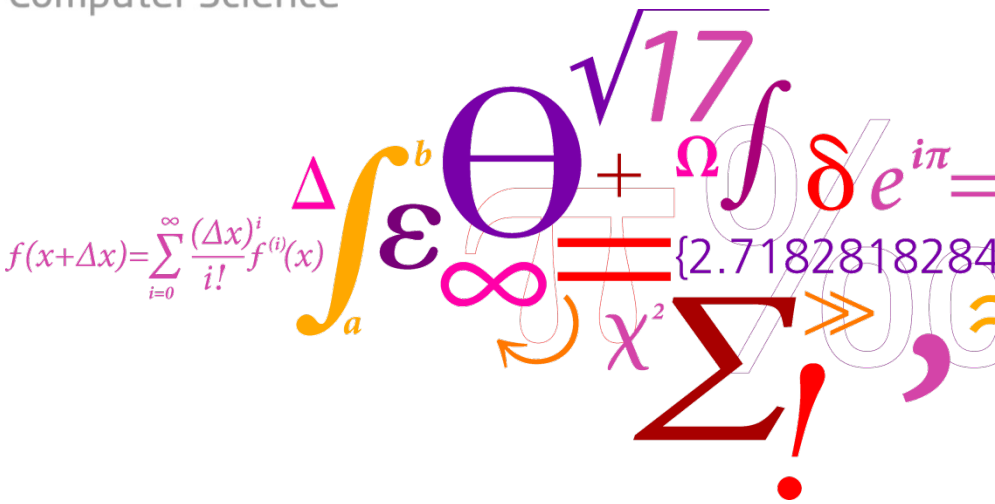
Software Engineering 2

A practical course in software engineering

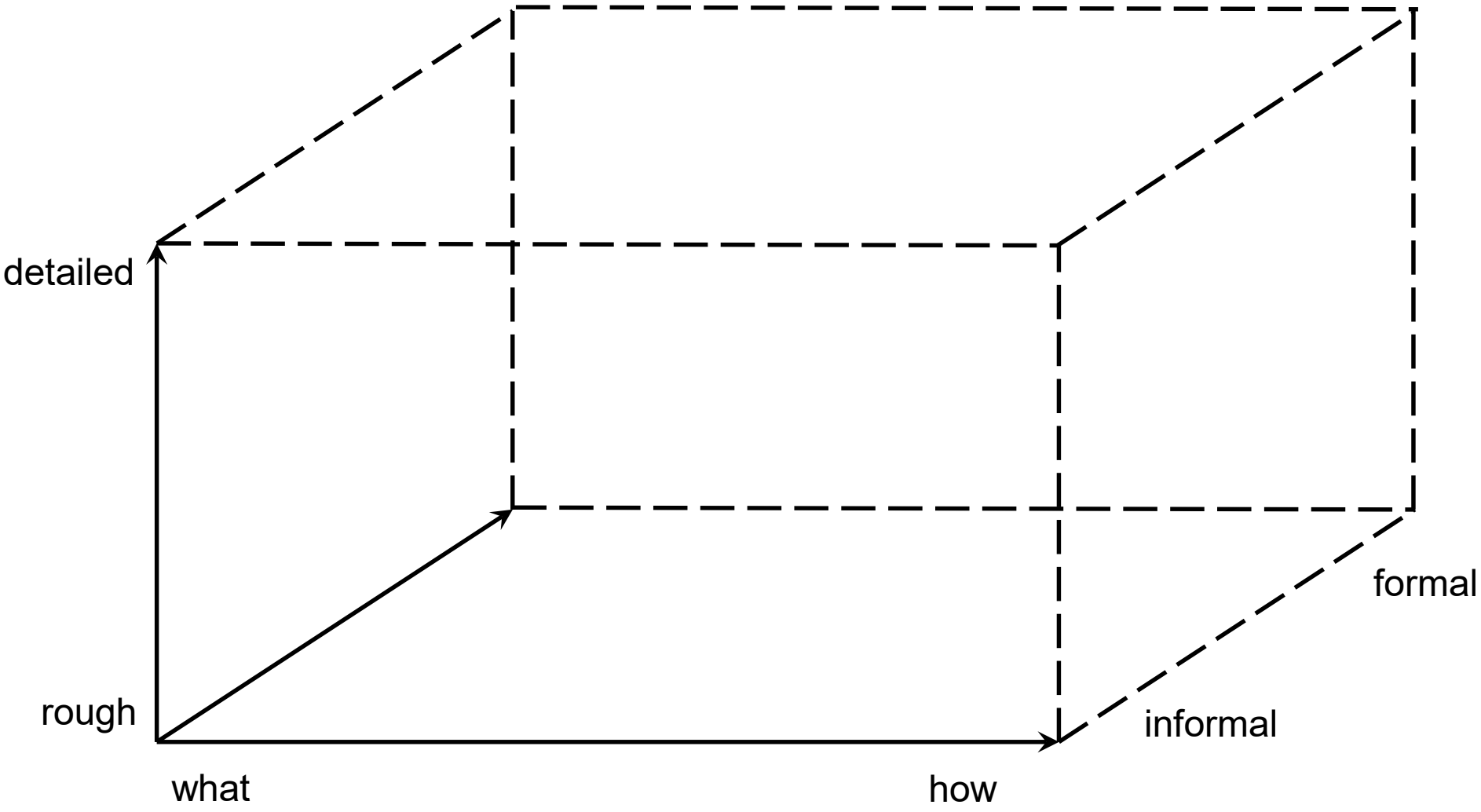
Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science



