

Expressing Requirements

John Reekie
University of Technology, Sydney

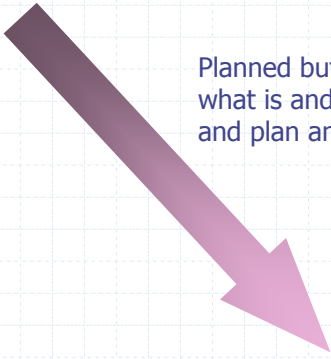
Contributors
Lian Loke, University of Technology, Sydney

“Requirement”

- ◆ That which a system shall do.
- ◆ Seems simple enough, but there are a lot of difficulties in this seemingly-simple task:
 - Technology changes constantly
 - Stakeholders may not even be aware of what they need
 - User needs change
 - The actual system is the result of a complex set of trade-offs (legal, market, organizational, as well as technical) which may not be fully understood.
- ◆ There are different responses to this difficulty.

Responses to requirements difficulties

Extensive requirements gathering and formalization in order to minimize chances of change later



Planned but iterative. Understand what is and is not known now, and plan and adapt accordingly.

"On the fly" requirements elicitation in close collaboration with users and customers

Freeze requirements

◆ Requirements are determined completely in an early phase of the project. They are then "frozen" and form a contractual agreement between the customer and the supplier.

◆ Issues:

- Not always realistic
- Doesn't support changing requirements
- May lead to more expensive or brittle system

On-the-fly requirements

- ◆ Since requirements will change, determine any requirement informally and then only right before it is about to be implemented.
- ◆ Exemplified by agile development methods
 - XP – “User stories”
 - Scrum – Prioritize the product backlog for each “sprint”
- ◆ Issues
 - Requires dedication to a particular way of working
 - Relies on skilled individuals
 - Needs constant interaction with the “customer.”
 - Quality of the customer can make or break the project.

Be flexible

- ◆ Different projects have different needs:
 - Scale
 - Safety
 - Complexity
 - Cost-sensitivity
 - Market-driven (innovation)
- ◆ Compare:
 - Digital camera
 - Desktop chat client
 - Nuclear power station control system
 - Digital television studio

Consider:

- ◆ The degree to which requirements can be determined and at what stage of the development process
- ◆ The amount of effort that is cost-effectively put into gathering and documenting requirements (vs benefits in terms of reliability and customer satisfaction)
- ◆ Notations that can be shared and understood – development teams, customers, marketing, management
- ◆ Notations that mesh well with design and implementation activities

In this subject...

- ◆ We will use three techniques, which can be used to *elicit* and *document* requirements
 - Usage scenarios
 - Design sketches
 - Essential use cases
 - Detailed use cases
- ◆ These are not the only ones
- ◆ Usage scenarios and design sketches are informal. Use cases are somewhat formal, and are part of what is commonly known as “object-oriented analysis.”
- ◆ Don’t forget that the Unified Process **does not** require requirements analysis to be completed before other workflows can be started!

Usage scenarios

- ◆ An informal description of a particular activity associated with the use of the system.
- ◆ Often includes "personas" --- characters to represent different users.
- ◆ Not part of any particular methodology.

"Sally inserted her card into the ATM and withdrew \$100. She was in a hurry, so was a bit annoyed that it took so long."

(Discuss!)

Why scenarios?

- ◆ Informal descriptions are relatively quick (and therefore cheap) to write
- ◆ Easily understood, discussed, and written by non-technical stakeholders
- ◆ Captures more of the context of system development:
 - Business needs
 - User goals and desires
 - Emotional response to system activity
 - External constraints
- ◆ Closely related to established traditions (in non-software fields) for dealing with human goals and with uncertainty:
 - Participatory design
 - Scenario thinking

Limits of scenarios

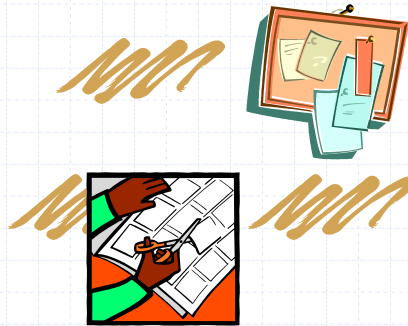
- ◆ Not suitable for contractual development styles
 - “You **said** the system would do *this*” – scenarios don’t have enough detail
 - Scenarios are selective, not exhaustive
- ◆ Hard to formalize (for those processes that insist on formalizing everything)
- ◆ Not amenable to automatic interpretation
 - But then again, most requirements methods aren’t
- ◆ Conclusion: use in the early phases, as a way to understand system context and stakeholder goals and needs. Add more formal requirements methods and artifacts if and when needed.

