Inf2: SEPP
Lecture 6: Requirements in Plan-driven vs Agile Processes, Software Project vs Software Product Engineering

Cristina Adriana Alexandru

School of Informatics
University of Edinburgh

Last lectures

- Requirements engineering (RE), with subactivities and techniques
  - General, classic way of presenting this topic
  - *Not only for a certain type of software (software project or software product)*
  - *Not only for a certain type of software development process (plan-driven or agile)*
- Use cases and use case diagrams
  - Often associated to documentation which is more heaviweight in plan-driven, but ...
  - *Use cases can be varied in granularity and also used in agile!*
  - UML invented and promoted as part of a plan-driven process, but it is a *modelling language* building *abstractions* and so *can be used in any software development process*!

Summary

- RE in plan-driven vs agile software development processes
- RE in software projects vs software products
  - New techniques: personnas, scenarios, user stories

Plan-driven development processes- Some more pieces to the puzzle

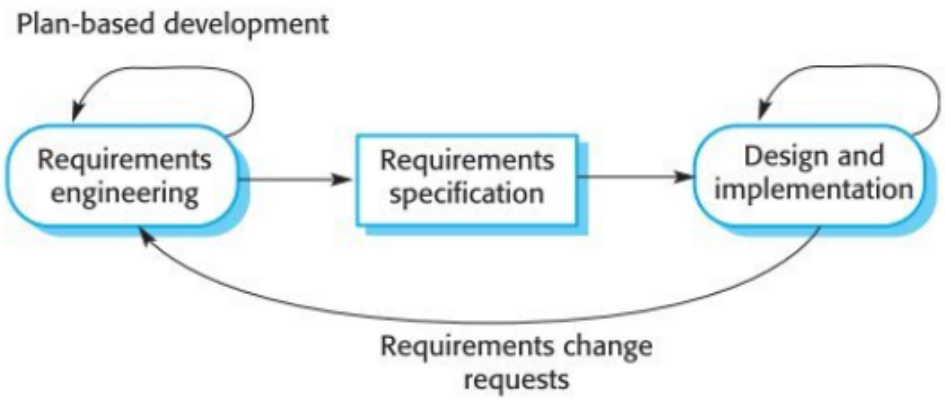
- Reminder: following a plan, 'heavyweight' documentation, reticence to change

Additionally:

- Different SE activities seen as separate stages, each with its output which is used for planning next stage
- Formal documents produced between the different stages
- Iteration within the stages themselves

RE in plan-driven development processes

- Getting requirements 'right' and then producing a thorough requirements specification document
- This document feeds into design and implementation activities
- There can still be iteration back to RE activity (depending on the process)



Taken from: Sommerville, I., 2016. 'Software Engineering 10'. Harlow: Pearson Education Limited.

Agile development processes- Some more pieces to the puzzle

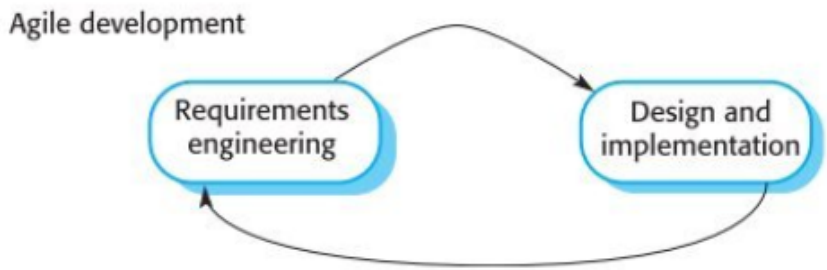
- Reminder: values more individuals and interaction, producing working software ('lightweight' documentation), customer collaboration, responding to change

Additionally:

- Iterative and incremental, working software after each iteration
- Focus on design and implementation ('working software')
- Most important unfinished features (i.e. requirements) chosen for each iteration
- Each iteration occurs *across* activities, with activities interleaved: includes a little bit of requirements engineering, a little bit of design, a little bit of implementation, then repeat

RE in agile development processes

- Happening at the same time with design and implementation in *each* iteration
- Broken down and dispersed within each iteration, omnipresent
- Details are discovered and unfold slowly
- No prescribed way to document requirements, 'just enough'



Taken from: Sommerville, I., 2016. 'Software Engineering 10'. Harlow: Pearson Education Limited.