Dimensions

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Example documents

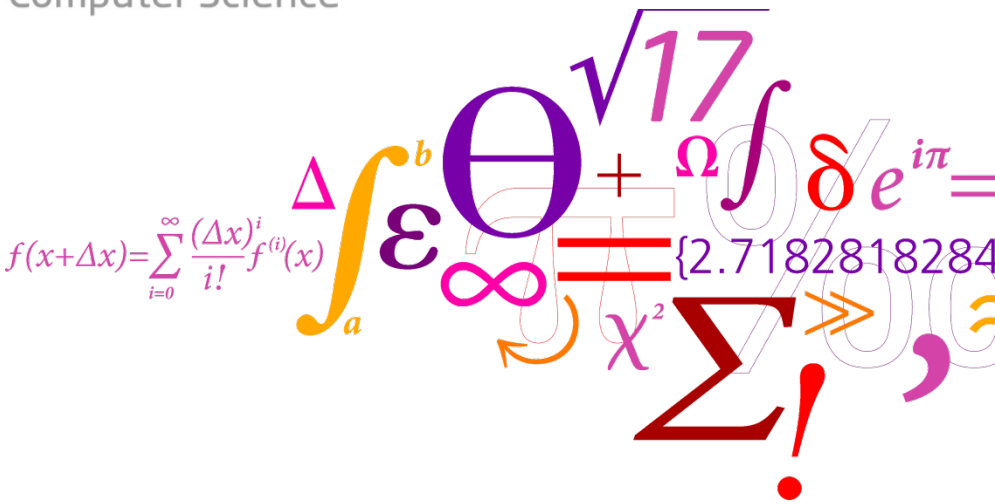
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Product objective
- Product use
- Use cases
- User story
- Domain model
- Code (implementation)
- Test
- Prototype
- GUI definition
- GUI mockup
- ...
- Design
- Architecture
- Data base schema
- XML Schema
- OOA
- OOD
- Systems specification
- Requirements specification
- Formal model
- Handbook

