introduction
    * Gathering Requirements
3. Requirements Documents (Waterfall)
    * SMART criterion
    * "7-Sins of a Specifier"

3. User Stories

4. Other Requirements Modelling Techniques
    * Functional / Non-functional requirements
    * Constraints

3. Agile: User Stories

User Stories

- In Jira, we will model each requirement as a User Story
- *User Stories* look at the requirements from a particular user's view point.
- General form: "As a [persona], I [want to], [so that]."
- Example: "As the departmental manager, I need to be able to view staff sign-in and sign-off times, so I can calculate their monthly pay-cheque."

- We should try to create our "Stories" in Jira like this
- (see, https://en.wikipedia.org/wiki/User_story)

User Stories

- General form: “As a [persona], I [want to], [so that].”
- A user story is the smallest unit of work in an agile framework. It’s an end goal, not a feature, expressed from the software user’s perspective.
- The purpose of a user story is to articulate how a piece of work will deliver a particular value back to the customer.

User Stories

* As Max, I want to invite my friends, so we can enjoy this service together.
* As Sascha, I want to organize my work, so I can feel more in control.
* As a manager, I want to be able to understand my colleagues progress, so I can better report our success and failures.
* As an investor, I want to be able to see how the total value of my funds varied over time, so that I can see if my investment was a wise one or not

User Stories

- Pros:
  - Stories keep the end focus on the user
  - A good story will clearly indicate when it can be considered “done”.
- Consider: “As the departmental manager, I need to be able to view staff sign-in and sign-off times, so I can calculate their monthly pay-cheque.”
  - End user?
  - When is this done?

User Stories

Password recovery story:

* "_____ needs to be able to recover their password automatically, so that ____"

User Stories

- User stories are fundamental to Jira and our project
  - See https://www.atlassian.com/agile/project-management/user-stories for more details
- Try and put all "Stories" into the format "As a [persona], I [want to], [so that]."
- User stories are also the building blocks of larger agile frameworks like epics, themes, and initiatives
  - See https://www.atlassian.com/agile/project-management/epics-stories-themes

This lecture

1. Agile vs Waterfall
2. Requirements Introduction
    * Gathering Requirements
3. Requirements Documents (Waterfall)
    * SMART criterion
    * "7-Sins of a Specifier"

3. User Stories

4. Other Requirements Modelling Techniques
    * Functional / Non-functional requirements
    * Constraints

4. Other methods of modelling Requirements

-try to include one or two of these in your Jira requirements

Use Cases

- *Use Cases* are descriptions of how the system will be used to meet its intended goals.
  - Example Use case:
    - Use-case Title: "Item Purchase"
    - Primary Actor: website user
    - (Story): The user should be able to browse items for sale, select item to place in a virtual shopping cart, enter delivery-address and payment details, and later receive the item by post
  - Example Use case:
    - Use-case Title: "Password recovery"
    - Primary Actor: System administrator
    - (Story):The system admin can reset any users password, but cannot view existing passwords. The system admin would do this when a colleague complains that they've lost their password. The SA would then inform the user verbally what their password was reset to.

Further reading:

- see CE202 week 3+4, or https://en.wikipedia.org/wiki/Use_case

Use-Case Diagram

* A use-case diagram shows the relationship between the user and one or more different use cases.

