Requirements Engineering

Software Engineering

Leiden University 2007-2008

Michel Chaudron

Requirements Engineering

Based on Selections from

- Chapter 4 from *Object-Oriented Software Engineering* by Lethbridge & Laganiere
- *Requirements Engineering: A Good Practice Guide* by Ian Sommerville & Pete Sawyer
- *Generative Programming* by Czarnecki

What, Why, Who, When, Where, How?

Requirements engineering

- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work 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 of the work so cripples the resulting system if done wrong.

No other part is more difficult to rectify later".

Fred Brooks, "No Silver Bullet",
IEEE Computer,1987
Author of The Mythical Man-month

Understanding the problem

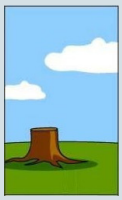
What the customer explained

What the project leader understood

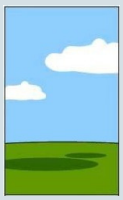
What the analyst designed

What the consultant defined

What the programmer made



How it was maintained



What was documented

What was charged

What was installed

What the client needed

Developers' View of Users	Users' View of Developers
Users don't know what they want. Users can't articulate what they want. Users have too many needs that are politically motivated. Users want everything right now. Users can't prioritize needs. Users refuse to take responsibility for the system. Users are unable to provide a usable statement of needs. Users are not committed to system development projects. Users are unwilling to compromise. Users can't remain on schedule.	Developers don't understand operational needs. Developers place too much emphasis on technicalities. Developers try to tell us how to do our jobs. Developers can't translate clearly stated needs into a successful system. Developers say no all the time. Developers are always over budget. Developers are always late. Developers ask users for time and effort, even to the detriment of the users' important primary duties. Developers set unrealistic standards for requirements definition. Developers are unable to respond quickly to legitimately changing needs.

Learning from each other

Users, customers, managers, domain experts, and developers share different skills, backgrounds, and expectations.

7

Developing a shared vision

Requirements emerge from a process of co-operative learning in which they are explored, prioritized, negotiated, evaluated, and documented.

8

The 10 top reasons for **not** doing requirements

10. We don't need requirements, we're using objects/java/web/….
9. The users don't know what they want
8. We already know what the users want
7. Who cares what the users want?
6. We don't have time to do requirements
5. It's too hard to do requirements
4. My boss frowns when I write requirements
3. The problem is too complex to write requirements
2. It's easier the change the system later than to do the requirements up front
1. We have already started writing code, and we don't want to spoil it

Volere Requirements Resources http://www.volere.co.uk

9

"I held my entire program up for 4+ weeks due to unclear, unwritten requirements. Took some heat for that in the beginning, but the deep dive requirements effort is highlighting a Silicon spin we didn't know about, standards that we don't support, other postlaunch requirements nobody considered…all of this causing us and mgmt to question the viability of the product. BTW, this is all stuff we wouldn't have realized until it smacked us in the face 6 months from now. Spending a month now prevented us from spending millions before a conscious decision."

From : Reflections on a Successful Corporate Requirements Engineering Training Curriculum, Erik Simmons, Intel Corporation, 2005

10

Stakeholder issues

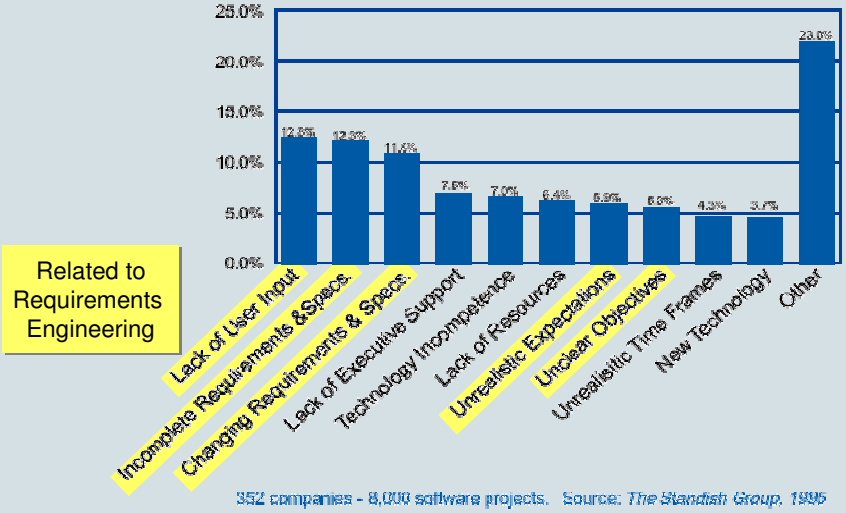
Steve McConnell, in his book *Rapid Development*, details a number of ways users can inhibit requirements gathering:

- Users don't understand what they want or users don't have a clear idea of their requirements
- Users won't commit to a set of written requirements
- Users insist on new requirements after the cost and schedule have been fixed.
- Communication with users is slow
- Users often do not participate in reviews or are incapable of doing so.
- Users are technically unsophisticated
- Users don't understand the development process.
- Users don't know about present technology.