II. Agile Development

See lecture 2 (and 8)!
But some additional slides

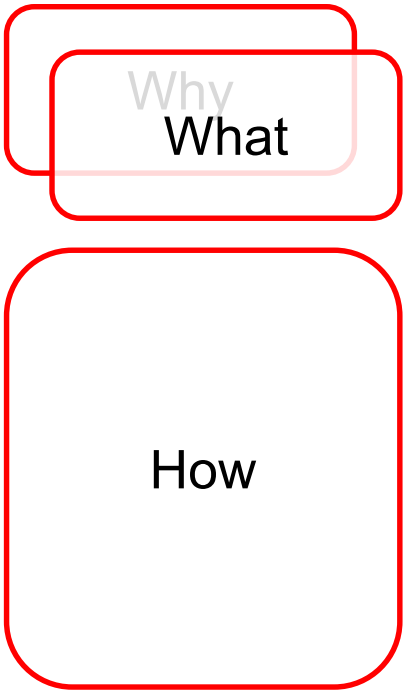
DTU Compute
Department of Applied Mathematics and Computer Science



1. Motivation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

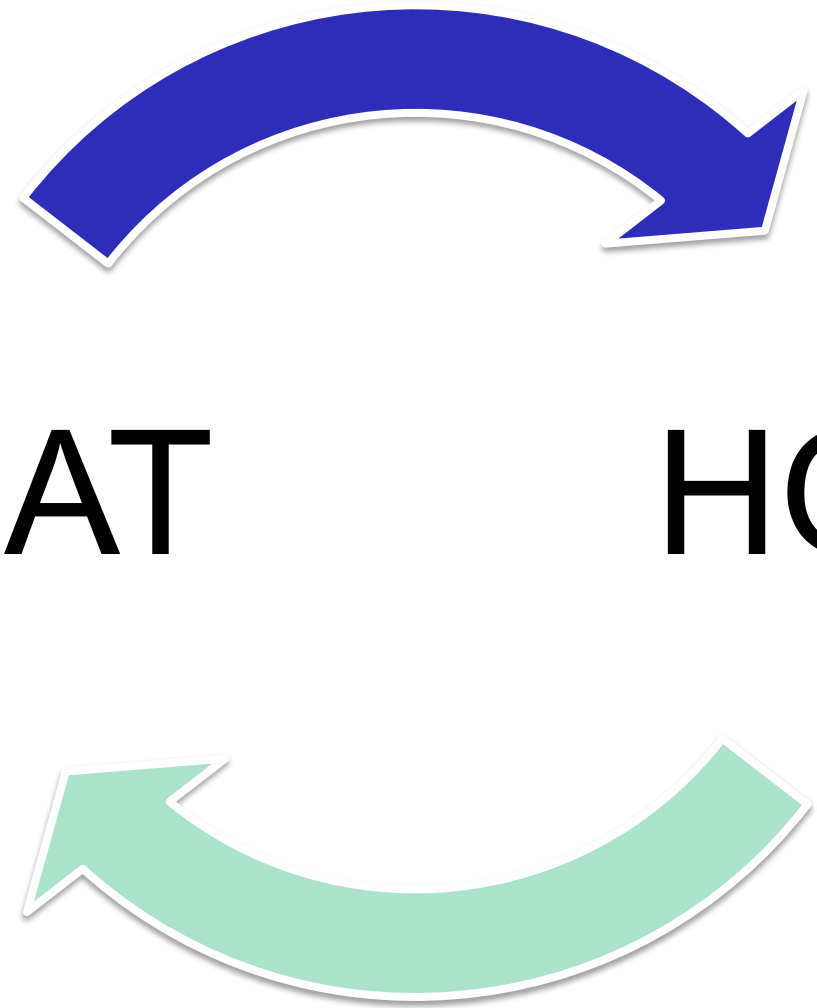
Conceive
Design
Implement
Operate



Co-evolution

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

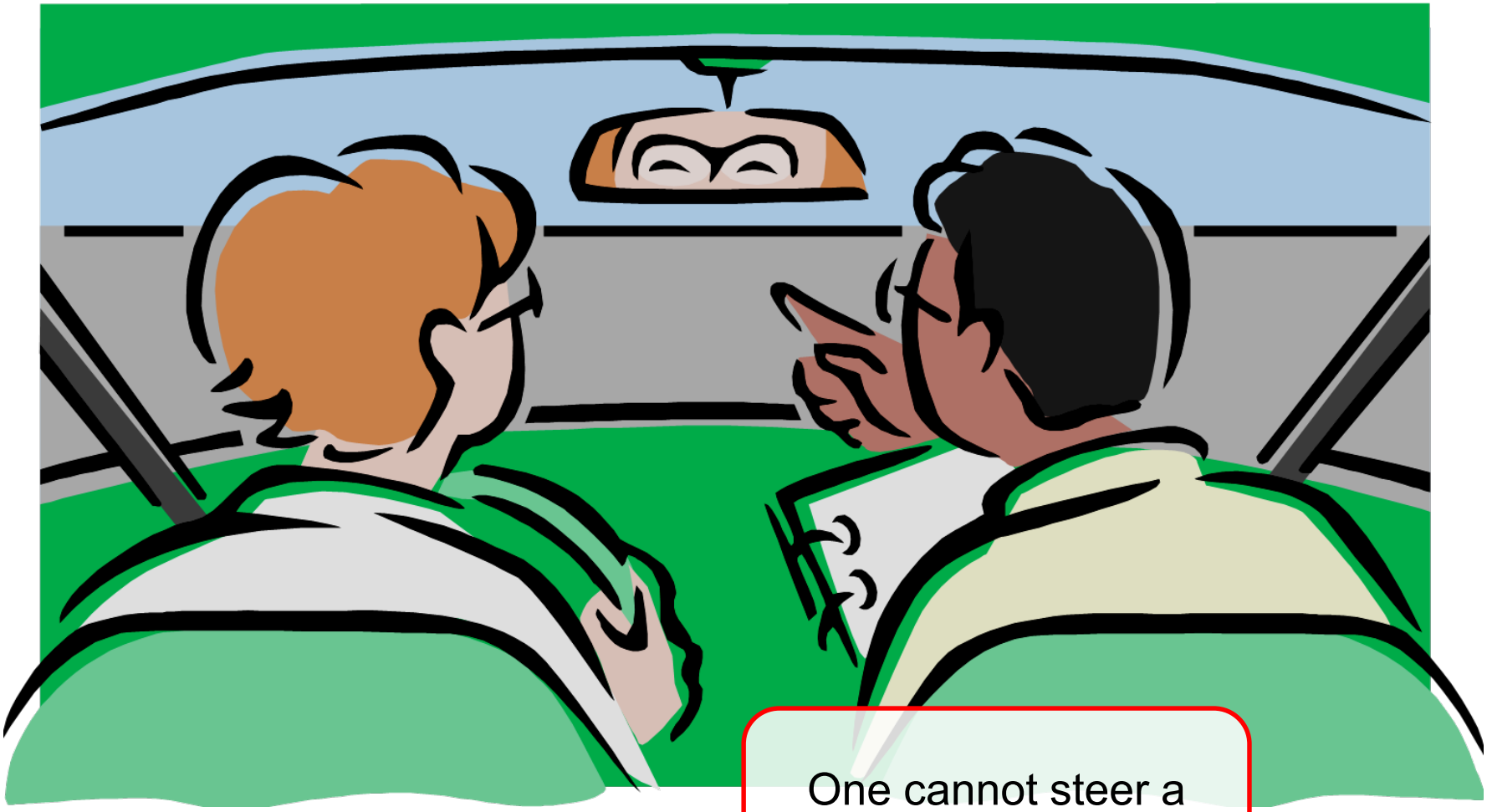
"What" should the software do?



"How" is it realized?

Driving a car

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Gall's law

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

"A complex system that works is invariably found to have evolved from a simple system that worked. A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over with a working simple system."

John Gall: Systemantics: An essay on how systems work, and especially how they fail. General Systemantics Press, Ann Arbor, Michigan, 1975.

Main message

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Experience shows (for complex systems):

- **CDIO** does not work purely sequentially
- Once we have implemented a system (**how**), we get a (much) better understanding of **what** the system should do!
- Software developments is very much about managing risk
- Development process needs adjustments while we are going

→ **Agile Software Development**

Agile manifesto

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

„We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.“

Kent Beck et al. 2001

2. Background

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Four variables of software development
- Cost of change

Agile development

- Values
- Principles
- Basic activities

The following presentation follows the book:
Kent Beck: Extreme Programming Explained, Addison-Wesley 1999.

