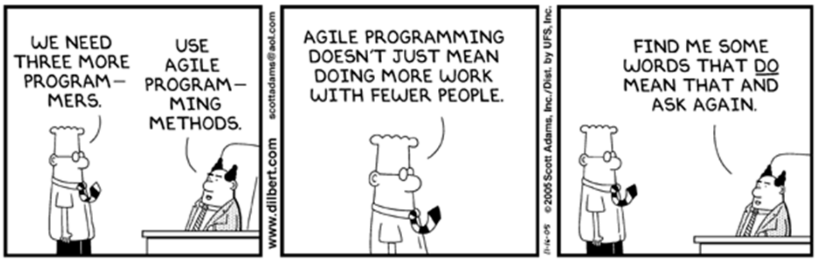
D7025E

Agile Programming

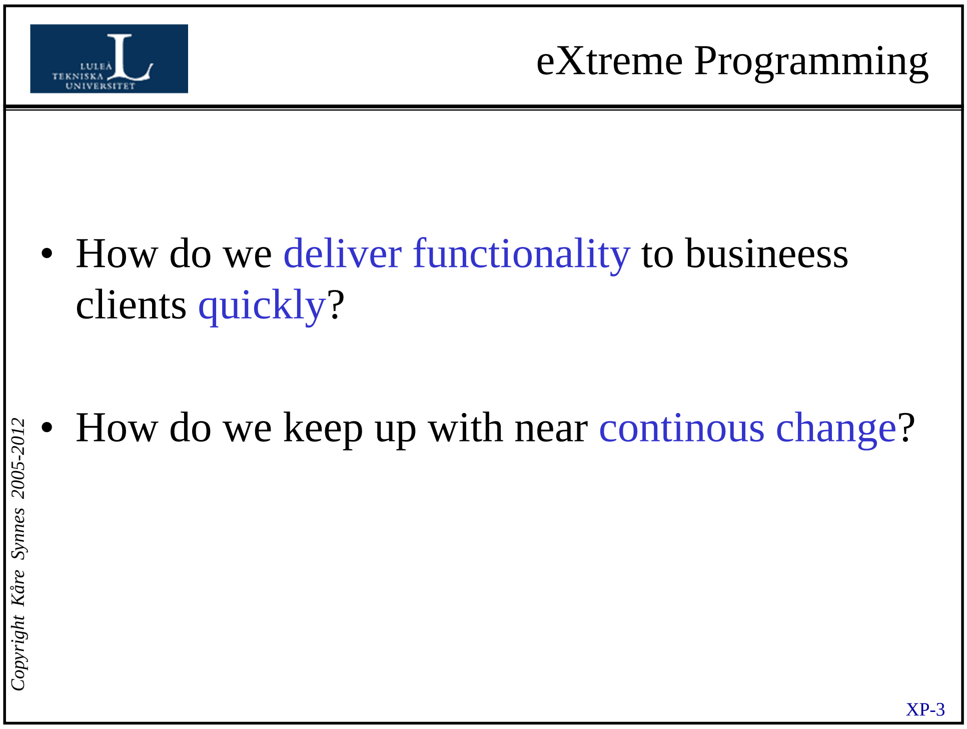
eXtreme programming (and SCRUM)


WE NEED THREE MORE PROGRAM-MERS.
USE AGILE PROGRAM-MING METHODS.
scottadams@aol.com
www.dilbert.com
AGILE PROGRAMMING DOESN'T JUST MEAN DOING MORE WORK WITH FEWER PEOPLE.
©2005 Scott Adams, Inc./Dist. by UFS, Inc.
FIND ME SOME WORDS THAT DO MEAN THAT AND ASK AGAIN.

