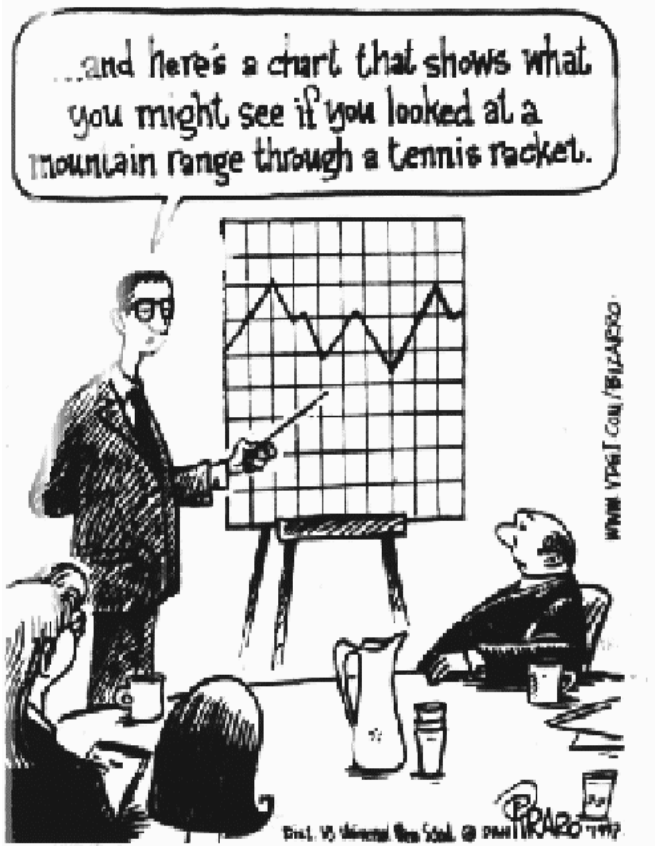
# 350102
# GENERAL INFORMATION & COMMUNICATION TECHNOLOGY II (GENICT)

## - SW ENGINEERING PROCESS MODELS -





Instructor: Peter Baumann

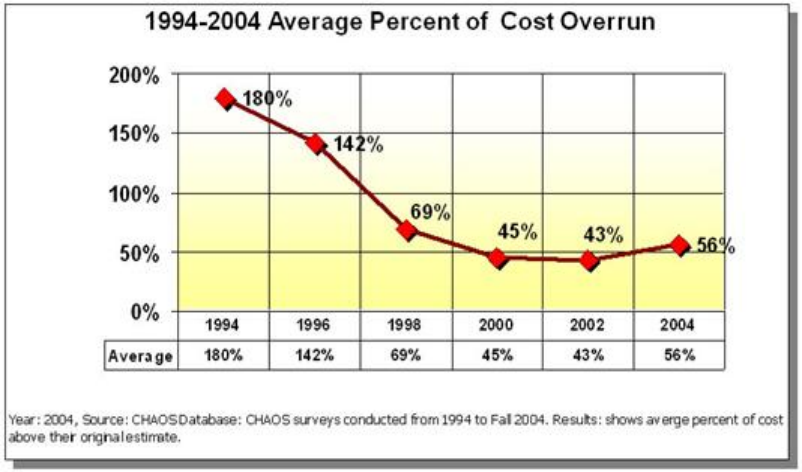
email: p.baumann@jacobs-university.de

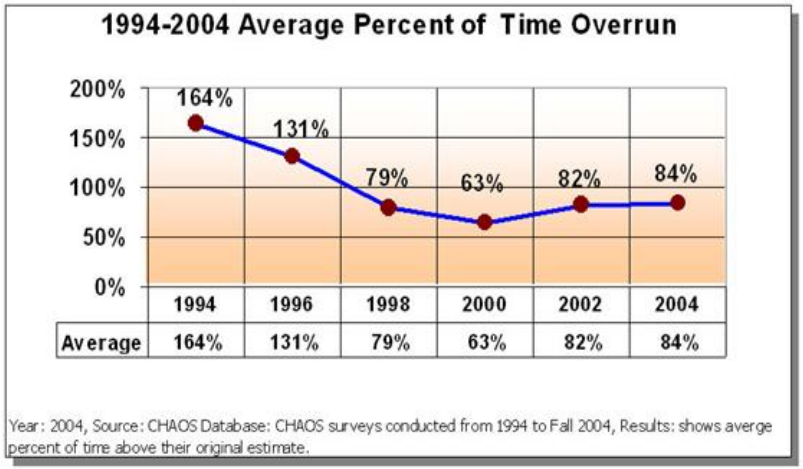
tel: -3178

office: room 88, Research 1

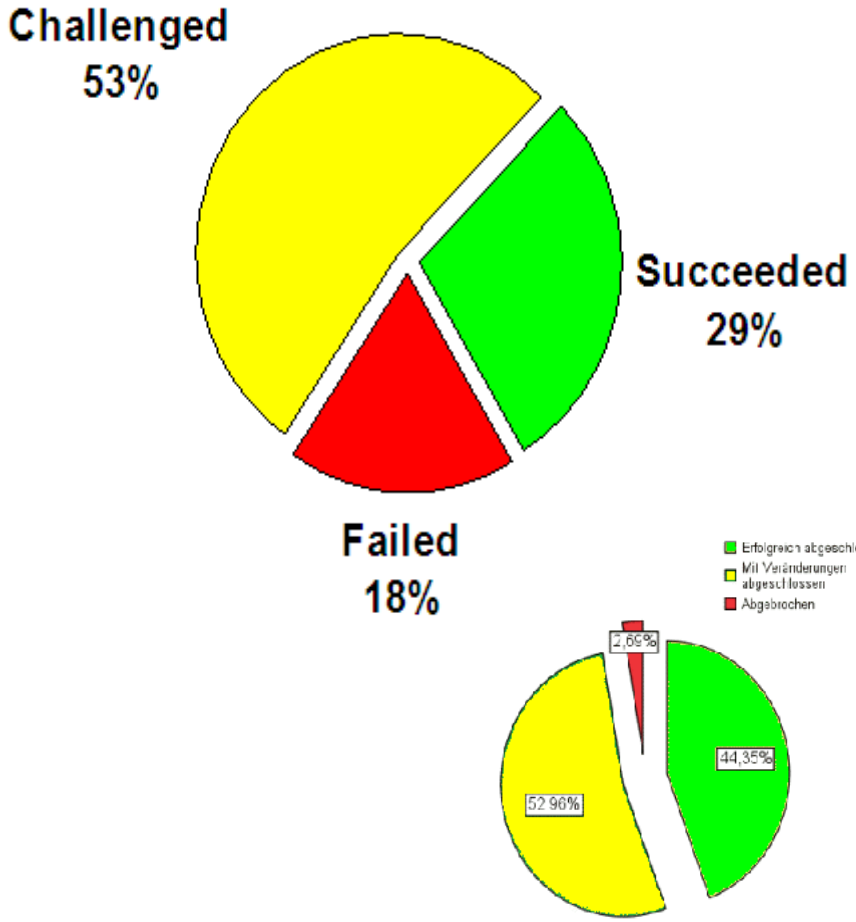
# Project Sucess/Failure Rate

JACOBS UNIVERSITY




[CHAOS Report, Standish Group]



# Top 10 Project Failure Factors: *Lack of...*

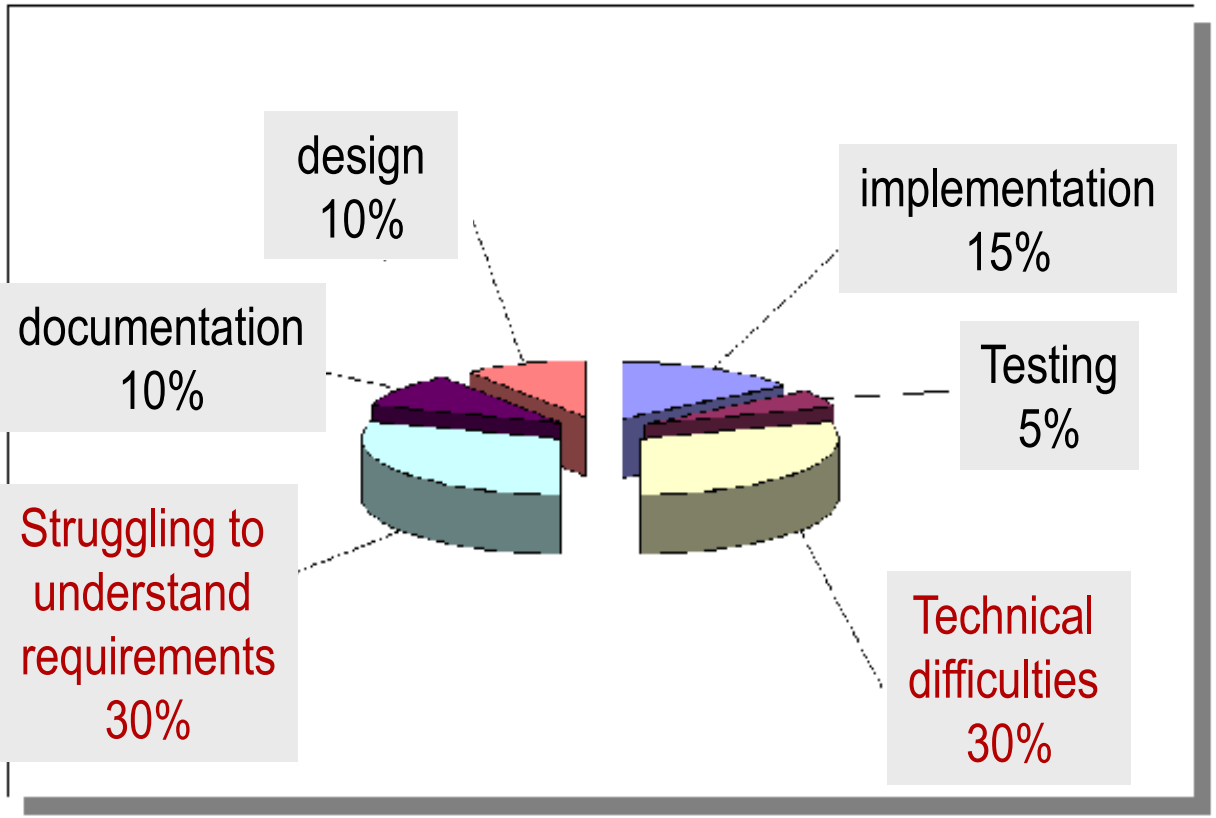
JACOBS UNIVERSITY

1. Executive support (18%)
2. User involvement (16%)
3. Experienced project manager (14%)
4. Clear business objectives (12%)
5. Minimized scope (10%)
6. Standard software infrastructure (8%)
7. Firm basic requirements (6%)
8. Formal methodology (6%)
9. Reliable estimates (5%)
10. Other criteria (5%)

[CHAOS Report, Standish Group International, Inc.]