Variables

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Four variables of software development process

- Cost
- Time
- Quality
- Scope

Basically, three variables can be freely chosen (e.g. by customer). The fourth is a result of the pick of the three (i.e. decided by develoment team).

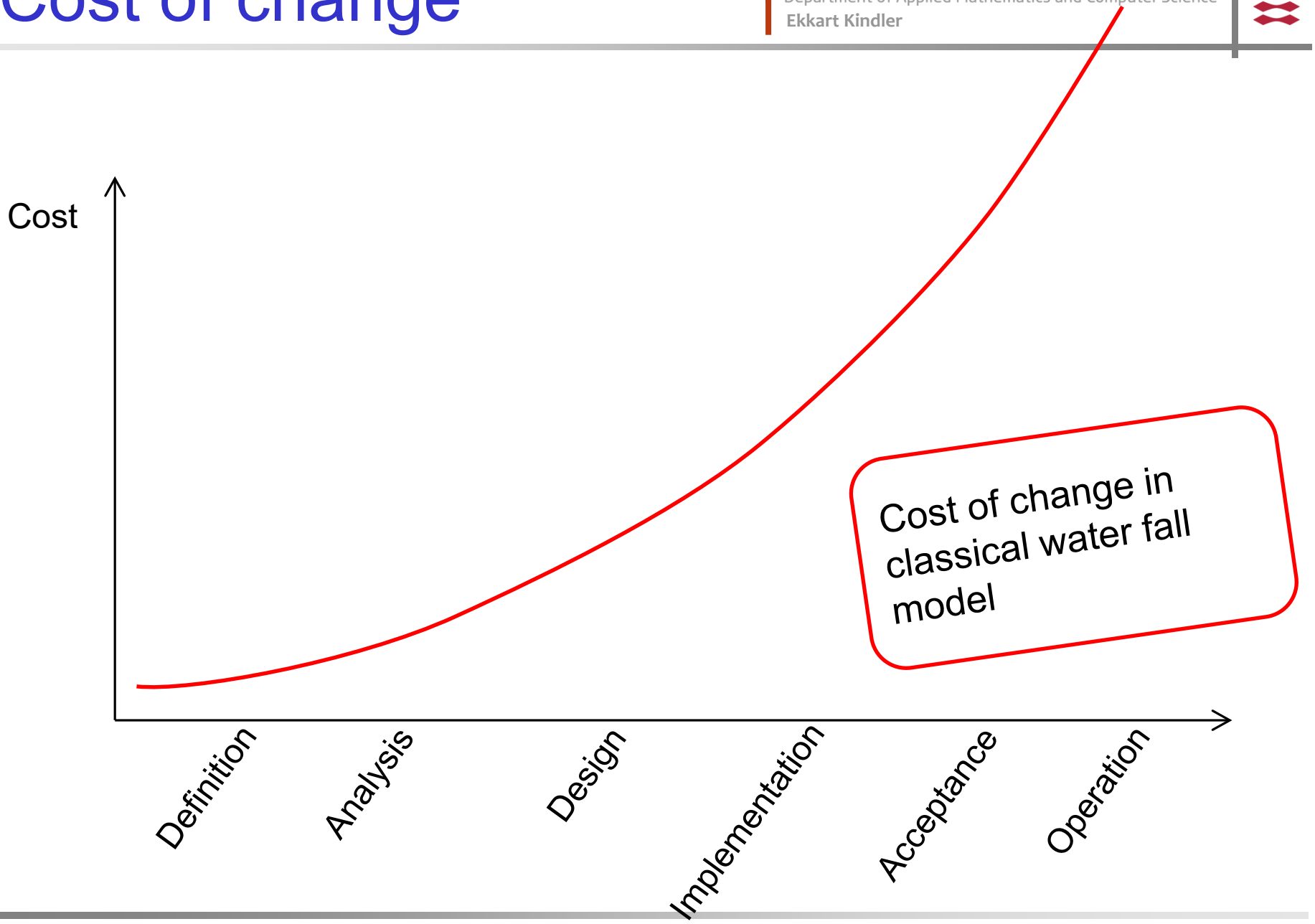
Note

- There is no way the customer could pick all four variables
- Even three variable are not fully independent
- More developers do not necessarily speed up the process (and not in a linear way)

