

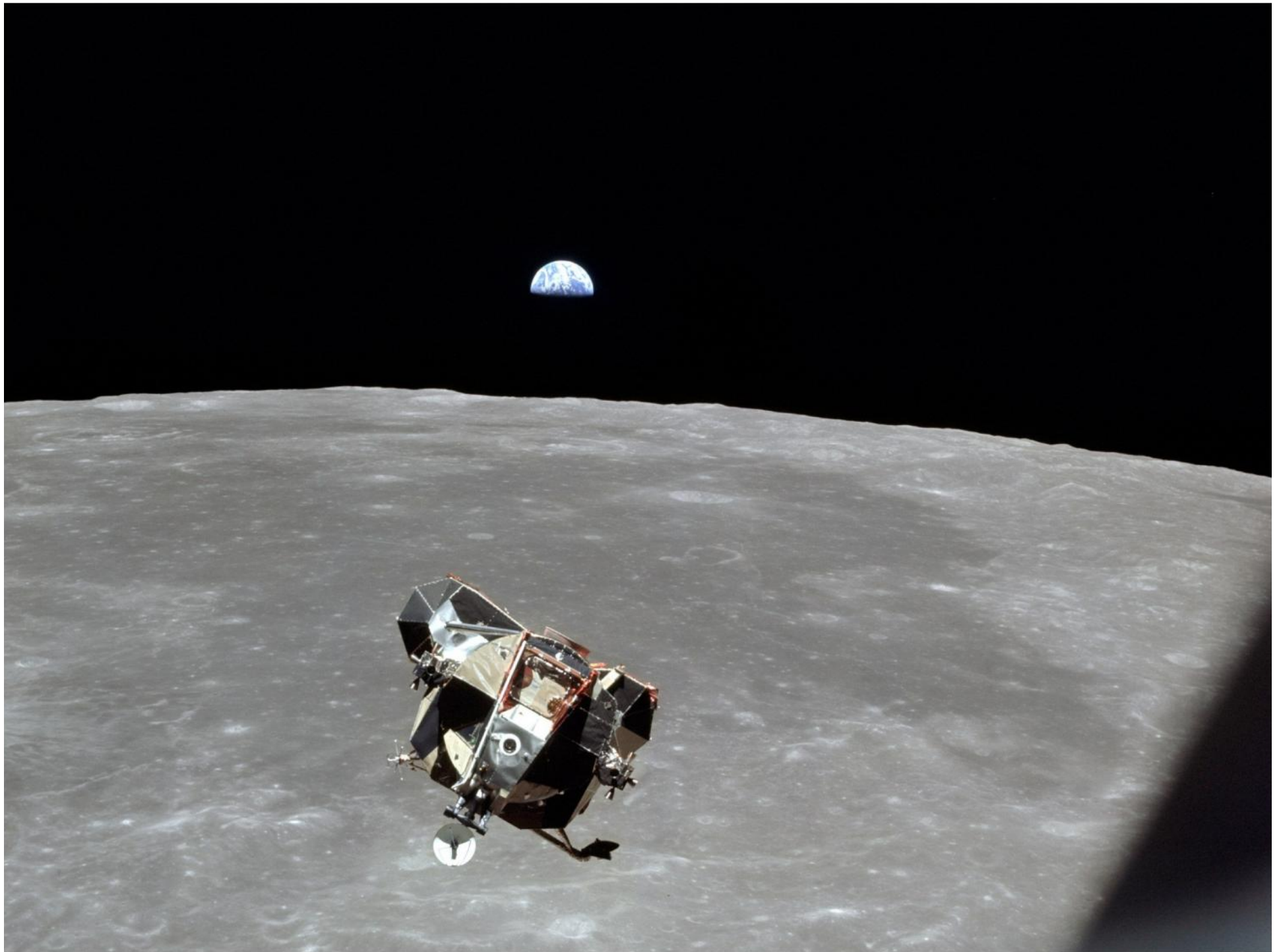
Requirements Engineering

A 21st century approach



02264

Harald Störrle



Chapter 1



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 1: Introduction

DTU course 02264

Agenda

Abstract

- Starting from concrete examples, we will argue what can be gained from Requirements Engineering (RE).
- Conversely, inadequate RE comes with the risk of substantial socio-economic cost so that our professional responsibility as engineers alone demands great care in RE, even though software engineers may not yet face the same legal consequences of malpractice as other kinds of engineers, say.
- We will introduce a working definition of RE and justify the subtitle of the course.

Contents

1. Software Faults are a Paramount Problem
2. Fighting Software Faults through Requirements Engineering
3. A Working Definition of Requirements Engineering
4. Requirements Engineering in the 21st Century



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 1.1:

Software Failures are a Paramount Problem

DTU course 02264

LH2904 Accident in Warsaw



On 14.9.1993, flight LH2904 crashed in Warsaw when landing, killing 2 and injuring 68.



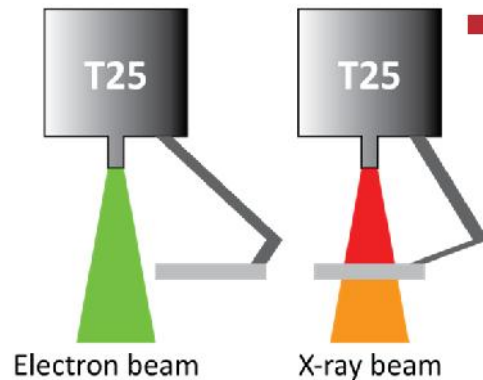
- **In 1993, an A320-211 landed in Warsaw under heavy rain, overshot the runway at high speed, crashed, and burst into flames.**
 - The two pilots were killed, 68 people on board were hurt, 51 of them seriously.
- **The plane did not slow down because it took the plane's computer 9s to switch from „airborne“ into „ground“ mode.**
- **Until then, thrust reversal, spoiler deployment, and wheel brakes were blocked .**
 - The state transition is triggered if both rear wheels turn quickly enough and carry at least 12t of weight.
 - Due to strong sideways wind, the left wheel bounced off the ground several times.
 - Due to aquaplaning, the wheels did not turn fast enough to trigger the state transition.

LH2904 Accident in Warsaw

- As a consequence, the trigger has been changed (minimum weight reduced to 2t), and spoilers and thrust reversal is not coupled to wheel spin any more.



Therac-25



Different planned operation modes of the Therac-25 (top), and the fatal configuration (bottom).

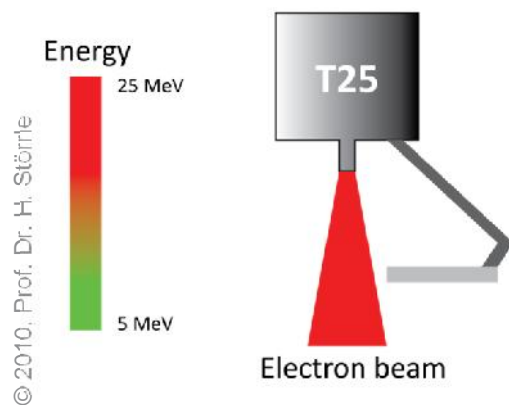
- **Radiation therapy is one of the three standard treatments for cancer.**

- The Therac-25 linear accelerator emits electron beams that can be turned into x-rays by increasing the beam energy and moving a tungsten target into the beam.
- Barring the beam by the target will significantly reduce the effective energy delivered, so in this mode, the beam has to be much more powerful to achieve the same result.
- Exposing patients to the high-energy beam without the target results in massive radiation overdoses.