# Where Time Really Is Spent In Practice

JACOBS UNIVERSITY




*Source: unknown*

# Software Project Management (PM)

JACOBS UNIVERSITY

- Project Management = activities to ensure that result is delivered
  - on time
  - on schedule
  - in accordance with requirements of customer and vendor (!)
- Core: planning & monitoring
- needed because software development always subject to
  - vendor budget & schedule constraints
  - changes

# What Fills a PM's Day

JACOBS UNIVERSITY

- Proposal writing
- Customer (and sales, and marketing) communication
- Project planning and scheduling
- Project costing
- Project monitoring and reviews
- Personnel selection and evaluation
- Report writing and presentations

- Probably most time-consuming activity
- Continuous, regularly revisited
- Various types of plan

# The Project Plan

JACOBS UNIVERSITY

- Project plan sets out:
  - The resources available to the project
    *…who?*
  - The work breakdown
    *…what?*
  - A schedule for the work
    *…when?*

- Project plan structure:
  - Introduction
  - Project organisation
  - Risk analysis
  - Hardware & software resource requirements
  - Work breakdown
  - Project schedule
  - Monitoring & reporting mechanisms

# Types of Project Plan

JACOBS UNIVERSITY

| Plan | Description |
|---|---|
| Quality plan | Describes the quality procedures and standards that will be used in a project. See Chapter 27. |
| Validation plan | Describes the approach, resources and schedule used for system validation. See Chapter 22. |
| Configuration management plan | Describes the configuration management procedures and structures to be used. See Chapter 29. |
| Maintenance plan | Predicts the maintenance requirements of the system, maintenance costs and effort required. See Chapter 21. |
| Staff development plan. | Describes how the skills and experience of the project team members will be developed. See Chapter 25. |

*cf. Sommerville Chapters!*

# Project Planning Process

JACOBS UNIVERSITY

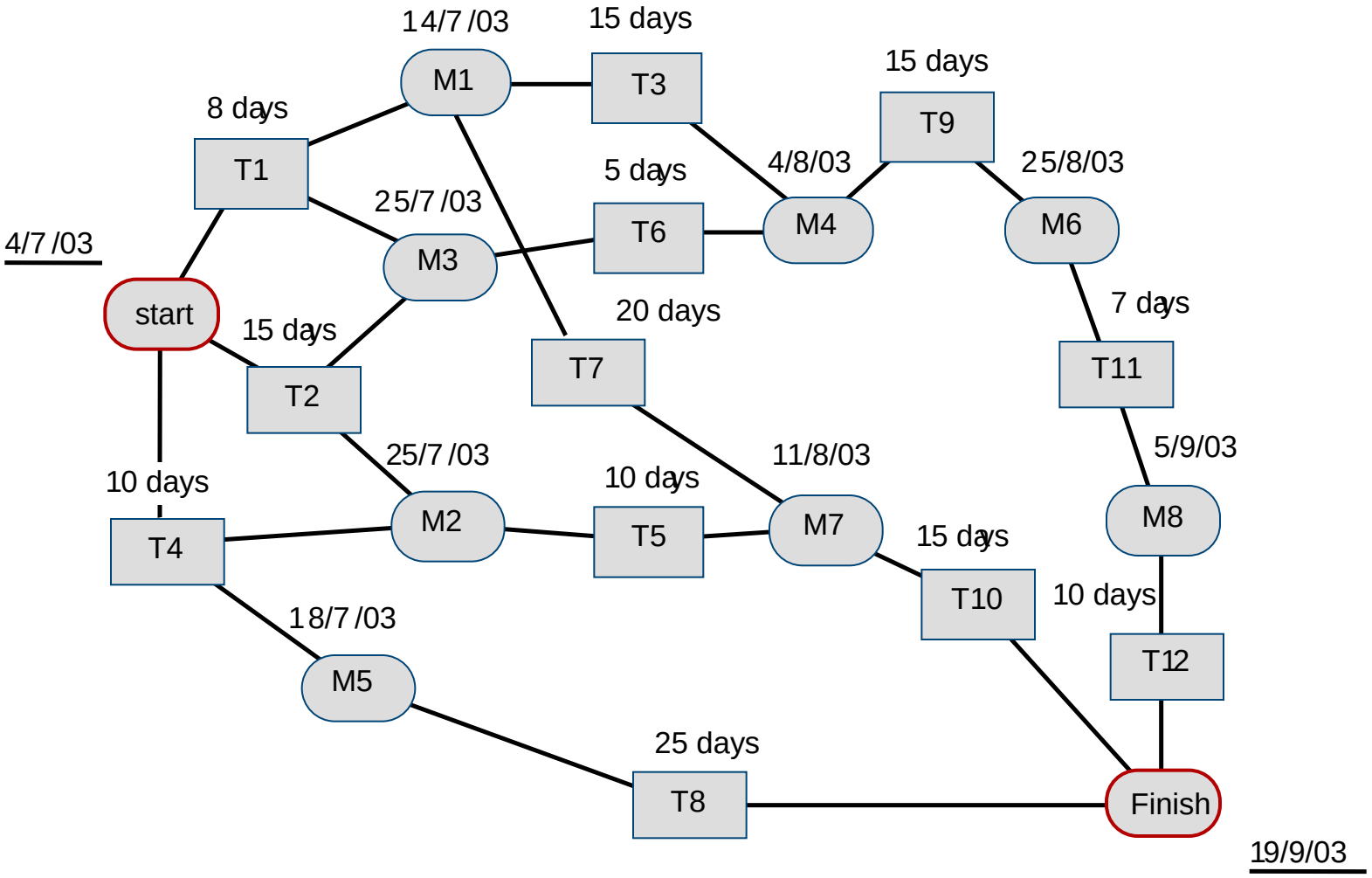
```
Establish project constraints
Make initial assessments of the project parameters
Define project milestones and deliverables
Draw up project schedule
while project has not been completed or cancelled
loop
        Initiate activities according to schedule
        Wait ( for a while )
        Review project progress
        Revise estimates of project parameters
        Update the project schedule
        Re-negotiate project constraints and deliverables
        if ( problems arise ) then
                Initiate technical review and possible revision
        end if
end loop
```

# Tabular Task Durations & Dependencies

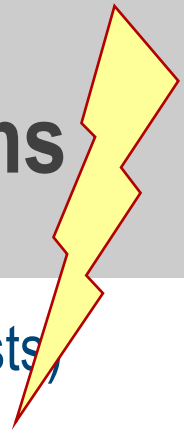
JACOBS
UNIVERSITY

| Activity | Duration (days) | Dependencies |
|---|---|---|
| T1 | 8 | |
| T2 | 15 | |
| T3 | 15 | T1 (M1) |
| T4 | 10 | |
| T5 | 10 | T2, T4 (M2) |
| T6 | 5 | T1, T2 (M3) |
| T7 | 20 | T1 (M1) |
| T8 | 25 | T4 (M5) |
| T9 | 15 | T3, T6 (M4) |
| T10 | 15 | T5, T7 (M7) |
| T11 | 7 | T9 (M6) |
| T12 | 10 | T11 (M8) |