- Agile helps adjusting these variables dynamically (and get the most value for the customer):
  - Short iterations
  - More practise in estimating user stories (developers)
  - Prioritization of user user stories (customer)

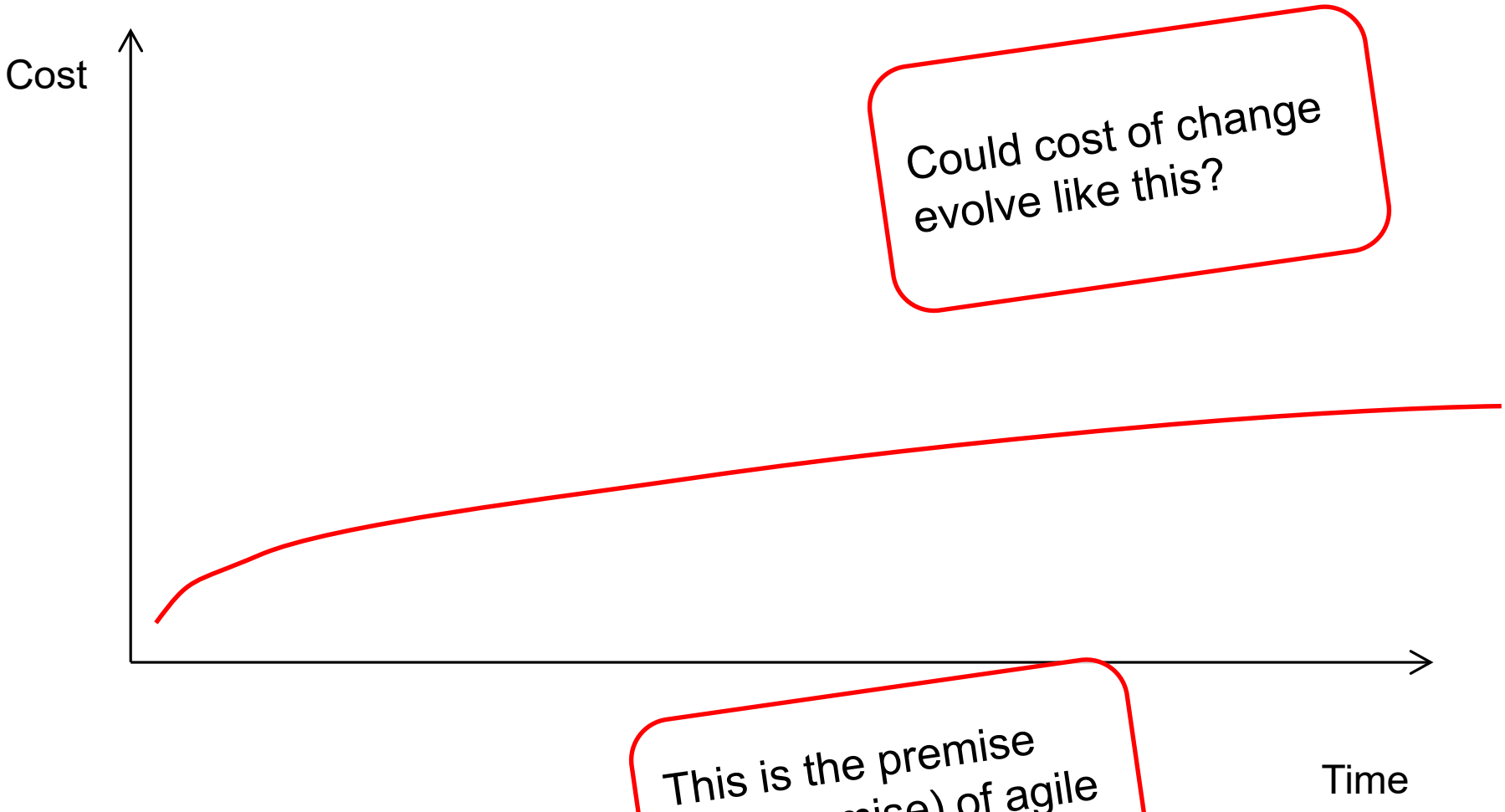
Cost of change

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Cost of change

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Agile Values

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

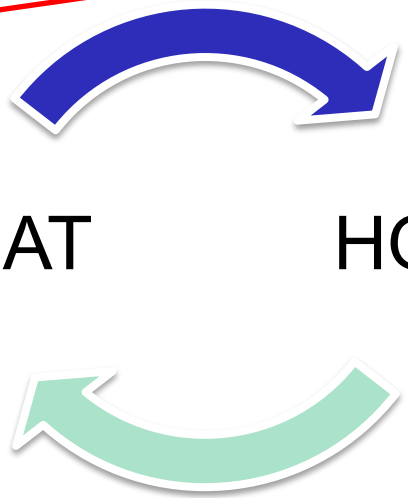
- Communication

Talking

- Simplicity

Better a simple thing that serves the purpose today, than a complicated thing that might never serve a purpose

- Feedback



But a lot more than that

- Courage

Agile Principles

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Values give some orientation and criteria for a successful agile development
- But, values are too vague for defining concrete practices

→ Therefore, agile builds on principles

Fundamental Principals

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Rapid feedback
- Assume Simplicity
- Incremental change
- Embracing change
- Quality work

Other principles

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Teach learning
- Small initial investment
- Play to win
- Concrete experiments
- Open, honest communication
- Work with people's instincts, not against them
- Accepted responsibilities
- Local adaptation
- Travel light
- Honest measurements

Activities

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Coding
- Testing
- Listening
- Designing

Agile development and practices help structuring these activities, so that you can keep doing them indefinitely.

3. Agile Practices (overview)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- On-site customer
- **Small/short releases** 2-3 week
- **Planning game**

Agile Practices

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- **Coding standards**
- **Testing**
- **Continuous integration**

Agile Practices

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- **Pair programming**
- **Simple design**
- **Refactoring**

Agile practices