- **In the 1980's, this happened repeatedly, killing at least 3 patients.**

- The error was caused by operators typing in commands to the control terminal faster than they could be processed.
- There were inadequate error messages („Malfunction 54“). The software was (partially) reused from the Therac-20 which had a hardware interlock preventing this problem.

[IEEE Computer, Vol. 26, No. 7, July 1993, pp. 18-41]



Ariane 5, Flight 501



On the 4.6.1996,
Ariane 501 exploded
after 40s flight.



- On her maiden flight, the Ariane 5 (flight 501) went out of control after 40s and was auto-destructed for safety reasons.
 - The rocket and its payload had cost 500mio US\$.
- **36.7s after lift-off, both of the doubly redundant flight control computers switched off due the same overflow from converting the horizontal speed from a 64bit float into a 16bit signed int.**
- **The software worked exactly as specified, but it had been specified (and created) for the Ariane 4 which was flying much slower than the Ariane 5.**
 - Ironically, the software causing the error was not even required for flight, but only for launch preparation. It was kept active the first 40s after lift-off to be able to quickly resume the launch procedure after count down standby.

ALG-II Padding Error



A large portion of the German unemployment benefits did not reach their recipients when the new system was put into operation in 2004.

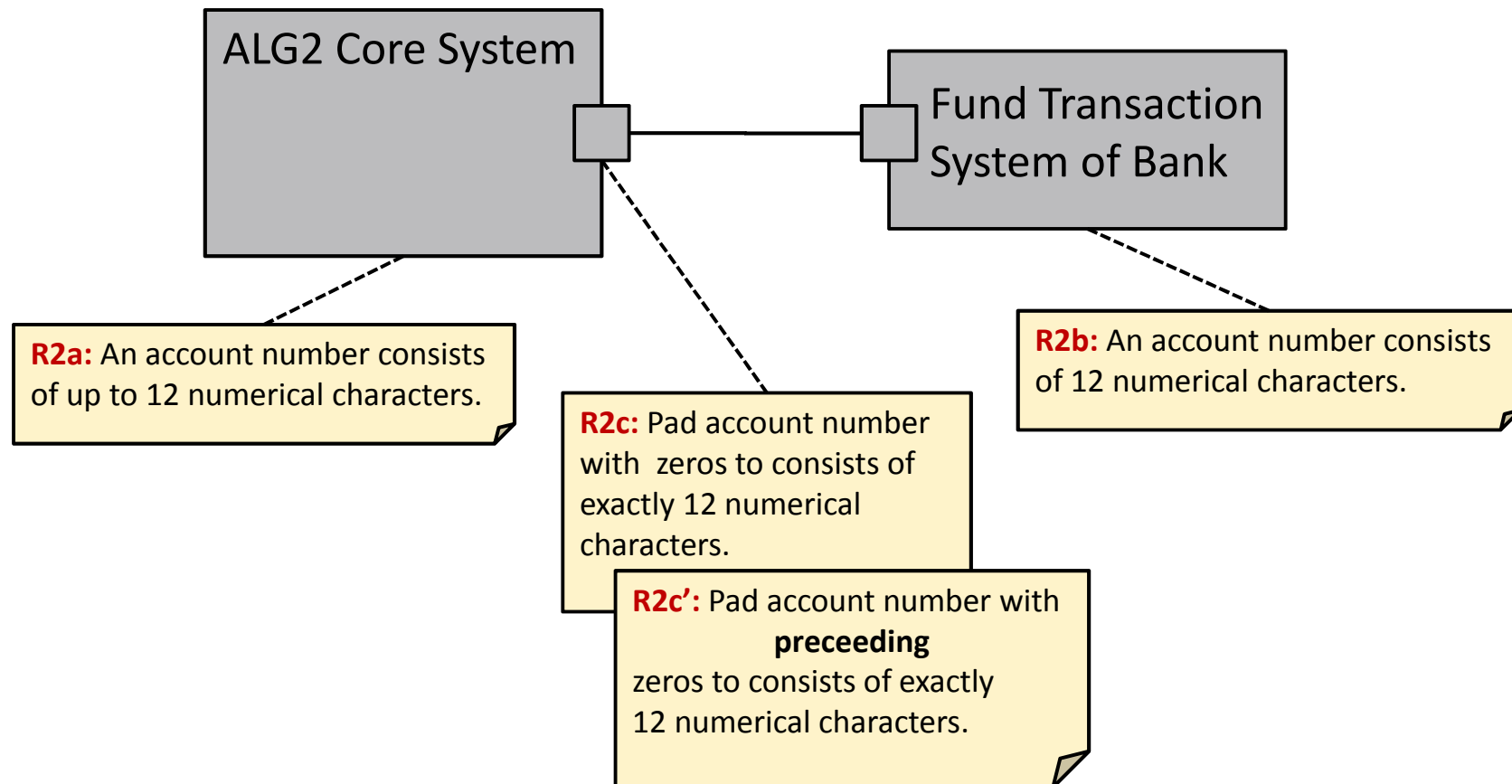


- **In 2004, a new system was put to operation to handle unemployment benefits in Germany.**
 - This is one of the largest e-government systems in Europe: as of 2004, there were 2.6m accounts maintained by 16k users paying out approx. 1.3bn€/month.
- **To achieve uniform length of account numbers, a post-processing system padded numbers with zeros – but at the wrong end!**

12345	→ 0012345	✓	12345	→ 1234500	✗
123456	→ 0123456	✓	123456	→ 1234560	✗
- **A lot of fund transfers failed and the benefits did not reach their intended recipients.**
 - When sending out the money as cash cheques by paper mail, it was discovered that another system truncated the street name in a way that many of the letters couldn't be delivered.

ALG-II Padding Error

- The problem could have been caused by a simple programming mistake, but what really happened is that a thoughtless programmer interpreted an underspecified requirement wrongly.



DB Datacenter breakdown



The complete IT backbone of Deutsche Bahn broke down on 14.1.2009 leaving tens of thousands of travelers stranded.

21	: Zug fällt aus	Zug fällt
7	: Zug fällt aus	Zug fällt
4	: Zug fällt aus	Zug fällt
20	: Zug fällt aus	Zug fällt
11	: Zug fällt aus	Zug fällt
8	: Zug fällt aus	Zug fällt
3	: Zug fällt aus	Zug fällt
6	: Zug fällt aus	Zug fällt
7	: Zug fällt aus	Zug fällt
23	: Zug fällt aus	Zug fällt

- On the 14.1.2009, the complete IT backbone of the German national railway system gradually broke down and couldn't be restarted.
- It took two days before train traffic would be back to normal.
 - During routine maintenance works, the power supply for a regional data center in Berlin was cut off crashing the entire facility.
 - Other systems and eventually the overall network failed domino-style.
 - In the end, there were no more ticket sales via any channel, and no more train and travel information.
- The problem turned out to be that the crash had left corrupted data in some data bases that crashed other dependent systems and prevented a restart.
- Apparently, nobody had ever considered the case of a restart.

- Adding (and satisfying) either of the following requirements would have avoided the problem.