# Activity Network



# Potential Scheduling Problems



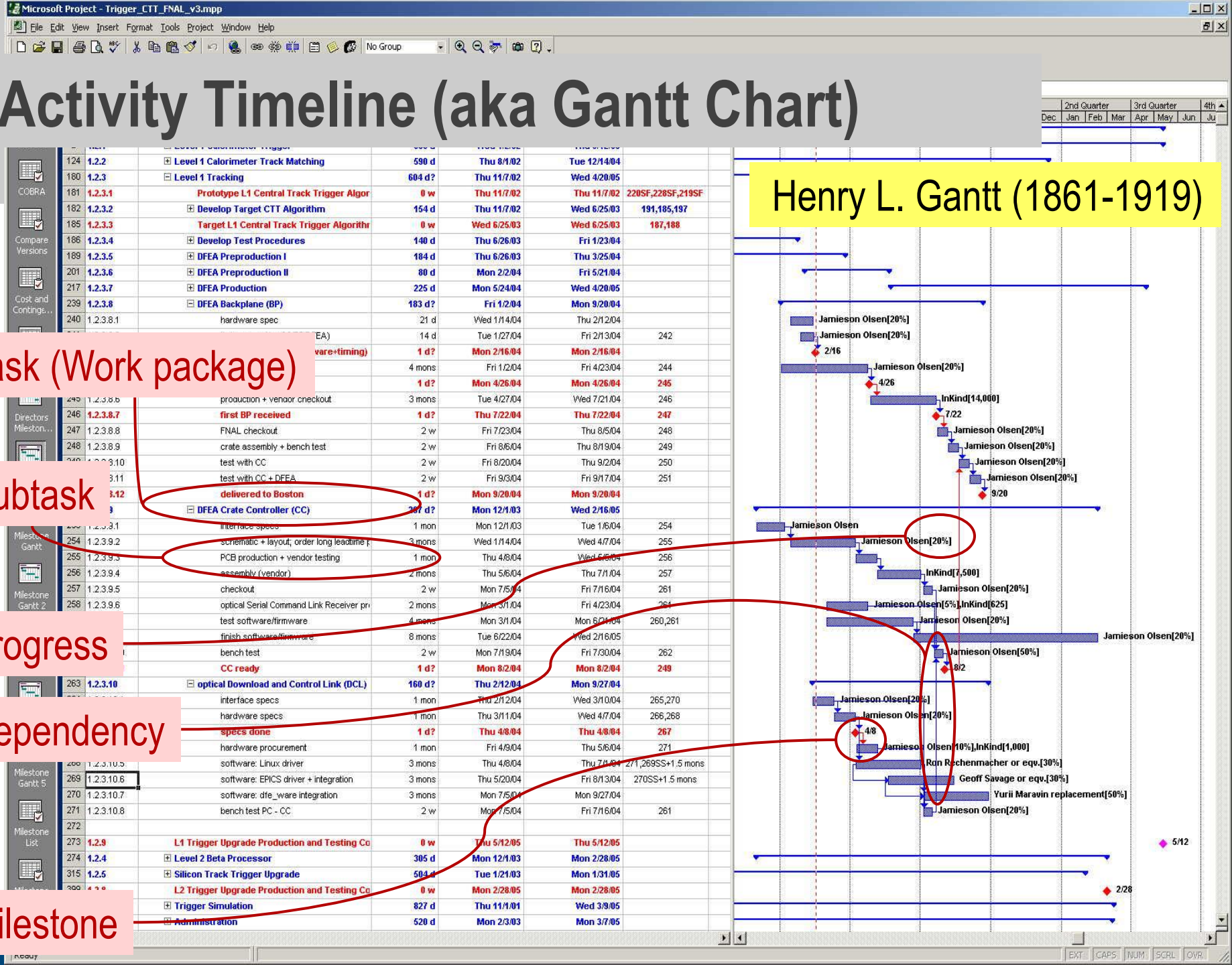
JACOBS UNIVERSITY

- Estimating difficulty of problems (hence, costs)
- Productivity !~ #people working on a task
- Adding people to a late project makes it later
  - communication overheads!
- The unexpected always happens!
  - Always allow contingency in planning

© Scott Adams, Inc./Dist. by UFS, Inc.

- *...as a partial little remedy, let's seek (tool) support*

# Activity Timeline (aka Gantt Chart)




| | | | | | | |
|---|---|---|---|---|---|---|
| 124 | 1.2.2 | Level 1 Calorimeter Track Matching | 590 d | Thu 8/1/02 | Tue 12/14/04 | |
| 180 | 1.2.3 | Level 1 Tracking | 604 d? | Thu 11/7/02 | Wed 4/20/05 | |
| 181 | 1.2.3.1 | Prototype L1 Central Track Trigger Algor | 0 w | Thu 11/7/02 | Thu 11/7/02 | 220SF,228SF,219SF |
| 182 | 1.2.3.2 | Develop Target CTT Algorithm | 154 d | Thu 11/7/02 | Wed 6/25/03 | 191,185,197 |
| 185 | 1.2.3.3 | Target L1 Central Track Trigger Algorithr | 0 w | Wed 6/25/03 | Wed 6/25/03 | 187,188 |
| 186 | 1.2.3.4 | Develop Test Procedures | 140 d | Thu 6/26/03 | Fri 1/23/04 | |
| 189 | 1.2.3.5 | DFEA Preproduction I | 184 d | Thu 6/26/03 | Thu 3/25/04 | |
| 201 | 1.2.3.6 | DFEA Preproduction II | 80 d | Mon 2/2/04 | Fri 5/21/04 | |
| 217 | 1.2.3.7 | DFEA Production | 225 d | Mon 5/24/04 | Wed 4/20/05 | |
| 239 | 1.2.3.8 | DFEA Backplane (BP) | 183 d? | Fri 1/2/04 | Mon 9/20/04 | |
| 240 | 1.2.3.8.1 | hardware spec | 21 d | Wed 1/14/04 | Thu 2/12/04 | |
| | | EA) | 14 d | Tue 1/27/04 | Fri 2/13/04 | 242 |
| | | vare+timing) | 1 d? | Mon 2/16/04 | Mon 2/16/04 | |
| | | | 4 mons | Fri 1/2/04 | Fri 4/23/04 | 244 |
| | | | 1 d? | Mon 4/26/04 | Mon 4/26/04 | 245 |
| 245 | 1.2.3.8.6 | production + vendor checkout | 3 mons | Tue 4/27/04 | Wed 7/21/04 | 246 |
| 246 | 1.2.3.8.7 | first BP received | 1 d? | Thu 7/22/04 | Thu 7/22/04 | 247 |
| 247 | 1.2.3.8.8 | FNAL checkout | 2 w | Fri 7/23/04 | Thu 8/5/04 | 248 |
| 248 | 1.2.3.8.9 | crate assembly + bench test | 2 w | Fri 8/6/04 | Thu 8/19/04 | 249 |
| | 3.10 | test with CC | 2 w | Fri 8/20/04 | Thu 9/2/04 | 250 |
| | 3.11 | test with CC + DFEA | 2 w | Fri 9/3/04 | Fri 9/17/04 | 251 |
| | 3.12 | delivered to Boston | 1 d? | Mon 9/20/04 | Mon 9/20/04 | |
| | 9 | DFEA Crate Controller (CC) | 287 d? | Mon 12/1/03 | Wed 2/16/05 | |
| | 3.1 | interface specs | 1 mon | Mon 12/1/03 | Tue 1/6/04 | 254 |
| 254 | 1.2.3.9.2 | schematic + layout; order long leadtime p | 3 mons | Wed 1/14/04 | Wed 4/7/04 | 255 |
| 255 | 1.2.3.9.3 | PCB production + vendor testing | 1 mon | Thu 4/8/04 | Wed 5/5/04 | 256 |
| 256 | 1.2.3.9.4 | assembly (vendor) | 2 mons | Thu 5/6/04 | Thu 7/1/04 | 257 |
| 257 | 1.2.3.9.5 | checkout | 2 w | Mon 7/5/04 | Fri 7/16/04 | 261 |
| 258 | 1.2.3.9.6 | optical Serial Command Link Receiver pr | 2 mons | Mon 3/1/04 | Fri 4/23/04 | 261 |
| | | test software/firmware | 4 mons | Mon 3/1/04 | Mon 6/21/04 | 260,261 |
| | | finish software/firmware | 8 mons | Tue 6/22/04 | Wed 2/16/05 | |
| | | bench test | 2 w | Mon 7/19/04 | Fri 7/30/04 | 262 |
| | | CC ready | 1 d? | Mon 8/2/04 | Mon 8/2/04 | 249 |
| 263 | 1.2.3.10 | optical Download and Control Link (DCL) | 160 d? | Thu 2/12/04 | Mon 9/27/04 | |
| | | interface specs | 1 mon | Thu 2/12/04 | Wed 3/10/04 | 265,270 |
| | | hardware specs | 1 mon | Thu 3/11/04 | Wed 4/7/04 | 266,268 |
| | | specs done | 1 d? | Thu 4/8/04 | Thu 4/8/04 | 267 |
| | | hardware procurement | 1 mon | Fri 4/9/04 | Thu 5/6/04 | 271 |
| | 3.10.5 | software: Linux driver | 3 mons | Thu 4/8/04 | Thu 7/1/04 | 271,269SS+1.5 mons |
| 269 | 1.2.3.10.6 | software: EPICS driver + integration | 3 mons | Thu 5/20/04 | Fri 8/13/04 | 270SS+1.5 mons |
| 270 | 1.2.3.10.7 | software: dfe_ware integration | 3 mons | Mon 7/5/04 | Mon 9/27/04 | |
| 271 | 1.2.3.10.8 | bench test PC - CC | 2 w | Mon 7/5/04 | Fri 7/16/04 | 261 |
| 272 | | | | | | |
| 273 | 1.2.9 | L1 Trigger Upgrade Production and Testing Co | 0 w | Thu 5/12/05 | Thu 5/12/05 | |
| 274 | 1.2.4 | Level 2 Beta Processor | 305 d | Mon 12/1/03 | Mon 2/28/05 | |
| 315 | 1.2.5 | Silicon Track Trigger Upgrade | 504 d | Tue 1/21/03 | Mon 1/31/05 | |
| | | L2 Trigger Upgrade Production and Testing Co | 0 w | Mon 2/28/05 | Mon 2/28/05 | |
| | | Trigger Simulation | 827 d | Thu 11/1/01 | Wed 3/9/05 | |
| | | Administration | 520 d | Mon 2/3/03 | Mon 3/7/05 | |

# Wrap-Up: Project Management

JACOBS UNIVERSITY

- Good project management essential for project success
  - intangible nature of software causes problems for management
- Managers have diverse roles
  but most significant activities are planning, estimating and scheduling
  - iterative processes which continue throughout the course of a project
- Projects are broken into tasks with deliverables at predefined milestones
  - Gantt chart, PERT chart for project activities, their durations and staffing
- Risk management for
  - identifying risks which may affect the project
  - planning → risks do not develop into major threats

# Commonalities & Differences

JACOBS UNIVERSITY

- SW & other engineering projects share commonalities:
  - Many activities not peculiar to software management
    →
    many techniques of engineering PM equally applicable to sw PM
  - Technically complex engineering systems tend to suffer from same problems as software systems: collaboration; deadlines; customers; …
- On the other hand, software projects are different from projects in other disciplines:
  - product is intangible
  - product is uniquely flexible
  - Software engineering not recognized as an engineering discipline with the sane status as mechanical, electrical engineering, etc.
  - software development process not standardised (well, not completely)
  - Many software projects 'one-off' projects