11

Why Software Projects Fail

Example of empirical research

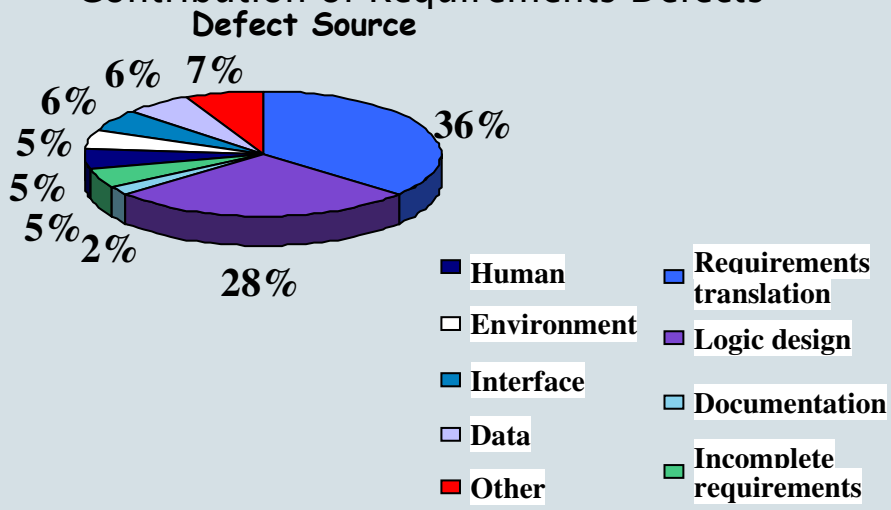


Related to Requirements Engineering

352 companies - 8,000 software projects. Source: *The Standish Group, 1995*

12

Contribution of Requirements Defects



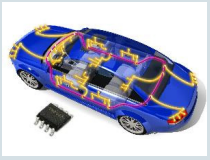
13

Why Requirements Engineering?

- Scope the problem
- Understand the problem
  - for the client as well as the architect
- Basis for design
- Contract between client/user and builders
  - agreement on what has to be built

14

Understand the Domain



What is important?

Which things are stable and which change?

How does the project add to an organizations' success

15

Initial Steps in RE process

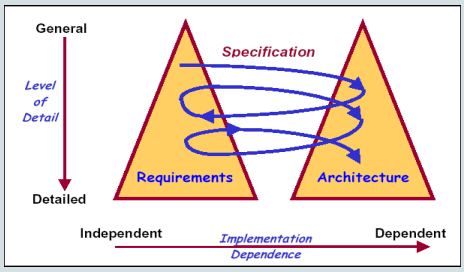
- What are the drivers?
  - Stakeholders & concerns
- What are the constraints?
  - Economical/technical/organisational
- What is the scope of the system?

16

Twin Peaks Process

Separate but concurrent development of requirements & architecture

WHAT: problem structuring

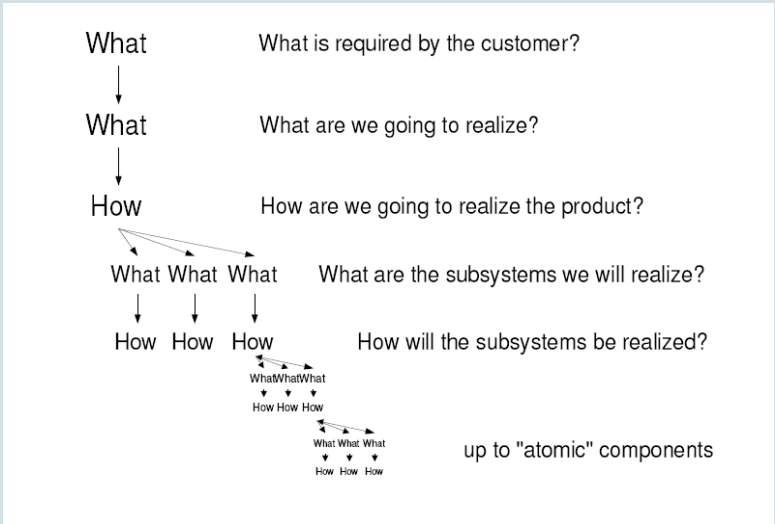


HOW: solution structuring

Progressing understanding of architecture & design provides a basis for discovering further system requirements and vice versa

There is interaction between available solutions and requirements

17



Slide by Gerrit Muller, ESI, 2007

18

What is a Requirement ?

- A statement about the proposed system that all stakeholders agree must be made true in order for the customer's problem to be adequately solved.
  - Short and concise piece of information
  - Says something about the system
  - All the stakeholders have agreed that it is valid
  - It helps solve the customer's problem
  - Contract between customer and builder

19

Example Requirement Template

Requirement #: Requirement Type: Event/use case #:

Description:

Rationale:

Source:
Fit Criteria:

Customer Satisfaction: Customer Disatisfaction:
Dependencies: Conflicts:
Supporting Materials:
History:

Volere

20

Errors

Up to 30-50% of the errors found further downstream the development process are due to errors in the requirements.