R3: The regular operations personnel shall be able to start or restart the system in less than 1h.

R3': Starting the system may not depend on the status of other systems or the integrity of the system's input data.

R3'': There is a function to roll back to the last consistent state in less than 15 minutes.
The last consistent state is never older than 1h.

- But very likely, the system has grown during operation over many years in an unstructured way.
- Probably, it has never been planned and analyzed systematically.

The Tacoma Narrows Bridge (1940)

Puget Sound, Washington State, USA



Comparison of Accidents

	LH2904 (Warsaw)	Therac-25	Ariane 501	ALG-2	DB Datacenter
Material Damage	2 dead 68 hurt	3 dead many injured	500mio US\$	Many thousands without cash	Many thousands stranded
Immaterial Damage	considerable	massive	considerable	massive	massive
Operator Fault	partially	yes	no	no	no
Software Fault	no	partially	no	yes	no
RE Fault	yes	yes	yes	yes	yes
Avoidable by Ideal RE	yes	yes	yes	yes	yes
Easily/ Cheaply Avoidable	partially	yes	yes	no	no

Why do these accidents happen?

- There are two possible sources of problems that might lead to this kind of accident.
- On the one hand, these accidents could show the limits of our scientific understanding of the world.
 - It could be, that we simply could not know this, that we simply could not possibly have prevented these accidents from happening, and that we had no way of limiting or mitigating their effects.
- On the other hand, these accidents could be due to embarrassingly mundane and simple, if not downright stupid, errors.
 - Almost always, this is the case.
 - *“One obvious lesson is that most accidents are not the result of unknown scientific principles but rather of a failure to apply well-known, standard engineering practices. A second lesson is that accidents will not be prevented by technological fixes alone, but will require control of all aspects of the development and operation of the system.”*

[Nancy Leveson: "Safeware: System Safety and Computers" Addison-Wesley, 1995]

Requirements Engineering Is Hard

- 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.
- Therefore, the most important function that the software builder performs for the client is the iterative extraction and refinement of the product requirements. For the truth is, the client does not know what he wants. The client usually does not know what questions must be answered, and he has almost never thought of the problem up the detail necessary for specification.
- Even the simple answer—"Make the new software system work like our old manual information-processing system" —is far too simple. One never wants exactly that. Complex software systems are, moreover, things that act, that move, that work. The dynamics of that action are hard to imagine. So in planning any software-design activity, it is necessary to allow for an extensive iteration between the client and the designer as part of the system definition.
- I would go a step further and assert that it is really impossible for a client, even working with a software engineer, to specify completely, precisely, and correctly the exact requirements of a modern software product before trying some versions of the product.



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 1.2:

Fighting Software Failures through Requirements Engineering

DTU course 02264

Why bother with RE?

- **Today, software is ubiquitous.**
 - Many consumer devices and appliances are really computers; contemporary vehicles and machines are mainly embedded systems; all critical infrastructures highly depend on IT systems, virtually *everything* today contains a processor.
- **Therefore, its quality (or lack thereof) affects us tremendously.**
 - The potential consequences range from minor inconveniences to global disaster.
- **The most cost-effective way to achieve better quality is via RE.**
 - This fact has been proven again and again over the last 30 years, but it is still not generally acknowledged.
- **RE by itself is no solution, but every solution will contain RE.**
 - It may be called differently or hidden behind something else, but is there.
- **Therefore every person involved in creating software-intensive systems must be knowledgeable in RE.**

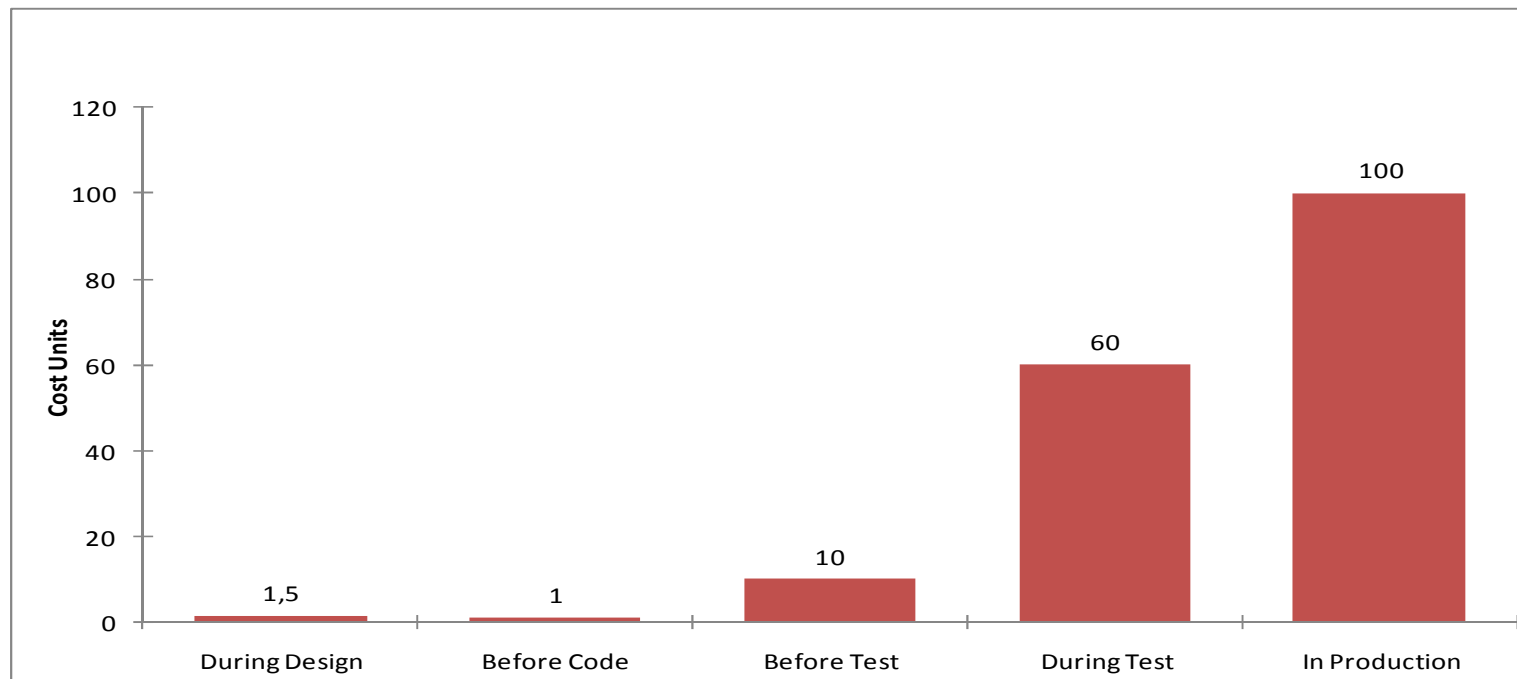
Better Software is expensive

- **Creating (better) software is expensive and only makes sense economically, if it significantly exceeds the likely cost.**
- **The most cost-effective way to achieve better quality software is almost always through systematic Requirements Engineering.**
 - Implementing the right requirements necessarily has a better Return on investment (ROI) than implementing the wrong requirements, no matter how cheap that is.
 - Also, there have been many studies on the economic benefit of RE over the last 30 years, and the result was always in favor of RE.
 - Of course, RE can still fail, diverge, or be overly expensive.
- **Looking at the way software is produced, it is obvious that projects fail mostly due to lack of and faults in Requirements Engineering.**