JACOBS UNIVERSITY

# Software Process Models

Sommerville, Chapters 4, 17
Pressman


Instructor: Peter Baumann

email: p.baumann@jacobs-university.de

tel: -3178

office: room 88, Research 1


# The Software Process

JACOBS UNIVERSITY

- Software process =
  a structured set of activities required to develop a software system
  - Specification
  - Design
  - Validation
  - Evolution.
- software process model =
  abstract representation of a process
  - description of a process from some particular perspective

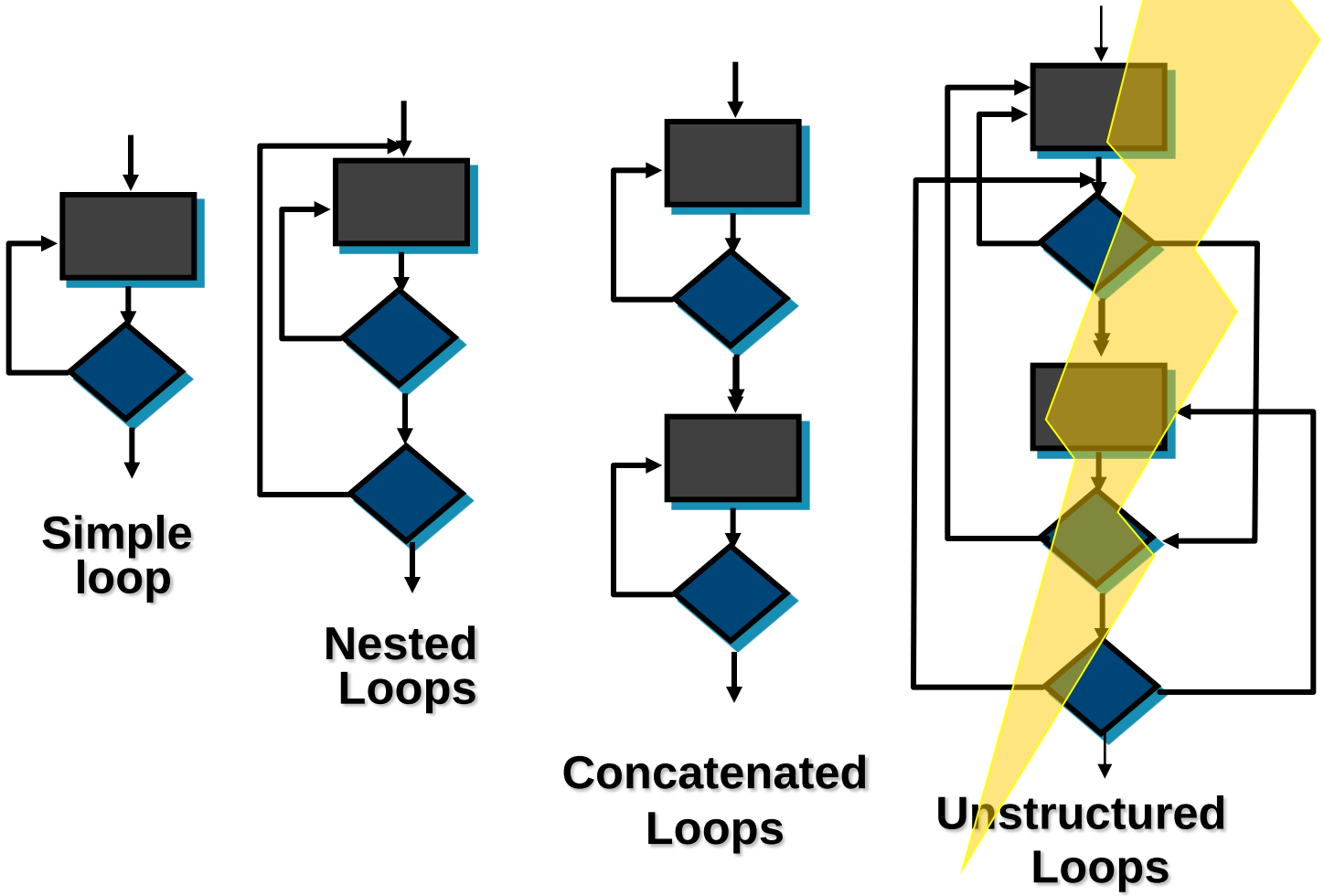
# Software Crisis

JACOBS UNIVERSITY

- early days of CS: difficulty of writing useful & efficient computer programs in the required time
- Reason: rapid increases in computer power, complexity of problems that could be tackled
  - existing methods neither sufficient nor up to the mark
- Issues:
  - Projects running over-budget, over-time
  - Software inefficient, of low quality, not meeting requirements
  - Projects unmanageable, code difficult to maintain
  - Software was never delivered

*When was that?*

# Structured Programming: Loops



# So What Can Go Wrong?

JACOBS UNIVERSITY

- Viking Venus spacecraft: tiny bug in FORTRAN code
  - Correct: DO 20 I = 1,100
  - program code: DO 20 I = 1.100

# For Geeks: Bad Stuff Goes in C++, Too

JACOBS UNIVERSITY

```
for ( count = 0, *templateList = myClass_New ( templateCount, char *);
      *templateList
      && count < templateCount
      && ( ( *templateList)[count] = aux_Duplicate (templates[count] ) );
      count++ );
```

- documenting this takes longer than writing a clear version of the code.
- no error handling at all!
- *How to do better?*

# Software Crisis: Response

JACOBS UNIVERSITY

- Structured programming
  - Functions, blocks...all is better than `goto`!
  - Avoid spaghetti code
  - Later: object-oriented programming
- Defensive programming
  - Better check twice
    – in particular across interfaces!
  - Runtime checks, safer PLs
- Academia: correctness proofs
- Systematic testing

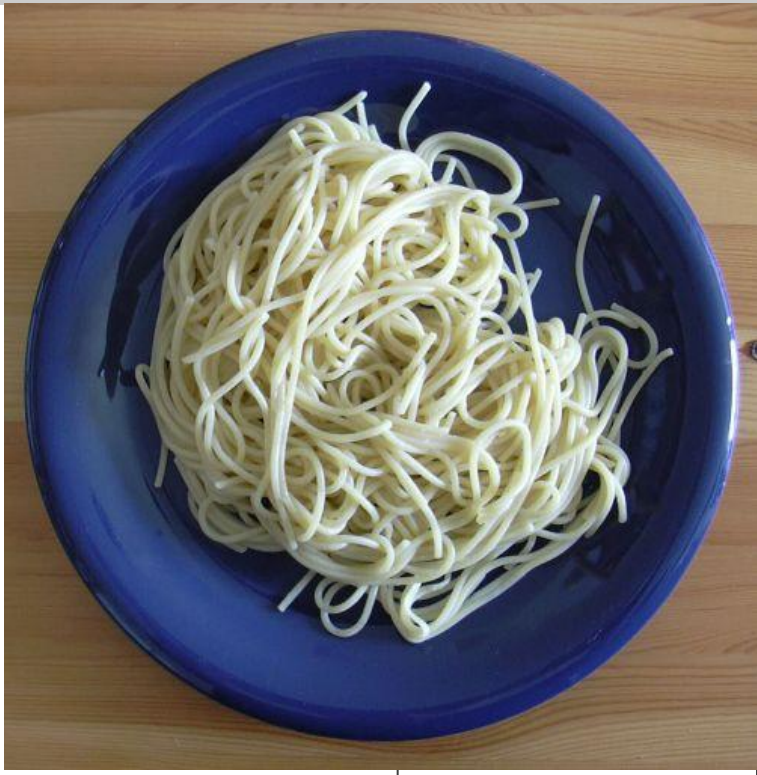


Image: Wikipedia
– check it out!

# Classical Software Process Models

JACOBS UNIVERSITY

- Waterfall model
  - Separate and distinct phases of specification & development
- Evolutionary development
  - Specification, development and validation are interleaved
- Component-based software engineering
  - The system is assembled from existing components
- *…plus many variants*
  - e.g. formal development:
    waterfall-like process, but using formal specification refined through several stages to an implementable design

# Roadmap

JACOBS UNIVERSITY

- SE process management
  - Waterfall model
  - Incremental methods
  - Agile/XP methods
  - Iterative / spiral methods (eg, RUP)
  - Evolutionary methods
  - V-Model
- CMMI

*Note:*
*deviates somewhat from Sommerville's classification, relies on Kal Toth (see later)*

