Engineering

RW344: Software Design

Bernd Fischer
bfischer@cs.sun.ac.za
Requirements Engineering
Topics

- requirements
- requirements errors
- requirements specification
Requirements engineering

Main concern: **what should the system do?**

Main result: **software requirements specification**

== description of desired system functionality
Requirements engineering is difficult
“The hardest single part of building a software system is deciding precisely what to build. No other part [...] is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part [...] so cripples the resulting system if done wrong. No other part is more difficult to rectify later.”

F. P. Brooks
Requirements engineering is difficult

Experiment: formal inspection of requirements (10 pages) for centralized railroad traffic controller

- written by experienced project leader in domain
  - author believed that teams would find only few errors
- inspected by 10 teams of 4 software engineers

More information:
Requirements engineering is difficult

Experiment: formal inspection of requirements (10 pages) for centralized railroad traffic controller
• written by experienced project leader in domain
  – author believed that teams would find only few errors
• inspected by 10 teams of 4 software engineers
• **92 errors**, some very **serious**, were found!

More information:
Requirements engineering is difficult

Experiment: formal inspection of requirements (10 pages) for centralized railroad traffic controller
- written by experienced project leader in domain
  - author believed that teams would find only few errors
- inspected by 10 teams of 4 software engineers
- **92 errors**, some very **serious**, were found!
- each team found only **35.5 errors on average**!
  - i.e., it **missed 56.5** to be found downstream
- many errors were found by only one team!
- errors of greatest severity found by fewest teams!

More information:
Requirements engineering is difficult

- 80% of interface faults and 20% of implementation faults due to requirements [Perry & Stieg 1993]
- 85% of defects due to requirements
  - 49% due to incorrect assumptions
  - 29% due to omitted requirements
  - 13% due to inconsistent requirements [Young 2001]
Requirements engineering is difficult

- 80% of interface faults and 20% of implementation faults due to requirements [Perry&Stieg 1993]
- 85% of defects due to requirements
  - 49% due to incorrect assumptions
  - 29% due to omitted requirements
  - 13% due to inconsistent requirements [Young 2001]
- 1.9 faults per page of specifications, 0.9 faults per page of design, 0.3 faults per page of code! [JPL]
Requirements engineering is difficult

- 80% of interface faults and 20% of implementation faults due to requirements  [Perry&Stieg 1993]
- 85% of defects due to requirements
  - 49% due to incorrect assumptions
  - 29% due to omitted requirements
  - 13% due to inconsistent requirements  [Young 2001]
- 1.9 faults per page of specifications, 0.9 faults per page of design, 0.3 faults per page of code!  [JPL]
- 54% of all errors found after coding and unit testing
  - 83% due to requirements and design
  - 17% due to coding stage  [Boehm]
Requirements errors are common and stubborn.

So: requirements errors...

- ... cause most defects
- ... are relatively more common than code errors
- ... are found out late
Requirements errors are common, stubborn, and expensive.

Requirements errors are more expensive to fix later:

[Graph showing cost of fixing requirements errors at different stages (Requirements, Analysis, Design, Coding, Postdelivery).]

1x  5x  10x  50x

[Stecklein 2004]
Requirements errors are common, stubborn, and expensive.

Requirements errors are more expensive to fix later:

- fixes in requirements can be done in text

[Stecklein 2004]
Requirements errors are more expensive to fix later:

- fixes in requirements can be done in text
- other fixes require
  - localization
  - change of
    ▶ design
    ▶ code
    ▶ documentation
  - test of change itself
  - system regression test

[Stecklein 2004]
Requirements errors are common, stubborn, and expensive.

Requirements errors are more expensive to fix later:

- fixes in requirements can be done in text
- other fixes require
  - localization
  - change of
    - design
    - code
    - documentation
  - test of change itself
  - system regression test
- worst case: system re-design and re-development

[Stecklein 2004]
Requirements errors are more expensive to fix later:

- fixes in requirements can be done in text

- other fixes require
  - localization
  - change of
    - design
    - code
    - documentation
  - test of change itself
  - system regression test

- worst case: **system re-design and re-development**
Requirements errors are more expensive to fix later:

- fixes in requirements can be done in text
- other fixes require
  - localization
  - change of
    - design
    - code
    - documentation
  - test of change itself
  - system regression test
- worst case: system re-design and re-development
Requirements
Requirement
Requirement: a thing that is needed or wanted.

Oxford Dictionary
Requirement: a thing that is needed or wanted.

Oxford Dictionary

Requirement: a singular documented physical and functional need that a particular design, product or process must be able to perform.

Wikipedia
**Requirement**

*Requirement*: a **thing that is needed or wanted**.

---

*Requirement*: a **singular documented physical and functional need** that a particular design, product or process must be able to perform.

---

*Requirement*: a **condition or capability** that must be met or **possessed by a system** or system component **to satisfy a contract**, standard, specification, or other formally imposed document.

---

**Oxford Dictionary**

**Wikipedia**

**IEEE Standard Glossary of Software Engineering Terminology**
A requirement is a description of one service or constraint of the system that the user desires.

Requirements are...

• ... aimed at different audiences: user requirements vs system requirements
• ... aimed at different aspects of the system: functional vs non-functional vs domain
• ... described with different formalisms: natural language vs graphical vs formal methods
User requirements

• written primarily for system users who do not necessarily have detailed technical knowledge
  – managers & users (customer) – to define system
  – managers (contractor) – to plan system development
  – developers (contractor) – to specify system reqs.
User requirements

• written primarily for system users who do not necessarily have detailed technical knowledge
  – managers & users (customer) – to define system
  – managers (contractor) – to plan system development
  – developers (contractor) – to specify system reqs.
• often in natural language plus diagrams
User requirements

• written primarily for system users who do not necessarily have detailed technical knowledge
  – managers & users (customer) – to define system
  – managers (contractor) – to plan system development
  – developers (contractor) – to specify system reqs.
• often in natural language plus diagrams
• often complemented with rationale to help developers understand
  – the application domain
  – why the requirement is stated in its current form
  – possible effects of requirements changes
User requirements

• written primarily for system users who do not necessarily have detailed technical knowledge
  – managers & users (customer) – to define system
  – managers (contractor) – to plan system development
  – developers (contractor) – to specify system reqs.
• often in natural language plus diagrams
• often complemented with rationale to help developers understand
  – the application domain
  – why the requirement is stated in its current form
  – possible effects of requirements changes
• also called requirement definition
User requirements - example

1. The web-based library information system shall allow users to search books and audio-visual materials through a catalogue page.

Source:
1. The web-based library information system shall allow users to search books and audio-visual materials through a catalogue page.

Source:
1. The web-based library information system shall allow users to search books and audio-visual materials through a catalogue page.


Source:
1. The web-based library information system shall allow users to search books and audio-visual materials through a catalogue page.


which catalogue?