Cost/Benefit of RE

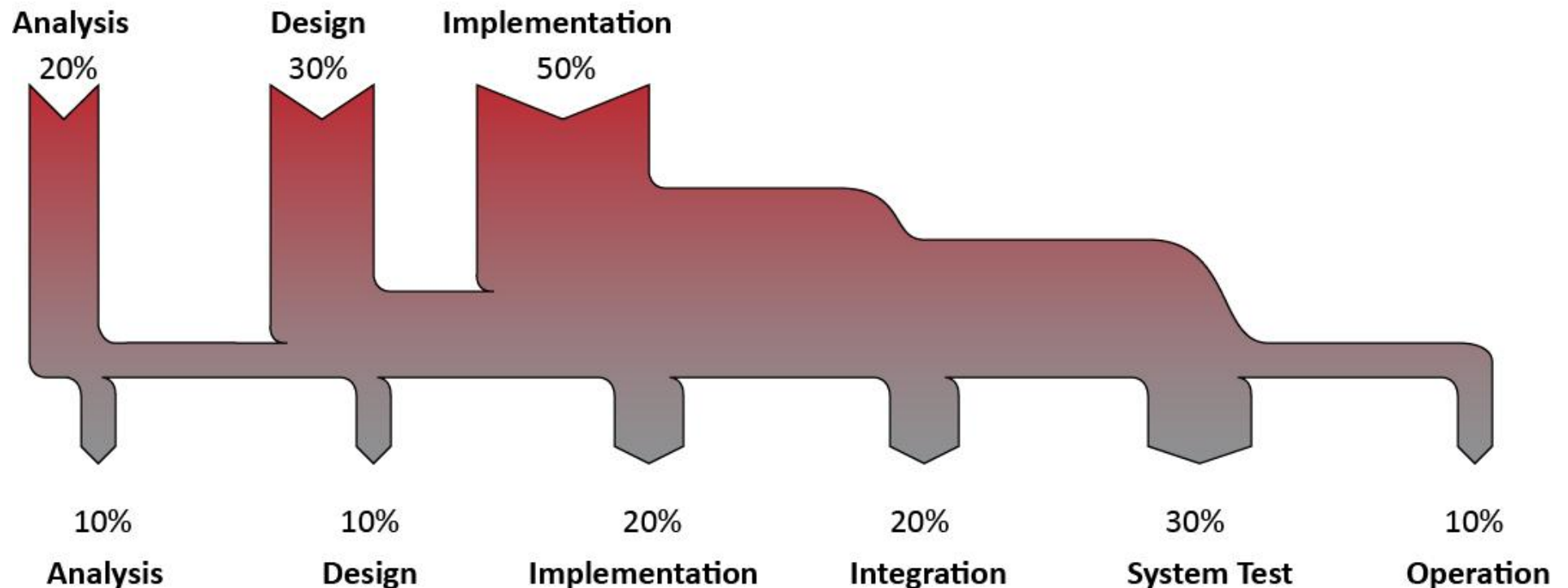
- Professional RE may be expensive, but it is still cheaper than not doing it: many empirical studies have confirmed that RE has a (large) positive ROI.

“Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase.”
[Boehm, Basili: Software Defect Reduction Top 10, S. 135]



Reason for Cost/Benefit of RE

Introduction of Faults

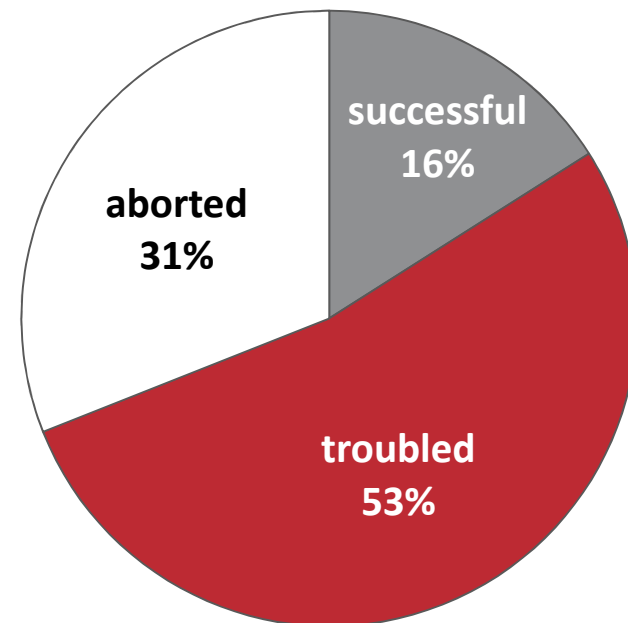


Fault Recognition and Removal

The longer an error remains undetected the more follow up errors and aftereffects it causes, and the more it costs.

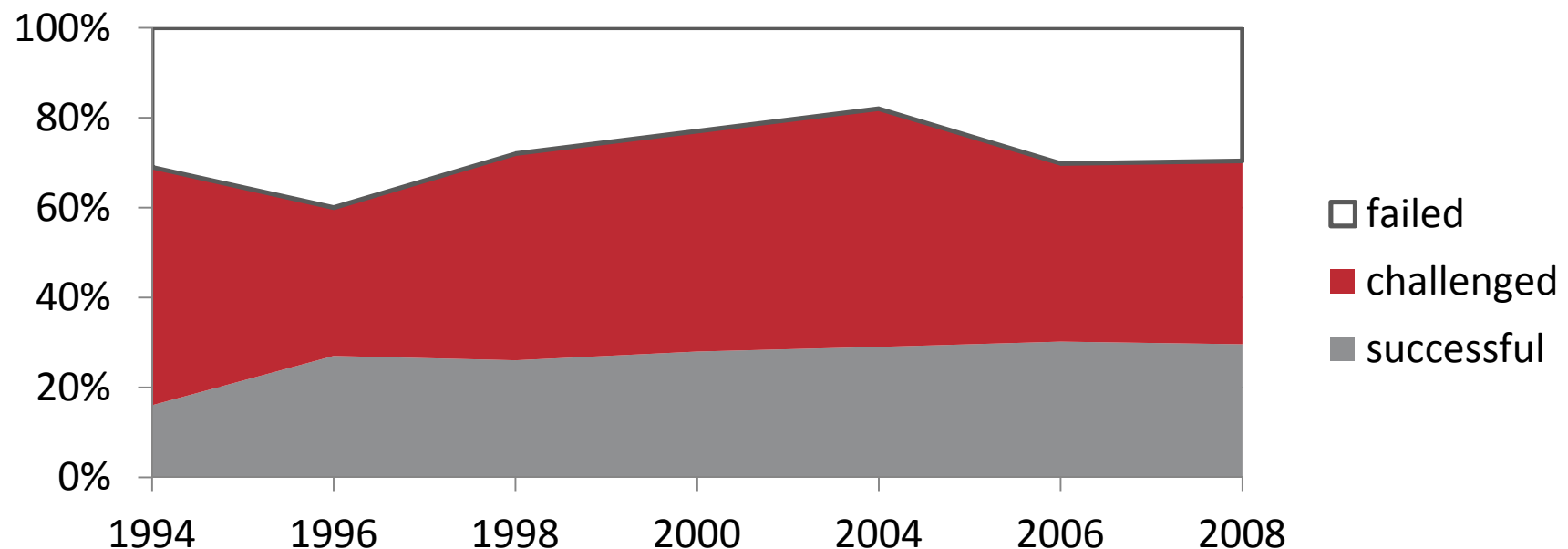
Another factor for RE efficiency

- Since 1994, the consultancy Standish Group surveys the software project market every two or three years.
 - Their results are published in a paper aptly called the CHAOS report.
- In 1994, 250.000 projects have been surveyed.
 - They exhibited a large average overrun of cost (~800%),
 - and time (~222%).
 - Almost 50% of all projects surveyed had more than 100% time overrun.
- A project is considered successful, if it delivers
 - acceptable functionality and quality
 - in time and on budget.