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Metaphors:
  Simple story (and terminology) how the system should work; speak in simple pictures

- Collective ownership:
  Anyone can change, anyone has obligation to change (if value increases)

- 40-hour week,

Synergy of Practices

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- The different practices support each other
- One practice's weakness is the others strength
- You cannot pick/chose practices arbitrarily

Example

Short Releases would be impractical, unless

- **Planning Game** helps identifyng the user stories with most value
- **Coninuous Integration** allows deploying its with minimal effort
- **Testing** would guarante low (or desired) defect rate
- **Simple Design** allows you what is needed now

4. Details

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Planning Game
- Testing

- Organizing facilities

Pair programming

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

is **NOT**

- one person programming, another only watching
- one peson typing
- a tutoring session for the other person
- always pairing up with the same person
- programming with the buddy you best get along with

Pair programming

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

is

- communication and interaction
- getting a second opinion
- critically reviewing the others work
  - correctness
  - simplicity
  - avoid tunnel view
- learing from each other
- dynamic

Use one machine (one keyboard and one mouse) only

BTW: Reviewing is one of the most powerful techniques to identify problems and issues

If you are not an expert, ask one to join you for the task at hand

Agile Test Strategy

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Test driven development:

- **Before implementing functionality**, you write its interface and tests for it
- Unit test are **automated**, and **run every time** new code is checked in
- **Everybody** is responsible for **fixing** broken code (code which results in failed tests)

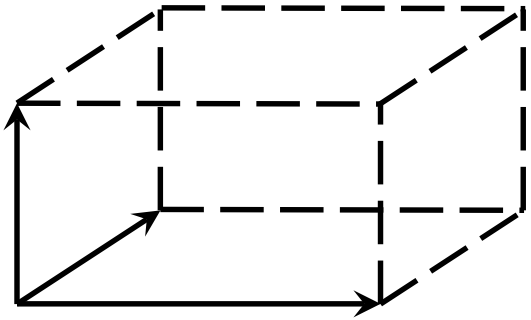
Agile Test Strategy

See lecture 8: Test Driven Development

Test driven development:

