Projektmanagemant im Softwarebereich: AG Reinert

Knut Reinert,
April 2009
Berlin

With some help from..

Projektmanagemant im Softwarebereich

Knut Reinert,
April 2009
Berlin

Qualifikationsziele:

Erwerb von allgemeinen Kenntnissen über die Anwendung von Software im beruflichen Alltag mit größeren Nutzergruppen, insbesondere praktische Erfahrungen mit typischen Problemen mit Software aus dem weiteren Umfeld der Bioinformatik und mit Lösungsansätzen zu deren Überwindung.

Inhalte:

Verwendung von für den zu erwartenden Berufsalltag typischer Software für ein typisches Projekt.

Auswahl passender Software aus einer vorgegebenen Kollektion bzw. Anpassung oder Entwicklung fehlender Softwaremodule.

Erarbeitung von Lösungsstrategien im Team

Versuch einer Lösungsumsetzung.

SEQAN (www.seqan.de)

Knut Reinert,
April 2009
Berlin

Freie

Home
Docu
Down
Conta

2 How to find a pattern

To find all occurrences of a pattern in an indexed String or a StringSet, SeqAn provides the Finder class (see searching tutorial), which is also specialized for indices. The following example shows, how to use the Finder class specialized for our index to search the pattern "be".

```
Finder< Index<String<char> > > myFinder(myIndex);
while (find(myFinder, "be"))
    cout << position(myFinder) << " ";
```

```
11 2
```

The finder object myFinder was created with a reference to myIndex. The function find searches the next occurence of "be" and returns true if an occurrence was found and false after all occurrences have been passed. The posititon of an occurrence in the text is returned by the function position called with the Finder object. Please note that in contrast to online-search algorithms (see searching tutorial) the returned occurrence positions are not ascending. As you can see in the code example above, the pattern "be" was passed directly to the function find function. This is a shortcut for indices and could be also written in a longer way (already known from the searching tutorial):

```
Finder< Index<String<char> > > myFinder(myIndex);
Pattern< String<char> >        myPattern("be");
while (find(myFinder, myPattern))
    cout << position(myFinder) << " ";
```

```
11 2
```

Click here for an example of a Finder for an enhanced suffix array and for a q-gram index.

3 Suffix trees

We consider an alphabet Σ and a sentinel character $ that is smaller than every character of Σ. A suffix tree of a given non-empty string s over Σ is a directed tree whose edges are labeled with non-empty substrings of s$ with the following properties:

1. Each outgoing edge begins with a different letter and the outdegree of an internal node is greater than 1.

2. Each suffix of s$ is the concatenation of edges from the root to a leaf node.

The official SeqAn paper was published in the BMC Bioinformatics journal:

A. Döring, D. Weese, T. Rausch, K. Reinert, *SeqAn - An efficient, generic C++ library for sequence analysis*, BMC Bioinformatics 2008, **9**:11

In this paper we describe the design and content of SeqAn and demonstrate its use by giving

Begleitseminar

Knut Reinert,
April 2009
Berlin

Block 1 (02.04.):

- Prinzipien und Werkzeuge des Softwaredesigns, Werkzeuge zum Programmieren (Bug-tracking Systeme, Debugger, Profiler, Memory analyzer, documentation tools),
- Einführung in make/VC++ project files
- Überblick über Seqan

Block 1a (06.04-09.04): C++ Blockkurs (nicht vorgeschrieben)

Block 2 (14.04.-17.04.):

- Das SeqAn-Tutorial. Mit 5 Einheiten: 1. (Sequences, files, searching), 2. (alignments, multiple alignments), 3. (Indices), 4. (graphs), 5. (Aufgabenverteilung zum Praktikum).

Block 3 (17.04-27.04.): Ausarbeitung und Präsentation des Projektplanes

Block 4 (27.04.-08.06): Programmieren und Dokumentieren der Module, Erstellen eines Abschlussberichtes

Project: Simple Sequence Assembler

Knut Reinert,
April 2009
Berlin

1) project management
2) de-novo repeat detection
3) filter for overlap computation
4) overlapper and construction of overlap graph
5) fragment layout and contig construction
6) scaffolding and consensus
7) (repeat resolution)

All modules will use functionality of SeqAn!

Block 1

Knut Reinert,
April 2009
Berlin

- Einführung in Software Engineering (Reinert)
- Einführung in Software Werkzeuge (Weese)
- Einführung in SeqAn build system, generic programming und SeqAn

Knut Reinert,
April 2009
Berlin

Basierend auf Folien von
Prof. Dr. Oscar Nierstrasz.
Vollständige Vorlesung unter:
http://www.iam.unibe.ch/~scg/Teaching/ESE/

Why Software Engineering?

Knut Reinert,
April 2009
Berlin

A naive view:

Problem Specification $\xrightarrow{\textit{coding}}$ Final Program

But ...

- Where did the *specification* come from?
- How do you know the specification corresponds to the *user's needs*?
- How did you decide how to *structure* your program?
- How do you know the program actually *meets the specification*?
- How do you know your program will always *work correctly*?
- What do you do if the users' *needs change*?
- How do you *divide tasks up* if you have more than a one-person team?

What is Software Engineering?

Knut Reinert,
April 2009
Berlin

Some Definitions and Issues

"state of the art of developing quality software on time and within budget"

- Trade-off between perfection and physical constraints
  - SE has to deal with real-world issues
- State of the art!
  - Community decides on "best practice" + life-long education

What is Software Engineering?

Knut Reinert,
April 2009
Berlin

"multi-person construction of multi-version software"

— Parnas

- Team-work
  - Scale issue ("program well" is not enough) + Communication Issue
- Successful software systems must evolve or perish
  - Change is the norm, not the exception

What is Software Engineering?

