L6 - From inception to elaboration

CS3028 - Principles of Software Engineering

Ernesto Compatangelo

Department of Computing Science

UNIVERSITY OF ABERDEEN

6.1 Reminding past issues and mapping them to current topics

Where are we now?

Software development paradigms

⇒ The Unified Process (UP) paradigm

⇒ UP phases and UP disciplines (activities) within each phase

⇒ Inception (first UP phase)

⇒ Elaboration (second UP phase)

⇒ Moving from inception to elaboration

⇒ Requirements beyond user stories

6.2 Summarising inception

A few words of wisdom about inception

- There is no well-defined borderline between inception and elaboration
  - Some (experienced) teams can decide go more in depth during inception, others (like yourself) will refrain from doing so
  - If there are $\mathcal{N}$ textbooks/documents describing inception there will be around $\mathcal{N}+1$ different interpretations of inception
  - Once you have examined the most critical aspects, just move on
- Even if a well-defined borderline between inception and elaboration does not exist, inception outcomes are quite clear

Moving to elaboration

- When can we declare inception realistically over?
  - For professionals, after between one and two weeks
  - For newcomers/part-timers, after between two and four weeks
- What needs to be worked out more in detail before we can declare inception over?
  - A more detailed specification of the critical requirements
  - Small chunks of code that demonstrate the usability of the envisaged technologies in the critical technical cases

6.3 Requirements: detailed focus

A closer look at requirements

- Functional requirements can be captured as **user stories**:
  - A **one-sentence description** of the functionality
  - **Informal materials** about the story, stored in your mind, which are used to flesh out the details of the story
  - **Tests** that convey and document details and that can be used to determine when a story is complete
- However, there are situation where user stories are not enough:
  - Some functional requirements capture user-system interactions that are too complex to fit in a single sentence / short paragraph
  - Requirements specify a big system developed by many analysts, designers and coders

Reminding requirements expressed as user stories

- A user story describes a functionality that will be valuable to either a user or a purchaser of the software system it contributes to specify
- A user story is a single sentence; if the requirement needs more sentences, it's because it represents more user stories and thus more requirements
- A user story can be complemented by short notes and test cases; ideally these should be written on the back of a single card whose front only contains the user story.

Requirements more in depth: the FURPS+ model

- Non-functional requirements are too wide and important to be collapsed under such a generic label; hence the FURPS+ model has been introduced which classifies requirements as,
  - **Functional**: ditto
  - **Usability**: human factors, help, documentation
  - **Reliability**: frequency of failure, recoverability, predictability
  - **Performance**: response times, throughput, accuracy, availability, resource usage
  - **Supportability**: adaptability, maintainability, internationalisation, configurability

Specifying the '+' in FURPS+

The '+' in FURPS+ denotes further categories of non-functional requirements that often underpin the non-functional 'ilities', namely:

- **Implementation**: resource limitations (HD, RAM), languages and tools, HW, OS
- **Interfacing**: constraints imposed by the interface with other (external, often legacy) systems
- **Operations**: how to manage the system in its operational setup
- **Packaging**: how to package the system for distribution (anything from jars to a single CD-ROM with SW protection)
- **Legal**: licensing, disclaimers, IPR protection, patenting

6.4 Requirements specification: heavyweight alternatives to user stories

Requirements specification alternatives

- We realise at this point that lightweight user stories are not enough.

  Do we have more comprehensive ways to specify requirements?

  - We can follow the 'old school' (US, 80s-90s) and embrace the **IEEE Standard 830-1998** (Revision of the IEEE Std 830-1993)
  - We can go 'the UP way' and express functional (and their associated non-functional) requirements by way of **use cases**
  - We can adapt & simplify the IEEE Standard 830-1998 and blend it with the UP approach
- To some extent, both of these heavyweight specification methods are industrial standards for large applications. The choice depends on a number of factors (*e.g.*, company standards, contract clauses. . .)

6.5 The IEEE Standard 830-1998

- The standard can be downloaded from the CS3015 website:

  `information/abdn.only/IEEE-STD-830-1998.pdf`
- The standard expects a Software Requirements Specification (SRS) document to include a detailed description of
  1. External interfaces (GUI, HW, SW, Comms)
  2. Functional requirements (either by mode, or class, or object, or feature, or stimulus, or functional hierarchy. . .
  3. Performance requirements
  4. Design constraints (*at this stage?*)
  5. Software system attributes
  6. Other requirements (whatever doesn't fit anywhere else)

6.6 Preparing for the topic ahead

Next week...

Requirements during elaboration: introducing UP use cases

More specifically, we will focus on:

- Requirements elicitation
- How requirements are organised in UP artifacts
- The UP notion of use case