RE in software projects

Reminder: In software projects, requirements provided as a contract (requirements specification document)/changed by paying customer; large, long-lived systems.

- Plan-driven best choice (see Lecture 2)
- 'Pure' agile incompatible with the need to stick to a requirements specification document
  Solution:
  - Contract missing the requirements specification document
  - Customer paying for time and not functionality
  - Disagreements and conflict possible.

RE in software products

Reminder: In software products, features (i.e. requirements) decided/ changed by dev team; small-medium sized system for several potential customers

- Important considerations:
  1. Need to attract customers to buy product
  2. Need to beat competition
  3. Speedy delivery to the market of the essence
  4. Reactivity to changes (needs, competitors with new features)
  5. Few products have innovated without customer input; Fewer have then kept their customers.
- Points 1-4 motivate the need for agile, ideally suited
- 'Pure' plan-driven not good
- Point 5 motivates the need to find out potential user needs

RE in software products: Understanding user needs

Interviews, surveys, facilitated meetings expensive

For small companies, informal user analysis and discussions preferred: asking potential users about their work, the software that they use, its strengths and weaknesses

Then, personnas, scenarios and user stories can be used to *suggest* requirements (IMPORTANT! Requirements *decided* by development company)

Understanding user needs: Personnas

'Imagined users'/character portraits describing a type of user

Can be derived from informal discussions, and should be cross-checked against the user data

Numerous templates available for defining a persona

Common components of a personna:

- Personalisation: name, personal circumstances, stock photograph
- Job-related (if business product): about their job and (if necessary) what it involves
- Education: background, level of technical skills and experience
- Relevance: motivation for using the product, what may want to do with it

Understanding user needs: Personnas

Example (for GymClass, an online gym class booking system):

Francesca, a fitness coach

Francesca, age 23, is a part-time fitness coach in Central Gym. She is also a full-time 4th year Masters student in Psychology in the local university. Originally from Italy, she has grown up in a small town with a long tradition for sporting competitions, and has always been passionate about sports and exercise. She has a diploma in Psychology from the University of Pisa. Since moving to the UK in 2014, Francesca has occupied several part-time jobs before finally ending up in the current position which she sees as a perfect fit as it allows her to keep being active while also observing human psychology and its relationship with sports and exercise. She is single, has a very busy lifestyle, and is not very comfortable with technology.
Francesca has been struggling with the current gym class booking system, which she finds very unfriendly and unintuitive, and this is why she often relies on her more tech-savvy colleagues in making class bookings for the gym members. However, she is glad that the system will soon be replaced with GymClass which promises to be more catered towards the needs of CentralGym coaches, and she is happy to give it a go. Now that she is leading her own Aerobics class, she would be particularly interested in using it for setting up the class and managing her list of attendees.

Understanding user needs: Personnas

Advantages of personnas:

- Help explain why the system would be useful and give examples of what users may want to do with it
- Help team reach shared vision about users, their skills and motivations (otherwise inconsistencies leading to inconsistent requirements possible)
- Developers empatising/stepping into the users' shoes

Disadvantage: will overlap after a number of personnas. Sommerville advises using max 5, which will help find the key system features.

Understanding user needs: Scenarios

Narrative explaining the context (the user's problem), and an imagined way that the user may address it (on or outside the system), written from the perspective of the user

Can be based on personnas or real users

For software products: Sommerville suggests using high-level scenarios, different to use case scenarios!