Requirements errors are typically non-clerical.

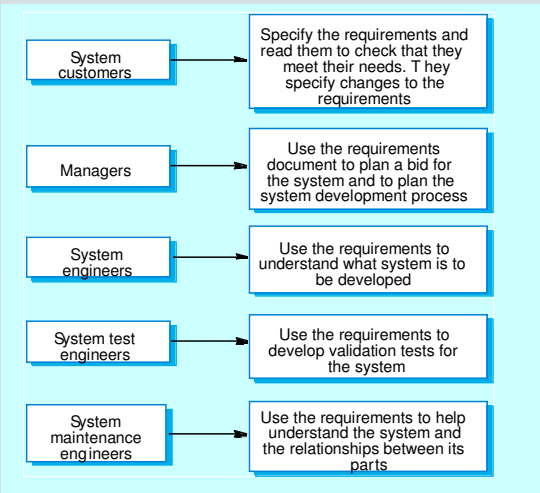
incorrect facts	49%
omissions	31%
inconsistencies	13%
ambiguities	5%

Requirements errors can be detected.

- Review by authors 23%
- Review by others 10%

21

Users of a requirements document



22

Types of requirements

- User requirements:

  The description of the functions that the system has to fulfil for its environment in terms of the users interacting with the system, e.g. in the form of *use cases*.

- Software requirements:

  The software requirements are a translation and a more precise description of the user requirements, in terms for software engineers.

  Functional and extra-functional requirements

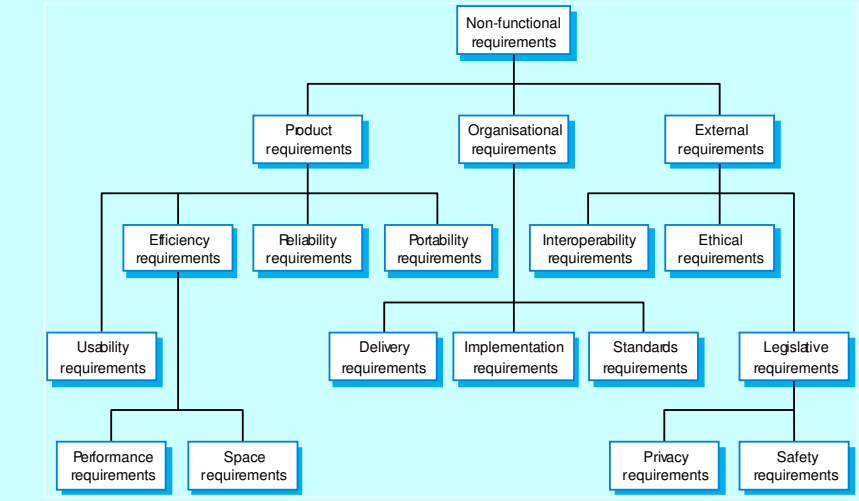
23

Types of Requirements

- Functional requirements
  - Describe *what* the system should do
- Extra-functional requirements
  - *ilities: Availability, Security, Reliability, Timeliness,
  - Capacity
- *Constraints* that must be adhered to during execution

24

Types of extra-functional req'rements



Functional requirements

- What *inputs* the system should accept
- What *outputs* the system should produce
- What data the system should *store* that other systems might use
- What *computations* the system should perform

Examples

- The system shall allow users to search for an item by title, author, or ISBN.

Defines system functionality.

- If an item is not returned within the period of load, then the person who loans the item will be fined Euro 1 per week.

Defines (causal) relations between system functions.

27

Examples of XFR: Reliability

Typically expressed in terms of

for repairable systems

- **Mean Time Between Failures (MTBF)**
  - Number of hours that pass before a component fails
  - E.g. 2 failures per million hours:
    - MTBF = 10^6 / 2 = 0,5 * 10^6 hr

For non-repairable systems

- **Mean Time To Failure (MTTF)**
  - Mean time expected until the first failure of a system
  - Is a statistical value over a long period of time

- **Mean Time To Repair (MTTR)** Availability

28

Examples XFR: Maintainability

Maintainability

The average person time required to fix a category 3 defect (including testing and documentation upgrade) shall not exceed two person days.

29

System Quality Attributes

- Time To Market
- Cost and Benefits
- Projected life time
- Targeted Market
- Integration with Legacy System
- Roll back Schedule

} Business Community view

- Performance
- Availability
- Usability
- Security

} End User's view

- Maintainability
- Portability
- Reusability
- Testability

} Developer's view

30

Constraints

Constraints are not negotiable

Constraints concerning the *environment and technology* of the system.

- Platform
- Technology to be used

Constraints concerning the *project plan and development methods*

- Development process (methodology) to be used
- Cost and delivery date
  - Often put in contract or project plan instead

31

Constraints

Constraint restrict how the requirements are to be implemented.

