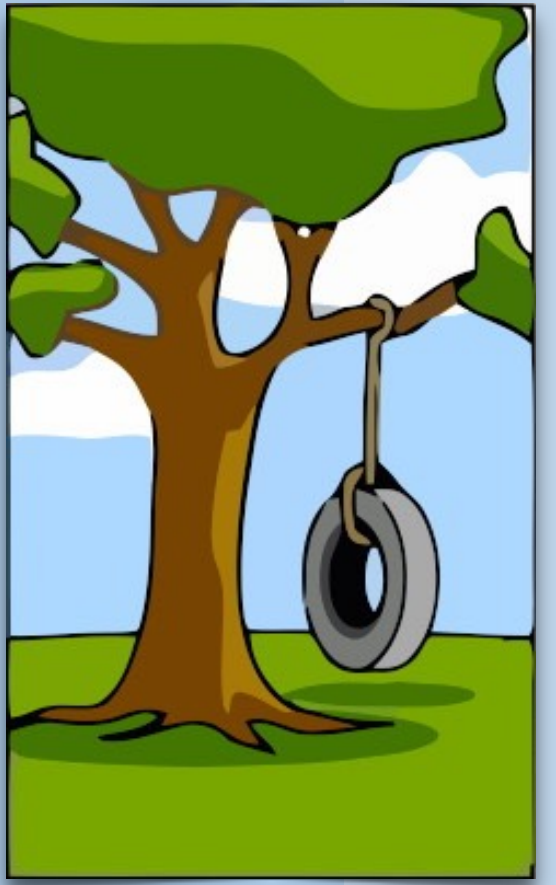
Introduction to Software Engineering

Requirements Collection



Roadmap

> The Requirements Engineering Process
> Use Cases
> Functional and non-functional requirements
> Evolutionary and throw-away prototyping
> Requirements checking and reviews

Sources

> *Software Engineering*. Ian Sommerville. Addison-Wesley, 10th edition, 2015

> *Software Engineering: A Practitioner's Approach*. Roger S. Pressman. McGraw Hill; 8th edition, 2003.

> *Objects, Components and Frameworks with UML*, D. D'Souza, A. Wills, Addison-Wesley, 1999

Roadmap

> **The Requirements Engineering Process**
> Use Cases
> Functional and non-functional requirements
> Evolutionary and throw-away prototyping
> Requirements checking and reviews

Zeitschema

Kommission: ____________________

Bitte ankreuzen wo Sie keinenfalls mitmachen können, und senden Sie das ausgefüllte Formular bis ___________ ans Dekanat zurück.

	Jan 2002																									Feb 2002							
	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	1	2	3	4	5	6	7	8
8:00 - 9:00																																	
9:00 - 10:00																																	
10:00 - 11:00																																	
11:00 - 12:00																																	
12:00 - 13:00																																	
13:00 - 14:00																																	
14:00 - 15:00																																	
15:00 - 16:00																																	
16:00 - 17:00																																	
17:00 - 18:00																																	

Bemerkungen: ____________________ Unterschrift: ____________________

This is the old paper form that the Dean's office of our Faculty used to use to schedule meetings. Every participant in the meeting would be sent (by post) such a form to fill in. The Dean's office would collect the responses and try to schedule a meeting.

Electronic Time Schedule

"So, basically we need a form for the time schedule that can be distributed by eMail, a place (html) where I can deposit these forms after they have been filled out, and an algorithm that calculates a few possible meeting times, possibly setting priorities to certain persons of each committee (since there will always be some time schedule overlaps). It would also be great if there were a way of checking whether everybody of the relevant committee has really sent their time schedule back and at the same time listing all the ones who have failed to do so. An automatic invitation letter for the committee meeting to all the persons involved, generated through this program, would be even a further asset."

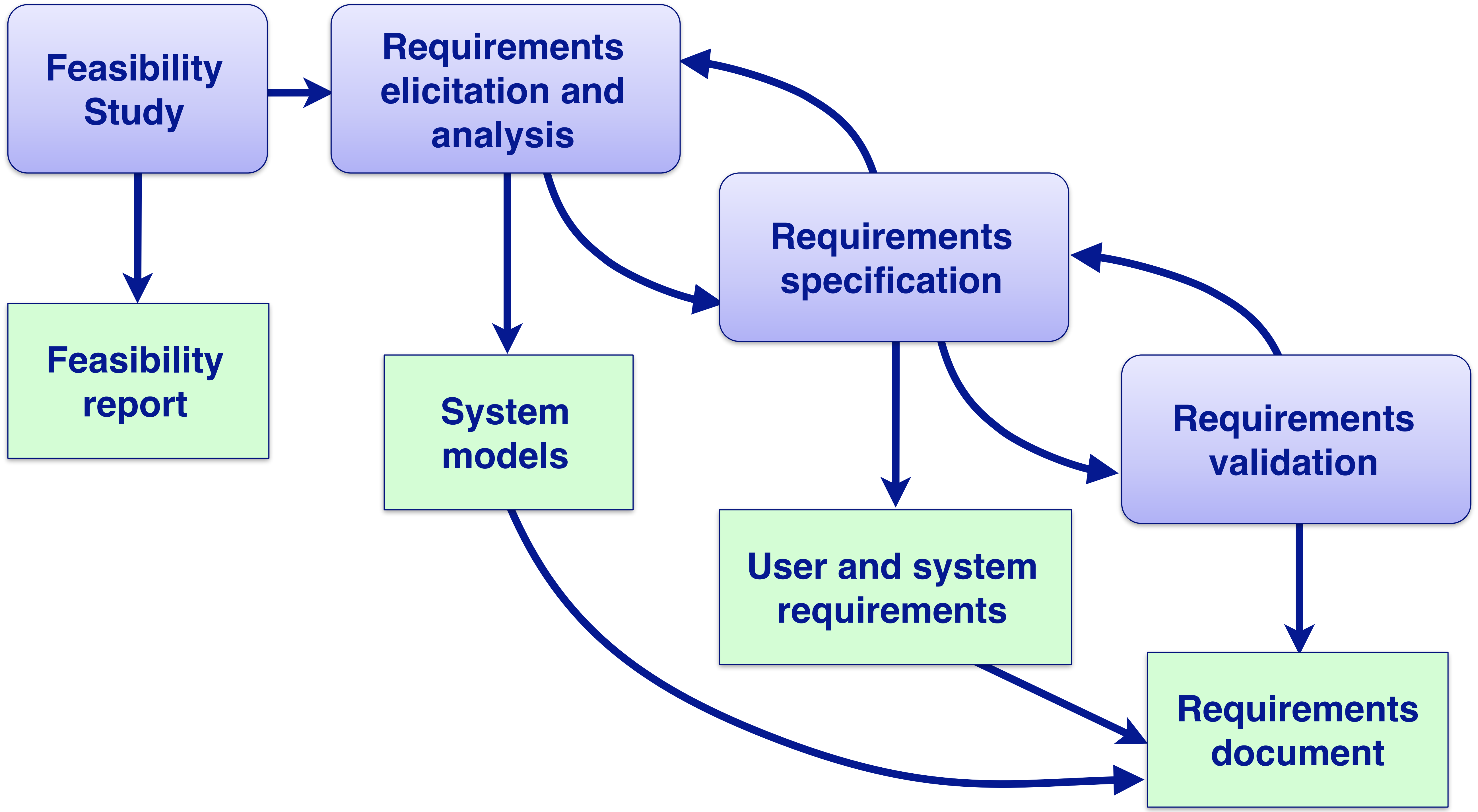
How can we transform this description into a requirements specification?

