

# Software Development Overview

Joachim von Hacht

# Software Development Is ...

- ... **non-trivial!**
- ... problem solving
- ... a young engineering discipline
  - Somewhat of an art
- ...in between very informal (dynamic/chaotic)
- ...short of mathematical tools (formulas)
- ...normally a group task
- ...highly dependent on communication

To handle the complexity a **software development process** is used (opposite: "Happy hacking")

# Software development process

- ... is a framework that is used to structure, plan, and control the process of developing an program (system).

# Current Situation

- Many attempts have been made to...
  - ...developing processes (aka methodologies)
  - ...define the activities (tasks)
  - ...developing "best practices"
  - ...developing tools
  - ...
- ... but still there's no **"Silver Bullet"**
  - Albeit, many impressing results...

# Process Philosophies

- Big design up front (BDUF), heavy process
  - Everything is specified before starting to implement (the traditional engineering approach)
  - Pros: Will possible save time later on
  - Cons: Hard to handle changes (design obsolete before we even begin)
- Agile development, lightweight process
  - Build incrementally (in small steps) and learn
  - Pros: Quick adaption to changes/problems
  - Cons: Insufficient design and documentation (missing general aspects of the problem)
- ... and many others
  - See Wikipedia for a list
  - Latest hype: Scrum

# Philosophy In Course

- We use a basic agile process
  - There should always be something to run (after a short start period)
  - We'll build incrementally (adding functionality, etc. )  
= **iterative development**
  - We test continuously
    - **Daily build and smoke** = The software should be built and the tests run after every workday
  - In between we re-factor the code base
    - No new functionality but a better structure
- Small design up front
  - We'll have some code to run
  - Try to learn from it and define a basic design before implementing

# Process Phases

- Requirement Elicitation
  - What are we going to build?
- Analysis
  - Build an model of it.

This has nothing  
with computers  
or programming  
to do

- System design
  - Use previous steps to create a high level design of a solution
- Detailed design
  - Refine/extend/transform system design to a more detailed solution.
- Implementation
  - Implement it.

Not strictly ordered.  
Have to do "loops", work  
in parallel, revise, etc.

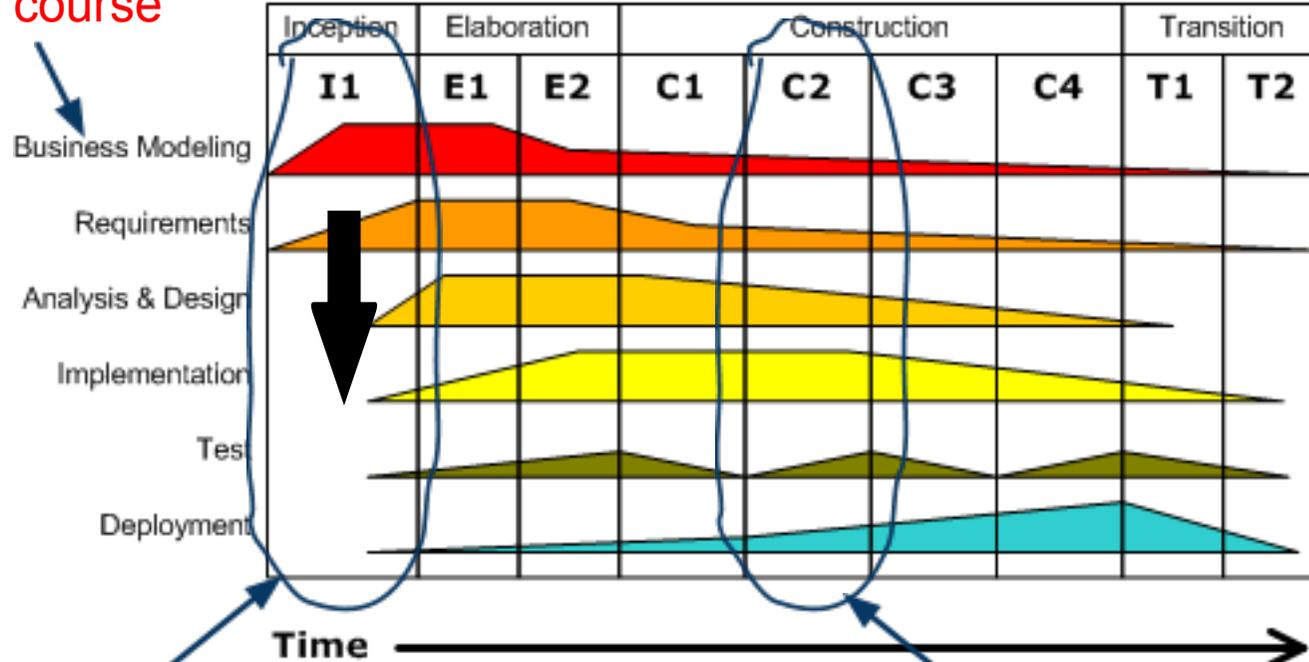
# Iterative Development (real life version)

One iteration all phases top-down

## Iterative Development

Business value is delivered incrementally in time-boxed cross-discipline iterations.