Is this a good requirement?

Source:
1. The web-based library information system shall allow users to search books and audio-visual materials through a catalogue page.

  → needs refinement

Source:
1. The library information system shall allow users to search the library catalogue for books and audio-visual materials.
1. The library information system shall allow users to search the library catalogue for books and audio-visual materials.

**Rationale:** *Allowing users to find books and AV-materials through the information systems saves both users’ and librarians’ time.*
1. The library information system shall allow users to search the library catalogue for books and audio-visual materials.

Rationale: *Allowing users to find books and AV-materials through the information systems saves both users’ and librarians’ time.*

1.1 The library information system shall allow users to search using any combination of the following fields:
1.1.1 author
1.1.2 title
1.1.3 publication year
## User requirements - example

1. The library information system shall allow users to search the library catalogue for books and audio-visual materials.

   *Rationale:* Allowing users to find books and AV-materials through the information systems saves both users’ and librarians’ time.

1.1 The library information system shall allow users to search using any combination of the following fields:

   1.1.1 author
   1.1.2 title
   1.1.3 publication year

1.2 The search shall return a book or audio-visual material if its library catalogue entry exactly matches each of the fields specified by the user (see 1.1).
System requirements

- written primarily for system developers as starting point for design
System requirements

• written primarily for system developers as starting point for design

• structured document setting out detailed, technical descriptions of the system services
  – adds detail to the user requirements
  – should be consistent with it
System requirements

• written primarily for system developers as starting point for design

• structured document setting out detailed, technical descriptions of the system services
  – adds detail to the user requirements
  – should be consistent with it

• sometimes also used as a contract between client and contractor
  – tension between precision and detail (for design) and openness (for contracting)
System requirements

• written primarily for system developers as starting point for design
• structured document setting out detailed, technical descriptions of the system services
  – adds detail to the user requirements
  – should be consistent with it
• sometimes also used as a contract between client and contractor
  – tension between precision and detail (for design) and openness (for contracting)
• also called requirement specification
User requirement

1. The software shall provide a means of representing and accessing external files created by other tools.
### User requirement

1. The software shall provide a means of representing and accessing external files created by other tools.

### System requirements
Example

User requirement

1. The software shall provide a means of representing and accessing external files created by other tools.

System requirements

1.1 The user should be provided with facilities to define the type of external files.
1.2 Each external file type may have an associated tool which may be applied to the file.
1.3 Each external file type may be represented as a specific icon on the user’s display.
1.4 Facilities should be provided for the icon representing an external file type of be defined by the user.

...
Functional requirements

A functional requirement defines a function of the system or one of its components.

- how the system should respond to particular inputs
- how the system should react in particular situations
- may also state what the system should not do
Functional requirements

A **functional requirement** defines a function of the system or one of its components

- how the system should respond to particular inputs
- how the system should react in particular situations
- may also state what the system should *not* do

Ideally: each line of code **traceable** to a functional requirement
Non-functional requirements

A non-functional requirement imposes constraints on the design or the implementation.

- often the “-ilities”
- typically cross-cutting or emergent properties:
  - affect whole system: where do you implement usability or reliability?
Non-functional requirements

A non-functional requirement imposes constraints on the design or the implementation.

- often the “-ilities”
- typically *cross-cutting* or *emergent properties*:
  - affect whole system: where do you implement usability or reliability?
- difficult to meet...
- ... but failure can render entire system useless
Non-functional requirements can also constrain

- **process**: how the system is developed
  - “The system shall be implemented in Java SE8.”
  - “The system development shall follow the rules in DO-178B.”

- **environment**: hardware, software, legal
  - “The system shall run on mobile devices with at least 16GB RAM that run Android KitKat 4.4.4.”
  - “The system shall not disclose to the operators of the system any personal information about customers apart from their name and reference number.”
Non-functional requirements

- Product requirements
  - Usability requirements
    - Performance requirements
  - Efficiency requirements
  - Reliability requirements
  - Portability requirements
- Organizational requirements
- External requirements
  - Interoperability requirements
  - Ethical requirements
  - Legislative requirements
    - Privacy requirements
    - Safety requirements
FURPS

- **Functionality**: capability, ...
- **Usability**: human factors, consistency, documentation, responsiveness, ...
- **Reliability**: availability, recoverability, predictability, accuracy, ...
- **Performance**: speed, efficiency, resource consumption, scalability, ...
- **Supportability**: testability, flexibility, localizability, ...

More information:
FURPS

- **Functionality**: capability, ...
- **Usability**: human factors, consistency, documentation, responsiveness, ...
- **Reliability**: availability, recoverability, predictability, accuracy, ...
- **Performance**: speed, efficiency, resource consumption, scalability, ...
- **Supportability**: testability, flexibility, localizability, ...

standardized way to organize (non-functional) requirements

More information:
Domain requirements

Application domains shape the system requirements:

• *functional*: complex computations or processes, ...
• *non-functional*: user expectations, standards, ...
Domain requirements

Application domains shape the system requirements:

• **functional**: complex computations or processes, ...
• **non-functional**: user expectations, standards, ...

Domain requirements are often the hardest:

• **understandability**: domain requirements are often expressed in domain-specific terms which are not understood by software engineers
• **implicitness**: domain experts often do not think to make the domain requirements explicit
  – they understand the area and requirements well
• **cross-cutting**
Requirements Elicitation
Requirements are in general not just “discovered” or “gathered”... but must be “elicited”!
Requirements elicitation techniques

Wide variety of approaches and techniques:

- **observational**: observe users of existing system
  - ethnographic methods
- **analytical**: analyze documents, forms, processes
  - domain analysis
- **inquisitive**: interview users
  - role playing
- **speculative**: suggest and evaluate functionality
  - brainstorming
  - rapid prototyping
Requirements elicitation techniques

Wide variety of approaches and techniques:

- **observational**: observe users of existing system
  - ethnographic methods
- **analytical**: analyze documents, forms, processes
  - domain analysis
- **inquisitive**: interview users
  - role playing
- **speculative**: suggest and evaluate functionality
  - brainstorming
  - rapid prototyping

Usually several methods must be applied.
Stakeholders and viewpoints

A **stakeholder** is somebody who **interacts with** or **is affected by** the system:

- different types of **users**
- **customer**
- **administrator**
- ...
A **stakeholder** is somebody who **interacts with or is affected by** the system:

- different types of **users**
- customer
- administrator
- ...

Stakeholders in computerized cash register?
Stakeholders and viewpoints

A **stakeholder** is somebody who interacts with or is affected by the system:

- different types of **users**
- **customer**
- **administrator**
- ...

Different stakeholders have different viewpoints on the system, and so different requirements!
Stakeholders and viewpoints

A _stakeholder_ is somebody who _interacts with_ or _is affected by_ the system:

- different types of _users_
- _customer_
- _administrator_
- ...