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

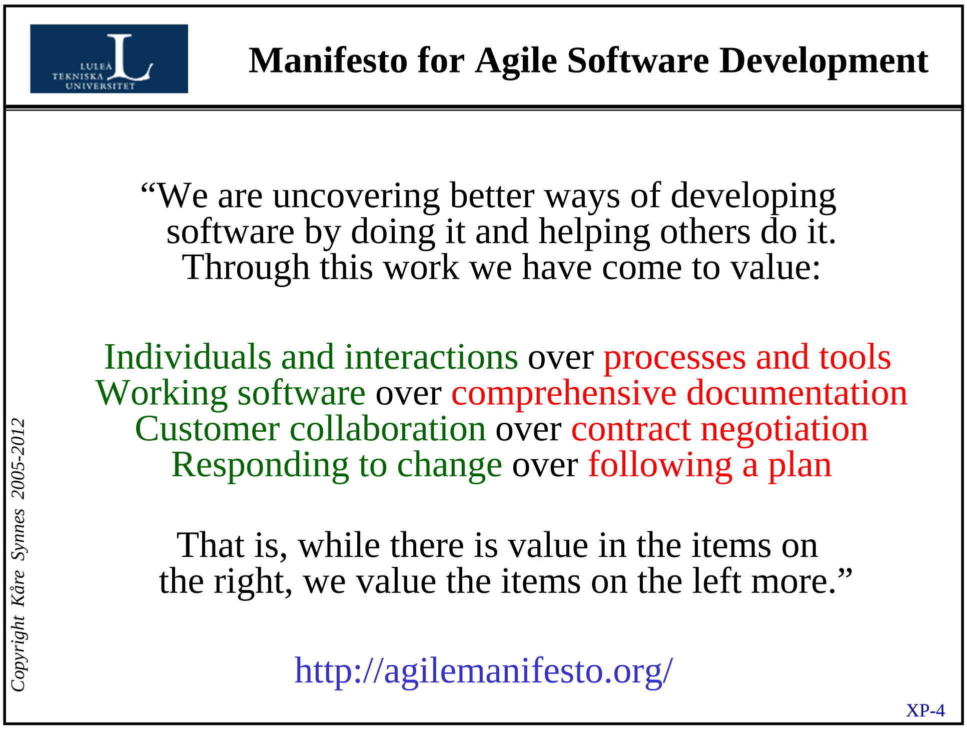
React to changing requirements!

- To date, most methodologies have treated software development as a manufacturing process, with the software proceeding along the requirements-analysis-design-code-test-maintain assembly line.
- This approach has an important assumption - that the shape of the finished product is known before the process begins.
- Most modern software projects cannot satisfy this assumption. The customer is specifying something completely new, and needs constant feedback to validate their choices.
- In turn, the programmers need to have a methodology that welcomes changing requirements so that they can react to feedback.



eXtreme Programming

- How do we deliver functionality to busineess clients quickly?
- How do we keep up with near continous change?



Manifesto for Agile Software Development

"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."

http://agilemanifesto.org/

XP Principles

- **Communication.** An XP team thrives on shared understanding of the problem and the software, and the most efficient and effective method of achieving shared understanding is face-to-face communication. Anything that obstructs efficient communication needs to be removed.
- **Simplicity.** Simplicity is the art of maximizing the amount of work not done. Dee Hock, former CEO of Visa International, says *"Simple, clear purpose and principles give rise to complex, intelligent behavior. Complex rules and regulations give rise to simple, stupid behavior".*
- **Courage.** Successful software teams need to operate on the edge of chaos - they need to go as fast as they possibly can without losing control. This means that sometimes they fail. If people are scared to fail then they'll go too slowly.
- **Feedback.** Often project teams and their customers don't realize they're in trouble until a short time before delivery. XP teams get frequent feedback - week to week by delivering working software, but also minute to minute through testing tools and any other mechanism they can implement.

The 12 Practices

Coding-oriented practices

#1 Testing

#2 Coding standards

#3 Common metaphor

#4 Refactoring

Design-oriented practices

#5 Simple design

#6 Small releases

#7 Continuous integration

Social, Psychological, and Organizational Practices

#8 The planning game

#9 Pair programming

#10 Collective ownership of code

#11 40-hour week

#12 On-site customer

Bonus Practices

- Small steps*
- Stand-up meetings*
- Continuous learning*

Coding-oriented Practice
#1: Testing

- Traditional:
  programmer performance = kloc
  tester performance = defects found
  - No one interested in reducing defects before testing!