- **Before implementing functionality**, you write its interface and tests for it (tests first)

This helps with becoming clear of what should be implemented and how exactly the interface should be!
Raises level of detail and makes things technical



Agile Test Strategy

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Test driven development:

- **Before implementing functionality**, you write its interface and tests for it
- Unit test are **automated**, and **run every time** new code is checked in

Indicator of quality all the time!

Agile Test Strategy

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Test driven development:

- **Before implementing functionality**, you write its interface and tests for it
- Unit test are **automated**, and **run every time** new code is checked in
- **Everybody** is responsible for **fixing** broken code (code which results in failed tests)

100% success rate (for unit tests) is the norm!

Testing

Two sources of tests:

- **Programmers**:
  Unit tests for everything which potentially or likely could be wrong (or which went wrong at a some time)
- **Customers** (maybe implemented by programmer):
  Functional tests for user stories

Business/Development

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

In agile,

- **business drives** decisions,
- which are **informed by development** (implications, cost risk)

Developers work on the impossible and with hardly understood risks and implications!

Too much focus on technology instead of business value

Too much power for either business or development:

too much effort and risk for too little value

Organizing Facilities

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Group work and group / meeting:
  - Arrangement of tables
  - Use projectors and blackboards
  - Make status of current user stories and tasks visible
  - Show metrics (tests, completion status)

  - Be clear about: Are you having a group discussion or are you working individually (in pairs)
  - Make sure everybody knows what their tasks are until the next meeting

Reminder from lecture 2

Development Strategy

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Continuous Integration:

- **All coded** is **integrated** within hours
- In a release all tests run 100%
- If a unit test fails, there is nothing more important than fixing it

Reminder from lecture 2

Development Strategy

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Collective Ownership:

- Complicated code will not survive for long
- Feeling of influence and responsibility on project
- Spreads knowledge of the system
  (together with pair programming)

Development Strategy

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Pair Programming:

- Increases communication
  → understand what you are doing
- (Positive) controlling
- Spreads knowledge of the system
  (together with collective ownership)

Design Strategy

Driven by all four values of agile

In agile,

- do not design for things that might be useful only later
- keep things as simple as possible (for what you need now)
- refactor when need will be

”Design through refactoring”

Simplicity heuristics

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- ”Code and tests communicate everything you want it to communicate”
- No duplicate code

This includes comments. Don’t forget reasonable “JavaDocs” in your code!

- Fewest possible classes
- Fewest possible methods

Design Strategy

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

In agile,

- design and architecture models (graphical representation of codes design and architecture) are used for discussion only and are not saved,
- design and architecture will evolve anyway

In this course, however, **you are required to document the design and architecture** in a systems specification! This is part of the learning objective.

Roles

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Programmer
- Customer
- Tester
- Tracker

All team members are writing tests. But the testers are responsible for **running** the (non-automated) **tests on a regular basis** – and **post the test results**
→ Testing

Keeps track of history (estimates, test) and helps people to reflect on them
→ **Monitoring and controlling**

Roles

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Programmer
- Customer

- Tester
- Tracker

- Coach

helps team improving the process, the practices, design, …

- Consultant

has deep technical knowledge (or is able to work it out)

- Big Boss

conveys courage, confidence and a bit of insistance

80:20 Rule (Pareto principle)

It might be slightly of 90:10, 70:30, but we call it 80:20 anyway

80% of the effect comes from 20% of the cause

In Software Engineering:

- 20% of the features give you 80% of the value
- 80% of the users use only 20% of the features
- 80% of the code done in 20% of the time

In turn, the 20% of the remaining code needs 80% of the time! Therefore estimations, often, are wrong!

- 80% of the bugs found in 20% of the code

Keep the 80:20 rule in mind for your decisions!