I received a request from the Dean’s office some years ago to explore an electronic solution to this problem. (Note that this was before Doodle appeared on the scene!)

Have a careful look at this email. Is there enough information here to start a design?

- Who are the stakeholders in this project?
- What are the possible “use cases” and “scenarios”?
- What are the technical and non-technical requirements?
- What should the system optimize? (Reduced work for the stakeholders? More optimal scheduling? …)

The Requirements Engineering Process



This figure from Sommerville (2000 edition) illustrates some of the key activities in requirements collection. An important part of the analysis is to capture *domain knowledge* (AKA “system models”). In our example, we have to understand at some level what kind of meetings take place in the Faculty, who participates in them, how the meetings are run, and so on.

Then we need to collect specific user requirements (use cases and scenarios), and also understand system requirements (what kinds of platforms do we need to consider). In the end, we have to produce (and maintain) documentation of these models and requirements.

Note that these activities are iterative and incremental, not sequential.

Requirements Engineering Activities

Feasibility study	Determine if the *user needs* can be *satisfied* with the *available technology* and *budget*.
Requirements analysis	Find out *what system stakeholders require* from the system.
Requirements definition	*Define the requirements* in a form understandable to the customer.
Requirements specification	Define the requirements in *detail*. (Written as a contract between client and contractor.)

“Requirements are for users; specifications are for analysts and developers.”

Be aware that the terms “requirements collection”, “requirements elicitation”, “requirements analysis”, “requirements definition”, and “requirements specification” can be highly ambiguous and confusing. What is important is that *the requirements engineer must interact with the customer* and the other stakeholders to collect the requirements, and to document them in a way that is understandable to everyone.

At some point, these requirements may be turned into “specifications” that may form the basis for a work contract.

That’s why we say, “Requirements are for users; specifications are for analysts and developers.”

What would each of these activities look like for the electronic time schedule project?

Requirements Analysis

Sometimes called *requirements elicitation* or *requirements discovery*

Technical staff work with customers to determine

> the application *domain*,

> the *services* that the system should provide and

> the system's operational *constraints*.

Involves various *stakeholders*:

> e.g., end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc.

For the electronic time schedule project, we would have to carry out interviews with various stakeholders: the administrative head of the Dean's office, selected committee heads and members, and so on. As we will see, an effective way to solicit requirements is to discuss concrete *use cases* and *scenarios* with the stakeholders, and then compare them for consistency.