- **Interface Requirements**.
  How external interfaces with other systems must be done.
- **Communication Interfaces**.
  The networks and protocols to be used.
- **Hardware Interfaces**.
  The computer hardware the software is to execute on.
- **Software Interfaces**.
  How the software should be compatible with other software: applications, compilers, operating systems, programming languages, database management systems.
- **User Interfaces**.
  Style, format, messages

32

Requirements on Requirements (1)

Each individual requirement should be

- **Important/necessary** for the solution of the current problem
- **Unique**
- **Unambiguous**
- **Logically consistent**
- **Not over-constrain the design** of the system
- **Atomic**: not consist of multiple separate requirements

33

Requirements on Requirements (2)

The set of requirements together should be:

- **Complete**
- Expressed using a **clear and consistent notation**
  - at the same level of detail
- Without duplication

34

Requirements on Requirements (3)

- **S** **Specific**
  To-the-point, precise
- **M** **Measurable**
  Quantifiable and verifiable
- **A** **Acceptable** (to the stakeholders)
  Accessible, understandable (for the user)
  Achievable (technically/planning/economically)
- **R** **Realistic**
  Deducible to the real business drivers
- **T** **Testable**

35

Let's consider

- "All communication between client and server is secure"
- "It is easy to extend"
- "The system should respond quickly"
- "The user should not have to wait more than a few second …"
- "Determine solution within 0.3 sec"
- "The system should be [illegible]"
- "The system can handle 100 concurrent [illegible]"
- "The system should be state-of-the-art …"

not measurable

not precise

vague:to what?

attainable

subjective

time-dependent;
means something else tomorrow

doing what?

36

Requirements Prioritization

37

The Cost of Traditional BRUF

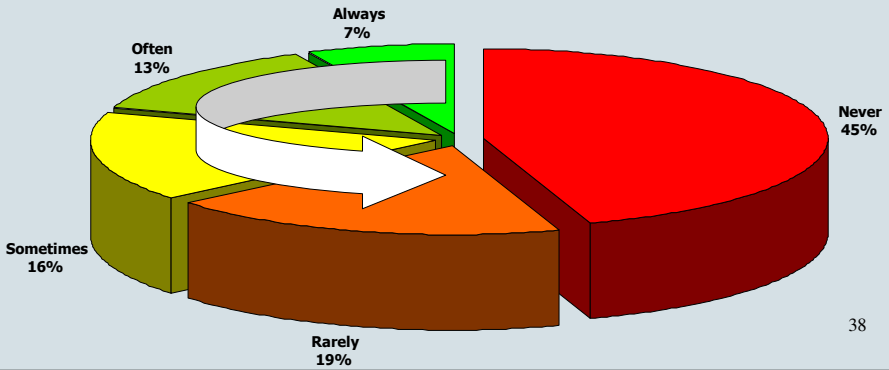
Big Requirements Up Front

Source: Jim Johnson of the Standish Group, Keynote Speech XP 2002

Pie chart shows percentage of functionality used by stakeholders

"Successful" Projects Still Have Significant Waste

Pareto-rule applies: 20% of functionality delivers 80% of value



38

Prioritizing Requirements

- **MIL STD**:
  - Must have, will have, may have
- **RUP: MoSCoW**
  Must have
  Should have
  Could have
  Won't have

Criteria: indicate importance

Alternative criteria: volitility, cost to realize, risk, ..

39

Cost-Value Prioritization of Requirements

Motivation for Prioritization:

- Focus development effort
  - Allocate resources based on importance
- Make trade-offs between conflicting goals, such as quality, cost and time-to-market

40

Cost-Value Prioritization of Requirements

Process:

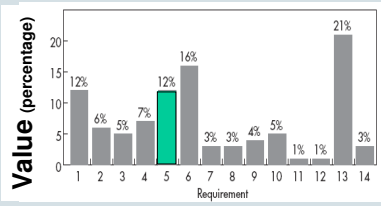
1. Review **requirements** for **clarity** and **completeness** (by Requirements Engineers)
2. Assess **relative value** of requirements in pair wise manner (Customers and users)
3. Assess **relative cost** of realizing requirements in pair wise manner (by experienced SW Engineers)
4. Calculate (value, cost)-pairs (using AHP*)
5. Plot requirements as (value, cost)-pairs
6. Prioritize