Different stakeholders have different viewpoints on the system, and so different requirements!

**Identifying the stakeholders is the first important step in requirements engineering!**
Example: meal ordering system

Notes from the first requirements elicitation meeting with the hospital staff

Attendees hospital: Head of hospital, Supervisor of ward 1, Chef of kitchen

Overall goal (according to Head of hospital): “We want a system that makes taking food orders of patients and getting them to the kitchen easier and more efficient.”

Particulars:

- Each patient needs to order for the next day food from a given menu.
- The menu is provided by the kitchen every day and needs to be provided to the individual wards so they can give it to their patients.
- All wards get the same menu.
- Patients choose their dishes for the next day from current menu.
- Each patient chooses three meals, one breakfast, one lunch and one dinner.
- For security reasons, an identification is needed for the staff to use the system.
- The system needs to save who did what with the system and also make sure that only authorized personal can work with the software.
- The Supervisor emphasizes that the system needs to be easy to use, since the nurses are not computer experts. The kitchen agrees.
- Management adds that it has to be cheap and that it is necessary that the staff is able to use the system quickly.
- What is the estimated size?
  - Hospital has 7 wards; each ward has one supervisor and a variable number of nurses.
  - Each nurse only works on one ward.
  - The patients can vary daily but the largest ward can only hold 100 patients
- Patients are only allowed to order food when they are still in the hospital the next day.
- Patients get discharged after breakfast.
- There needs to be a way to check the patients availability.
- A mobile device for taking orders is necessary so the nurses can take it along during the morning visits.
- Kitchen: the process for the menu is that the kitchen starts to make a new menu which is released after finishing the menu. Each menu should contain at least one dish for each meal. After release the menu is available for the ward manager and can be printed locally. The release of the menu needs to be before 08:00 so that the patients can still choose their meals for the next day.
- The kitchen cannot handle more than four different dishes for any meal.
- A patient can change his order for the next day. The new information is saved. Also information about the nurse who took this new order needs to be accounted for. All information needs to be saved who did what.
- If errors occur the system should inform the user what and why something went wrong.
- There is only one menu available for any given day.
- The kitchen receives a list of all orders at about 17:00 so that they can order according amounts of ingredients. They will prepare the food the next day.
- The dishes are then served the next day by the nurses of the ward.
- Orders should not be deleted as long as the patient is still in hospital, so when complains or similar things happen it can be looked into who has done what.
Example: meal ordering system

Notes from the first requirements elicitation meeting with the hospital staff

Attendees hospital: Head of hospital, Supervisor of ward 1, Chef of kitchen

Overall goal (according to Head of hospital): “We want a system that makes taking food orders of patients and getting them to the kitchen easier and more efficient.”

Particulars:
- Each patient needs to order for the next day food from a given menu.
- The menu is provided by the kitchen every day and needs to be provided to the individual wards so they can give it to their patients.
- All wards get the same menu.
- Patients choose their dishes for the next day from current menu.
- Each patient chooses three meals, one breakfast, one lunch and one dinner.
- For security reasons, an identification is needed for the staff to use the system.
- The system needs to save who did what with the system and also make sure that only authorized personal can work with the software.
- The Supervisor emphasizes that the system needs to be easy to use, since the nurses are not computer experts. The kitchen agrees.
- Management adds that it has to be cheap and that it is necessary that the staff is able to use the system quickly.
- What is the estimated size?
  - Hospital has 7 wards; each ward has one supervisor and a variable number of nurses. Each nurse only works on one ward.
  - The patients can vary daily but the largest ward can only hold 100 patients
- Patients are only allowed to order food when they are still in the hospital the next day.
- Patients get discharged after breakfast.
- There needs to be a way to check the patients availability.
- A mobile device for taking orders is necessary so the nurses can take it along during the morning visits.
- Kitchen: the process for the menu is that the kitchen starts to make a new menu which is released after finishing the menu. Each menu should contain at least one dish for each meal. After release the menu is available for the ward manager and can be printed locally. The release of the menu needs to be before 08:00 so that the patients can still choose their meals for the next day.
- The kitchen cannot handle more than four different dishes for any meal.
- A patient can change his order for the next day. The new information is saved. Also information about the nurse who took this new order needs to be accounted for. All information needs to be saved who did what.
- If errors occur the system should inform the user what and why something went wrong.
- There is only one menu available for any given day.
- The kitchen receives a list of all orders at about 17:00 so that they can order according amounts of ingredients. They will prepare the food the next day.
- The dishes are then served the next day by the nurses of the ward.
- Orders should not be deleted as long as the patient is still in hospital, so when complaints or similar things happen it can be looked into who has done what.

Identify the stakeholders!
Requirements Analysis
Use case analysis is a systematic approach to develop requirements, based on use cases:
Use case analysis is a **systematic** approach to develop requirements, based on **use cases**:

A **use case** is a generalized description of a **one high-level interaction** between the **system** and one or more stakeholders or **actors** in order to achieve a **user goal**.
Use case analysis

Use case analysis is a **systematic** approach to develop requirements, based on **use cases**:

A use case is a generalized description of one **high-level interaction** between the system and one or more stakeholders or **actors** in order to achieve a **user goal**.
Use case analysis is a **systematic** approach to develop requirements, based on use cases:

A **use case** is a generalized description of one high-level interaction between the **system** and one or more stakeholders or **actors** in order to achieve a **user goal**.

Examples:
- change order
- pay with credit card
- return a book on time
- return a book too late
Use case diagrams – UML notation
Use case diagrams – UML notation

actor

© uml-diagrams.org
Use case diagrams – UML notation

actor

use case
Use case diagrams – UML notation

- **system name**: Online Shopping
- **system boundary**: «Subsystem»
- **actor**: Registered Customer, Web Customer, New Customer
- **use case**: View Items, Make Purchase, Checkout, Client Register, «include»
Use case diagrams – UML notation

- **System name**
- **System boundary**
- **Actor**
- **Use case**

 Actors can be systems.
Use case diagrams – UML notation

- **system name**: Online Shopping
- **system boundary**: «Subsystem»
- **actor**: Registered Customer, New Customer, Web Customer
- **stereotype**: «Service»
- **inheritance**: «include»
- **use case**: View Items, Make Purchase, Checkout, Client Register
- **actors can be systems**: Identity Provider, Credit Payment Service, PayPal
Use case diagrams – UML notation

- **system name**
- **system boundary**
- **actor**
- **inheritance**
- **inheritance & inclusion**
- **stereotype**
- **actors can be systems**

The diagram illustrates the use case for Online Shopping with various actors, use cases, and stereotypes.
Use case analysis

Use cases can be used to derive initial requirements:

• identify **actors** as subset of the stakeholders
  – not all stakeholders actually interact with the system
• identify all ways each actor **uses** the system
• identify similar uses by different actors
  – restructure use cases by generalization, extension, inclusion
• define **system boundary**
Example: meal ordering system

Notes from the first requirements elicitation meeting with the hospital staff

Attendees hospital: Head of hospital, Supervisor of ward 1, Chef of kitchen

Overall goal (according to Head of hospital): “We want a system that makes taking food orders of patients and getting them to the kitchen easier and more efficient.”