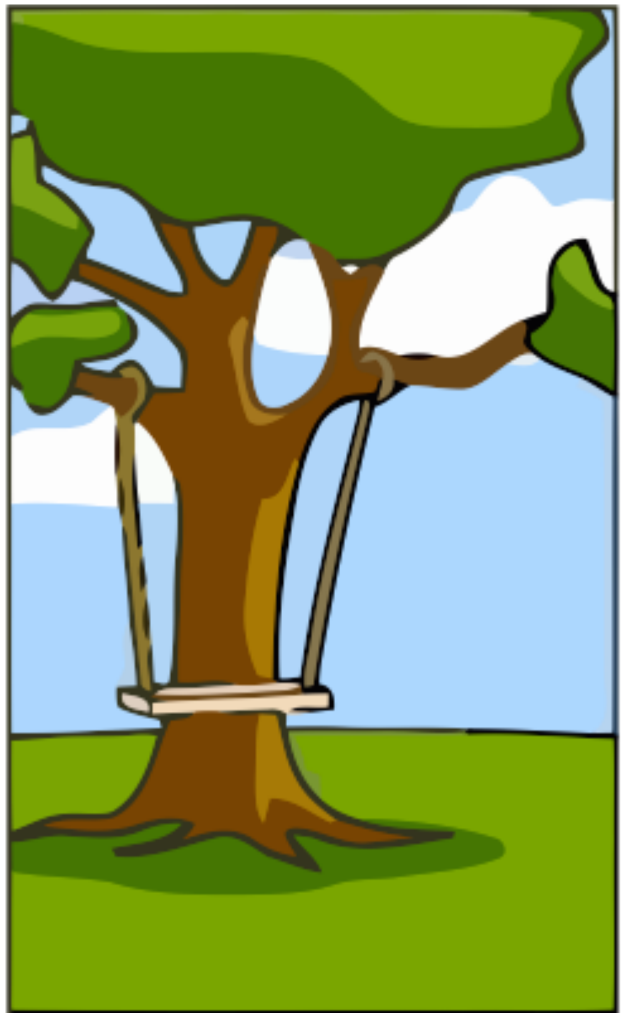
Problems of Requirements Analysis

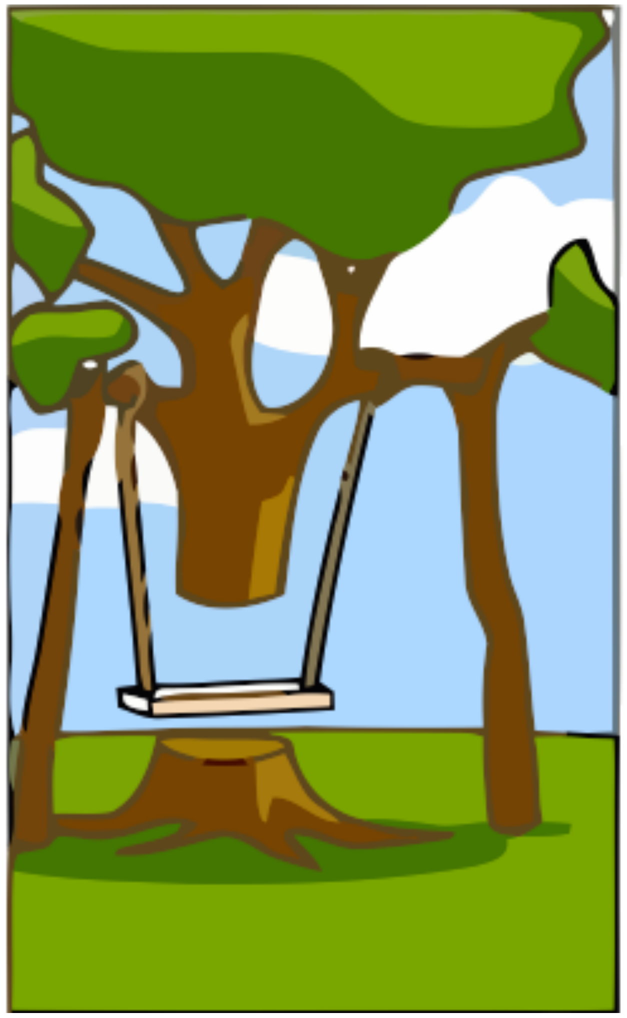
Various problems typically arise:

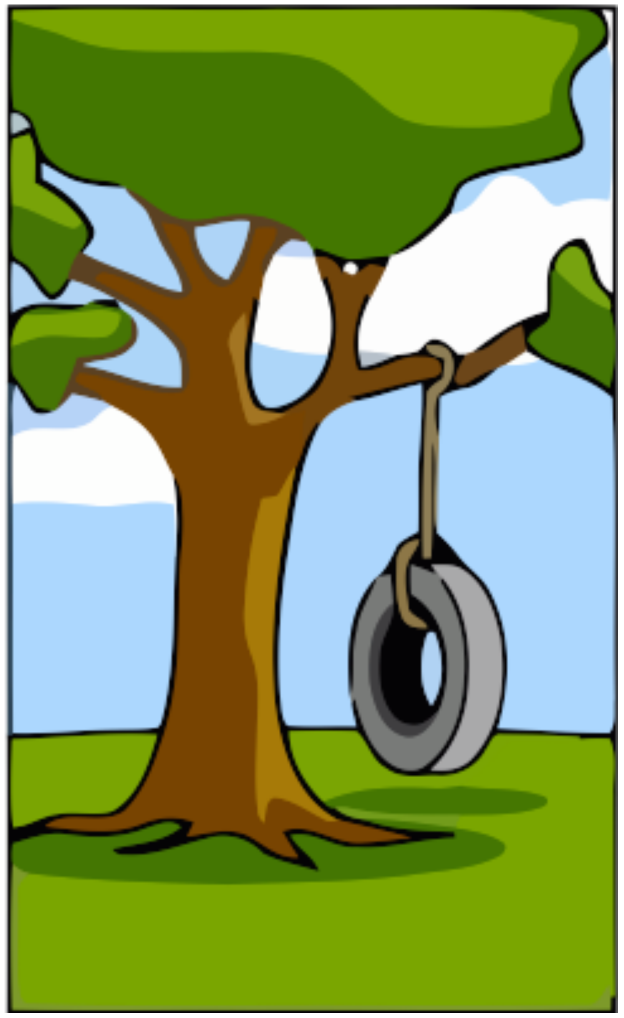
- Stakeholders *don’t know* what they really want
- Stakeholders express requirements *in their own terms*
- Different stakeholders may have *conflicting requirements*
- *Organisational and political factors* may influence the system requirements
- The *requirements change* during the analysis process.
- *New stakeholders* may emerge.

In our project, stakeholders may not be aware of possible alternative technical solutions. Each person comes to the project with their own perspective, and may only have a vague idea of the requirements of other stakeholders.

How the Customer explained it


How the Project Leader understood it


How the Analyst designed it


What the Customer really needed

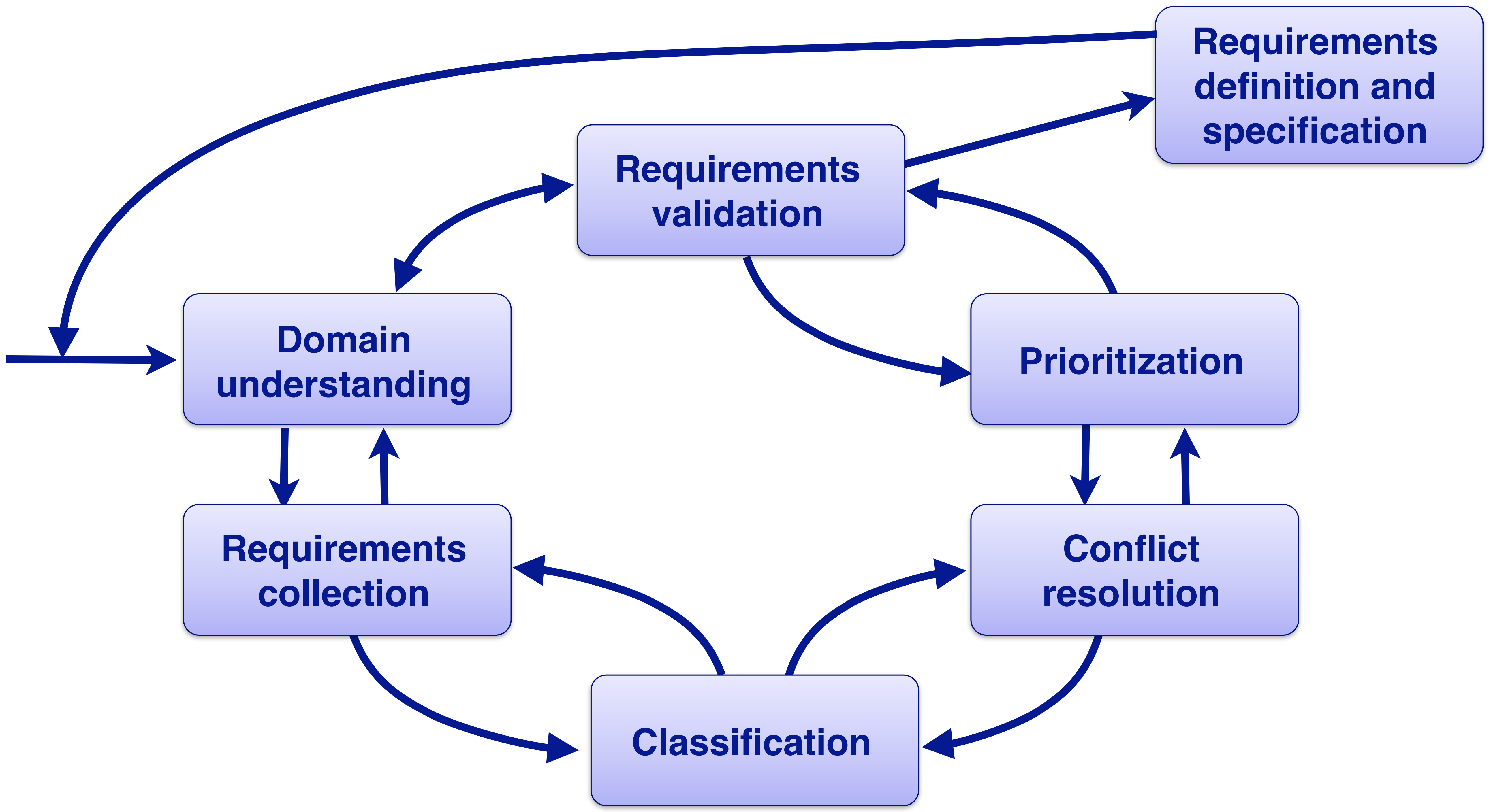
There are many versions of this ancient cartoon. The key thing not to lose sight of is “what the customer needs”! Since there may be many stakeholders, it is important to figure out who is driving the project and what the business case is.