Knut Reinert,
April 2009
Berlin

"software engineering is different from other engineering disciplines"

— Sommerville

- Not constrained by physical laws
  - limit = human mind
- It is constrained by political forces
  - balancing stake-holders

Software Development Activities

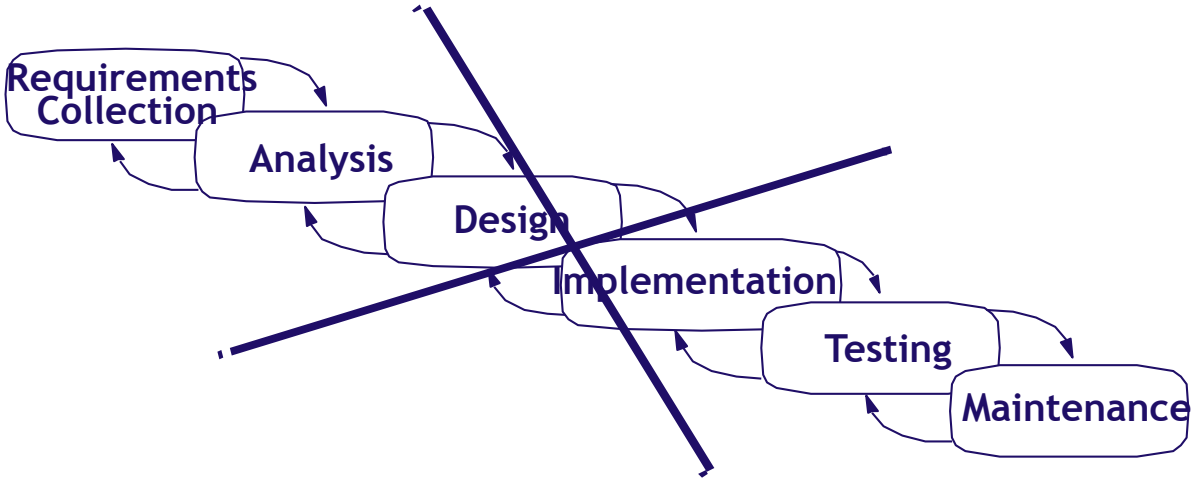
Knut Reinert,
April 2009
Berlin

Requirements Collection	Establish customer's needs
Analysis	Model and specify the requirements ("what")
Design	Model and specify a solution ("how")
Implementation	Construct a solution in software
Testing	Validate the solution against the requirements
Maintenance	Repair defects and adapt the solution to new requirements

The Classical Software Lifecycle

Knut Reinert,
April 2009
Berlin

The classical software lifecycle models the software development as a step-by-step "waterfall" between the various development phases.



The waterfall model is unrealistic for many reasons:

- requirements must be *frozen too early* in the life-cycle
- requirements are *validated too late*

Problems with the Software Lifecycle

- "Real projects rarely follow the sequential flow that the model proposes. *Iteration* always occurs and creates problems in the application of the paradigm"

- "It is often *difficult* for the customer *to state all requirements* explicitly. The classic life cycle requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects."

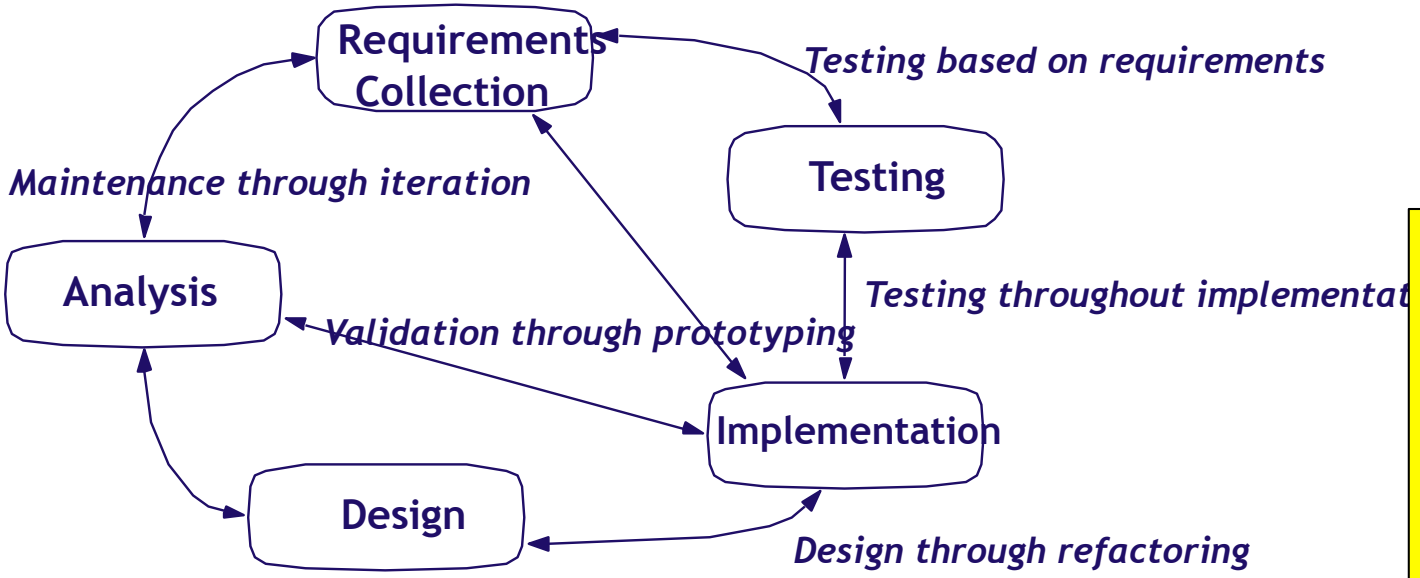
- "The customer must have patience. A *working version* of the program(s) will not be available until *late in the project* timespan. A major blunder, if undetected until the working program is reviewed, can be disastrous."

  — Pressman, SE, p. 26

Iterative Development

Knut Reinert,
April 2009
Berlin

In practice, development is always iterative, and *all* activities progress in parallel



If the waterfall model is pure fiction, why is it still the dominant software process?

Iterative Development

Knut Reinert,
April 2009
Berlin

Plan to *iterate* your analysis, design and implementation.

— You won't get it right the first time, so *integrate*, *validate* and *test* as frequently as possible.

"You should use iterative development only on projects that you want to succeed."