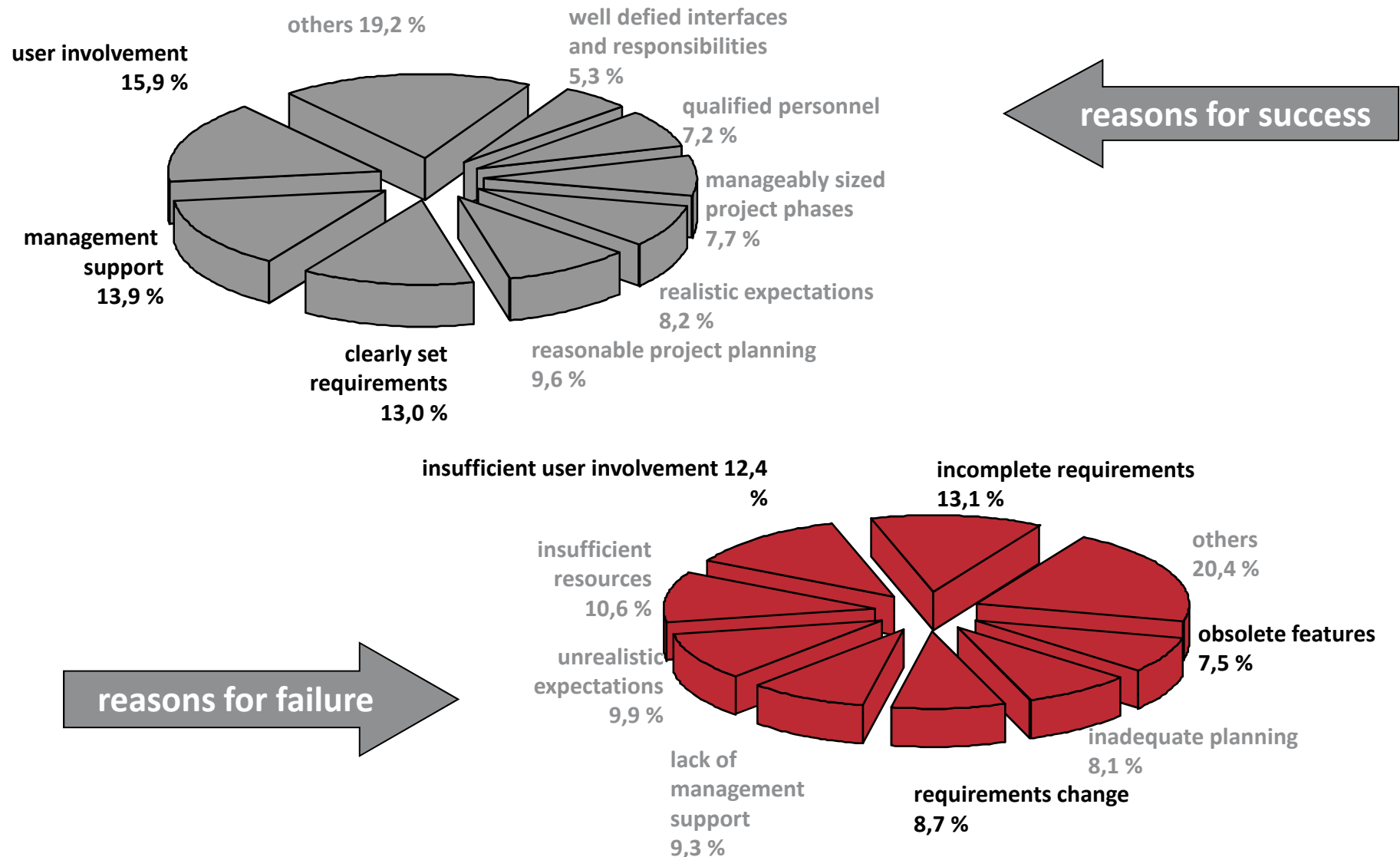


The CHAOS report over time

- **At first sight, the situation seems more or less stable since 1994.**
 - When looking closer, a slight improvement can be detected.
 - This is probably just a statistical artifact, but we cannot find out for sure, since the report's methodology is not known in sufficient detail.
 - In fact, there has been substantial critique concerning the validity of the CHAOS report's methodological soundness.



Requirements are a key factor



Benefit of Requirements Engineering

- Requirements Engineering as a discipline is effective and efficient in creating high quality software that does address the client needs.
- RE is also effective for making projects successful and, if lacking, a main factor to failure.
- **This fact is still not generally acknowledged.**
 - E.g. in the last years many people in the agile camp deny the necessity of requirements, notwithstanding the fact that they do quite a lot of RE activities, if only under a different name.
- **A rational and mature discipline of Software Engineering will incorporate a lot of RE; RE by itself is no solution, but every solution will contain RE.**

Benefit of RE

- **The benefit of professional Requirements Engineering to preventing software faults is very large.**
 - It is probably the single largest contribution,
 - it is cost-effective (i.e., relatively cheap), and
 - it is no magic – even you can do it! :-)
- **In fact, no sizable piece of software can be created without handling the requirements adequately.**
- **However, Requirements Engineering is a necessary condition for fault-free software, but not a sufficient condition!**
 - Other things have to come in, too, like Software Process, Architecture, and, of course Coding and Testing.

Alice: Would you tell me, please, which way I ought to go from here?

The Cat: That depends a good deal on where you want to get to.

Alice: I don't much care where.

The Cat: Then it doesn't much matter which way you go.

Alice: ...so long as I get somewhere.

The Cat: Oh, you're sure to do that, if only you walk long enough.



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 1.3:

A working definition of Requirements Engineering

DTU course 02264

Example: how RE reduces complexity

Goals

We want to do precise symbolic computations like differentiation. Thus we need rational numbers (\mathbb{Q}) instead of floating point numbers.