Requirements evolution

> Requirements *always evolve* as a better understanding of user needs is developed and as the organisation’s objectives change

> It is essential to *plan for change* in the requirements as the system is being developed and used

The Requirements Analysis Process



Another diagram from Sommerville (2000 edition).

Don’t take this diagram too literally, but consider it as a rough guideline to the key activities:

- Domain understanding: gather & understand *domain knowledge*
- Collection: document *use cases & scenarios* by *interaction* with stakeholders
- Classification: into coherent clusters
- Conflict resolution: i.e., due to *different stakeholders*
- Prioritization: i.e., importance, value …
- Validation: checking consistency, completeness …

What would these activities look like for our timetable project?

Roadmap

- > The Requirements Engineering Process
- > **Use Cases**
- > Functional and non-functional requirements
- > Evolutionary and throw-away prototyping
- > Requirements checking and reviews

Use Cases and Scenarios

A *use case* is the *specification* of a *sequence of actions*, including *variants*, that a system (or other entity) can perform, *interacting with actors* of the system”.

- login to the timetable system
- define a new committee
- schedule a meeting

A *scenario* is a *particular trace of action occurrences*, starting from a known initial state.

- go to the timetable web site; enter your login; click on “forgot password” …
- enter “new committee”; enter committee name; select committee members from database; indicate committee chair; …

Ivar Jacobson described this approach to software development in his classic book:

Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, 1992

The essence of the approach is to document with the stakeholders the “use cases” that capture the key interactions with the system to be built. Each use case is then broken down into “scenarios” that document the different ways in which the use may play out. These scenarios can then be simulated later to validate possible designs.

Note that this approach captures *how* the system will be used, rather than just listing the desired “features”, which expresses only *what* the system should do.

Jacobson joined Rational in 1995 and OOSE was incorporated into UML.

Use Cases and Viewpoints ...

Stakeholders represent different problem *viewpoints*.

- —Interview as many *different* kinds of stakeholders as possible/ necessary
- —Translate requirements into *use cases* or “stories” about the desired system involving a fixed set of actors (users and system objects)
- —For each use case, capture *both typical and exceptional* usage scenarios

Users tend to think about systems in terms of “features”.

- —You must get them to tell you *stories* involving those features.
- —Use cases and scenarios can tell you if the requirements are *complete and consistent*!

Stakeholders in the timetable system would be committee members, the Dean, the Dean's secretary, the system administrator, ...

Exceptional scenarios cover cases where information is missing, or errors arise: committee members are not already known to the system; an email address is missing; no suitable meeting date can be found …

"*Features*" are functionalities from the user perspective: I can login; set my profile; find the committees I am involved in; ask to schedule a meeting. Features need to be turned into use cases so that individual scenarios can be simulated. Only this way can you be gain confidence that all the requirements are covered.

Unified Modeling Language

UML is the industry standard for documenting OO models

Class Diagrams	visualize *logical structure* of system in terms of *classes, objects and relationships*
Use Case Diagrams	show external *actors and use cases* they participate in
Sequence Diagrams	visualize *temporal message ordering* of a *concrete scenario* of a use case
Collaboration (Communication) Diagrams	visualize *relationships* of objects exchanging messages in a *concrete scenario*
State Diagrams	specify the *abstract states* of an object and the *transitions* between the states