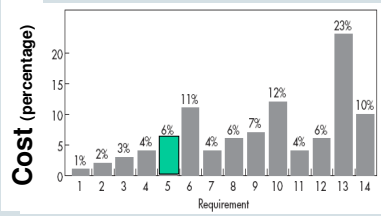
* Analytic Hierarchy Process

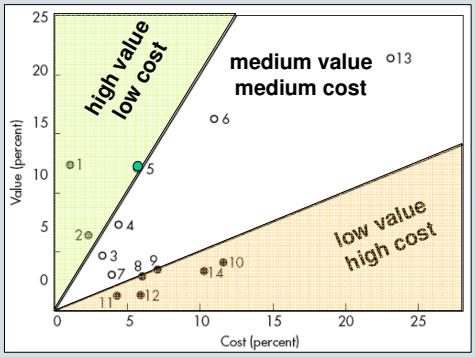
41

Requirements Prioritization Example

- 14 Requirements







42

THE ANALYTIC HIERARCHY PROCESS

To make decisions, you identify, analyze, and make trade-offs between different alternatives to achieve an objective. The more efficient the means for analyzing and evaluating the alternatives, the more likely you'll be satisfied with the outcome. To help you make decisions, the Analytic Hierachy Process compares alternatives in a stepwise fashion and measures their contribution to your objective.[1]

AHP in action. Using AHP for decision making involves four steps. We'll assume here that you want to evaluate candidate requirements using the criterion of value.

Step 1. *Set up the* n *requirements in the rows and columns of an* n × n *matrix.* We'll assume here that you have four candidate requirements: Req1, Req2, Req3, and Req4, and you want to know their relative value. Insert the *n* requirements into the rows and columns of a matrix of order *n* (in this case we have a 4 × 4 matrix).

Step 2. *Perform pairwise comparisons of all the requirements according to the criterion.* The fundamental scale used for this purpose is shown in Table A.[1] For each pair of requirements (starting with Req1 and Req2, for example) insert their determined relative intensity of value in the position (Req1, Req2) where the row of Req1 meets the column of Req2. In position (Req2, Req1) insert the reciprocal value, and in all positions in the main diagonal insert a "1." Continue to perform pairwise comparisons of Req1–Req3, Req1–Req4, Req2–Req3, and so on. For a matrix of order *n*, $n \cdot (n-1)/2$ comparisons are required. Thus, in this example, six pairwise comparisons are required; they might look like this:

	Req1	Req2	Req3	Req4
Req1	1	1/3	2	4
Req2	3	1	5	3
Req3	1/2	1/5	1	1/3
Req4	1/4	1/3	3	1

Step 3. *Use averaging over normalized columns to estimate the eigenvalues of the matrix* (which represent the criterion distribution). Thomas Saaty proposes a simple method for this, known as averaging over normalized columns.[1] First, calculate the sum of the *n* columns in the comparison matrix. Next, divide each element in the matrix by the sum of the column the element is a member of, and calculate the sums of each row:

	Req1	Req2	Req3	Req4	*Sum*
Req1	0.21	0.18	0.18	0.48	*1.05*
Req2	0.63	0.54	0.45	0.36	*1.98*
Req3	0.11	0.11	0.09	0.04	*0.34*
Req4	0.05	0.18	0.27	0.12	*0.62*

Then normalize the sum of the rows (divide each row sum with the number of requirements). The result of this computation is referred to as the *priority matrix* and is an estimation of the eigenvalues of the matrix.

$$\frac{1}{4} \cdot \begin{pmatrix} 1.05 \\ 1.98 \\ 0.34 \\ 0.62 \end{pmatrix} = \begin{pmatrix} 0.26 \\ 0.50 \\ 0.09 \\ 0.16 \end{pmatrix}$$

Step 4. *Assign each requirement its relative value based on the estimated eigenvalues.* From the resulting eigenvalues of the comparison matrix, the following information can be extracted:

- Req1 contains 26 percent of the requirements' total value,
- Req2 contains 50 percent,
- Req3 contains 9 percent, and
- Req4 contains 16 percent.

43

AHP consistency

Result consistency. If we were able to determine precisely the relative value of all requirements, the eigenvalues would be perfectly consistent. For instance, if we determine that Req1 is much more valuable than Req2, Req2 is somewhat more valuable than Req3, and Req3 is slightly more valuable than Req1, an inconsistency has occurred and the result's accuracy is decreased. The redundancy of the pairwise comparisons makes the AHP much less sensitive to judgment errors; it also lets you measure judgment errors by calculating the consistency index of the comparison matrix, and then calculating the consistency ratio.

Consistency index. The consistency index (CI) is a first indicator of result accuracy of the pairwise comparisons. You calculate it as $CI = (\lambda \max - n)/(n-1)$. $\lambda \max$ denotes the maximum principal eigenvalue of the comparison matrix. The closer the value of $\lambda \max$ is to *n* (the number of requirements), the smaller the judgmental errors and thus the more consistent the result. To estimate $\lambda \max$, you first multiply the comparison matrix by the priority vector:

$$\begin{pmatrix} 1 & 1/3 & 2 & 4 \\ 3 & 1 & 5 & 3 \\ 1/2 & 1/5 & 1 & 1/3 \\ 1/4 & 1/3 & 3 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0.26 \\ 0.50 \\ 0.09 \\ 0.16 \end{pmatrix} = \begin{pmatrix} 1.22 \\ 2.18 \\ 0.37 \\ 0.64 \end{pmatrix}$$