— Martin Fowler, UML Distilled

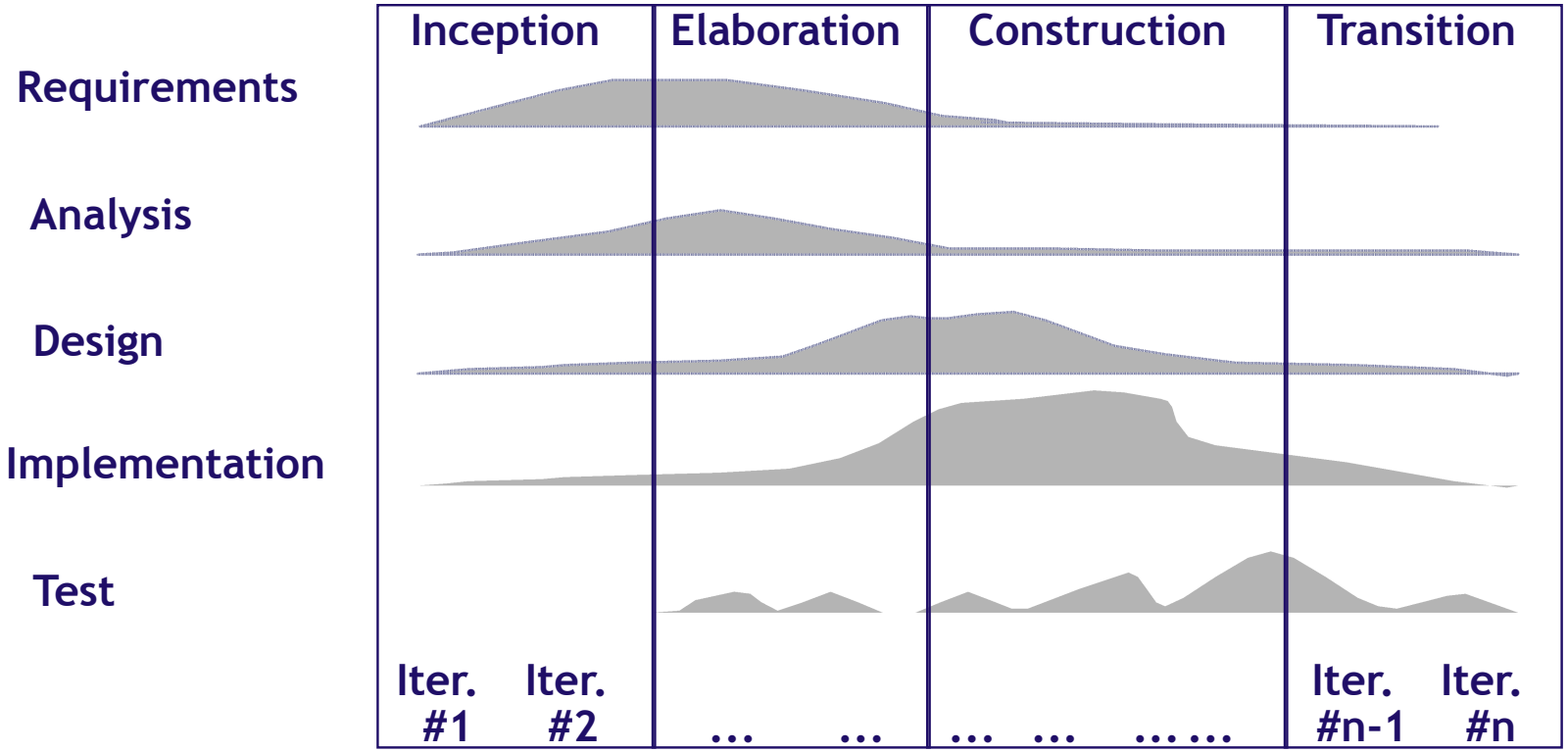
Iterative Development

Plan to *incrementally* develop (i.e., prototype) the system.

- If possible, *always have a running version* of the system, even if most functionality is yet to be implemented.
- *Integrate* new functionality as soon as possible.
- *Validate* incremental versions against user requirements.