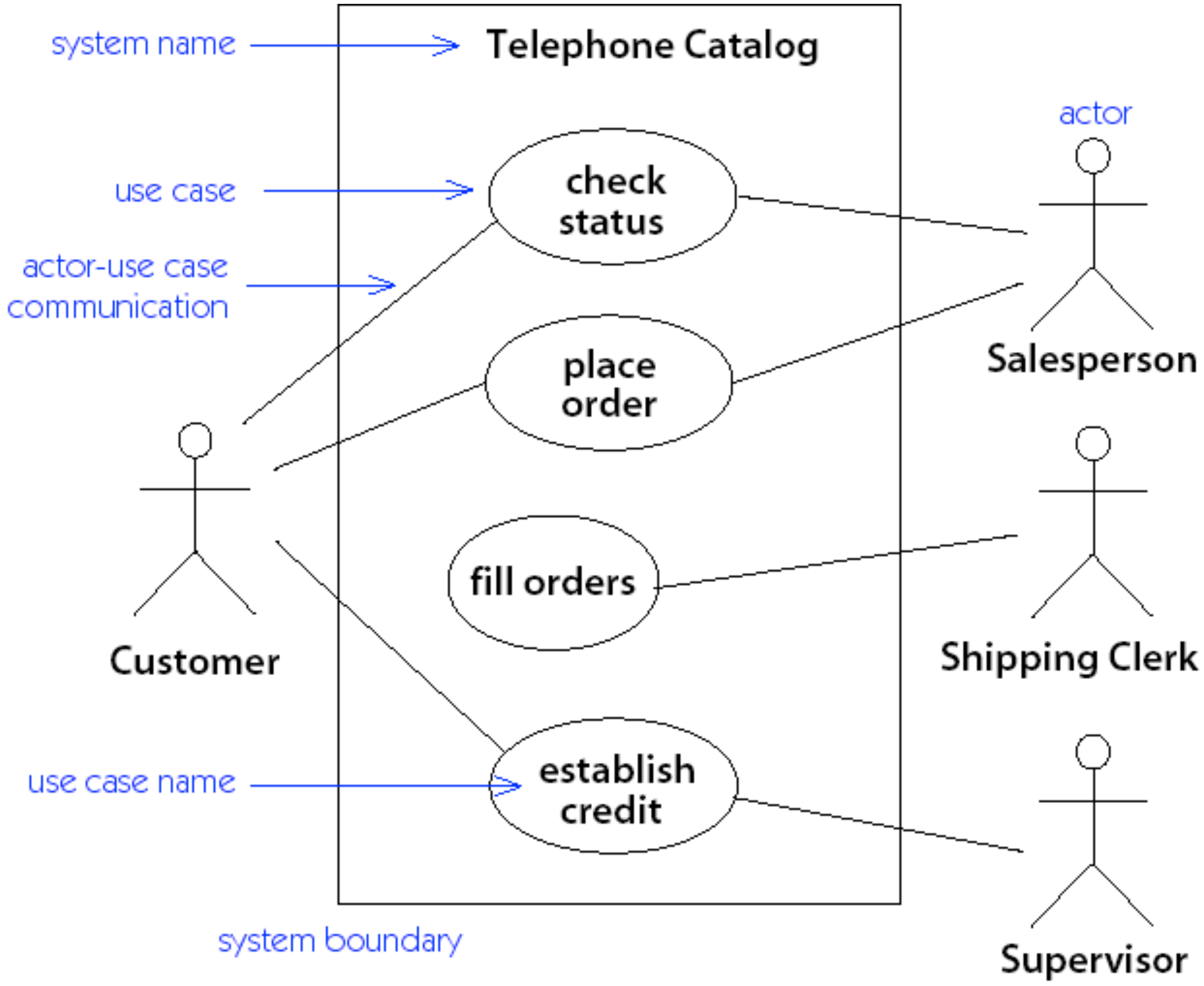
In the timetable project, class diagrams can be used both to model the actual domain, as well as to describe the design of the software system.

Use case diagrams describe at a high level the main activities supported by the system (defining a new committee, adding members to a committee, scheduling a meeting, etc.)

Sequence diagrams and Collaboration diagrams can be used to document a particular scenario of a use case (i.e., the concrete steps).

State diagrams are useful for describing the evolution of a process (for example, the lifecycle of a committee, or the steps taken to organize a particular meeting).

Use Case Diagrams



More on this later …

Figure 5-1. *Use case diagram*

Use cases are described in chapter 5 of the UML reference:

http://scgresources.unibe.ch/Literature/Books/Rumb99aUMLreference.pdf

NB: there are dedicated lectures on UML later in this course

Sequence Diagrams

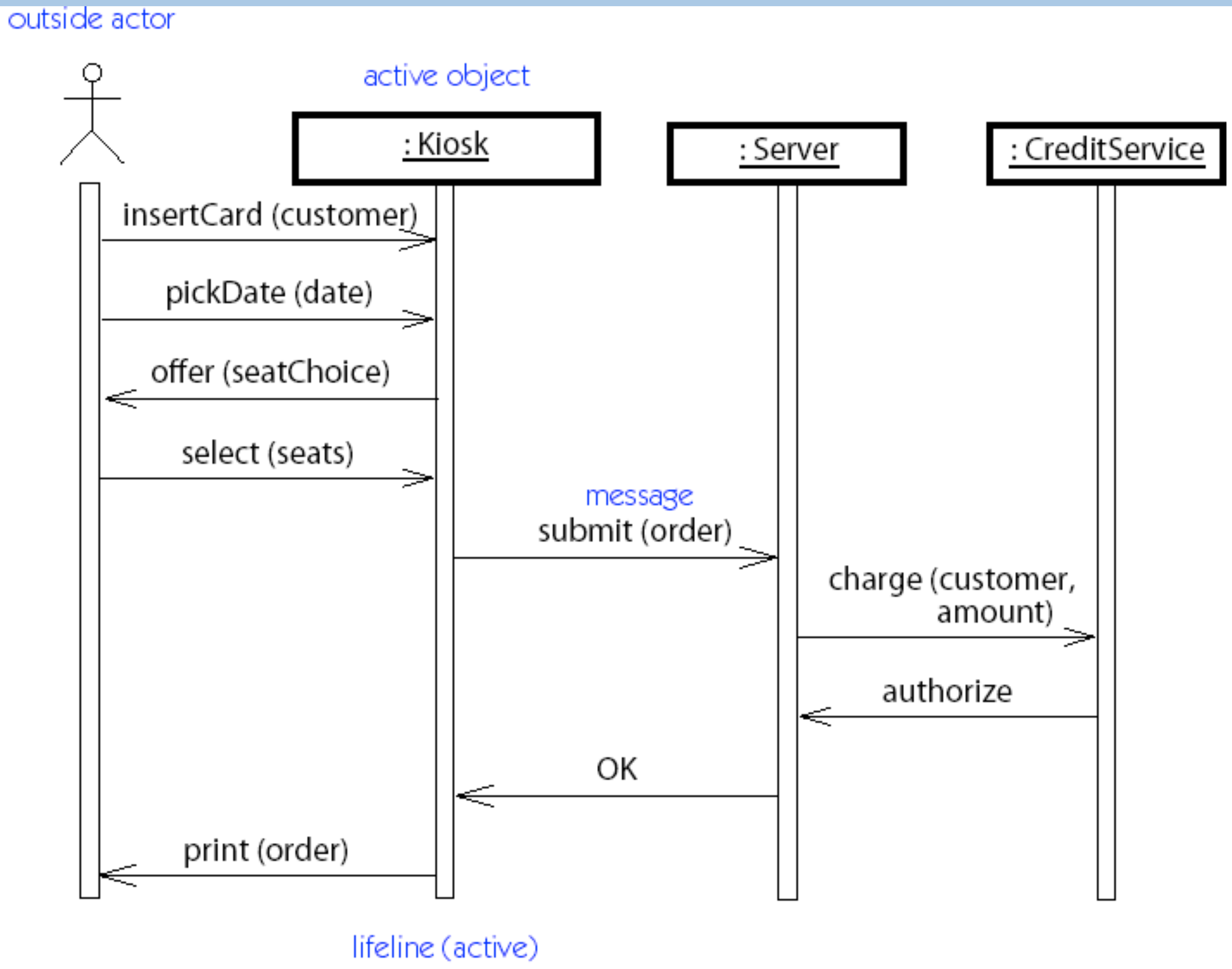


Figure 8-1. *Sequence diagram*

Writing Requirements Definitions

Requirements definitions usually consist of *natural language*, supplemented by (e.g., UML) *diagrams and tables*.