Understanding user needs: Scenarios

Most important components of a scenario:

- Brief statement of overall objective
- References to the personna involved
- What is involved in reaching the objective
- (Optional) Problem that cannot be addressed by current system
- (Optional) One way that the identified problem might be addressed

Understanding user needs: Scenarios

Example: Francesca's scenario: Setting up her Aerobics class on GymClass

Francesca is a fitness coach in Central Gym. She has recently been promoted to leading a new Aerobics class, which will take place during early mornings on weekdays to attract professionals and students. As with several other classes in Central Gym, the class should accept both gym members holding yearly memberships, as well as one-day gym members.
Francesca logs into her GymClass account with her staff number and password. The system recognises that she is working in the central Edinburgh gym, and provides a welcome screen which is populated with news on current classes, how busy it is, and popular sports and exercise blog posts. Francesca navigates to the area for classes, and the system shows a list of all existing classes. Her 'My classes' area is currently empty. She proceeds to choosing to create a new class. The system prompts her for the class details including class name, timetable, number of accepted attendees. At the next step, Francesca is asked for more advanced settings, and it is here that she needs to select opening ip up for one-day members. Finally, she has the choice of whether to make the class available for bookings, or save it for later. . . . Francesca also wants to add details about her profile for the class. As she never uses Flickr, she chooses the option to upload photos of herself from her computer. . . .

Understanding user needs: Scenarios

Advantages of scenarios:

- Like personnas, help reach shared understanding of system
- Way of brainstorming with the team what system should do; help move towards requirements
- Facilitate communication and stimulate design creativity
- Read naturally, provide context
- Easy to write and understand, and so useful to get users involved in their development

Disadvantages:

- Not specifications, so may be incomplete, lack detail
- Overlap

Sommerville recommends using 3-4 scenarios/personna.

Understanding user needs: User stories

Finer-grained narratives setting in a structured way a single thing that a user wants from the system

Describe a sequence of interactions with the system, without details of the interactions

Can be derived from scenarios, but more user stories may be needed for a complete description of functionality

Understanding user needs: User stories

Formats:

As a <role>, I <want/need> to <do something>

or

As a <role>, I <want/need> to <do something> so that <reason>

The latter is helpful if developers unfamiliar with what users do, and can help come up with alternatives for providing what the user needs.

Understanding user needs: User stories

Examples: Francesca's user stories

As a fitness instructor, I want to be able to set up my own classes on GymClass.

As a fitness instructor, I want to be able to make classes available to one-day gym members.

As a fitness instructor, I need to be able to upload photos of myself for my classes' instructor profile from my computer so that I don't need to use online image hosting services that I am not familiar with.

Understanding user needs: User stories

Advantages of user stories:

- Like personnas and scenarios, help reach shared understanding of system
- Like scenarios, way of brainstorming with the team what system should do; help move towards requirements
- Like scenarios, facilitate communication and stimulate design creativity
- If sufficiently detailed, can be used for planning the next iteration in agile

Disadvantages:

- Like scenarios, not specifications, so may be incomplete, lack detail

Scenarious are: more natural, easier to relate to and easier to understand by users; provide more context about user actions and way of working

Deciding requirements from scenarios and user stories

Look through scenarios and:

1. Identify actions denoted by active verbs (e.g. use, send, update, open, etc.)
2. Highlight their phrases
3. Think about what can support these actions on the system and how they could be implemented

User stories may immediately suggest requirements

Advantages/disadvantages of scenarios and user stories for suggesting requirements

Consider user needs

Can result in a software product that is accepted by users

But . . .

Lock in existing ways of working, by showing how users currently do things

Advice: start by using them, but then also think creatively about other/additional more efficient and interesting options

Reading

- Essential: On RE in agile vs. plan-driven: Sommerville SE Chapter 3 up until 3.1
- On scaling up agile (i.e. using it for large systems): Sommerville SE Chapter 3 Section 3.4 (Essential: especially until 3.4.3)
- Essential: On RE in software product engineering: Sommerville ESP Chapter 3