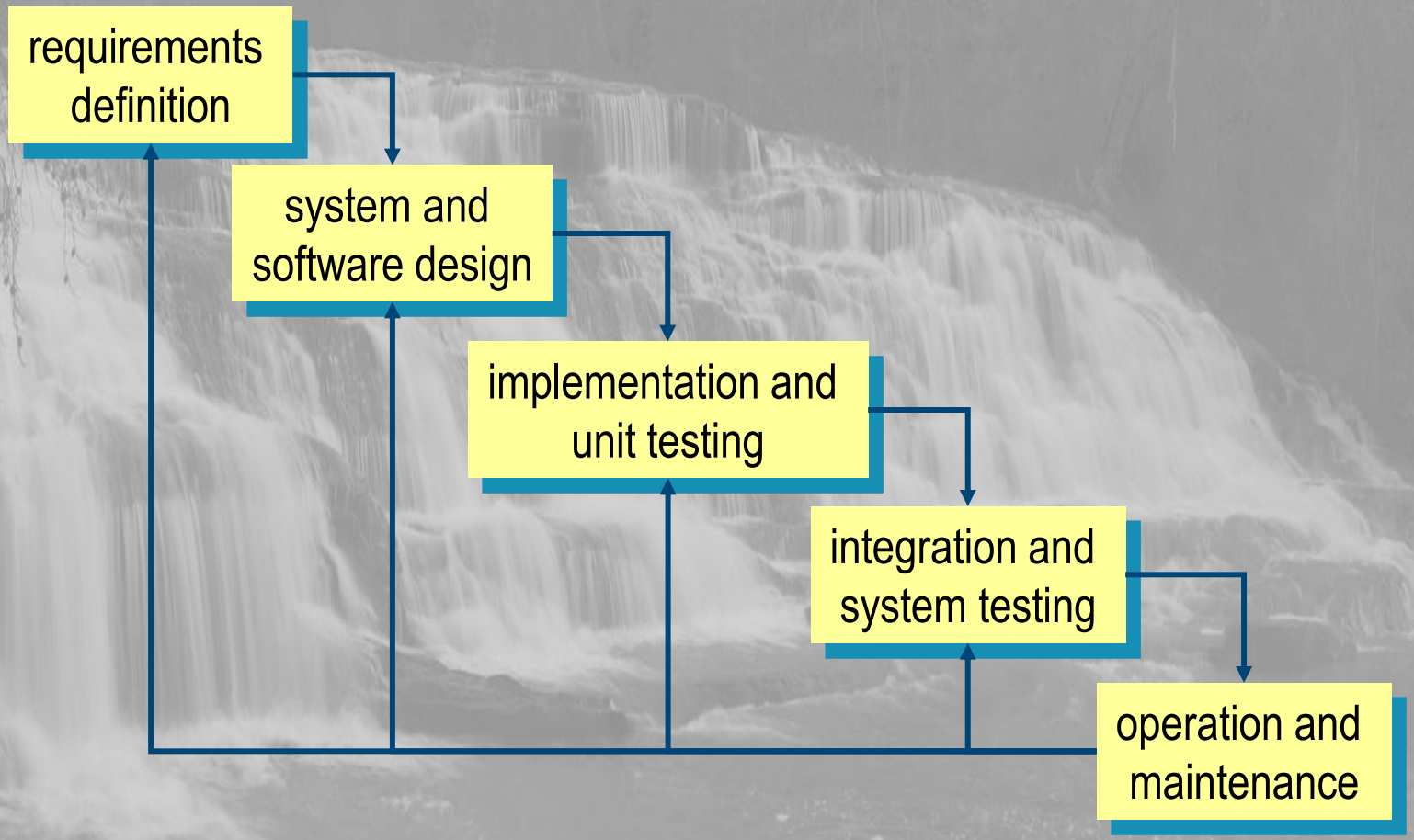
# Roadmap

JACOBS UNIVERSITY

- SE process management
  - Waterfall model
  - Incremental methods
  - Agile/XP methods
  - Iterative / spiral methods (eg, RUP)
  - Evolutionary methods
  - V-Model
- CMMI

# Waterfall Model

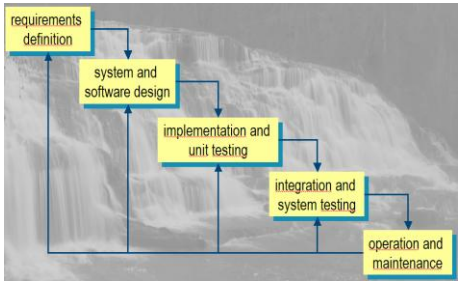
JACOBS UNIVERSITY



# Waterfall Model: Appraisal

JACOBS UNIVERSITY

- Partitioning into distinct stages
  → difficult to accommodate change after process is underway
  → Inflexible
  - One phase has to be complete before moving onto next phase



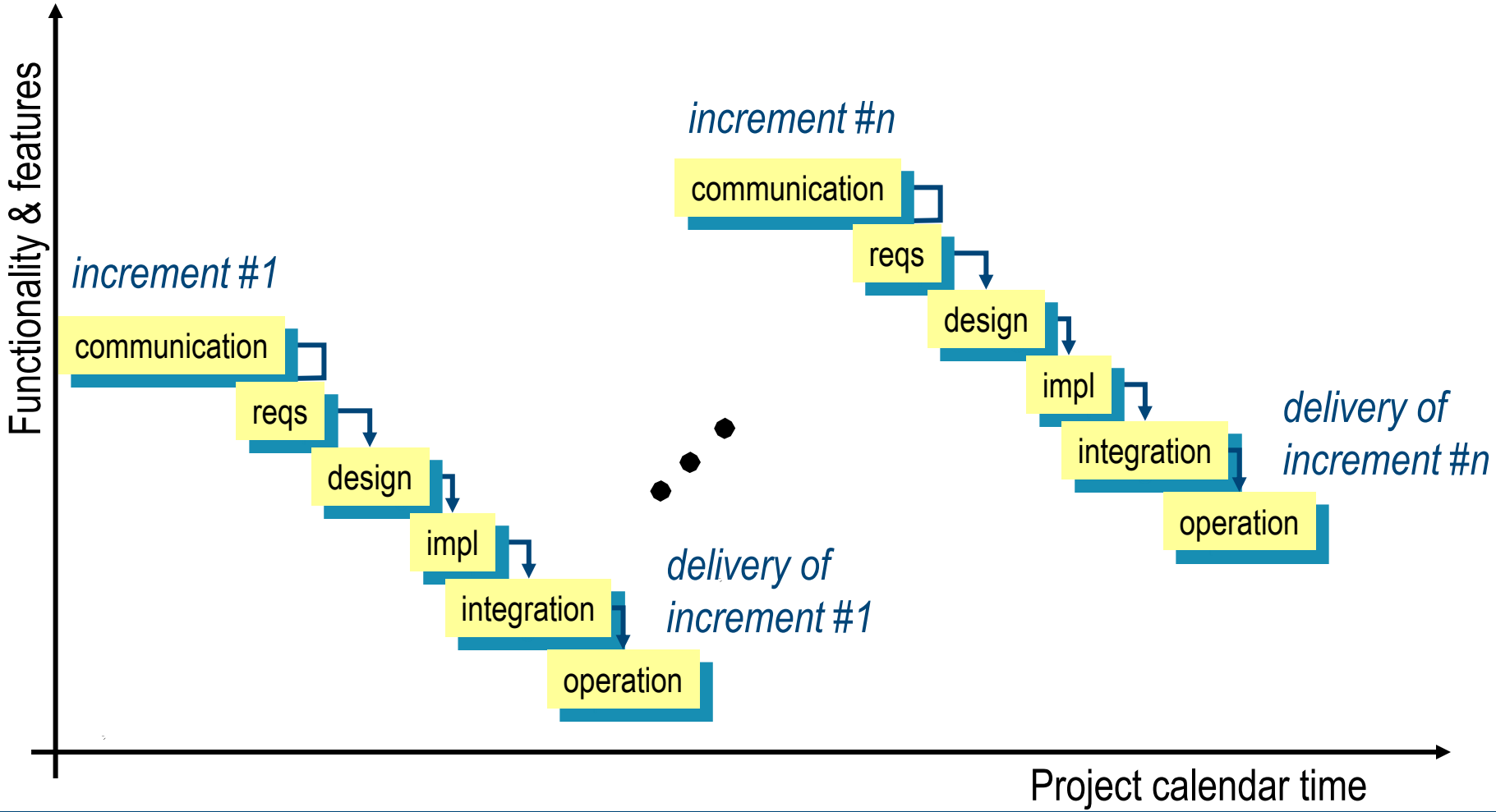

- Few business systems have stable requirements
  - changing customer requirements
  - Increased domain understanding
  - Unforeseen technical difficulties
- only appropriate when requirements well-understood and fairly stable
- *mostly used for large systems engineering projects (?)*
  *where system is developed at several sites*

# Roadmap

JACOBS UNIVERSITY

- SE process management
  - Waterfall model
  - Incremental methods
  - Agile/XP methods
  - Evolutionary methods
- CMMI