Three types of problems can arise:

- —**Lack of clarity**: It is hard to write documents that are both *precise and easy-to-read*.
- —**Requirements confusion**: *Functional and non-functional requirements* tend to be intertwined.
- —**Requirements amalgamation**: Several *different requirements* may be expressed together.

Roadmap

> The Requirements Engineering Process
> Use Cases
> **Functional and non-functional requirements**
> Evolutionary and throw-away prototyping
> Requirements checking and reviews

Functional and Non-functional Requirements

Functional requirements describe system *services* or *functions*

- Enter a new user into the timetable system
- Schedule a meeting

Functional requirements must be precise and unambiguous

Non-functional requirements are *constraints* on the system or the development process

- User data must remain confidential
- Finding a possible meeting date should be instantaneous (< 0.5s)

Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless!

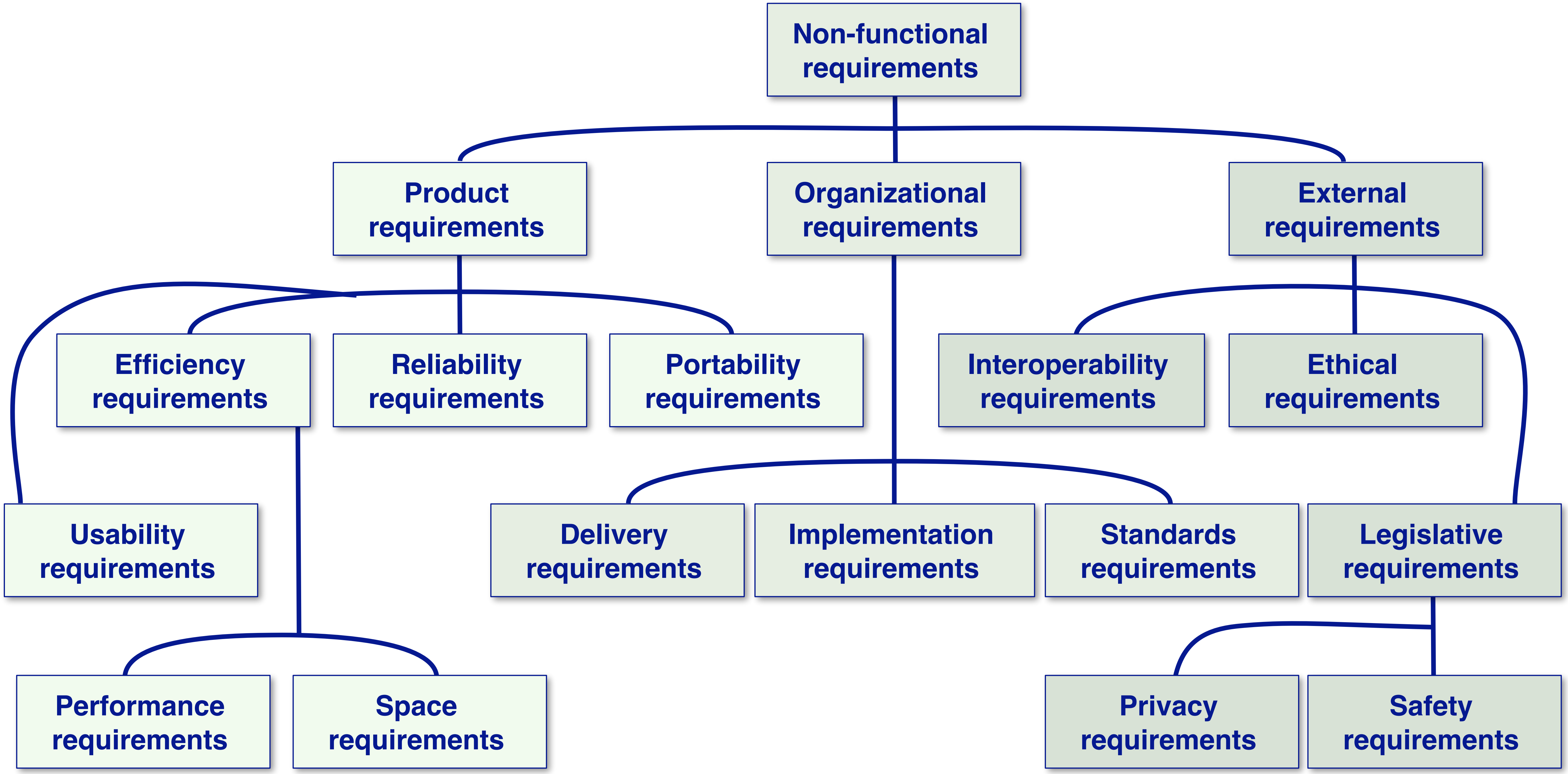
Functional requirements have to do with the *functions* computed by the system under construction. They can typically be tested using unit tests.

Non-functional requirements, on the other hand, express all the *constraints* that the system must fulfil. They may range from constraints like the platform that must be used, to run-time constraints like performance and throughput.

Non-functional Requirements

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

Types of Non-functional Requirements



Examples of Non-functional Requirements

Product requirement	The timetable system should run on all standard browsers, i.e., Safari, Firefox, Chrome, IE
Organisational requirement	Committees must be constituted according to University and Faculty regulations.
External requirement	User profile data is confidential and must confirm to Swiss privacy laws. Users have the right to audit their data and to request that their data be removed from the system.

Requirements Verifiability

Requirements must be written so that they can be *objectively verified*.

Imprecise:

- The timetable system should be *easy to use* and should be organised in such a way that *user errors are minimised.*
  - *Terms like "easy to use" and "errors shall be minimised" are useless as specifications.*

Verifiable:

- Users should be able to use the timetable system without having to read a user manual. Users should be able to enter their conflicts for a proposed meeting in under 2 minutes.