Requirements

Features: Normalize, add, multiply, negate, convert to Floats, print, ...
Qualities: Degree of precision and performance
Constraints: unskilled developers, time pressure

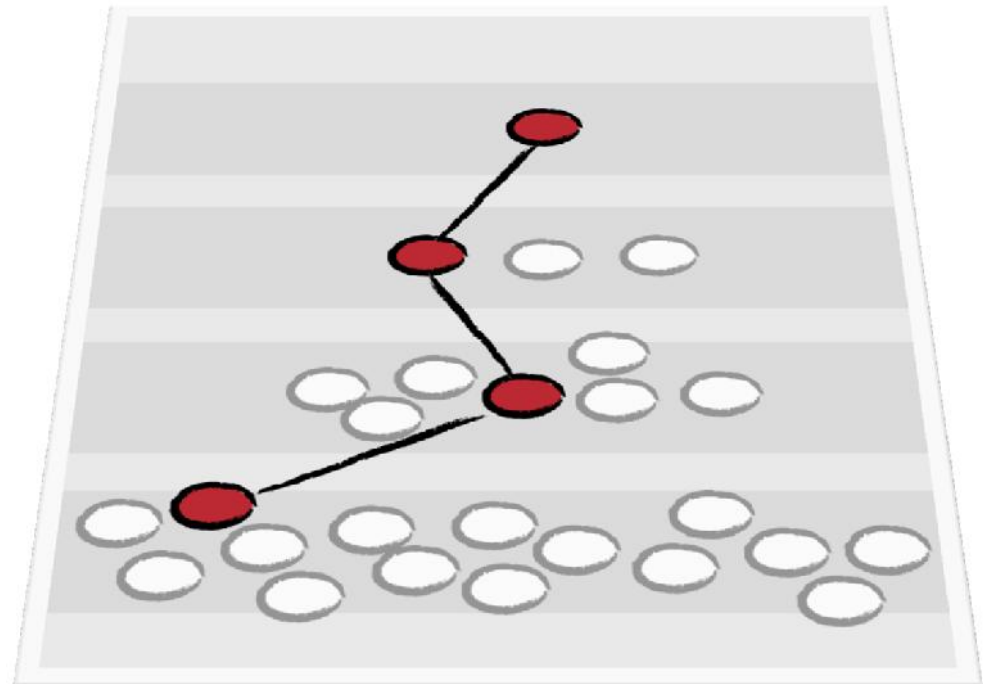
Design

Represent fractions as pairs of INT with the convention that the denominator is always positive.
Use Euklids algorithm for normalization

Implementation

Choose data & control structures, identifier names, messages, exceptions, tests, ...

There is an enormous number of alternatives to consider. It is impossible to find the “right” solution by just looking at that set.
A stepwise process of successive decisions makes the choices explicit, thus rational, traceable, and accountable. This way, the process becomes much easier.



RE terms and methods

Goals

Oriented towards company strategy and the competition, focuses on the market and application domain

Requirements

Oriented towards product or product line, its subsystems and components, or individual functions and properties

Design

Oriented towards solution, taking into account The solution space (i.e. technology).
May contain feasibility studies.

Implementation

The running systems: all decisions are fixed
Change incurs substantial effort

Prose and Diagrams

Multimedia, PowerPoint
Informal, communication oriented
Create and innovate

Structured Text

Controlled Languages, tabular text schemas, weak modeling languages with little formality
Capture and describe

Modeling Languages

UML, ARIS/EPC, RIVA/RAD, SDL/MSD, Matlab
Potentially full, but de facto often little formality
Elaborate and specify

Programming Languages

Java, C#, Cobol, PL/1
Executable (i.e. fully formal)
Formalize and determine all details

Positivistic view

"The choice of functional specifications [...] may be far from obvious, but their role is clear: it is to act as a logical firewall between two different concerns.

The one is the 'pleasantness problem', i.e. the question of whether an engine meeting the specification is the engine we would like to have; the other one is the 'correctness problem', i.e. the question of how to design an engine meeting the specification.

I firmly believe that whenever we succeed in erecting such a firewall, the effort will pay off handsomely. The reason for this belief of mine is that the two problems are most effectively tackled by totally different techniques."

[Edsger W. Dijkstra:
On the Cruelty of Really Teaching Computing Science,
CACM 12/1989 vol. 32 no.12 pp. 1398-1414]

Wishes & Needs

"Pleasantness Problem"

*Elicitation & Analysis
→ doing the right thing*

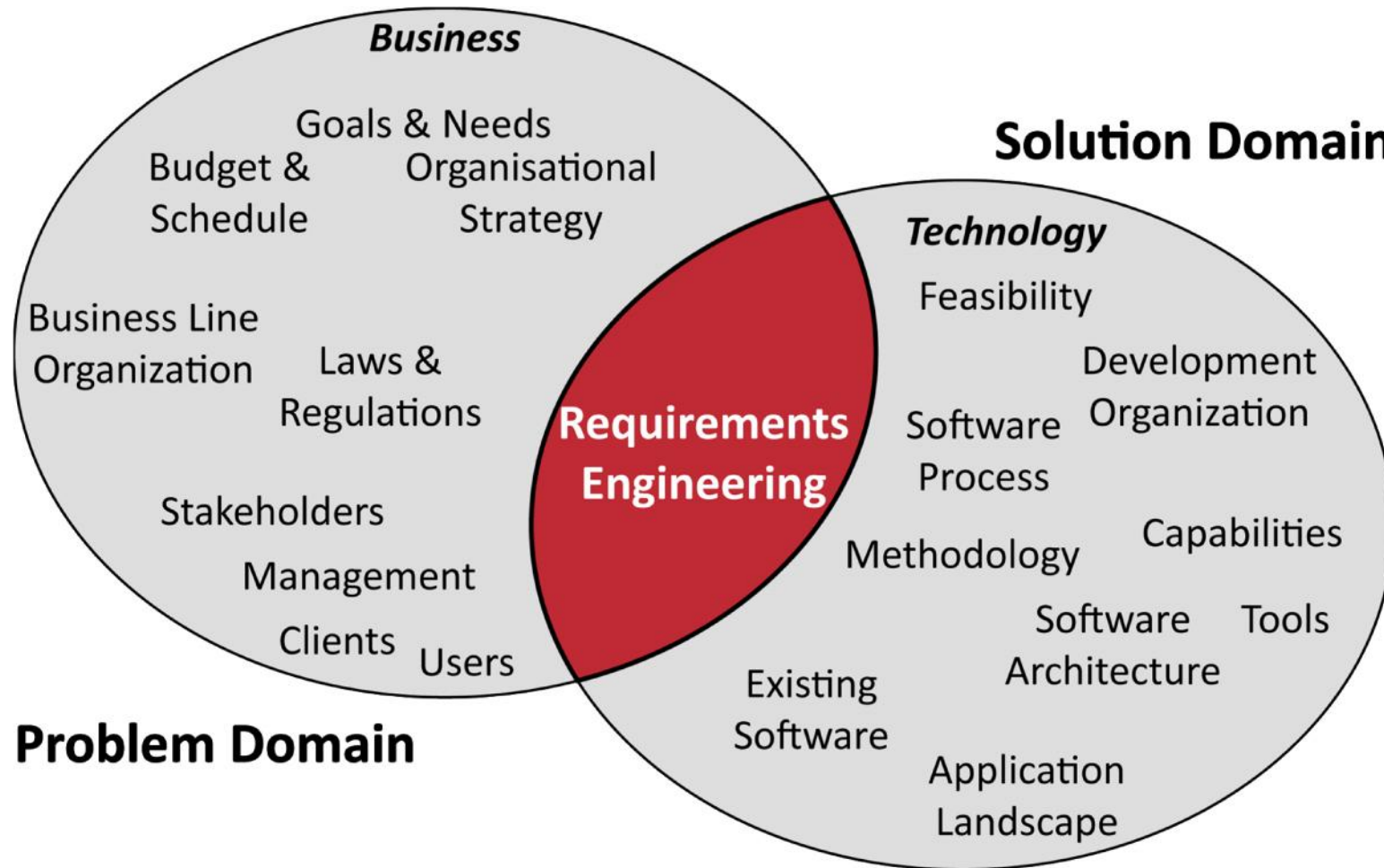
Firewall
(Specification)