# The Incremental Model

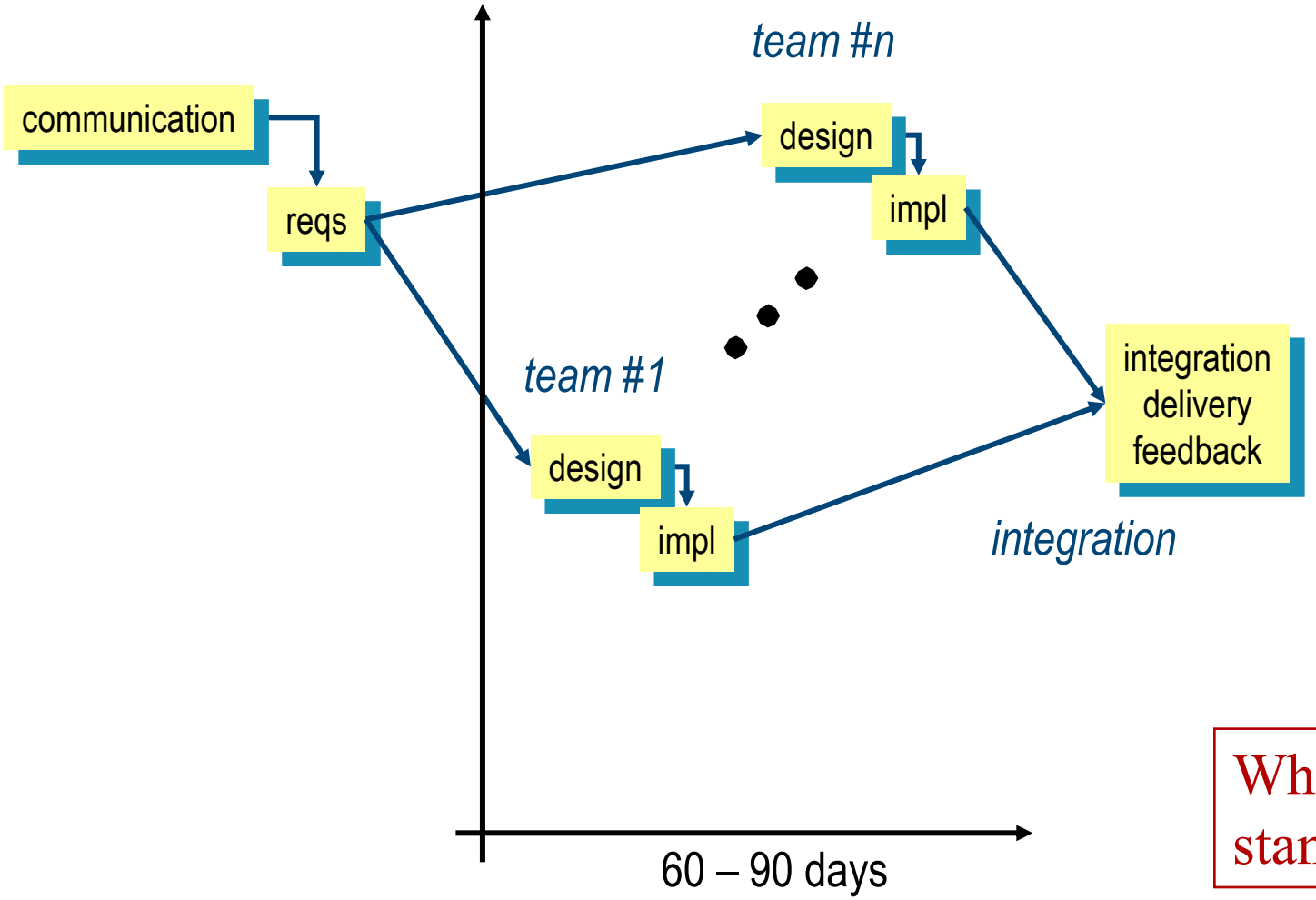


# Incremental Delivery

JACOBS UNIVERSITY

- development & delivery broken down into increments
  - each increment delivering part of the required functionality
- User requirements are prioritised
  - highest priority requirements included in early increments
- Once development of increment is started, requirements are frozen
  - requirements for later increments can continue to evolve

# Variant: The RAD Model

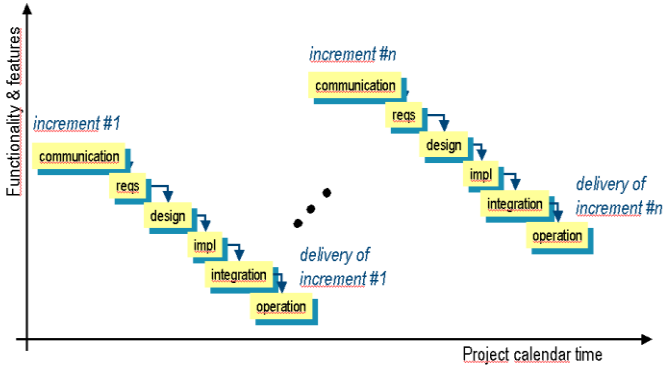



What does RAD stand for?

# Incremental Development: Appraisal

JACOBS UNIVERSITY

- Customer value delivered with each increment
  - system functionality is available earlier
- Early increments act as a prototype
  - help elicit requirements for later increments
- Lower risk of overall project failure
- Highest priority system services tend to receive most testing
  - Why?



# Roadmap

- SE process management
  - Waterfall model
  - Incremental methods
  - Agile/XP methods
  - Evolutionary methods
- CMMI

© Scott Adams, Inc./Dist. by UFS, Inc.

# The Manifesto for Agile Software Development

JACOBS UNIVERSITY

- *"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

# What is “Agility”? Loosely Speaking…

JACOBS UNIVERSITY

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed
- Yielding …
- Rapid, incremental delivery of software

# Principles of Agile Methods: CIPCS

JACOBS UNIVERSITY

- Customer involvement
  - customer closely involved
  - ...to provide & prioritise new system requirements + to evaluate iterations
- Incremental delivery
  - software developed in increments
  - customer specifying requirements to be included per increment
- People, not process
  - Recognize + exploit team skills
  - Leave team to develop own ways of working
- Embrace change
  - Expect system requirements to change
  - design system to accommodate these changes
- Maintain simplicity
  - Focus on simplicity in both software and development process
  - Wherever possible, actively work to eliminate complexity

# Extreme Programming

JACOBS UNIVERSITY

- An 'extreme' variation of iterative development based on very small increments
  - New versions may be built several times per day;
  - Increments are delivered to customers ~every 2 weeks;
  - All tests must be run for every build; build only accepted if all tests run successfully
- Relies on
  - constant code improvement
  - user involvement in the development team
  - pairwise programming
- Perhaps best-known & most widely used agile method
  - originally proposed by Kent Beck

# Pair Programming

JACOBS UNIVERSITY

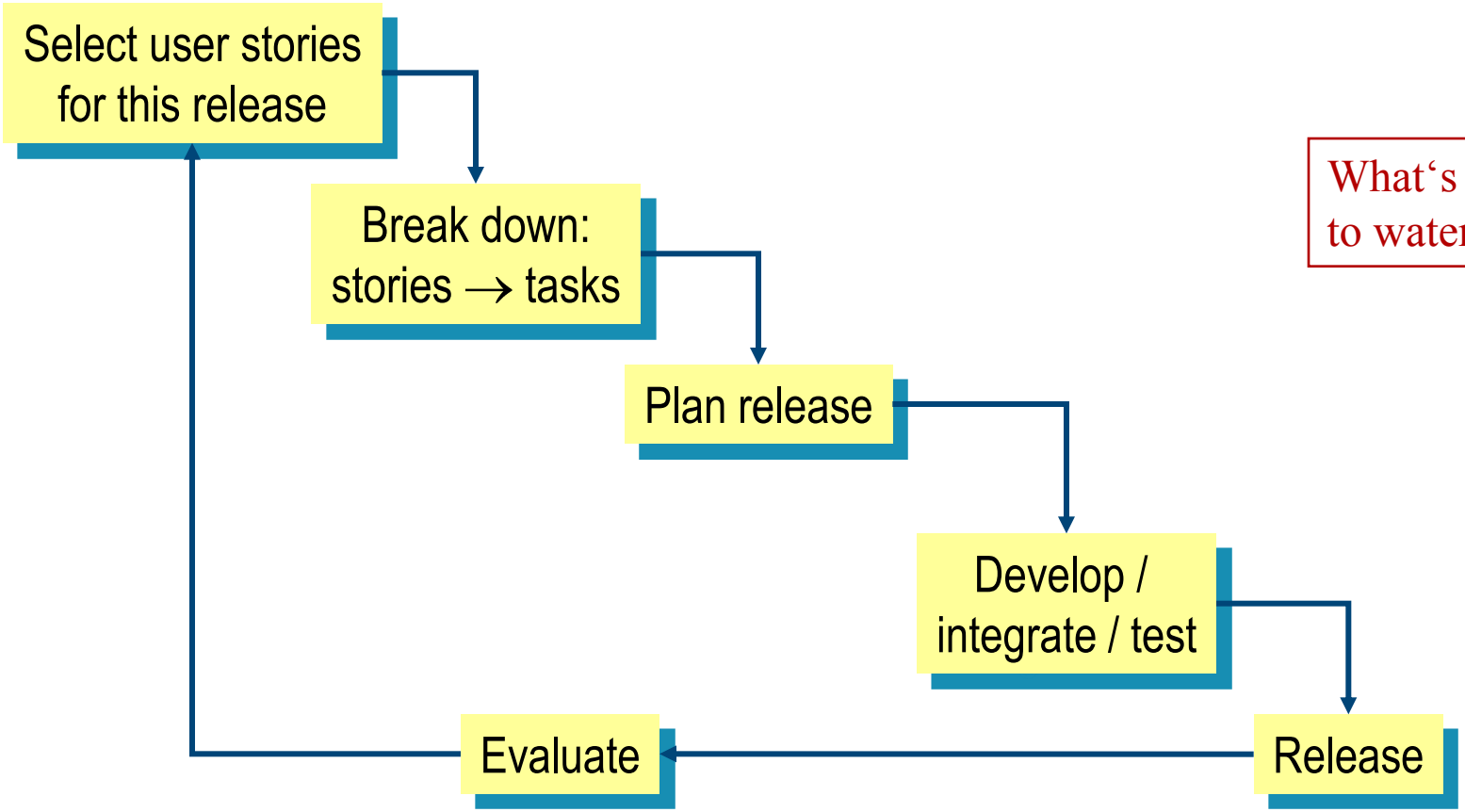
- programmers work in pairs, sitting together to develop code
  - helps develop common ownership of code
  - spreads knowledge across the team
  - Cross checking of all code
- informal review process
  - each line of code looked at by more than 1 person
- encourages refactoring
  - whole team can benefit
- *Measurements suggest that development productivity with pair programming is similar to that of two people working independently.*

# XP and Change

JACOBS UNIVERSITY

- Conventional wisdom: design for change
  - worth spending time & effort anticipating changes
  - reduces costs later in the life cycle
- XP, however, maintains that this is not worthwhile
  - cannot be reliably anticipated
- Rather, it proposes constant code improvement (refactoring)
  - make changes easier when they have to be implemented

# The XP Release Cycle




What's different to waterfall?