Verifiability is an important criterion that is typically checked in a formal review of the requirements document. Every single requirement must be expressed in such a way that it is possible to verify that the requirement is fulfilled or not.

The classic example of a poorly phrased requirement is that the software should be “easy to use”. Since “easy to use” is subjective, it is useless as a formal requirement.

Instead, the requirements should state explicit, measurable criteria, such as the amount of time it should take a new user to learn it, what the expected throughput of the system should be for a new or an experienced user, or the expected error rate.

Precise Requirements Measures (I)

Property	***Measure***
Speed	Time to define a new committee Time to enter one’s conflict Response time to identify potential meeting schedule Time to load the application
Size	Deployment size (MB) Source LOC Number of packages/classes
Ease of use	Training time Rate of errors made by trained users Number of help frames

Precise Requirements Measures (II)

Property	*Measure*
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Dependency on underlying platforms (which) Number of target systems

Roadmap

> The Requirements Engineering Process
> Use Cases
> Functional and non-functional requirements
> **Evolutionary and throw-away prototyping**
> Requirements checking and reviews

Prototyping Objectives

The objective of *throw-away prototyping* is to *validate or derive the system requirements*.

—Prototyping starts with that requirements that are *poorly understood*.

The objective of *evolutionary prototyping* is to deliver a *working system* to end-users.

—Development starts with the requirements that are *best understood*.

The purpose of all prototypes is to *reduce risk*.

An *evolutionary prototype* is simply a very *early version* of the software under development using an *iterative development lifecycle*. The risk that is reduced is that the wrong system will be developed. By delivering early demos as prototypes, dialogue with the stakeholders is maintained, which helps the development team focus on delivering maximum value to the customer.

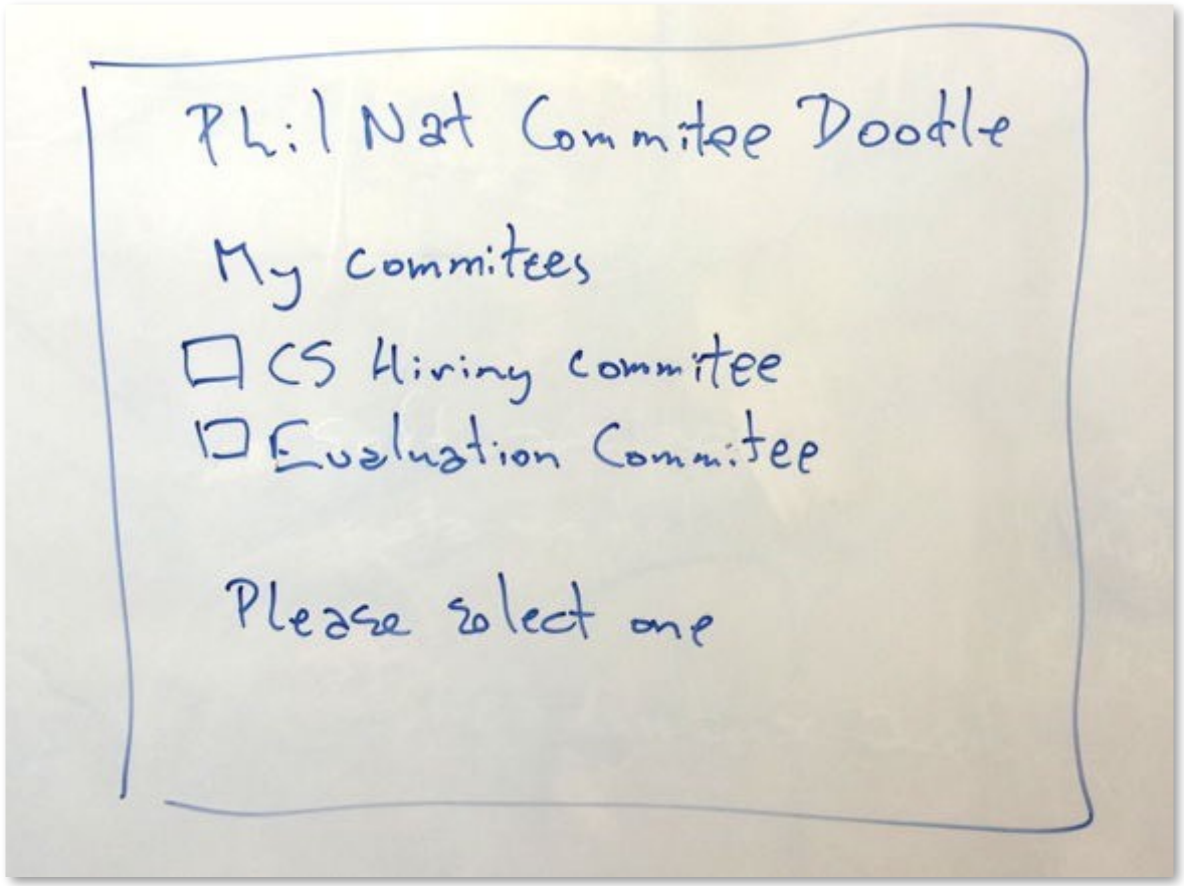
A *throwaway prototype* is not an early version of the software, but exists to *test out some idea* that is not well understood. The risk that is reduced is that the project will proceed under false assumptions. The prototype may test out any aspect of the project: user interface ideas, scenarios, feasibility of one technology or another. The prototype need not be running software. A good example is paper prototypes of the user interface.

Throw-away Prototyping

> Used to *reduce requirements risk*
  — The prototype is developed from an initial specification, delivered for experiment then discarded

> The throw-away prototype should not be considered as a final system
  — Various system characteristics may be left out
    – *A mockup of the timetable system may consist of static HTML pages*
  — There is no specification for long-term maintenance

Paper prototypes



Prototypes do not have to be executable or pretty!

A paper prototype is a cheap way to explore design options for a user interface. Each page can illustrate one step in the scenario of a use case.

A key advantage of paper prototypes is that no one can possible confuse them with the finished system! (Sometime very flashy software prototypes can give the false impression that development is much further along than it really is.)

Evolutionary Prototyping

> Can be seen as the *early versions of the system* under development in a spiral development lifecycle.