The Unified Process

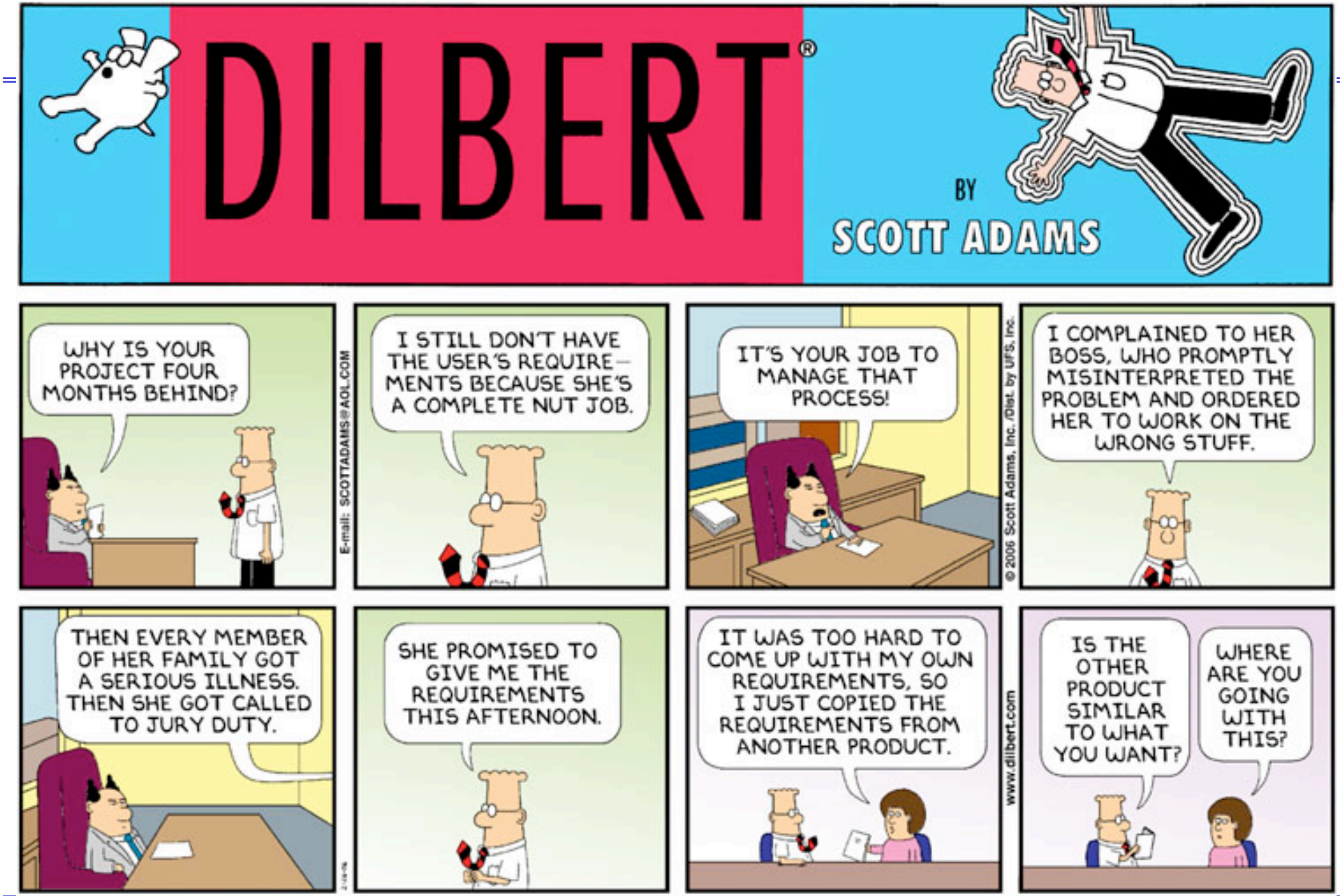
Knut Reinert,
April 2009
Berlin



How do you plan the number of iterations?
How do you decide on completion?

Knut Reinert,
April 2009
Berlin

Software Development Activities



Requirements Collection

Knut Reinert,
April 2009
Berlin

User requirements are often expressed *informally*:

— features

— usage scenarios

Although requirements may be documented in written form, they may be *incomplete*, *ambiguous*, or even *incorrect*.

Changing requirements

Requirements *will* change!

—*inadequately captured* or expressed in the first place

—user and business *needs may change* during the project

Validation is needed *throughout* the software lifecycle, not only when the "final system" is delivered!

—build constant *feedback* into your project plan

—plan for *change*

—early *prototyping* [e.g., UI] can help clarify requirements

Requirements Analysis and Specification

Knut Reinert,
April 2009
Berlin

Analysis is the process of specifying *what* a system will do.

—The intention is to provide a clear understanding of what the system is about and what its underlying concepts are.

The result of analysis is a *specification document*.

Does the requirements specification correspond to the users' actual needs?

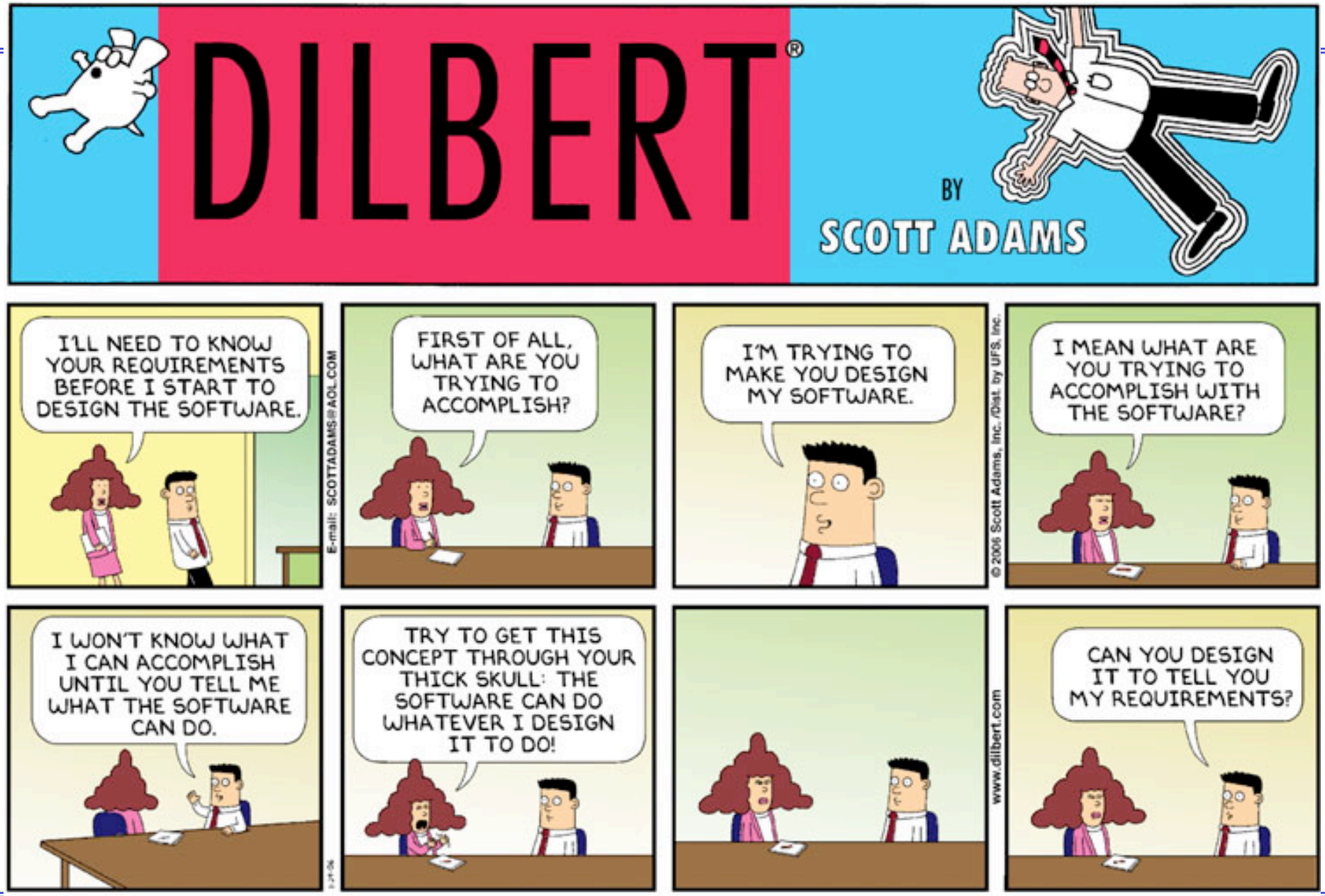
Requirements Engineering Activities

Knut Reinert,
April 2009
Berlin

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

"Requirements are for users; specifications are for analysts and developers."