# Extreme Programming Phases: *Planning*

JACOBS UNIVERSITY

- Begins with creation of "user stories"
  - Requirements recorded on Story Cards
  - Developers (!) break stories into 'Tasks'
  - Stories grouped for a deliverable increment determined by time available + relative priority
  - Agile team assesses each story and assigns a cost
  - Commitment on delivery date
- Incremental planning
  - After first increment, "project velocity" helps to define subsequent delivery dates for other increments

# Sample Story Card: Document Downloading

JACOBS UNIVERSITY

- *Downloading and printing an article*
  - *First, you select the article that you want from a displayed list. You then have to tell the system how you will pay for it - this can either be through a subscription, through a company account or by credit card.*
  - *After this, you get a copyright form from the system to fill in and, when you have submitted this, the article you want is downloaded onto your computer.*
  - *You then choose a printer and a copy of the article is printed. You tell the system if printing has been successful.*
  - *If the article is a print-only article, you can't keep the PDF version so it is automatically deleted from your computer*

# Extreme Programming Phases: *Design*

JACOBS UNIVERSITY

- KIS(S) principle
- For difficult design problems: suggests "spike solutions" = design prototype
- Encourages "refactoring" to achieve iterative refinement of internal program design

# Extreme Programming Phases: *Coding*

JACOBS UNIVERSITY

- unit tests before coding commences
- Encourages “pair programming”
- All developers expected to refactor code continuously + immediately
  - keeps code simple & maintainable

# Extreme Programming Phases: *Testing*

JACOBS UNIVERSITY

- Test-first development
- Automated test harnesses
  - run all unit tests each time new release is built
  - Incremental test development from scenarios
- User involvement in test development and validation
  - "Acceptance tests" defined by customer
  - executed to assess customer visible functionality

# Sample Test

JACOBS UNIVERSITY

- *Test 4: Test credit card validity*
- *Input:*
  - *string representing credit card number*
  - *two integers representing month and year when card expires*
- *Tests:*
  - *all bytes in string are digits*
  - *month between 1 .. 12, year ≥ current year*
  - *Using first 4 digits of credit card number: check that card issuer is valid by looking up card issuer table.*
  - *Check credit card validity by submitting card number & expiry date information to card issuer*
- *Output:*
  - *OK or error message indicating that the card is invalid*

# Extreme Programming Phases: *Integration*

JACOBS UNIVERSITY

- After each task: integration of result into whole system
- Check-in accepted only if *all* unit tests pass

# Consequences of Extreme Programming

JACOBS UNIVERSITY

- Incremental planning
  - Stories determined by time available + relative priority
- Small Releases
  - minimal useful set of functionality that provides business value is developed first
- Collective Ownership
  - pairs of developers work on all areas of system
  - no islands of expertise, all developers own all code
  - Anyone can change anything
- Simple Design: Enough design to meet current requirements and no more
- Simple code: Refactoring
- Sustainable pace
  - No large amounts of over-time – net effect often reduced code quality, medium term productivity
- On-site Customer
  - End-user representative available full time
  - Customer member of development team, responsible for bringing system

# Agile methods: Appraisal

- Team members may be unsuited to the intense involvement of agile methods
- Developers need to be experienced, not too different in expertise
- can be difficult to keep interest of customers involved in process

# Agile methods: Appraisal

JACOBS UNIVERSITY

- Maintaining simplicity requires extra work
- Contracts may be a problem
  - Prioritising changes can be difficult when there are multiple stakeholders
  - …as with other approaches to iterative development
- *Agile methods probably best suited to small/medium-sized business systems or PC products* = *short-term, highly flexible projects*

# Roadmap

JACOBS UNIVERSITY

- SE process management
  - Waterfall model
  - Incremental methods
  - Agile/XP methods
  - Evolutionary methods
- CMMI

# Evolutionary Development

JACOBS UNIVERSITY

- Exploratory development
  - work with customers
  - evolve final system from initial outline specification
  - *start with well-understood requirements, add new features as proposed by customer*
    → similar to incremental / iterative approach
- Throw-away prototyping
  - Goal: understand system requirements,
    not to build a deliverable
  - *start with poorly understood requirements to clarify what is really needed*

# Prototyping

JACOBS UNIVERSITY

- For some large systems, incremental development & delivery may be impractical
  - especially true when multiple teams working on different sites
- Alternative: Prototyping
  - experimental system developed as basis for formulating requirements
  - thrown away when system specification agreed
- prototype = initial version of a system used to
  - demonstrate concepts
  - try out design options
- prototype can be used in:
  - requirements engineering process → help with requirements elicitation & validation
  - design processes → explore options, develop UI design

back-to-back tests

# Throw-Away Prototypes

JACOBS UNIVERSITY

- Prototypes should be discarded after development as they are not a good basis for a production system:
  - may be impossible to tune the system to meet non-functional requirements
  - Prototypes normally undocumented
  - prototype structure usually degraded through rapid change
  - prototype probably will not meet normal organisational quality standards

# When Incremental Dev, When Prototype?

JACOBS UNIVERSITY

- incremental development: deliver working system to end-users
  - development starts with requirements best understood
- throw-away prototyping: validate or derive system requirements
  - prototyping process starts with requirements poorly understood

# Evolutionary Development: Appraisal

JACOBS UNIVERSITY

- Problems
  - Lack of process visibility
  - Systems are often poorly structured
  - Special skills (e.g. in languages for rapid prototyping) may be required
- Applicability
  - For small or medium-size interactive systems
  - For well isolated parts of large systems (e.g. the user interface)
  - For short-lifetime systems

JACOBS UNIVERSITY

# Capability Maturity Model Integration

Sommerville, Chapter 28


Instructor: Peter Baumann

email: p.baumann@jacobs-university.de

tel: -3178

office: room 88, Research 1

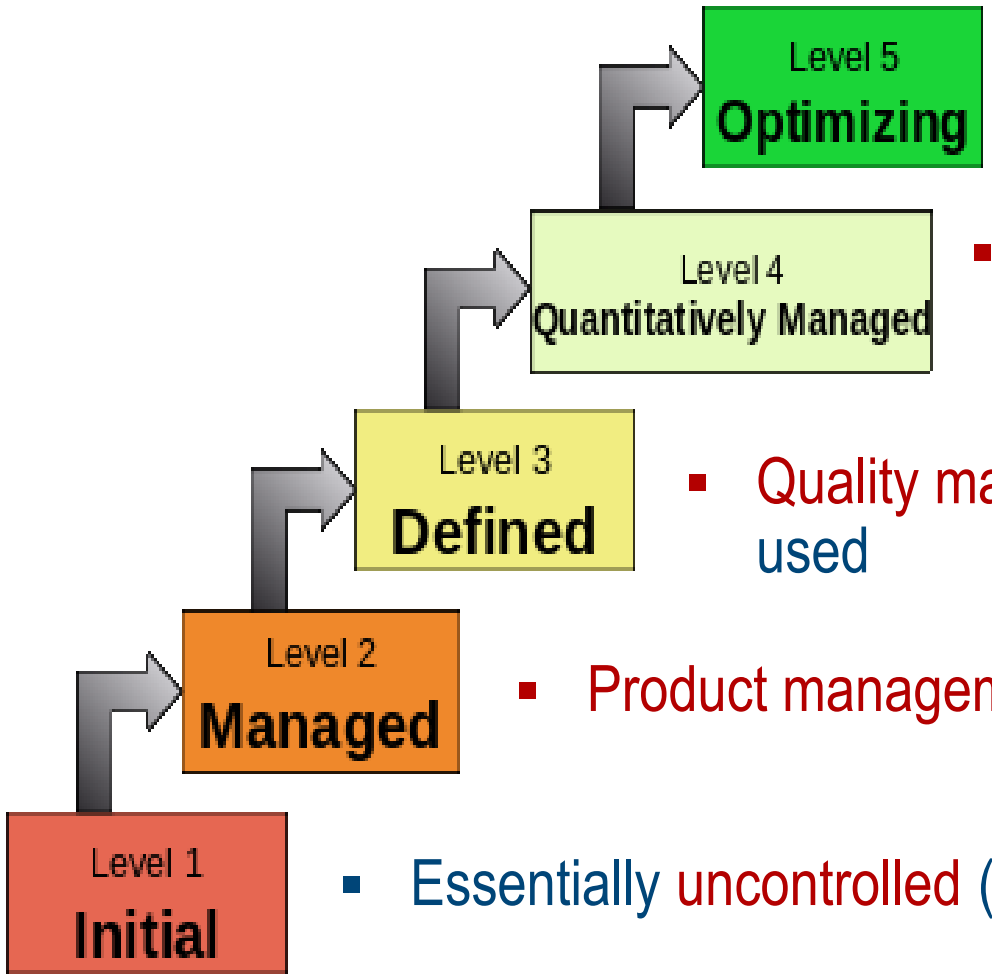

„In theory, there is no difference between theory and practice. In practice, there is.“
-- Yogi Berra (?)

# Process Capability Assessment

JACOBS UNIVERSITY

- To what extent do an organisation's processes follow best practice?
  - identify areas of weakness for process improvement
- various models; SEI most influential
  - Software Engineering Institute (SEI), www.sei.cmu.edu
  - SEI's mission: promote software technology transfer, particularly to US defence contractors
- CMM(I) framework measures process maturity, thereby helps with improvement
  - Capability Maturity Model (CMM) introduced in the early 1990s
  - Revised: Capability Maturity Model Integration (CMMI) introduced in 2001
  - See also: ISO/IEC 15504 (SPICE)

# CMM Organisational Maturity Levels




- Process improvement strategies defined & used
- Process management procedures & strategies defined & used
- Quality management strategies defined & used
- Product management procedures defined & used
- Essentially uncontrolled (each project a "one-time heroic act")

[Wikipedia]

# Problems with the CMM

JACOBS UNIVERSITY

- Model levels
  - Companies could be using practices from different levels at the same time but if all practices from a lower level were not used, it was not possible to move beyond that level
- Discrete rather than continuous
  - Did not recognise distinctions between the top and the bottom of levels
- Practices oriented
  - Concerned with how things were done (the practices) rather than the goals to be achieved

# CMMI

JACOBS UNIVERSITY

- CMMI = Capability Maturity Model Integration
  - integrated capability model
    that includes software and systems engineering capability assessment
- Components:
  - Process areas – 24 process areas that are relevant to process capability and improvement are identified. These are organised into 4 groups.
  - Goals – Goals are descriptions of desirable organisational states. Each process area has associated goals.
  - Practices – Practices are ways of achieving a goal;
    however, they are advisory and other approaches to achieve the goal may be used.