Particulars:

- Each patient needs to order for the next day food from a given menu.
- The menu is provided by the kitchen every day and needs to be provided to the individual wards so they can give it to their patients.
- All wards get the same menu.
- Patients choose their dishes for the next day from current menu.
- Each patient chooses three meals, one breakfast, one lunch and one dinner.
- For security reasons, an identification is needed for the staff to use the system.
- The system needs to save who did what with the system and also make sure that only authorized personal can work with the software.
- The Supervisor emphasizes that the system needs to be easy to use, since the nurses are not computer experts. The kitchen agrees.
- Management adds that it has to be cheap and that it is necessary that the staff is able to use the system quickly.
- What is the estimated size?
  - Hospital has 7 wards; each ward has one supervisor and a variable number of nurses.
    - Each nurse only works on one ward.
  - The patients can vary daily but the largest ward can only hold 100 patients
- Patients are only allowed to order food when they are still in the hospital the next day.
- Patients get discharged after breakfast.
- There needs to be a way to check the patients availability.
- A mobile device for taking orders is necessary so the nurses can take it along during the morning visits.
- Kitchen: the process for the menu is that the kitchen starts to make a new menu which is released after finishing the menu. Each menu should contain at least one dish for each meal. After release the menu is available for the ward manager and can be printed locally. The release of the menu needs to be before 08:00 so that the patients can still choose their meals for the next day.
- The kitchen cannot handle more than four different dishes for any meal.
- A patient can change his order for the next day. The new information is saved. Also information about the nurse who took this new order needs to be accounted for. All information needs to be saved who did what.
- If errors occur the system should inform the user what and why something went wrong.
- There is only one menu available for any given day.
- The kitchen receives a list of all orders at about 17:00 so that they can order according amounts of ingredients. They will prepare the food the next day.
- The dishes are then served the next day by the nurses of the ward.
- Orders should not be deleted as long as the patient is still in hospital, so when complains or similar things happen it can be looked into who has done what
Example: meal ordering system

Notes from the first requirements elicitation meeting with the hospital staff

Attendees hospital: Head of hospital, Supervisor of ward 1, Chef of kitchen

Overall goal (according to Head of hospital): “We want a system that makes taking food orders of patients and getting them to the kitchen easier and more efficient.”

Particulars:
- Each patient needs to order for the next day food from a given menu.
- The menu is provided by the kitchen every day and needs to be provided to the individual wards so they can give it to their patients.
- All wards get the same menu.
- Patients choose their dishes for the next day from current menu.
- Each patient chooses three meals, one breakfast, one lunch and one dinner.
- For security reasons, an identification is needed for the staff to use the system.
- The system needs to save who did what with the system and also make sure that only authorized personal can work with the software.
- The Supervisor emphasizes that the system needs to be easy to use, since the nurses are not computer experts. The kitchen agrees.
- Management adds that it has to be cheap and that it is necessary that the staff is able to use the system quickly.
- What is the estimated size?
  - Hospital has 7 wards; each ward has one supervisor and a variable number of nurses. Each nurse only works on one ward.
  - The patients can vary daily but the largest ward can only hold 100 patients
- Patients are only allowed to order food when they are still in the hospital the next day.
- Patients get discharged after breakfast.
- There needs to be a way to check the patients availability.
- A mobile device for taking orders is necessary so the nurses can take it along during the morning visits.
- Kitchen: the process for the menu is that the kitchen starts to make a new menu which is released after finishing the menu. Each menu should contain at least one dish for each meal. After release the menu is available for the ward manager and can be printed locally. The release of the menu needs to be before 08:00 so that the patients can still choose their meals for the next day.
- The kitchen cannot handle more than four different dishes for any meal.
- A patient can change his order for the next day. The new information is saved. Also information about the nurse who took this new order needs to be accounted for. All information needs to be saved who did what.
- If errors occur the system should inform the user what and why something went wrong.
- There is only one menu available for any given day.
- The kitchen receives a list of all orders at about 17:00 so that they can order according amounts of ingredients. They will prepare the food the next day.
- The dishes are then served the next day by the nurses of the ward.
- Orders should not be deleted as long as the patient is still in hospital, so when complains or similar things happen it can be looked into who has done what

Identify the actors!
Example: meal ordering system

Notes from the first requirements elicitation meeting with the hospital staff

Attendees hospital: Head of hospital, Supervisor of ward 1, Chef of kitchen

Overall goal (according to Head of hospital): “We want a system that makes taking food orders of patients and getting them to the kitchen easier and more efficient.”

Particulars:
- Each patient needs to order for the next day food from a given menu.
- The menu is provided by the kitchen every day and needs to be provided to the individual wards so they can give it to their patients.
- All wards get the same menu.
- Patients choose their dishes for the next day from current menu.
- Each patient chooses three meals, one breakfast, one lunch and one dinner.
- For security reasons, an identification is needed for the staff to use the system.
- The system needs to save who did what with the system and also make sure that only authorized personal can work with the software.
- The Supervisor emphasizes that the system needs to be easy to use, since the nurses are not computer experts. The kitchen agrees.
- Management adds that it has to be cheap and that it is necessary that the staff is able to use the system quickly.
- What is the estimated size?
  - Hospital has 7 wards; each ward has one supervisor and a variable number of nurses. Each nurse only works on one ward.
  - Patients can vary daily but the largest ward can only hold 100 patients.
- Patients are only allowed to order food when they are still in the hospital the next day.
- Patients get discharged after breakfast.
- There needs to be a way to check the patients availability.
- A mobile device for taking orders is necessary so the nurses can take it along during the morning visits.
- Kitchen: the process for the menu is that the kitchen starts to make a new menu which is released after finishing the menu. Each menu should contain at least one dish for each meal. After release the menu is available for the ward manager and can be printed locally. The release of the menu needs to be before 08:00 so that the patients can still choose their meals for the next day.
- The kitchen cannot handle more than four different dishes for any meal.
- A patient can change his order for the next day. The new information is saved. Also information about the nurse who took this new order needs to be accounted for. All information needs to be saved who did what.
- If errors occur the system should inform the user what and why something went wrong.
- There is only one menu available for any given day.
- The kitchen receives a list of all orders at about 17:00 so that they can order according amounts of ingredients. They will prepare the food the next day.
- The dishes are then served the next day by the nurses of the ward.
- Orders should not be deleted as long as the patient is still in hospital, so when complains or similar things happen it can be looked into who has done what.
Adding detail to use cases