Knut Reinert,
April 2009



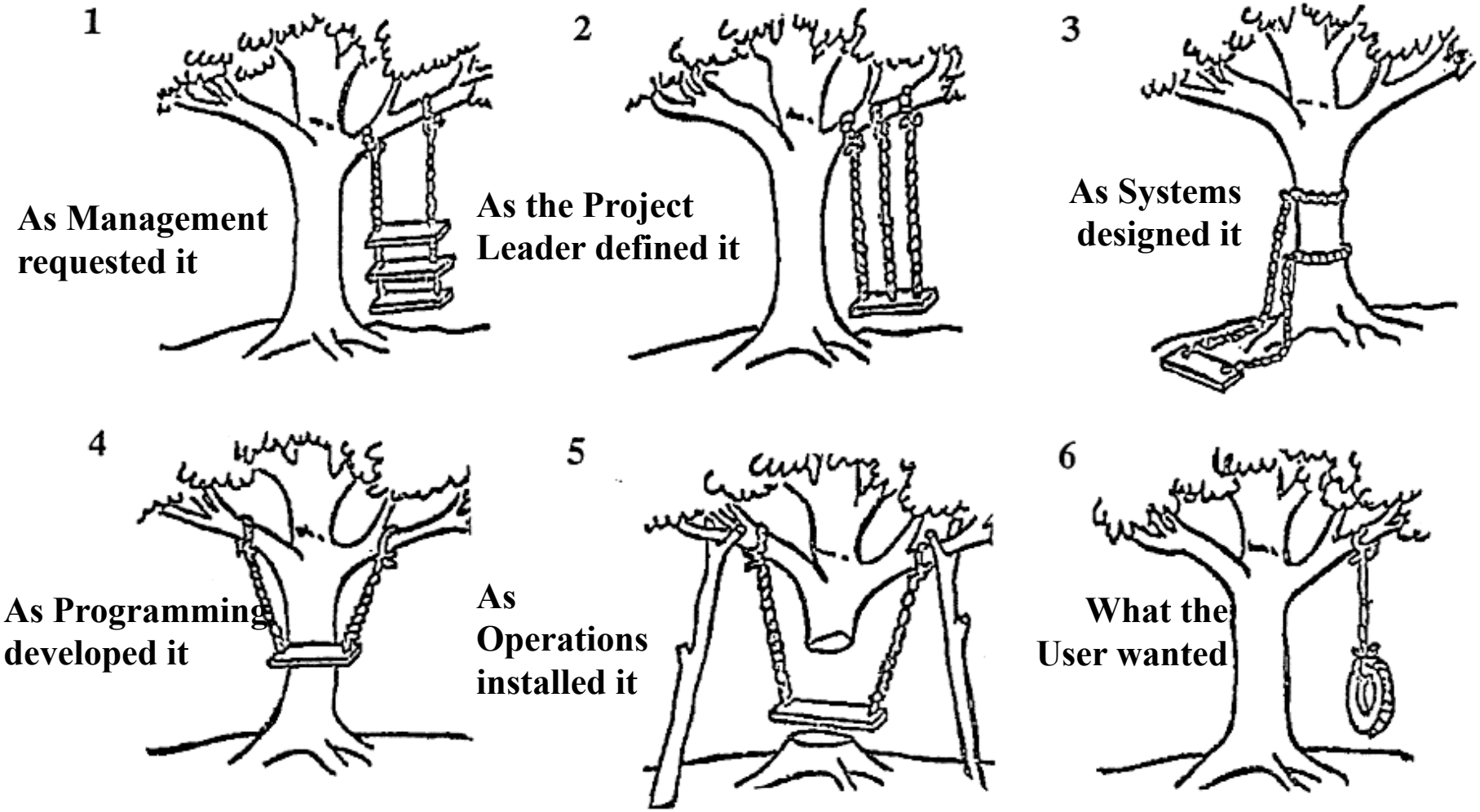
Problems of Requirements Analysis

Various problems typically arise:

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

Problems of Requirements Analysis

Knut Reinert,
April 2009
Berlin



Prototyping

Knut Reinert,
April 2009
Berlin

A *prototype* is a software program developed to test, explore or validate a hypothesis, i.e. *to reduce risks*.

An *exploratory prototype*, also known as a *throwaway prototype*, is intended to *validate requirements* or *explore design choices*.

- UI prototype – validate user requirements
- rapid prototype – validate functional requirements
- experimental prototype – validate technical feasibility

Prototyping

Knut Reinert,
April 2009
Berlin

An *evolutionary prototype* is intended to evolve in steps into a finished product.

iteratively "grow" the application, *redesigning* and *refactoring* along the way

First do it, then do it right, then do it fast.

Design

Knut Reinert,
April 2009
Berlin

Design is the process of specifying *how* the specified system behaviour will be realized from software components. The results are *architecture* and *detailed design documents*.

Object-oriented design delivers models that describe:

—how system operations are implemented by *interacting objects*

—how classes refer to one another and how they are related by *inheritance*

—*attributes* and *operations* associated to classes

Design is an iterative process, proceeding in parallel with implementation!

Knut Reinert,
April 2009
Berlin

Conway's law

"Organizations that design systems are constrained to produce designs that are copies of the communication structures of these organizations"

Implementation and Testing

Knut Reinert,
April 2009
Berlin

Implementation is the activity of *constructing* a software solution to the customer's requirements.

Testing is the process of *validating* that the solution meets the requirements.

The result of implementation and testing is a *fully documented* and *validated* solution.