Then you divide the first element of the resulting vector by the first element in the priority vector, the second element of the resulting vector by the second element in the priority vector, and so on:

$$\begin{pmatrix} 1.22 / 0.26 \\ 2.18 / 0.50 \\ 0.37 / 0.09 \\ 0.64 / 0.16 \end{pmatrix} = \begin{pmatrix} 4.66 \\ 4.40 \\ 4.29 \\ 4.13 \end{pmatrix}$$

44

Prioritization

- Estimation of relative weights
  - ratio-scale
- 100 $ approach
  - ratio-scale
- Ranking by comparing
  - (bubble)sorting – ordinal scale

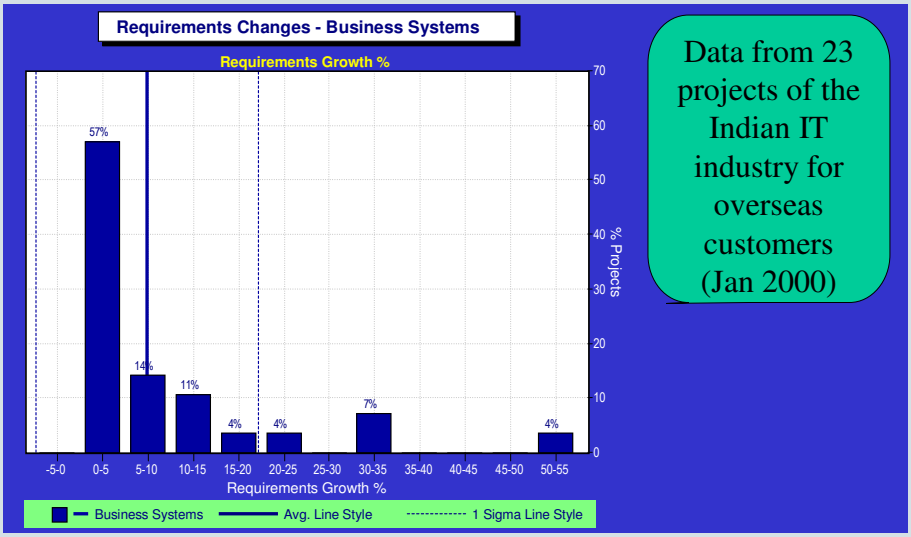
45

Managing Changing Requirements

- Requirements change because:
  - Business process changes
  - Technology changes
  - The problem becomes better understood
- Requirements analysis never stops
  - Continue to interact with the clients and users
  - The benefits of changes must outweigh the costs.
    - Certain small changes (e.g. look and feel of the UI) are usually quick and easy to make at relatively little cost.
    - Larger-scale changes have to be carefully assessed
      - Forcing unexpected changes into a partially built system will probably result in a poor design and late delivery
  - Some changes are enhancements in disguise
    - Avoid making the system *bigger*, only make it *better*

46

Requirement Changes



Data from 23 projects of the Indian IT industry for overseas customers (Jan 2000)