- eXtreme:
  - Develop **test before code** and let tests drive the development!
  - Lifecycle:
    Listen (requirements, interface), test, Code, test, Design (Refactoring), test
  - Automated tests! Instant feedback!

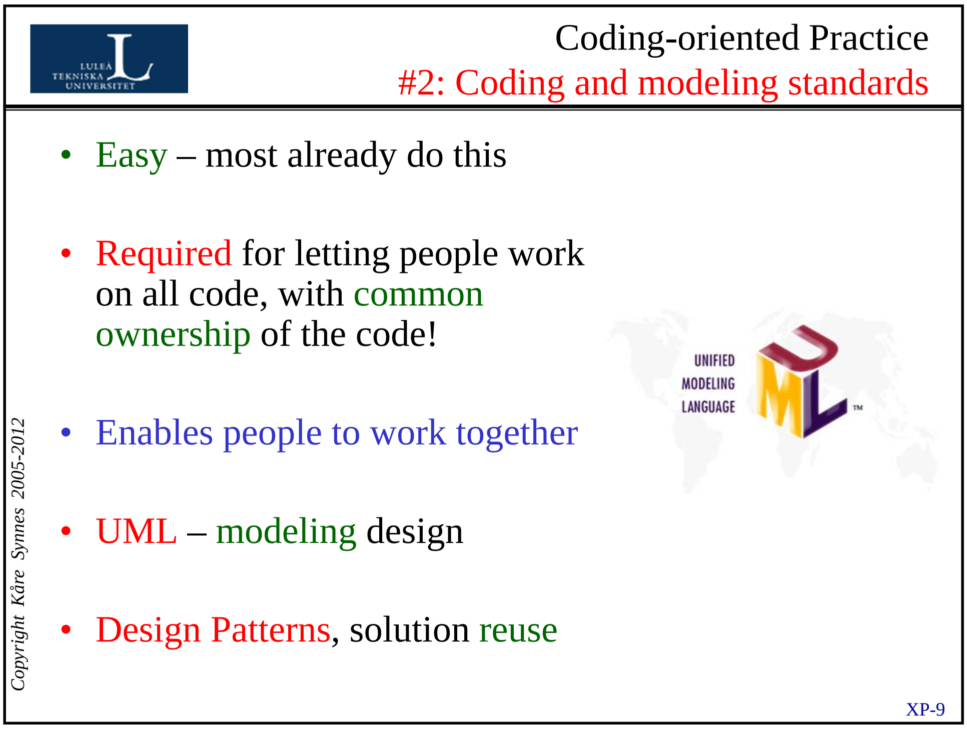
XP Activities
Testing

- All code **must** have unit/module tests.

- All code **must** pass all unit tests before it can be released.

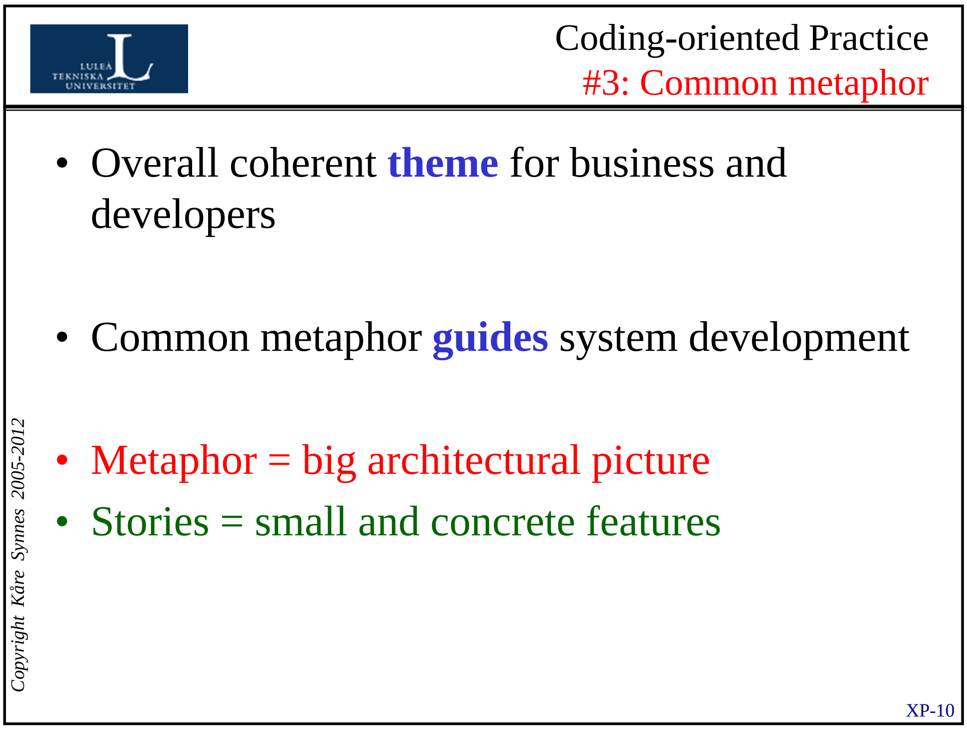
- When a bug is found tests **must** be created.