# CMMI Process Areas

JACOBS UNIVERSITY

Process areas – Goals – Practices

| | |
|---|---|
| Process management | Organisational process definition; Organisational process focus; Organisational training; Organisational process performance; Organisational innovation and deployment |
| Project management | Project planning; Project monitoring and control; Supplier agreement management; Integrated project management; Risk management; Integrated teaming; Quantitative project management |
| Engineering | Requirements management; Requirements development; Technical solution; Product integration; Verification; Validation |
| Support | Configuration management; Process and product quality management; Measurement and analysis; Decision analysis and resolution; Organisational environment for integration; Causal analysis and resolution |

# CMMI Goals

JACOBS UNIVERSITY

Process areas – Goals – Practices

- Goal:
  - Corrective actions are managed to closure when the project's performance or results deviate significantly from the plan.
  - Actual performance and progress of the project is monitored against the project plan.
  - The requirements are analysed and validated and a definition of the required functionality is developed.
  - Root causes of defects and other problems are systematically determined.
  - The process is institutionalised as a
- Process area:
  - Specific goal in Project Monitoring and Control
  - Specific goal in project monitoring and control
  - Specific goal in requirements development
  - Specific goal in causal analysis and resolution
  - Generic goal

# CMMI Practices

JACOBS UNIVERSITY

Process areas – Goals – Practices

- Practice
  - Analyse derived requirements to ensure that they are necessary and sufficient
  - Validate requirements to ensure that the resulting product will perform as intended in the user's environment using multiple techniques as appropriate.
  - Select the defects and other problems for analysis.
  - Perform causal analysis of selected defects and other problems and propose actions to address them.
  - Establish and maintain an organisational policy for planning and performing the requirements development process.
  - Assign responsibility and authority for performing
- Associated goal
  -
  - The requirements are analysed and validated and a definition of the required functionality is developed.
  - Root causes of defects and other problems are systematically determined.
  - The process is institutionalised as a defined process.
  -
  -

# CMMI Assessment

JACOBS UNIVERSITY

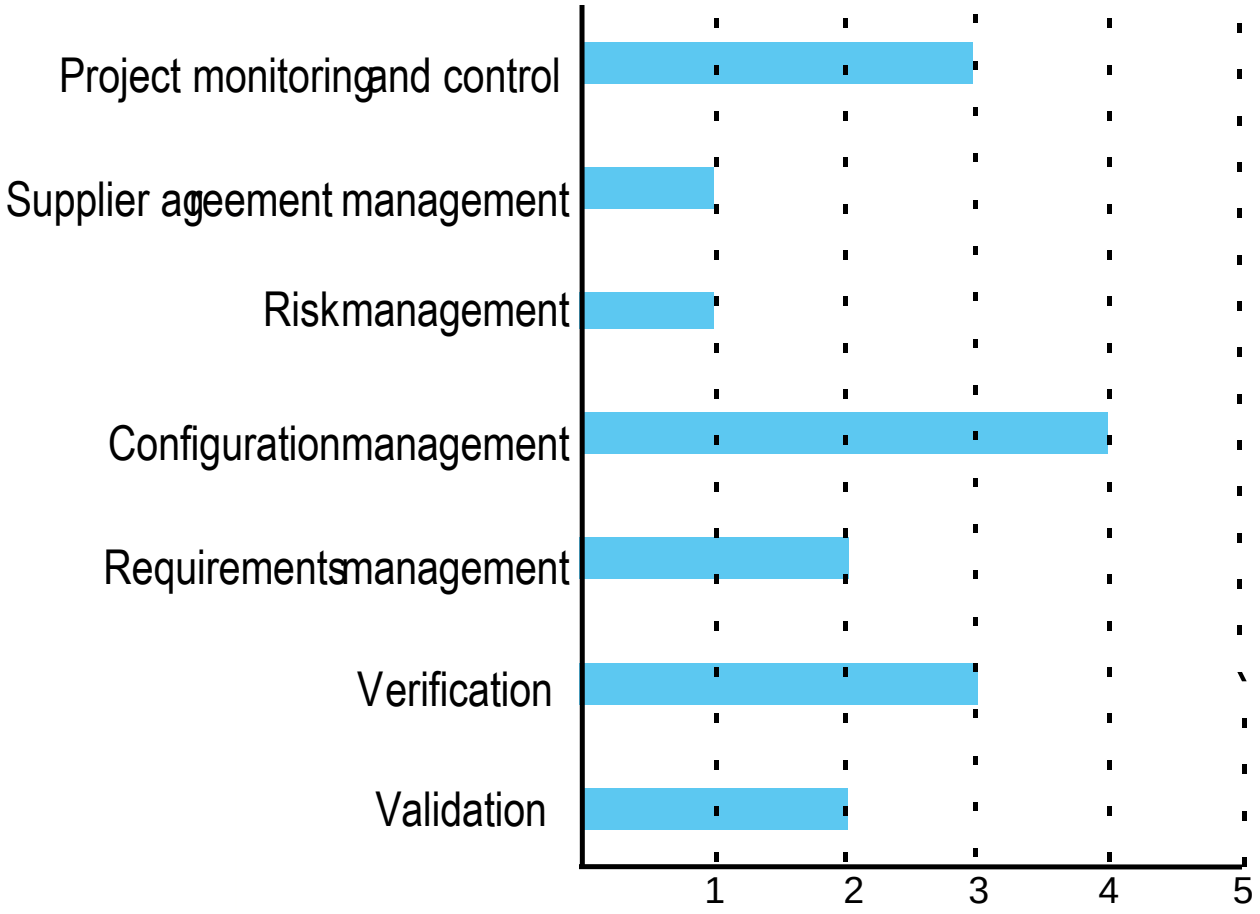
- Examines processes used in an organisation and assesses maturity in each process area
- Merged into one final "grade" using a 6-point scale:
  - Not performed;
  - Performed;
  - Managed;
  - Defined;
  - Quantitatively managed;
  - Optimizing.

# The Continuous CMMI Model

JACOBS UNIVERSITY

- First extension: staged CMMI model
  - Each maturity level has process areas and goals.
  - Eg, process area associated with "managed level" includes:
    *Requirements management; Project planning; Project monitoring and control; Supplier agreement management; Measurement and analysis; Process and product quality assurance*.
- Next extension: continuous CMMI model
  - finer-grain: considers individual or groups of practices, assesses their use
  - maturity assessment not a single value, but one maturity value per area
  - each process area: levels 1…5
  - Advantage:
    organisations can pick and choose process areas to improve according to their local needs

# Sample Process Capability Profile



# Open Source Maturity Model

JACOBS UNIVERSITY

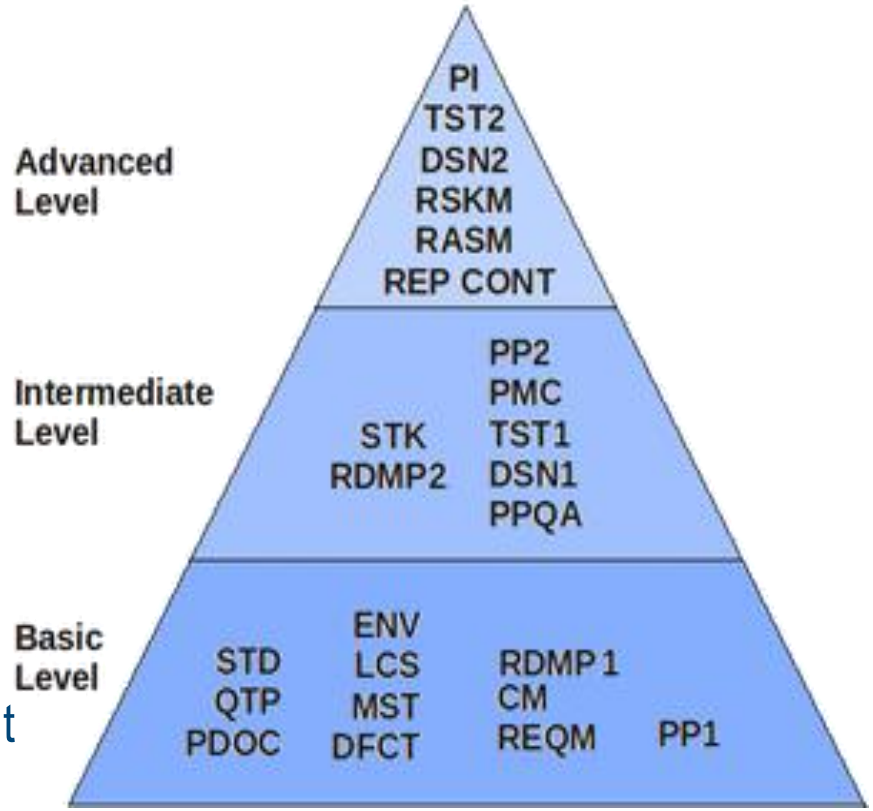
- **QualiPSo OpenSource Maturity Model (OMM)**
  - Methodology for assessing the Free/Libre Open Source Software (FLOSS) development process
  - see http://en.wikipedia.org/wiki/OpenSource_Maturity_Model


- helps in building trust in the development process of companies using or producing FLOSS
  - To enable use of FLOSS software in production, not only in prototypes

# QualiPSo Maturity Levels

JACOBS UNIVERSITY

- Basic (few necessary practices)
- Intermediate
- Advanced
  - PI – Product Integration
  - RSKM – Risk Management
  - TST2 – Test Part 2
  - DSN2 – Design 2
  - RASM – Results of third party assessment
  - REP – Reputation
  - CONT – Contribution to FLOSS Product from SW Companies




[QualiPSo]

# Wrap-Up

JACOBS UNIVERSITY

- CMM(I): assess IT company on its maturity wrt. managing its own processes
- Process improvement in CMM(I) based on reaching a set of goals related to good software engineering practice
- CMMI: summary value → detailed assessment on several parameters

**Real World Benefits: Lockheed Martin M&DS**

SW CMM ML2 (1993) to ML 3 (1996) to CMMI ML5 (2002)

1996 - 2002
- increased software productivity by 30%
- decreased unit software cost by 20%
- decreased defect find and fix costs by 15%