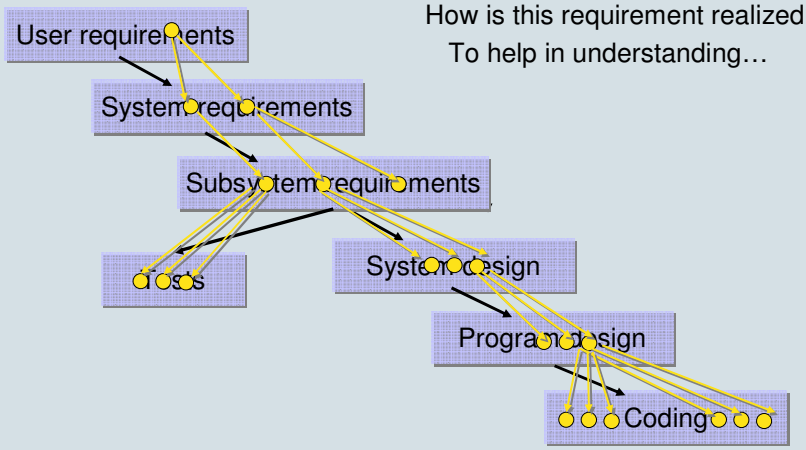
Traceability

- From req to arch choices/features
- From features to req's

- Check
  - Completeness of system
  - Analyze impact of changing requirements

48

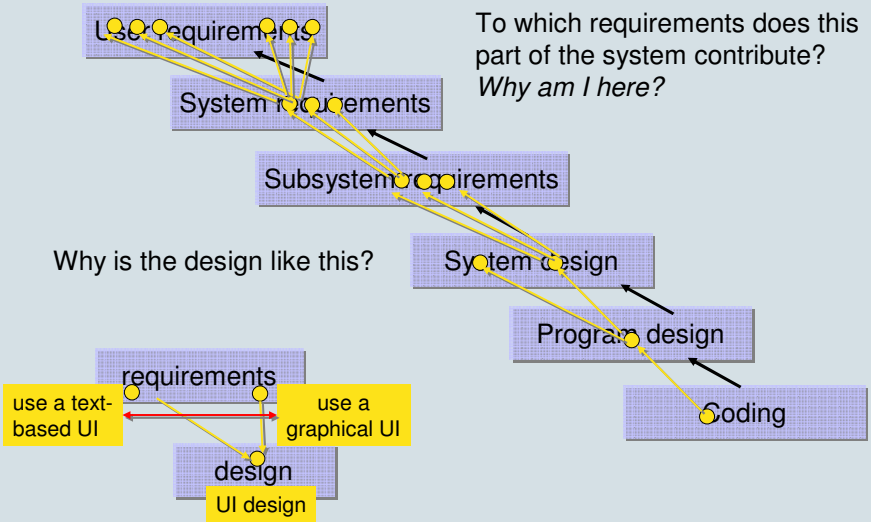
Forward Traceability



How is this requirement realized?
To help in understanding…

49

Backward Traceability



To which requirements does this part of the system contribute?
Why am I here?

Why is the design like this?

50

Why Traceability?

- Accountability: where did this requirement come from?
  - The source of a requirements may be needed for clarification, negotiation, conflict resolution
- Matching solution to problem
  - For monitoring completeness of system:
    - Acceptance test: are all requirements addressed?
    - are there unnecessary requirements/features?
- Analyze impact of changes (in req'mt's / design decions)
  - Change request: What parts of the design need to change, if a requirement changes?
- Reuse of requirements

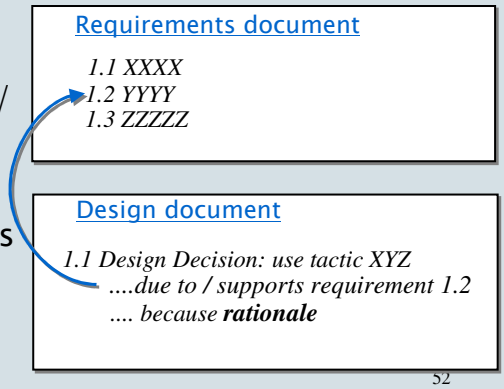
51

How Traceability: Hyperlinks

design document and requirements document contains hyperlinks to each other

Typical use:
interactive exploring / browsing req.docs

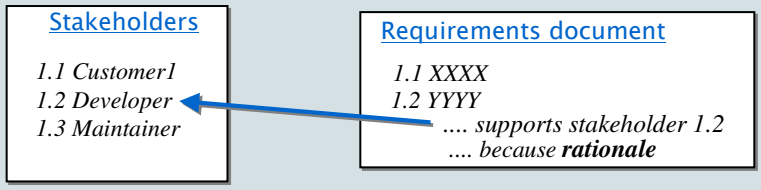
Using .html documents & browsers



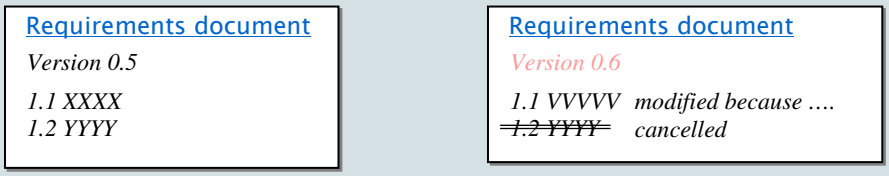
52

But also

- Trace the source of requirements



- Trace the history/evolution of requirements



53

How Traceability: Matrix

A matrix links requirement to design decisions

requirements

design decisions

Uses: database or spread-sheet

	1	2	3	4	5	6	..	..	..
1									
2		X							
3			X						
4			X		X	X			
5			X						
6									
7						X	X		
..									
..	X			X			X		

54

Req. Management Guidelines

Basic Guidelines:

1. Define policies for requirement management
2. Define traceability policies
3. Maintain a traceability manual

Intermediate Guidelines:

4. Use (automated) requirements management tool
5. Define change management policies
   - Maintain a change history
6. Identify global system requirements

Advanced Guidelines:

7. Measure requirements stability
   - Identify volatile requirements
8. Record rejected requirements

From: Sommerville & Sawyer

Traceability Research Questions

- How much traceability should one do?
- Can we automate traceability?
  - Matching keywords between design and req's?

Concluding Remarks

There is a lot more to requirements that meets the eye.

A lot of errors in system development can be traced to erroneous requirements. It pays to make an effort to check your requirements

Requirements evolve in concert with architectural decisions.

Domain Engineering helps developing system families

Lots of guidelines exist for doing requirements right! Use them!

57

Questions?

See you this afternoon & next week

[Gacek et al 1995] present the results of a survey of people who are somehow involved in software development processes (developers, customers, maintainers, aquisitioners, etc.).