Not in this course

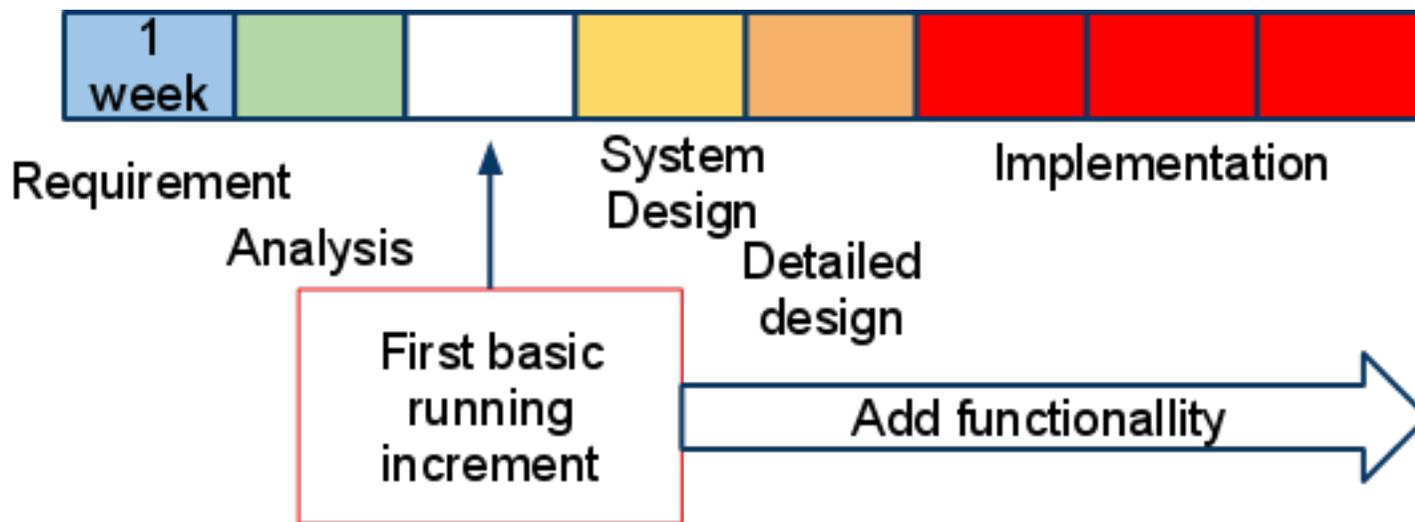


A primitive but running application with very limited functionality

This iteration has much more functionality

# Iterative Development (course version)

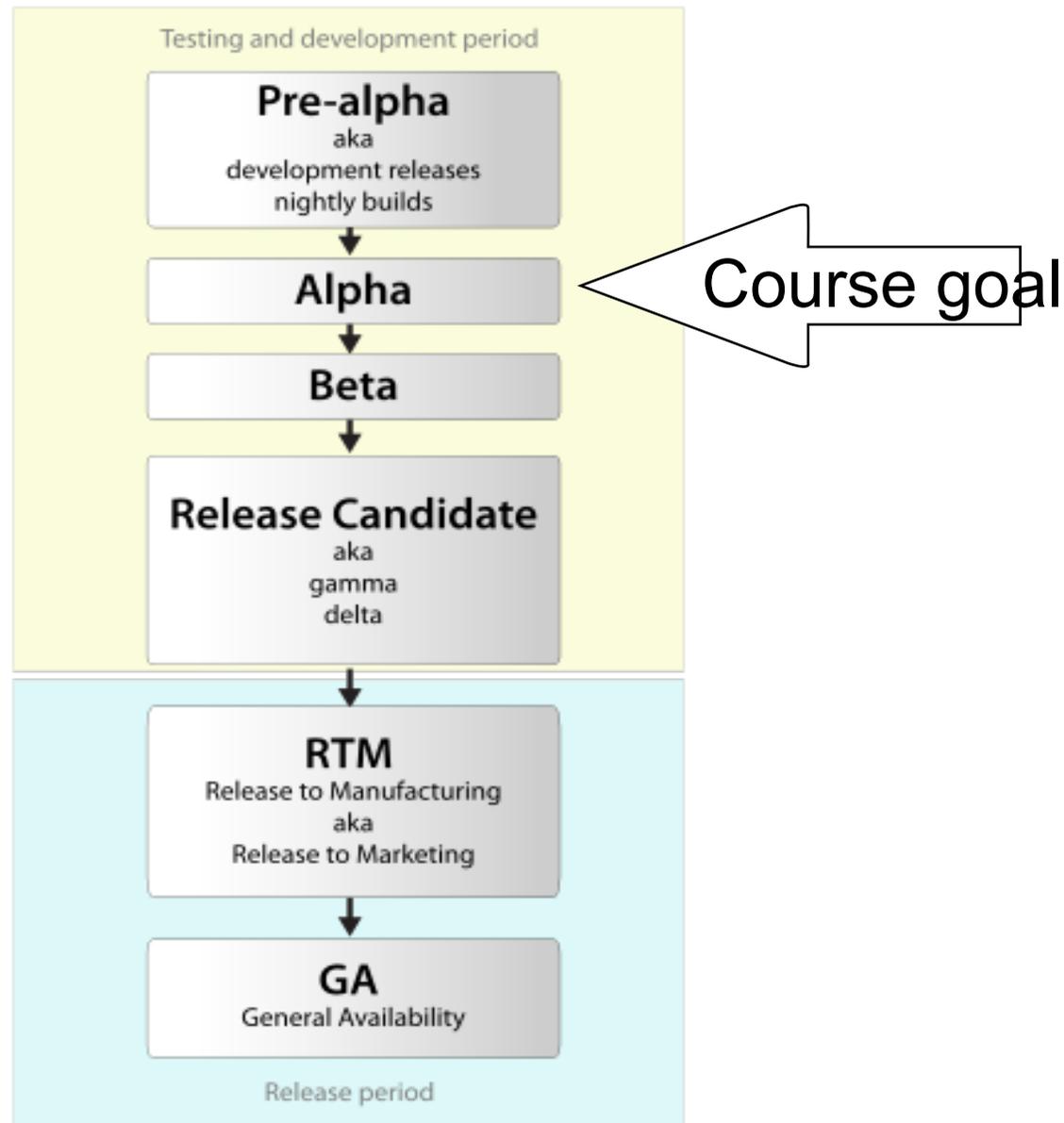
Process Outline (left to right)