* In Jira, you could attach a use-case diagram to a Story/Task

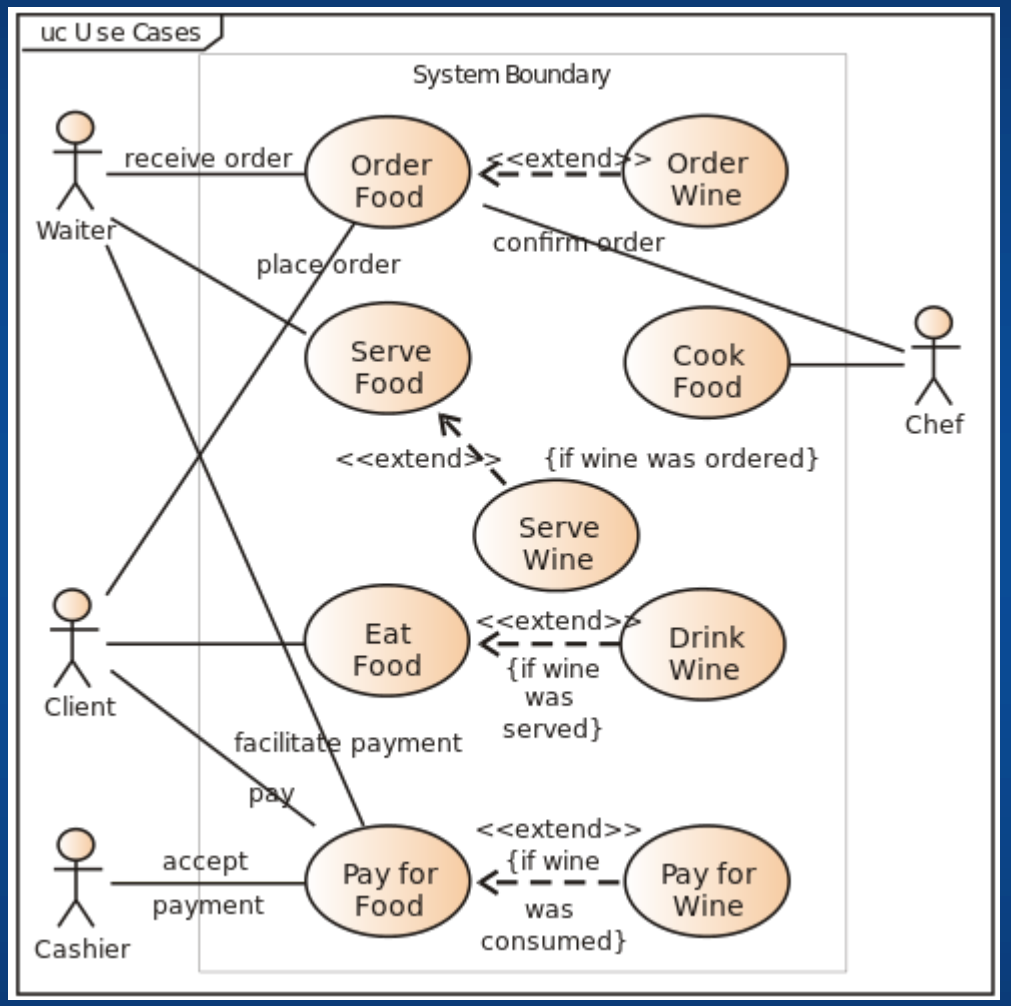


Image credit: https://en.wikipedia.org/wiki/Use_case_diagram

Modelling Techniques

Scenario-based Requirements

* These are scenarios of a use-case, illustrating how things are meant to work,
  * e.g. “when the hard-drive is running low on storage, the system administrator will receive an email alert”
  * e.g. “when the administrator user clicks here, window XYZ opens”

  * (What’s problematic with these 2 requirements?)

Personas

Some agile techniques use the concept of a *persona* to give developers a 'picture' of the user(s) of a product

- even going so far as to give those users a name
- E.g. Pete is a power-user who likes to use keyboard-shortcuts for all menu items
- E.g. Bill has vision problems and uses a screen reader for all web-browsing.

- (Further reading: http://www.agilemodeling.com/artifacts/personas.htm)

Personas

Imagine developing a touch sensitive system to be placed in a shopping mall.

* The system allows users to search for shops that sell the products they have come to the mall to buy

Q: Using "personas", who might we visualise as users of such a system?*

Requirements

- Requirements should define
  - What the product will do - Functional Requirements
  - How well it will do it – Non-functional Requirements
  - Any relevant constraints

Functional Requirements

These describe actions that the product must take if it is to be useful to a user.

* e.g.
  * The product shall detect when the air temperature falls below freezing
  * The product shall produce a schedule of all roads upon which ice is predicted to form within a given time parameter

Non-functional Requirements

These are properties or qualities that the product must have if it is to be acceptable to client or user.

- Cover such things as look and feel, usability, security, legal attributes
- Can be critical to a product's acceptance
- e.g.
  - The product must be able to determine "friend or foe" in less than 0.25 seconds
  - The product shall provide a pleasing user experience
  - The product shall be able to be used by travellers in the arrivals hall who do not speak the home language.
- Beware of "fluffy" requirements. (What is problematic about the second one here?)

Constraints

- Constraints are global requirements.
- Can be limitations.
  - on the project itself
  - or the eventual design of the product
- They are often documented as non-functional requirements.

e.g.

  - The product must be available at the beginning of the next academic year
  - The product shall operate as an IPad, IPhone, Android and Blackberry app.
- Beware of constraints that are not actually genuine business constraints. (What might be wrong with the last one?)*

Constraints and Non-functional Requirements

- Constraints and non-functional requirements can also be modelled as a user story
  - E.g. "Website must have 99.999% uptime" becomes
    - "As a user, I want the site to be available 99.999 percent of the time I try to access it, so that I don't get frustrated and find another site to use."

- Otherwise can represent non-functional requirements / constraints as a task;
  - but that's not as good practice

Further reading: https://www.mountaingoatsoftware.com/blog/non-functional-requirements-as-user-stories

Summary

Summary

We've covered:

* Agile vs. Waterfall
* Waterfall SRS document
  * SMART requirements
  * The 7 Sins of a specifier
* Agile User Stories
* Other requirement modelling techniques

Summary

Plus we've covered:

* Eliciting requirements from the client
* Functional / non-functional requirements
* Constraints

Further reading on Requirements Gathering

More info:

1. https://www.atlassian.com/agile/project-management/user-stories

2. Further reading: Wysocki "Effective Project Management : Traditional, Adaptive, Extreme", chapter 3.

Take the Requirements review quiz on Moodle

Next Online Presentation:

"Estimating Tasks and Stories"