Implemented System

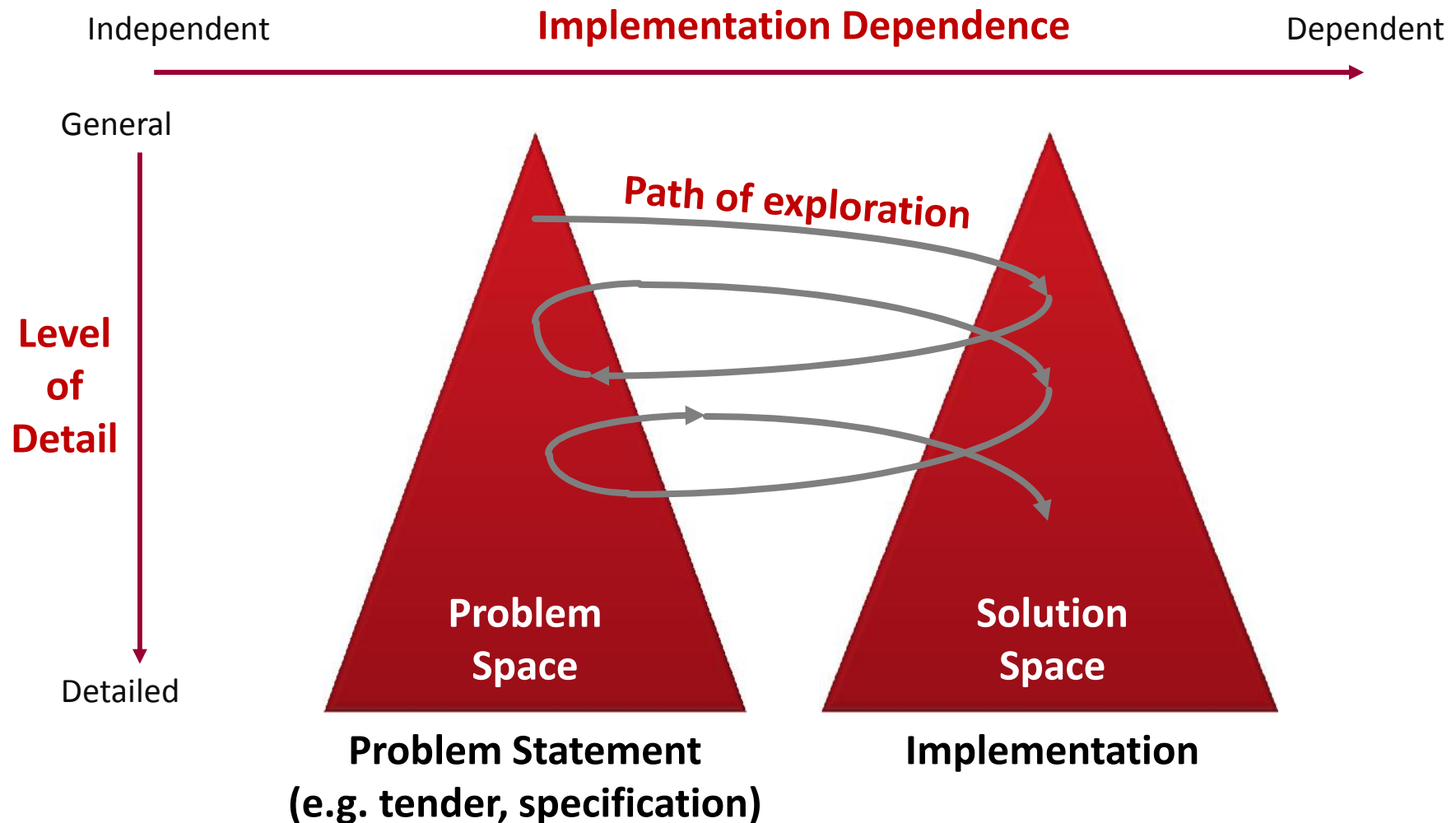
"Correctness Problem"

*Design & Implementation
→ doing it in the right way*

Locating Requirements Engineering



The “Twin Peaks” View



Definition „Requirement“

- A **Requirement** is a statement asserting a desired property of a product, process or the people involved in a process.

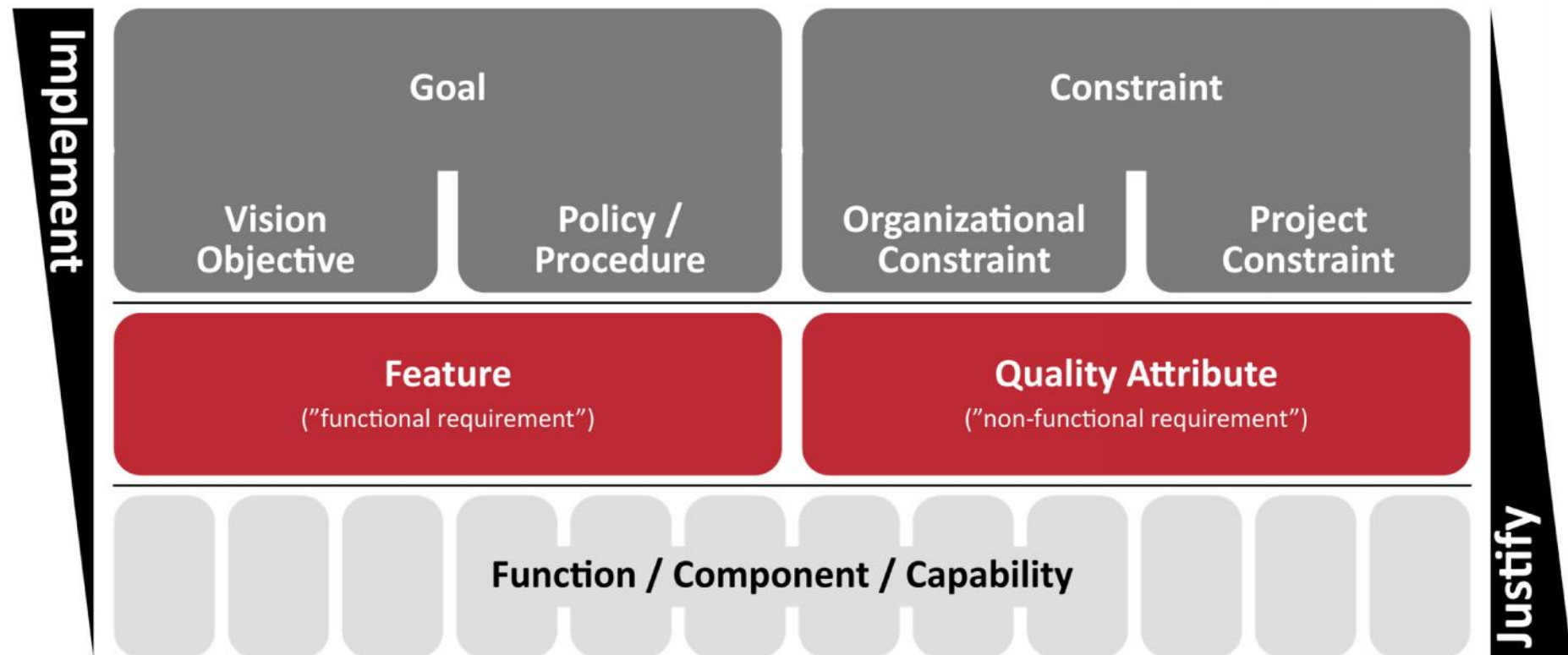
Requirement

- (1) A condition or capability needed by a user to solve a problem or achieve an objective.
- (2) 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 documents.
- (3) A documented representation of a condition or capability as in (1) or (2).