- Details see **road map** on course page

# Software Release Cycle

Normally you will not be able to present a GA application

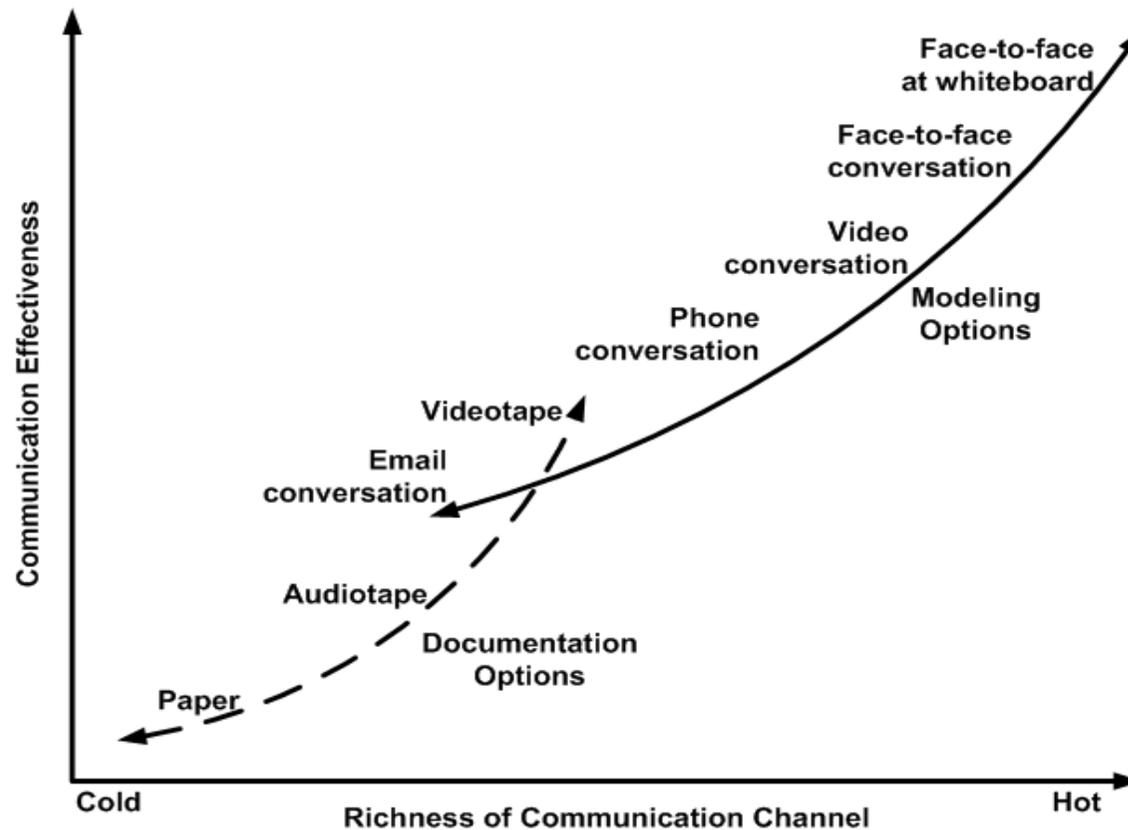


# Programming paradigms

- Fundamental style of computer programming
  - Functional: Program is a function
  - Structured: Programs is a collection of subroutines
  - Object oriented: Program is a model
  - ...others...
- We use the OO style
  - Some claimed benefits
    - The problem is composed of interacting entities (objects), not functions or subroutines. Mental picture and problem match.
    - It's a 1:1 mapping between problem and solutions, possible to use the same concepts from problem to code
    - **Traceability**, possible to trace the origin of the code all way back to problem.
    - Technical benefits compared to structured programming