There they found that, with respect to architects, the three major concerns were

"1) requirements traceability;

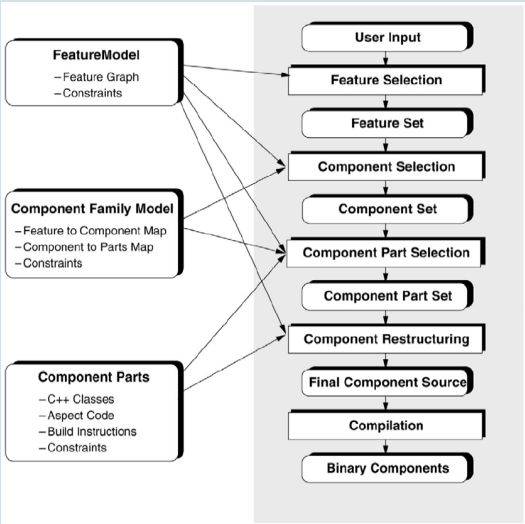
2) support of tradeoff analyses; and

3) completeness, consistency of architecture."

Gacek, C., Abd-Allah, A., Clark, B.K., and Boehm, B. (1995)

"On the Definition of Software System Architecture," in *Proceedings of the First International Workshop on Architectures for Software Systems - 17th ICSE*, Seattle, 24-25 April 1995, pp. 85-95.

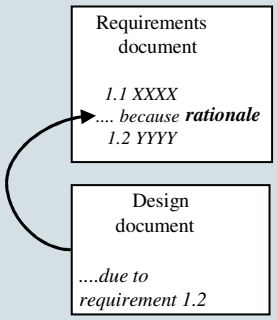
59



60

Requirements documents

- should be:
  - agreed to by all the stakeholders
  - sufficiently complete
  - well organized
    - Easy to read
    - Easy to maintain / change
  - clear
- Traceability:
  - use of hypertext may be usefull
    - for exploring/browsing req.docs



61

- **Analysis anti-patterns**
- : The Functional/Technical specification is given to the Development team on a napkin (i.e., informally, and with insufficient detail) which is fundamentally equivalent to having no specification at all.
- : All requirements are communicated to the development teams in a rapid succession of netmeeting sessions or phone calls with no Functional/Technical specification or other supporting documentation.
- : To write the Technical/Functional specification after the project has already gone live.

62

Don Gause lists the five most important sources of requirements failure as:

- failure to effectively manage conflict,
- lack of clear statement of the design problem to be solved,
- too much unrecognized disambiguation,
- not knowing who is responsible for what
- lack of awareness of requirements risk.

63

Through Requirements you are meant to find out and understand what users' intentions and need are.

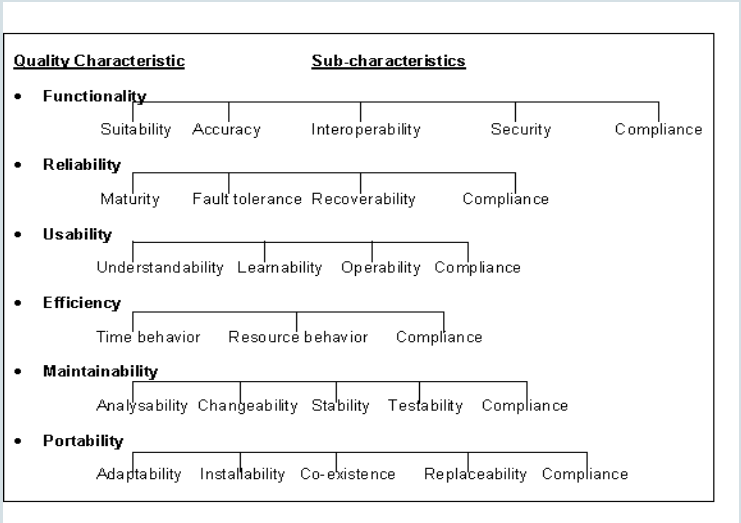
This may be different from what they say it is!

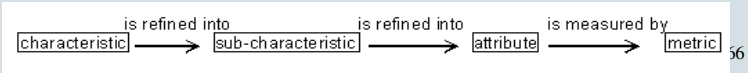
64

Ezelsbruggetje

- Het woord is waarschijnlijk afkomstig van het feit dat de ezel maar een heel klein randje nodig heeft om snel op de plek van bestemming te komen; een plank over een sloot volstaat al. Het woord ezelsbrug is al heel oud en bestond in het Latijn al (*pons asinorum*).
- English translation welcome …

65





66

STIMULUS–ENVIRONMENT–RESPONSE

'Formula' for scenario's

- Use case scenario
  Remote user requests a database report via the Web during peak period and receives it within 5 seconds
- Growth scenario
  Add a new data server during peak hours within a downtime of at most 8 hours.
- Exploratory scenario
  Half of the servers go down during normal operation without affecting overall system availability

A good scenario makes clear what the stimulus is and what the measurable response of interest is

67