- Early prototypes help to clarify requirements
  - *Who should initiate the meeting schedule? Who approves it?*
- Can be used to explore design options
  - *How should the timetable be displayed?*

Roadmap

- The Requirements Engineering Process
- Use Cases
- Functional and non-functional requirements
- Evolutionary and throw-away prototyping
- **Requirements checking and reviews**

Requirements Checking

Validity	Does the system provide the functions *which best support* the customer's needs?
Consistency	Are there any *requirements conflicts*?
Completeness	Are *all functions* required by the customer included?
Realism	Can the requirements be implemented given *available budget and technology*?

Validity: do the stakeholders agree on the needs?

Consistency: do the Dean's office and the Committees agree on the procedure to schedule a meeting? Who selects the possible dates? Who decides amongst the potential dates?

Completeness: have all stakeholders been interviewed?

Realism: where will the system be deployed? Who will maintain it?

Requirements Reviews

> *Regular reviews* should be held while the requirements definition is being formulated

> Both *client and contractor* staff should be involved in reviews

> Reviews may be *formal* (with completed documents) or *informal*.

  — *Good communications* between developers, customers and users can resolve problems at an *early stage*

Review checks

Verifiability	Is the requirement realistically *testable*?
Comprehensibility	Is the requirement properly *understood*?
Traceability	Is the *origin* of the requirement clearly stated?
Adaptability	Can the requirement be *changed* without a large *impact* on other requirements?

Sample Requirements Review Checklist

> Does the (software) product have a *succinct name*, and a *clearly described purpose*?
> Are the *characteristics of users* and of *typical usage* mentioned? (No user categories missing.)
> Are all *external interfaces* of the software explicitly mentioned? (No interfaces missing.)
> Does each specific requirement have a *unique identifier* ?
> Is each requirement *atomic* and *simply formulated* ? (Typically a single sentence. Composite requirements must be split.)
> Are requirements organized into *coherent groups* ? (If necessary, hierarchical; not more than about ten per group.)
> Is each requirement *prioritized* ? (Is the meaning of the priority levels clear?)
> Are all *unstable requirements* marked as such? (TBC=`To Be Confirmed', TBD=`To Be Defined')

http://wwwis.win.tue.nl/2M390/rev_req.html

Sample Requirements Review Checklist

- Is each requirement *verifiable* (in a provisional acceptance test)? (Measurable: where possible, quantify; capacity, performance, accuracy)
- Are the requirements *consistent* ? (Non-conflicting.)
- Are the requirements sufficiently *precise* and *unambiguous* ? (Which interfaces are involved, who has the initiative, who supplies what data, no passive voice.)
- Are the requirements *complete*? Can everything not explicitly constrained indeed be viewed as developer freedom? Is a product that satisfies every requirement indeed acceptable? (No requirements missing.)
- Are the requirements *understandable* to those who will need to work with them later?
- Are the requirements *realizable* within budget?
- Do the requirements express actual *customer needs* (in the language of the problem domain), *rather than solutions* (in developer jargon)?

http://wwwis.win.tue.nl/2M390/rev_req.html

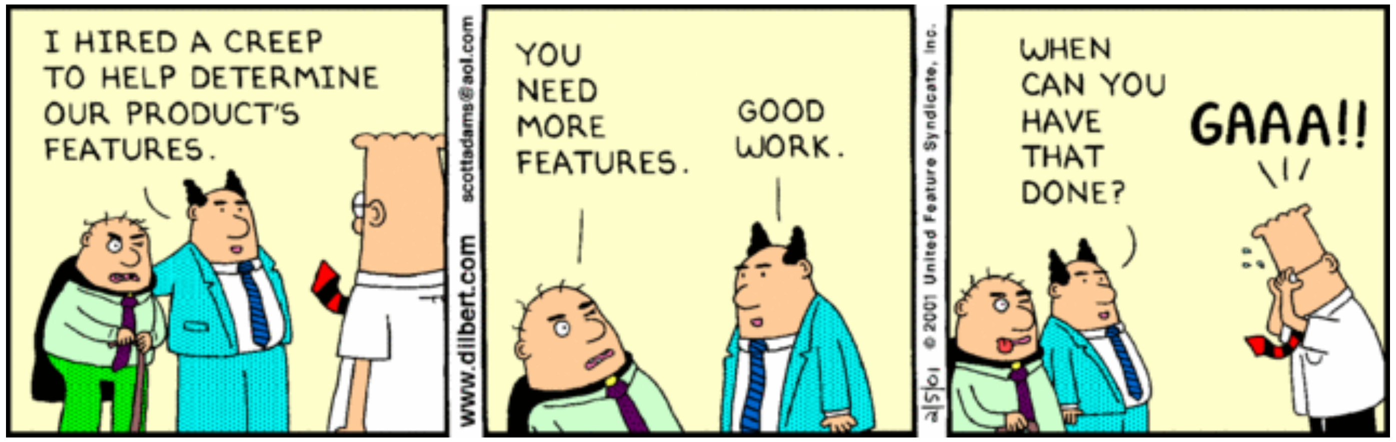
Traceability

To protect against changes you should be able to *trace back from every system component to the original requirement* that caused its presence

	C1	*C2*	…	…	*Cm*
req1		x			
req2	x				
…				x	
…					x
reqn		x	x		

A software process should help you keep this virtual table up-to-date

Simple techniques may be quite valuable (naming conventions, ...)


I HIRED A CREEP TO HELP DETERMINE OUR PRODUCT'S FEATURES.
YOU NEED MORE FEATURES.
GOOD WORK.
WHEN CAN YOU HAVE THAT DONE?
GAAA!!
www.dilbert.com
scottadams@aol.com
© 2001 United Feature Syndicate, Inc.

“*Feature creep*” refers to the uncontrolled addition of features to the system.

To avoid feature creep it is critical to have a clear business vision of what the system should achieve or optimize.

Traceability helps to keep feature creep under control by tracing new features to the actual requirements they should address.

In upcoming lectures we will see how agile methods address this issue by guiding both the development team and the customer towards features that bring the most *business value* to the customer.

What you should know!

- What is the difference between requirements analysis and specification?
- Why is it hard to define and specify requirements?
- What are use cases and scenarios?
- What is the difference between functional and non-functional requirements?
- What's wrong with a requirement that says a product should be "user-friendly"?
- What's the difference between evolutionary and throw-away prototyping?

Can you answer the following questions?

- Why isn’t it enough to specify requirements as a set of desired features?
- Which is better for specifying requirements: natural language or diagrams?
- How would you prototype a user interface for a web-based ordering system?
- Would it be an evolutionary or throw-away prototype?
- What would you expect to gain from the prototype?
- How would you check a requirement for “adaptability”?