Design sketches

- ◆ Create a *rough sketch* of some part of the user interface of a proposed system
- ◆ Use it as a basis for discussing user or customer needs. (These may remain hidden until a suitable artifact such as a prototype or sketch is provided.)
- ◆ The informal nature of a sketch help people to focus more on the functionality implied by the sketch and less on the appearance of the UI.
- ◆ Help to clarify functionality.
- ◆ Extend into paper prototypes to understand user needs and to refine user interaction, early and at relatively low cost.

Paper prototypes

- ◆ Low-cost exploration of user interaction
 - Focuses on the essential nature of the interaction
 - One person is user, the other "plays" computer
 - Gathers knowledge
 - Cheap and fast to "rebuild" interfaces



http://en.wikipedia.org/wiki/Paper_prototypes

Use cases

- ◆ A structured, formally-written description of a specific user interaction with the system. A single use case describes the activities of the user and the system in the course of accomplishing a particular task.
- ◆ Use cases are an accepted part of object-oriented analysis. They are not restricted to OO systems, though.
- ◆ Use cases are often written to formal templates. See *Writing Use Cases* by Alistair Cockburn for (much) more detail.
- ◆ There are many different templates and styles of use case. In this subject we will start with a particular variant called *Essential use cases* (*User-Centered Design*, Constantine and Lockwood)

http://en.wikipedia.org/wiki/Use_cases

An essential use case

User action

1. Identify self
4. Make choice
6. Take card
8. Take cash

System responsibility

2. Verify identity
3. Offer choices
5. Eject card
7. Dispense cash

(Discuss!)

What?

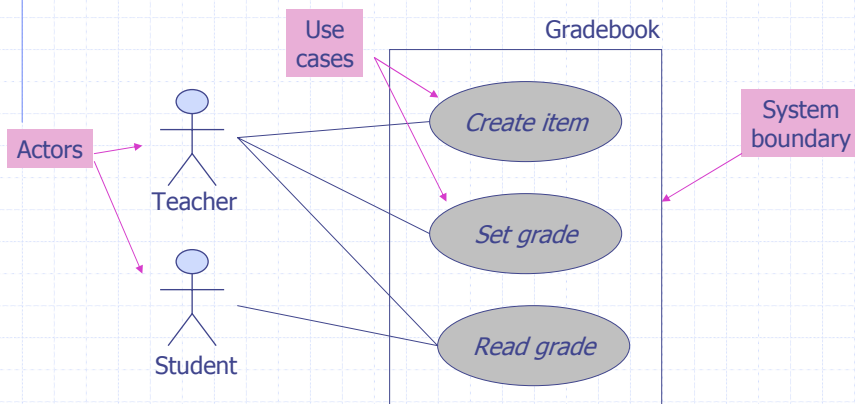
- ◆ The (essential) use case is divided into two columns
 - User intention
 - System responsibility
 - Write on a single index card
- ◆ Use terse phrases
- ◆ Keep it abstract: don't specify the user interface, just say what the user or system does in *domain-level* terms
 - "Pulled down the File menu..." BAD
 - "Opened the grades spreadsheet" GOOD

Detailed use cases

- ◆ Written in more detail than essential use cases
 - User actions
 - System responses
- ◆ Alternate paths
 - The main path is written as-is
 - Alternate paths written as a separate subset of actions and responses
 - Use them if (and only if) you need them
- ◆ Exceptions
 - Exception paths occur when something unexpected occurs
 - Again, use if (and only if) needed
- ◆ A detailed use case is a much more elaborate beast than an essential use case...
- ◆ Danger: don't design the user interface in the use cases...

Use-case diagrams

- ◆ Clarify the relationship of **actors** to **use cases**
- ◆ Use case diagrams are **not** use cases!!!



Limits of use cases

- ◆ Detailed use cases are **still** imprecise
 - If you really want precision, use mathematics
 - Trying to make use cases precise can be very expensive
 - Precise use cases are longer and more expensive to change (when requirements change)
- ◆ There is no standard style of use case.
- ◆ Use cases are not part of UML (really).

In this subject...

- ◆ You are expected to use all of these techniques as appropriate (as determined by the MUP and your own progress)
- ◆ Inception phase
 - Stakeholders and scenarios
 - Interface sketches
- ◆ Elaboration phase
 - Actors and essential use cases
 - Paper prototype
- ◆ Construction phase
 - Detailed use cases
 - Test cases derived from use cases