Design, Implementation and Testing

Knut Reinert,
April 2009
Berlin

Design, implementation and testing are iterative activities

—The implementation does not "implement the design", but rather the design document *documents the implementation*!

System tests reflect the requirements specification

Testing and implementation go hand-in-hand

—Ideally, test case specification *precedes* design and implementation

Maintenance

Maintenance is the process of changing a system after it has been deployed.

- *Corrective maintenance*: identifying and repairing *defects* *(tool used e.g. bug tracker)*
- *Adaptive maintenance*: *adapting* the existing solution to new platforms (e.g. new OS)
- *Perfective maintenance*: implementing *new requirements*

In a spiral lifecycle, everything after the delivery and deployment of the first prototype can be considered "maintenance"!

Maintenance activities

“Maintenance” entails:

- configuration and version management
- reengineering (redesigning and refactoring)
- updating all analysis, design and user documentation

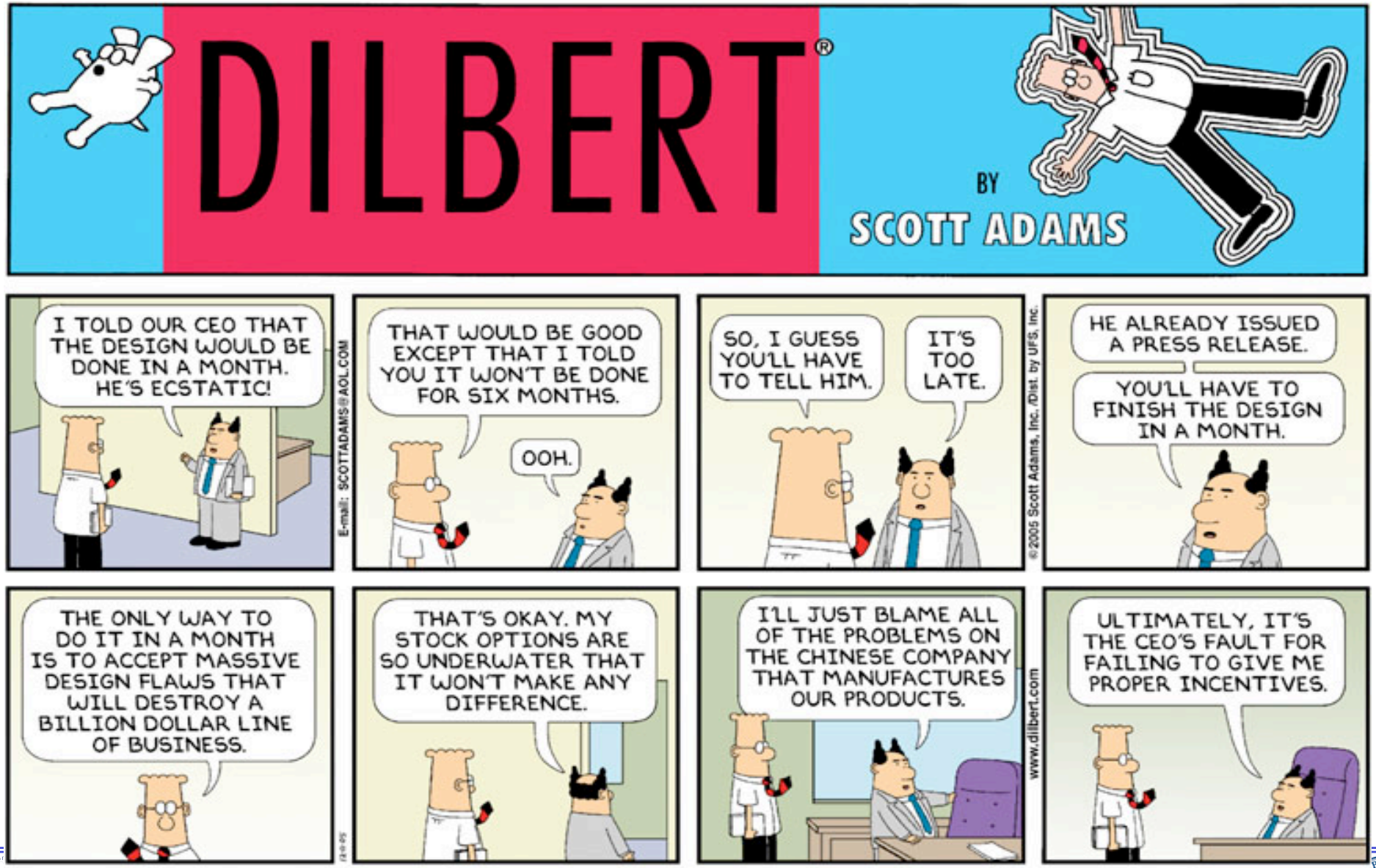
Repeatable, automated tests enable evolution and refactoring

Knut Reinert,
April 2009
Berlin

Project management

Project management

Knut Reinert,
April 2009
Berlin

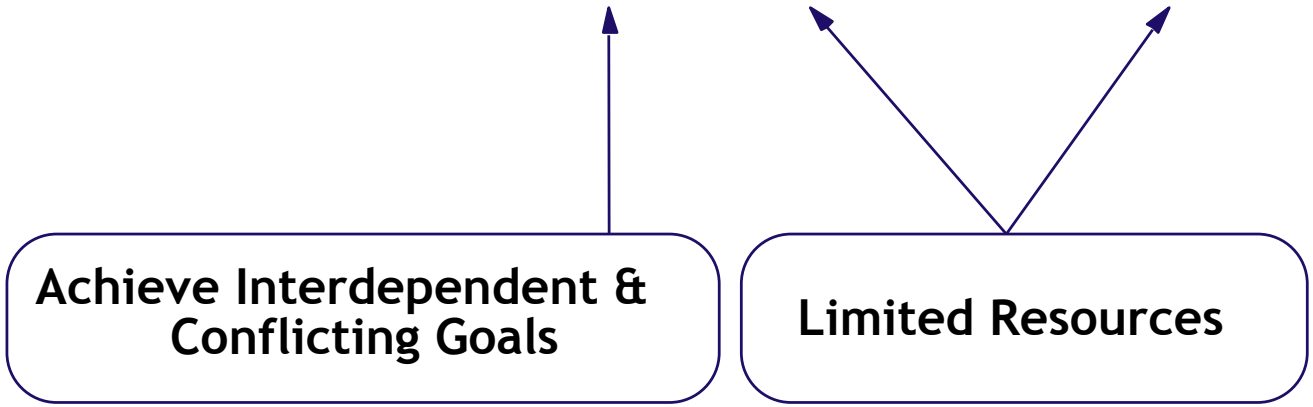


Why Project Management?