Requirements Analysis

- (1) The process of studying user needs to arrive at a definition of system, hardware, or software requirements.
- (2) The process of studying and refining system, hardware, or software requirements.

- The term „Requirement“ is used in many different contexts, and with many different meanings.
 - All of the following are (sometimes) called „Requirement“.
 - We will use „Requirement“ only in reference to those notions in the red boxes in the middle layer.



Definition “Requirements Engineering”

Not a phase
or stage!

Requirements Engineering (RE)

is a **set of activities** concerned with identifying and communicating the **purpose** of a software-intensive system, and the **contexts** in which it will be used.

Communication
is as important
as the analysis.
RE means
“organizing truth”.

The purpose determines the qualities and features of a system :
“form follows function”.

Designers need to know how and where the system will be used

Hence, RE acts as the bridge between the **real world needs** of users, customers, and other **constituencies** affected by a software system, and the **capabilities and opportunities** afforded by software-intensive technologies.

Requirements are partly about what is needed...

...and partly about what is possible

Need to identify all the stakeholders, not just the customer and user

Purpose of RE

Successful Requirements Engineering can contribute to a project success in four different ways.

- **Knowledge**

- Doing requirements engineering creates (or compiles) knowledge necessary to achieve the objective, including finding out what the objective is.

- **Control**

- Non-crosscutting features and quality attributes can be used as a unit of project planning and control in a very straightforward way. Most „agile“ development methods (e.g., XP, Scrum, FDD, and DSDM) critically rely on this RE usage.

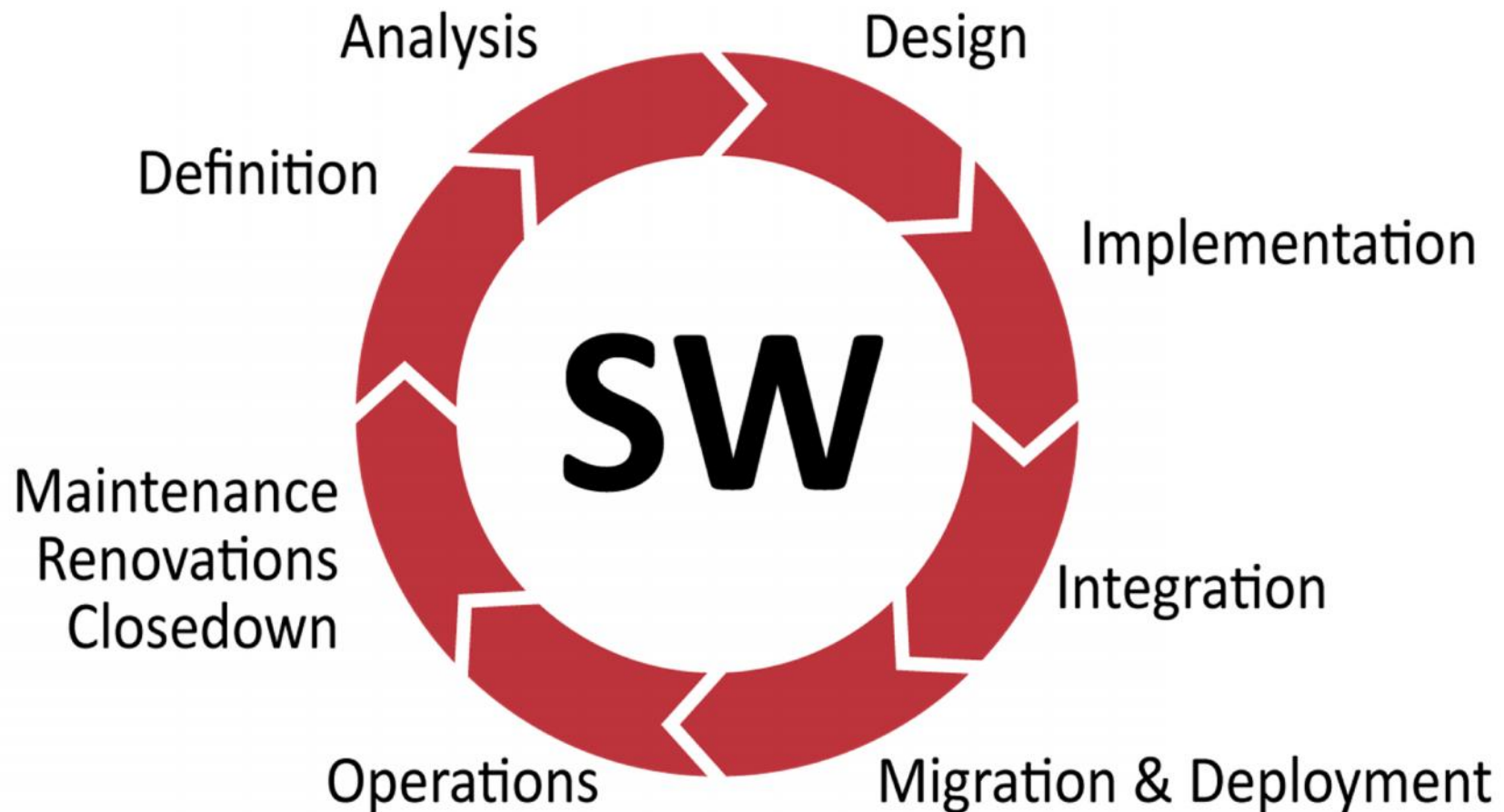
- **Contract**

- If the specification is written down in great detail and quality, it can be used easily in contractual agreements between client and supplier.

- **Consensus**

- If everybody agrees on an issue, there is no need for a specification. If people disagree, even a written and signed contract can be taken to court. With trust, knowledge is dispensable, only distrust requires knowledge.

The System/Software Lifecycle



In all phases, RE activities are necessary

The Early Software Lifecycle in Detail

Analysis (also: Requirements Analysis)

Understand, validate, and specify:

- the customer's goals and needs, and the system vision
- the system context, stakeholders, and domain architecture;
- the project's constraints and the system's requirements;
- the application domain, its entities, behaviors, and interactions.

Project Definition (also: Inception)

- Formulate a project vision, communicate it, and
- gain support from project sponsors.
- Conduct feasibility/viability studies for vital parts.
 - Select and tailor a development method/process,
 - set up a tool chain, and
 - create initial versions of project guidelines.
- Draft an initial realization plan.





Prof. Dr. Harald Störrle

Software Engineering Section
Department of Informatics and Mathematical Modeling
Technical University of Denmark
Richard Petersens Plads
Building 322, Room 024
DK-2800 Kgs. Lyngby

`hsto@imm.dtu.dk`

`www.imm.dtu.dk/~hsto`