Short or full *descriptions* complete use case diagrams:

- **name**: give a short, descriptive name to the use case
- **actors**: list the actors who can perform this use case
- **goals**: explain what the actor is trying to achieve
- **summary**: give a short informal description
- **preconditions**: state of the system before the use case
- **postconditions**: state of the system after completion
- **related use cases**: list generalizations, extensions, ...
- **steps**: describe each step using a 2-column format
  - actor’s actions and system’s responses
  - focus on interaction, not computation
Use case: Open file by name

• **actors:** user
• **steps:**

  *actor*                    *system*
  1. choose “Open...” command  2. display “File open” dialog
  3. type file name
  4. confirm selection
  5. remove dialog from display
Use case: Open file by name

- **actors**: user
- **steps**:
  1. choose “Open...” command
  2. display “File open” dialog
  3. type file name
  4. confirm selection
  5. remove dialog from display

formulate so that different alternatives are allowed
Use case: Exit car park, paying cash

- **actors**: car driver
- **goals**: to leave the car park after having paid the amount due in cash
- **summary**: when the car driver wants to leave, s/he must drive the car to the exit barrier and interact with the machine to pay the amount due and leave
- **preconditions**: the car driver must have entered the car park with a car and must have picked up a valid ticket on entry
- **postconditions**: ticket is invalid
- **related use cases**: Exit car park, paying with debit card
Use case: Exit car park, paying cash

- **Steps:**
  - **Actor**
    1. drive car to exit barrier, triggering a sensor
    4. insert ticket
    6. insert money
  - **System**
    2. detect presence of a car
    3. prompt driver to insert ticket
    5. display amount due
    7. return change (if any due)
    8. prompt driver to take change (if any due)
    9. raise barrier
  - **Actor**
    10. drive car through barrier, triggering a sensor
  - **System**
    11. lower barrier
Use case: Check out a book

- **actors:**
- **goals:**
- **summary:**
- **preconditions:**
- **postconditions:**
- **related use cases:**
- **steps:**

actor system
Structuring use cases

For larger systems uses cases should be structured:

- **generalization**: used to represent *several similar* use cases by a single case
  - specializations provide details of the original cases
  - cf. superclasses in class diagrams

- **extension**: used to make *optional* and *exceptional* cases explicit
  - keep the description of the basic use case simple

- **inclusion**: used to factor out *commonality* between different use cases
  - cf. method calls: avoid repeating details, abstraction
  - Examples: interaction validation, user authorization, ...
Structuring use cases – example

Source:
Example: meal ordering system

- Nurse
  - Take orders
  - Change orders

- Kitchen Staff
  - Identification
  - Receive orders
  - Create menu
  - Release menu

- Supervisor
Example: meal ordering system

Restructure (using inclusion)?
Requirements Specification
Requirements specifications should be …
Requirements for requirements

Requirements specifications should be …

- **correct**: each requirement is free from faults
Requirements for requirements

Requirements specifications should be ...

• **correct**: each requirement is free from faults
• **consistent**: no requirement conflicts with any other
Requirements specifications should be …

- **correct**: each requirement is free from faults
- **consistent**: no requirement conflicts with any other
- **precise, unambiguous, and clear**: each requirement has a single, easy-to-understand meaning
- **free of unwarranted design detail**: the specification does not prescribe the implementation
Requirements for requirements

Requirements specifications should be …
Requirements for requirements

Requirements specifications should be ...

- **relevant**: each requirement is pertinent to the problem and its solution
- **traceable**: each requirement can be traced to its origin in the problem environment
- **complete**: the requirements cover all aspects of the user function.
- **prioritized**: each requirement’s importance is given
Requirements for requirements

Requirements specifications should be ...

- **relevant**: each requirement is pertinent to the problem and its solution
- **traceable**: each requirement can be traced to its origin in the problem environment
- **complete**: the requirements cover all aspects of the user function.
- **prioritized**: each requirement’s importance is given
- **feasible**: each requirement can be realized with the available resources, and within the given constraints
- **testable**: for each requirement it is possible to determine whether it has been satisfied
Requirements specification methods

Requirements can be recorded in different ways:

- **natural language**
- **structured natural language**
  - standard forms or templates
- **requirements specification languages**
  - “programming-like” languages with abstract features
  - decision tables
- **graphical notations** (with text annotations)
  - SADT, DFD, ...
  - use cases
- **mathematical specification languages**
  - VDM, Z, B, Event-B, ...
  - automata-, logic-, or set-based
Requirements specification methods

Requirements can be recorded in different ways:

- **natural language**
- **structured natural language**
  - standard forms or templates
- **requirements specification languages**
  - “programming-like” languages with abstract features
  - decision tables
- **graphical notations** (with text annotations)
  - SADT, DFD, ...
  - use cases
- **mathematical specification languages**
  - VDM, Z, B, Event-B, ...
  - automata-, logic-, or set-based
Requirements specification methods

Requirements can be recorded in different ways:

- **natural language**
- **structured natural language**
  - standard forms or templates
- **requirements specification languages**
  - “programming-like” languages with abstract features
  - decision tables
- **graphical notations** (with text annotations)
  - SADT, DFD, ...
  - **use cases**
- **mathematical specification languages**
  - VDM, Z, B, Event-B, ...
  - automata-, logic-, or set-based
Requirements specification methods

Requirements can be recorded in different ways:

- **natural language**
- **structured natural language**
  - standard forms or templates
- **requirements specification languages**
  - “programming-like” languages with abstract features
  - decision tables
- **graphical notations** (with text annotations)
  - SADT, DFD, ...
  - use cases
- **mathematical specification languages**
  - VDM, Z, B, Event-B, ...
  - automata-, logic-, or set-based

Most common methods:

1970’s and 1980’s

Used mostly in safety-critical domains
Natural language requirements

Advantages:
Natural language requirements

Advantages:

• easy to understand
  – everybody can read...

• easy to implement
  – no specific tools required, Word or LaTeX will do

• flexible
  – no constraints on what can be expressed
Natural language requirements

Advantages:

• easy to understand
  – everybody can read...