Knut Reinert,
April 2009
Berlin

Almost all software products are obtained via *projects*.
(as opposed to manufactured products)

Project Concern = *Deliver on time* and *within budget*



The Project Team is the primary Resource!

What is Project Management?

Knut Reinert,
April 2009
Berlin

Project Management = *Plan the work* and *work the plan*

Management Functions

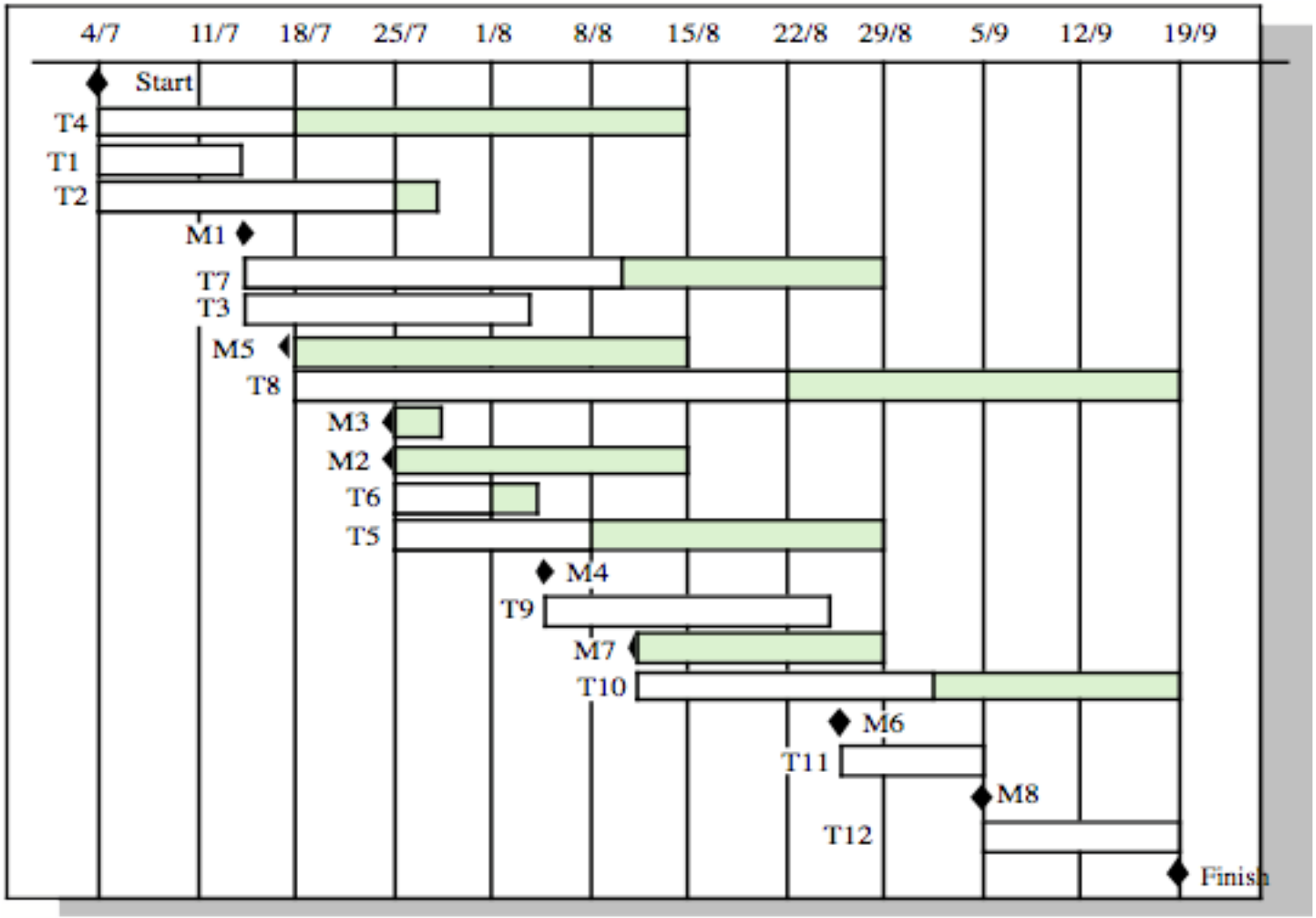
- *Planning*: Estimate and schedule resources
- *Organization*: Who does what
- *Staffing*: Recruiting and motivating personnel
- *Directing*: Ensure team acts as a whole
- *Monitoring (Controlling)*: Detect plan deviations + corrective actions