# Software Development and Communication

- Effective communication is a fundamental requirement for software development.



Copyright 2002-2005 Scott W. Ambler  
Original Diagram Copyright 2002 Alistair Cockburn

# Software Development Documentation

- The fundamental issue is communication
- Documentation should be concise: overviews/roadmaps are generally preferred over detailed documentation
- Documentation is as much a part of the system as the source code
- Document stable things, not speculative things
- The benefit of having documentation must be greater than the cost of creating and maintaining it
- Developers rarely trust the documentation, particularly detailed documentation because it's usually out of sync with the code
- ...

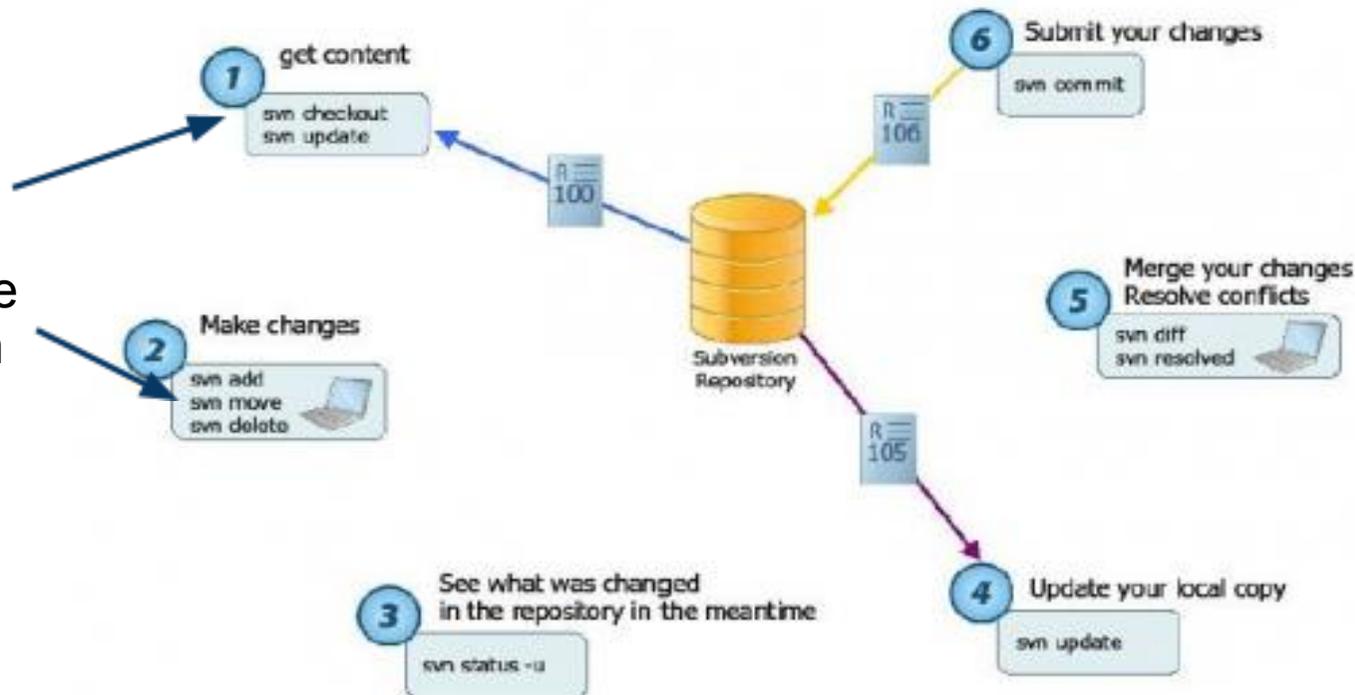
# Communication and Documentation in Course