• easy to implement
  – no specific tools required, Word or LaTeX will do

• flexible
  – no constraints on what can be expressed

Example

1. The library information system shall allow users to search the library catalogue for books and audio-visual materials.
Natural language requirements

Disadvantages
Natural language requirements

Disadvantages

• ambiguity and lack of clarity
  – synonyms
  – readers must interpret words the same way
  – precision requires longwinded prose

• over-flexible
  – not enough structure to enforce uniform style

• amalgamation and confusion
  – several requirements may be expressed together
  – functional and non-functional requirements mixed-up

• lack of modularisation
2.16.3.f While acting as the bus controller, the C&C MDM CSCI shall set the e, c, w, indicators identified in Table 3.2.16-II for the corresponding RT to “failed” and set the failure status to failed for all RT’s on the bus upon detection of transaction errors of selected messages to RTs whose 1553 FDIR is not inhibited in two consecutive processing frames within 100 milliseconds of detection of the second transaction error if a backup BC is available, the BC has been switched in the last 20 seconds, the SPD card reset capability is inhibited, or the SPD card has been reset in the last 10 major (10 second) frames, and either

1. the transaction errors are from multiple RT’s, the current channel has been reset within the last major frame, or
2. the transaction errors are from multiple RT’s, the bus channel’s reset capability is inhibited, and the current channel has not been reset within the last major frame.
2.16.3.f While acting as the bus controller, the C&C MDM CSCI shall set the e, c, w, indicators identified in Table 3.2.16-II for the corresponding RT to “failed” and set the failure status to failed for all RT’s on the bus upon detection of transaction errors of selected messages to RTs whose 1553 FDIR is not inhibited in two consecutive processing frames within 100 milliseconds of detection of the second transaction error if a backup BC is available, the BC has been switched in the last 20 seconds, the SPD card reset capability is inhibited, or the SPD card has been reset in the last 10 major (10 second) frames, and either

1. the transaction errors are from multiple RT’s, the current channel has been reset within the last major frame, or
2. the transaction errors are from multiple RT’s, the bus channel’s reset capability is inhibited, and the current channel has not been reset within the last major frame.
Structured Natural Language

Idea: **limit flexibility** of natural language with **forms** or **templates** in order to **increase uniformity** → easier to read and to check completeness
Structured Natural Language

Idea: limit flexibility of natural language with forms or templates in order to increase uniformity
→ easier to read and to check completeness

Typical format (cf. use cases):
• definition of the function or entity
• description of inputs and where they come from
• description of outputs and where they go to
• other entities used
• description of the action to be taken
• pre- and post-conditions (if appropriate)
• side effects (if any)
<table>
<thead>
<tr>
<th>Function</th>
<th>Search books by “Author”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Search books written by the author specified by the users.</td>
</tr>
<tr>
<td>Inputs</td>
<td>Format: LASTNAME, FIRST NAME</td>
</tr>
<tr>
<td></td>
<td>or LASTNAME, INITIAL</td>
</tr>
<tr>
<td>Source</td>
<td>Text string entered by users.</td>
</tr>
<tr>
<td>Outputs</td>
<td>Display a page containing all books written by specified author or “none” if no book is found.</td>
</tr>
<tr>
<td>Destination</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Requires</td>
<td>Central database storing books’ information.</td>
</tr>
<tr>
<td>Pre-condition</td>
<td>None</td>
</tr>
<tr>
<td>Post-condition</td>
<td>None</td>
</tr>
<tr>
<td>Side-effects</td>
<td>None</td>
</tr>
</tbody>
</table>
## Structured Natural Language

<table>
<thead>
<tr>
<th>Function</th>
<th>Add node</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Adds a node to an existing design. The user selects the type of node, and its position. When added to the design, the node becomes the current selection. The user chooses the node position by moving the cursor to the area where the node is added.</td>
</tr>
<tr>
<td><strong>Inputs</strong></td>
<td>Node type, Node position, Design identifier.</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>Node type and Node position are input by the user, Design identifier from the database.</td>
</tr>
<tr>
<td><strong>Outputs</strong></td>
<td>Design identifier.</td>
</tr>
<tr>
<td><strong>Destination</strong></td>
<td>The design database. Design is committed to database on completion of the operation.</td>
</tr>
<tr>
<td><strong>Requires</strong></td>
<td>Design graph rooted at input design identifier.</td>
</tr>
<tr>
<td><strong>Pre-condition</strong></td>
<td>Design is open and displayed on user’s screen.</td>
</tr>
<tr>
<td><strong>Post-condition</strong></td>
<td>Design is unchanged apart from the addition of a node of the specified type at the given position.</td>
</tr>
<tr>
<td><strong>Side-effects</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
Alternative requirements template

**Number:** a unique requirements number

**Use Case:** reference to use case using this requirement

**Introduction:** what is the requirement about?

**Rationale:** why is the requirement here?

**Source:** who came up with the requirement?

**Author:** who wrote it down?

**Inputs:** what comes in?

**Outputs:** what comes out?

**Required function:** what is the requirement?

**Related:** what other requirements are related?

**Conflicts:** requirements in conflict with this one

**Support Material:** documents, figures, tables, etc.

**Test Cases:** how do we test the requirement?

**Date:** when was the requirement modified last?
Tabular specifications

- condition/action tables

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sugar level falling ($r_2 &lt; r_1$)</td>
<td>$\text{CompDose} = 0$</td>
</tr>
<tr>
<td>Sugar level stable ($r_2 = r_1$)</td>
<td>$\text{CompDose} = 0$</td>
</tr>
<tr>
<td>Sugar level increasing and rate of increase decreasing</td>
<td>$\text{CompDose} = 0$</td>
</tr>
<tr>
<td>($</td>
<td>r_2 - r_1</td>
</tr>
<tr>
<td>Sugar level increasing and rate of increase stable or increasing</td>
<td>$\text{CompDose} =$</td>
</tr>
<tr>
<td>or increasing ($</td>
<td>r_2 - r_1</td>
</tr>
<tr>
<td></td>
<td>If rounded result = 0 then</td>
</tr>
<tr>
<td></td>
<td>$\text{CompDose} = \text{MinimumDose}$</td>
</tr>
</tbody>
</table>

- useful for systems with a number of possible alternative courses of action
  - particularly control systems
Example: meal ordering system

Notes from the first requirements elicitation meeting with the hospital staff

Attendees hospital: Head of hospital, Supervisor of ward 1, Chef of kitchen

Overall goal (according to Head of hospital): "We want a system that makes taking food orders of patients and getting them to the kitchen easier and more efficient."