But … keep the balance

Knut Reinert,
April 2009
Berlin

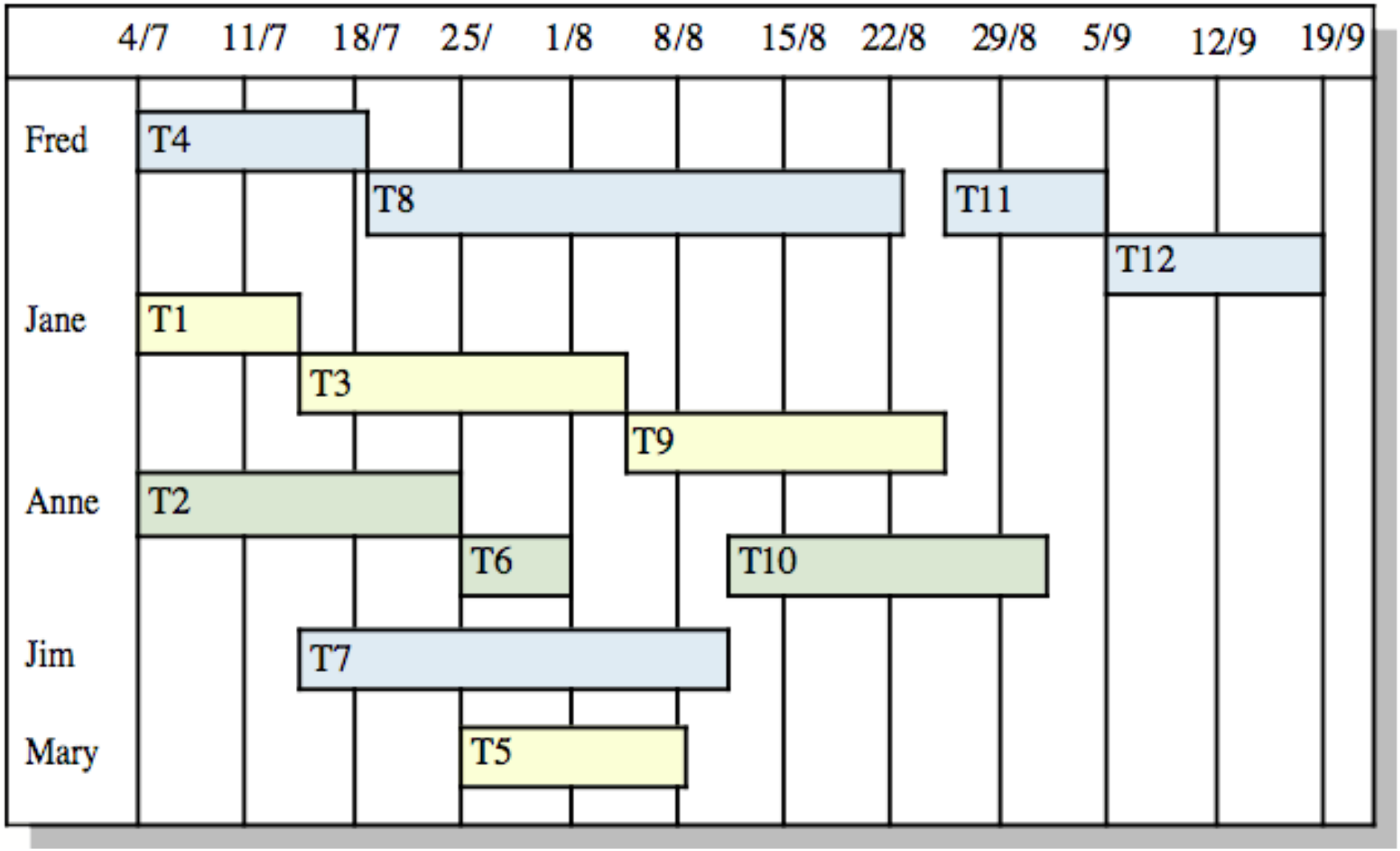
Gantt Chart: Activity Timeline

Knut Reinert,
April 2009
Berlin



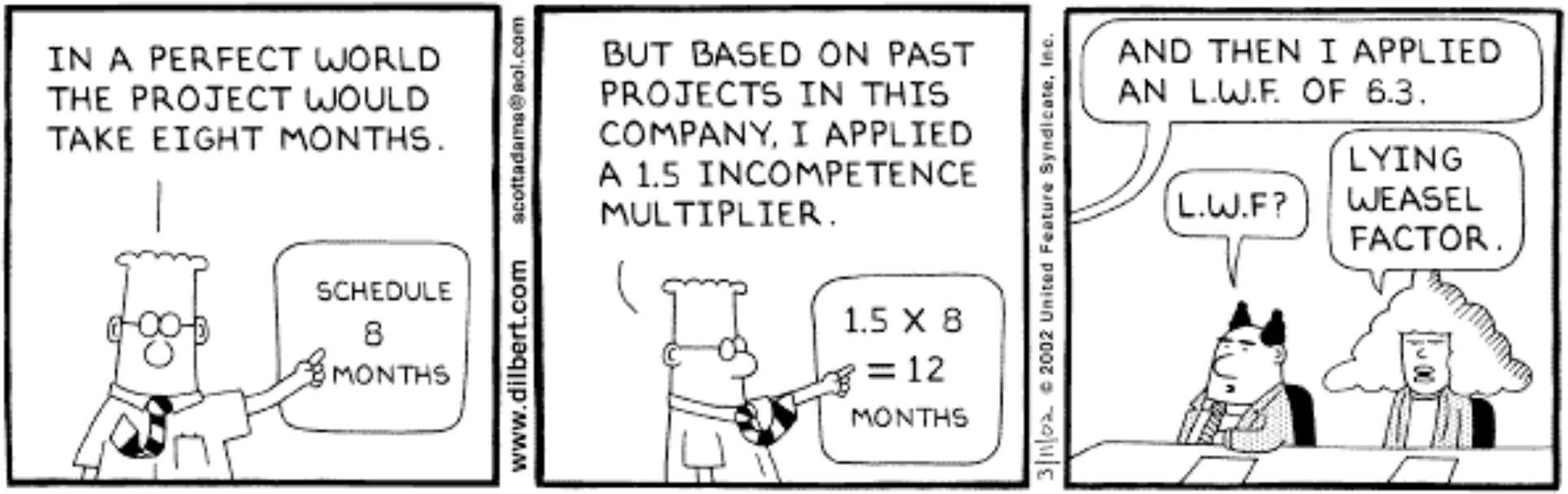
Gantt Chart: Staff allocation

Knut Reinert,
April 2009
Berlin



Be realistic!

Knut Reinert,
April 2009
Berlin



Coding tools

Knut Reinert,
April 2009
Berlin

- An editor or better an IDE (Integrated Development Environment), e.g. xcode, eclipse, VC++,...
- A debugger, e.g. ddd, gdb, ...
- A profiler and memory checker (for C++), .e.g gprof, valgrind, purify, quantify,....
- A documentation system, e.g. doxygen, dddoc, ...

Coding tools

Knut Reinert,
April 2009
Berlin

David and Tobias will later give an introduction into some of those tools