- Find a room with a whiteboard and gather
  - Don't spread the group!
- Use documentation to communicate between members
  - Document with a Purpose...
    - Really try to identify crucial points for understanding
    - Define important terms
  - ..it's not the size
- With high quality source code and a test suite to back it up you need a lot less system documentation
  - More later...
- Templates on course page

# Organizing Software Development

- Issue tracker
  - Possible use //TODO in Eclipse
  - Better: Google code (or similar)
- Version handling (for everything), Apache Subversion (svn)

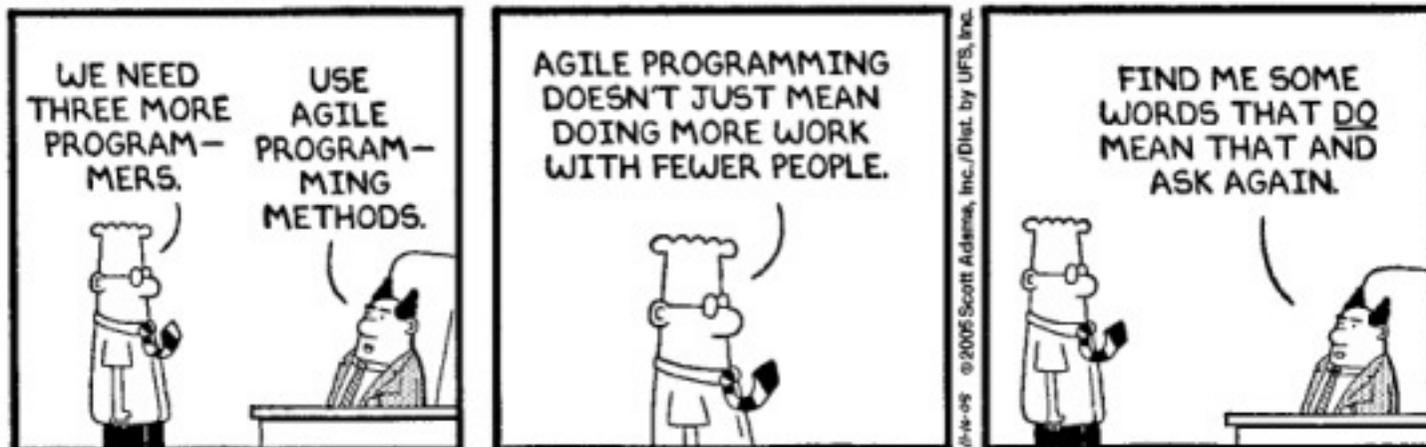
svn  
commands,  
written on  
command line  
(also a plugin  
for Eclipse)



# Hmmm

## DILBERT

BY SCOTT ADAMS



# Summary

- We will try to use a basic (agile) process as a guide during project
- The process has 5 steps
  - Each step has a few tasks, more later...
  - After first 2 steps there should be something to run
- Process not strictly ordered

# Checklist week 1

- Form a group
- Select a project and name it
- Mail group data to [hajo@chalmers.se](mailto:hajo@chalmers.se) (see Course PM)
- Set up project site (visit workshop Tuesday)
- First group meeting Thursday (schedule on course page)
  - Start working on "use cases"... to be continued...