Particulars:
- Each patient needs to order for the next day food from a given menu.
- The menu is provided by the kitchen every day and needs to be provided to the individual wards so they can give it to their patients.
- All wards get the same menu.
- Patients choose their dishes for the next day from current menu.
- Each patient chooses three meals, one breakfast, one lunch and one dinner.
- For security reasons, an identification is needed for the staff to use the system.
- The system needs to save who did what with the system and also make sure that only authorized personal can work with the software.
- The Supervisor emphasizes that the system needs to be easy to use, since the nurses are not computer experts. The kitchen agrees.
- Management adds that it has to be cheap and that it is necessary that the staff is able to use the system quickly.
- What is the estimated size?
  - Hospital has 7 wards; each ward has one supervisor and a variable number of nurses.
    - Each nurse only works on one ward.
  - The patients can vary daily but the largest ward can only hold 100 patients
- Patients are only allowed to order food when they are still in the hospital the next day.
- Patients get discharged after breakfast.
- There needs to be a way to check the patients availability.
- A mobile device for taking orders is necessary so the nurses can take it along during the morning visits.
- Kitchen: the process for the menu is that the kitchen starts to make a new menu which is released after finishing the menu. Each menu should contain at least one dish for each meal. After release the menu is available for the ward manager and can be printed locally. The release of the menu needs to be before 08:00 so that the patients can still choose their meals for the next day.
- The kitchen cannot handle more than four different dishes for any meal.
- A patient can change his order for the next day. The new information is saved. Also information about the nurse who took this new order needs to be accounted for. All information needs to be saved who did what.
- If errors occur the system should inform the user what and why something went wrong.
- There is only one menu available for any given day.
- The kitchen receives a list of all orders at about 17:00 so that they can order according amounts of ingredients. They will prepare the food the next day.
- The dishes are then served the next day by the nurses of the ward.
- Orders should not be deleted as long as the patient is still in hospital, so when complains or similar things happen it can be looked into who has done what
Writing Good Requirements
Bad requirements

Many projects suffer from...

• poorly **structured** requirements documents
• poorly **written** individual requirements
• poorly validated or **untestable** requirements
• poor **customer involvement**
• poor **prioritization**
  – analysis paralysis, scope creep, gold plating
• poor **change management**
  – entropic decay
Templates are Essential

• define **standard requirements templates**
  – easy to find information
  – nothing will be “forgotten”

• define a **standard document structure**
  – readers familiar with the document
  – acts as a checklist so that no sections are forgotten
1 Introduction
  1.1 Purpose
  1.2 Scope
  1.3 Definitions, acronyms, and abbreviations
  1.4 References
  1.5 Overview

2 Overall Description
  2.1 Product perspective
  2.2 Product functions
  2.3 User characteristics
  2.4 Constraints
  2.5 Assumptions and Dependencies

3 Specific Requirements
Appendixes
Glossary
Index
IEEE requirements document template

1 Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Definitions, acronyms, and abbreviations
   1.4 References
   1.5 Overview

2 Overall Description
   2.1 Product perspective
   2.2 Product functions
   2.3 User characteristics
   2.4 Constraints
   2.5 Assumptions and Dependencies

3 Specific Requirements

Appendixes
Glossary
Index
1 Introduction
  1.1 Purpose
  1.2 Scope
  1.3 Definitions, acronyms, and abbreviations
  1.4 References
  1.5 Overview

2 Overall Description
  2.1 Product perspective
  2.2 Product functions
  2.3 User characteristics
  2.4 Constraints
  2.5 Assumptions and Dependencies

3 Specific Requirements

Appendixes
Glossary
Index
1 Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Definitions, acronyms, and abbreviations
   1.4 References
   1.5 Overview

2 Overall Description
   2.1 Product perspective
   2.2 Product functions
   2.3 User characteristics
   2.4 Constraints
   2.5 Assumptions and Dependencies

3 Specific Requirements

Appendixes
Glossary
Index
1 Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Definitions, acronyms, and abbreviations
   1.4 References
   1.5 Overview

2 Overall Description
   2.1 Product perspective
   2.2 Product functions
   2.3 User characteristics
   2.4 Constraints
   2.5 Assumptions and Dependencies

3 Specific Requirements

Appendixes
Glossary
Index
1 Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Definitions, acronyms, and abbreviations
   1.4 References
   1.5 Overview

2 Overall Description
   2.1 Product perspective
   2.2 Product functions
   2.3 User characteristics
   2.4 Constraints
   2.5 Assumptions and Dependencies

3 Specific Requirements

Appendixes
Glossary
Index

Identifies the product and application domain.

Describes the external interfaces: system, user, HW, SW; site adaptation and HW constraints.

Summary of major functions.

Anything that limits the designer’s options.

All requirements go in here; main part of the document – different organizational styles.
Specific Requirements

How do we organize Section 3 (Specific requirements)?

• **depends** on the system under consideration
• organized to provide
  – clarity
  – changeability
  – etc.
• 8 examples suggested in IEEE Standard
3 Specific Requirements

3.1 External interface requirements
   3.1.1 User interfaces
   3.1.2 Hardware interfaces
   3.1.3 Software interfaces
   3.1.4 Communications interfaces

3.2 Functional requirements

3.3 Performance requirements

3.4 Design constraints

3.5 Software system attributes

3.6 Other requirements
3 Specific Requirements

3.1 External interface requirements
   3.1.1 User interfaces
   3.1.2 Hardware interfaces
   3.1.3 Software interfaces
   3.1.4 Communications interfaces

3.2 Functional requirements
   3.2.1 User class 1
      3.2.1.1 Requirement 1.1
      3.2.1.2 Requirement 1.2
      3.2.1.n Requirement 1.n
   3.2.2 User class 2
      3.2.2.1 Requirement 2.1
      3.2.2.2 Requirement 2.2
      3.2.2.m Requirement 2.m
   etc.

3.3 Performance requirements
3.4 Design constraints
3.5 Software system attributes
3.6 Other requirements
Specific Requirements by Mode

3 Specific Requirements

3.1 External interface requirements
   3.1.1 User interfaces
   3.1.2 Hardware interfaces
   3.1.3 Software interfaces
   3.1.4 Communication interfaces

3.2 Functional requirements
   3.2.1 Mode 1
      3.2.1.1 Requirement 1.1
      3.2.1.2 Requirement 1.2
      3.2.1.n Requirement 1.n
   3.2.2 Mode 2
      3.2.2.1 Requirement 2.1
      3.2.2.2 Requirement 2.2
      3.2.2.m Requirement 2.m
      etc.

3.3 Performance requirements

3.4 Design constraints

3.5 Software system attributes

3.6 Other requirements
3 Specific Requirements

3.1 External interfaces
   3.1.1 User interfaces
   3.1.2 Hardware interfaces
   3.1.3 Software interfaces
   3.1.4 Communications interfaces

3.2 Functional requirements

3.3 Performance requirements

3.4 Design constraints

3.5 Software system attributes

3.6 Other requirements

3.2. System features
   3.2.1. System feature 1
      3.2.1.1. Purpose of feature
      3.2.1.2. Stimulus/response sequence
      3.2.1.3. Functional requirements
         3.2.1.3.1. Functional requirement 1
         3.2.1.3.2. Functional requirement 2
         3.2.1.3.n. Functional requirement n

   3.2.2. System feature 2

   etc.
Requirements documents should NOT include...

• **project development plans**
  – cost, staffing, schedules, methods, tools, etc.
  – lifetime of SRS is entire operational life of the product
  – lifetime of development plans is much shorter

• **product assurance plans**
  – configuration management, V&V, test, QA, etc.
  – different audiences and timelines

• **design considerations**
  – except if the application domain constrains the design
    ▶ e.g., limited bandwidth or security concerns
  – focus on what, not on how
Writing style

Requirements are technical documents: brevity, clarity, consistency, and precision are more important than literary style!

• avoid requirements “fusion”
  – one functionality per requirement
• use consistent terminology
  – define terms in glossary
• \textbf{BE PRECISE} – no vague requirements
Writing style: BE PRECISE

- avoid **persuasive** words
  - *certainly, therefore, clearly, obviously, ...*
- avoid **vague** words
  - *some, most, sometimes, often, usually, normally, ordinarily, typically, customarily, ...*
- avoid **non-committal** words
  - *ought to, preferred, desirable, wanted, ...*
- avoid **and/or, etc.** like the plague
- avoid **generic** verbs
  - *handled, processed, rejected, skipped, ...*
- avoid **unquantifiable** words
  - *reduce, optimize, large, rapid, user-friendly, easy, simple, intuitive, robust, efficient, flexible, ...*
Use signifiers to denote the urgency of a requirement:

• **SHALL** means that the definition is an absolute requirement of the specification (i.e., *essential*)

• **SHOULD** means that there may exist valid reasons in particular circumstances to ignore a particular requirement (i.e., *desirable*)

• **MAY** means that a requirement is truly *optional*

More information:
• S. Bradner, Key words for use in RFCs to Indicate Requirement Levels, IETF RFC 2119, 1997
2.6 Withdrawal
If the card is accepted, the user has entered the correct PIN, and there are sufficient funds in the account, the amount of cash shall be dispensed. If the card is invalid (in which case it should be ejected), the PIN does not match the one required for the card (in which case a tone shall sound and the user given the option to try again—the tries shall be limited to 3), or the balance is insufficient (in which case a tone shall sound and the user shall have the opportunity to enter a new amount) cash shall not be dispensed.
2.6: The System shall support cash withdrawals by the user.

2.6.1: A withdrawal shall be allowed if and only if:
   a. The card can be validated (Req. 2.7).
   b. The PIN is valid for the card (Req 2.8).
   c. The funds in the card account exceeds the funds requested in the withdrawal.

2.6.2: If a withdrawal is allowed (2.6.1), the exact amount requested shall be dispensed.

Is this now a good requirement?
Customer involvement

Lack of user input is major (1 in 8) cause of project failures

Warning sign: user surrogates (user managers, marketing staff) supply all the requirements

- identify the various user classes
  - not all users are the same
- find “product champions” for specific user classes
Scope creep

scope creep: new requirements are continually added

• define the project scope clearly
  – expect some requirements growth in all projects;
    include buffers in project plan
• attach a person to each requirement
  – people are much less likely to add “the kitchen sink” if
    their name is there
Analysis paralysis and gold plating

Analysis paralysis: requirements development seems to go on forever

Gold plating: developers add unnecessary functionality

• identify key decision-makers to “call the shots”
• prioritize requirements
• trace requirements to use cases
plan for change:

- use version control system for requirements
- change impact analysis
  - identify related requirements in document
Requirements Validation
Requirements validation techniques

Goal: check requirements for **realism**, **consistency**, and **completeness**

- **prototyping**
  - use executable system model to check requirements
- **requirements reviews**
  - systematic manual analysis of the requirements.
- **test-case generation**
  - develop tests for requirements to check testability: “If you can’t test it, it is not a requirement!”
Review checks

- **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?
  - easier if the document is properly cross-referenced
- **verifiability**: is the requirement realistically testable?
Traceability

Requirements traceability is the linking between requirements and other artefacts:

- **use cases**
  - system validation (do we plan the right system?)

- **other requirements**
  - cross-referencing
  - user-to-system: capture hierarchical structure

- **code**
  - system validation (did we implement the right system?)

- **test cases**
  - verification (did we implement the system right?)
Traceability techniques

• assign a unique number to all requirements
  – and other items
• cross-reference using this unique number
• representation typically as **traceability matrix**
  – several matrices for different types of traceability links
Traceability techniques

• assign a unique number to all requirements
  – and other items
• cross-reference using this unique number
• representation typically as **traceability matrix**
  – several matrices for different types of traceability links
• main problems
  – link maintenance
  – link recovery
Requirements verifiability

- **requirements** must be **verifiable**, i.e., written so that their validity in the system can be checked by one or many test case(s)
  
  “If you can’t test it, it is not a requirement!”

- even more **difficult** for **non-functional** requirements
Requirements verifiability

- **requirements** must be **verifiable**, i.e., written so that their validity in the system can be checked by one or many test case(s)
  
  “If you can’t test it, it is not a requirement!”

- even more **difficult** for **non-functional** requirements

For example

4. The system should be easy to use by experienced engineers and should be organized in such a way that user errors are minimized.
Requirements verifiability

- **requirements** must be **verifiable**, i.e., written so that their validity in the system can be checked by one or many test case(s)
  
  “If you can’t test it, it is not a requirement!”

- even more **difficult** for **non-functional** requirements

For example

4. The system should be easy to use by experienced engineers and should be organized in such a way that user errors are minimized.

- How do you test “**easy-to-use**”?
- What is **minimized**?
Requirements verifiability

- requirements must be verifiable, i.e., written so that their validity in the system can be checked by one or many test case(s)
  “If you can’t test it, it is not a requirement!”
- even more difficult for non-functional requirements

Better:

4.1 Engineering staff with more than two years of professional experience shall be able to use all the system functions after a total of two hours training.
4.2 After this training, the average number of errors made by experienced engineers shall not exceed two per day.
Requirements verifiability

Many non-functional properties can be quantified:

- **speed**
  - processed transactions/second
  - user/event response time
  - screen refresh time

- **size**
  - minimum ram requirement
  - database size

- **ease of use**
  - average user training time
  - number of help frames

- **reliability**
  - mean time to failure
  - probability of unavailability
  - rate of failure occurrence
  - availability
Requirements verifiability

Many non-functional properties can be quantified:

- **robustness**
  - time to restart after failure
  - percentage of events causing failure
  - probability of data corruption on failure

- **portability**
  - % of target dependent statements
  - number of target systems
Summary

- requirements capture **what** a system shall do
  - but avoid design detail as much as possible
  - written in the user’s language (with all the problems…)
- requirements are **difficult** to find out and formalize
- requirements problems are
  - common
  - the most costly
  - the most difficult to correct (they are conceptual)
- use a **standard structure, forms, and checklists** for the document and the individual requirements
- write **verifiable** requirements
Wally, we don't have time to gather the product requirements ahead of time.

I want you to start designing the product anyway. Otherwise it will look like we aren't accomplishing anything.

Of all my projects, I like the doomed ones best.
Poor requirements are the source of all evil!