- Functional/acceptance tests are run **often** and the score is published!



Coding-oriented Practice
#2: Coding and modeling standards

- Easy – most already do this
- Required for letting people work on all code, with common ownership of the code!
- Enables people to work together
- UML – modeling design
- Design Patterns, solution reuse



Coding-oriented Practice
#3: Common metaphor

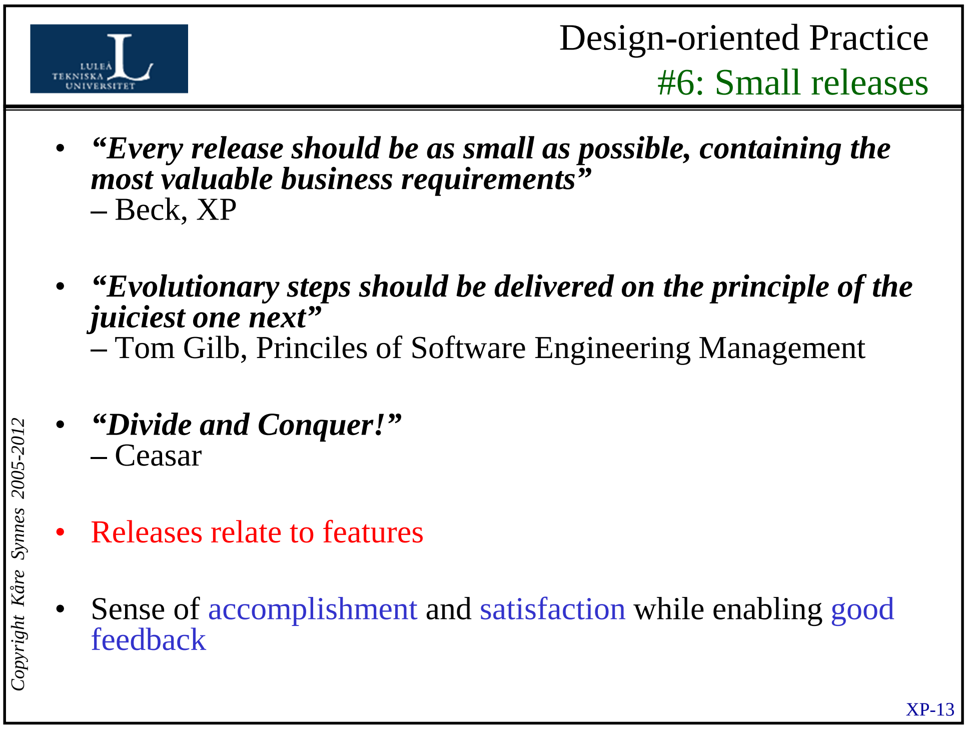
- Overall coherent **theme** for business and developers
- Common metaphor **guides** system development
- Metaphor = big architectural picture
- Stories = small and concrete features

Coding-oriented Practice
#4: Refactoring

- **Not no design, but continous design!**
- Reduce redundancy, eliminate unused functionality, simplify design
- Removes fear of change and builds confidence
- Nothing is set in stone if people can see a way to make it better and if it’s possible to do so

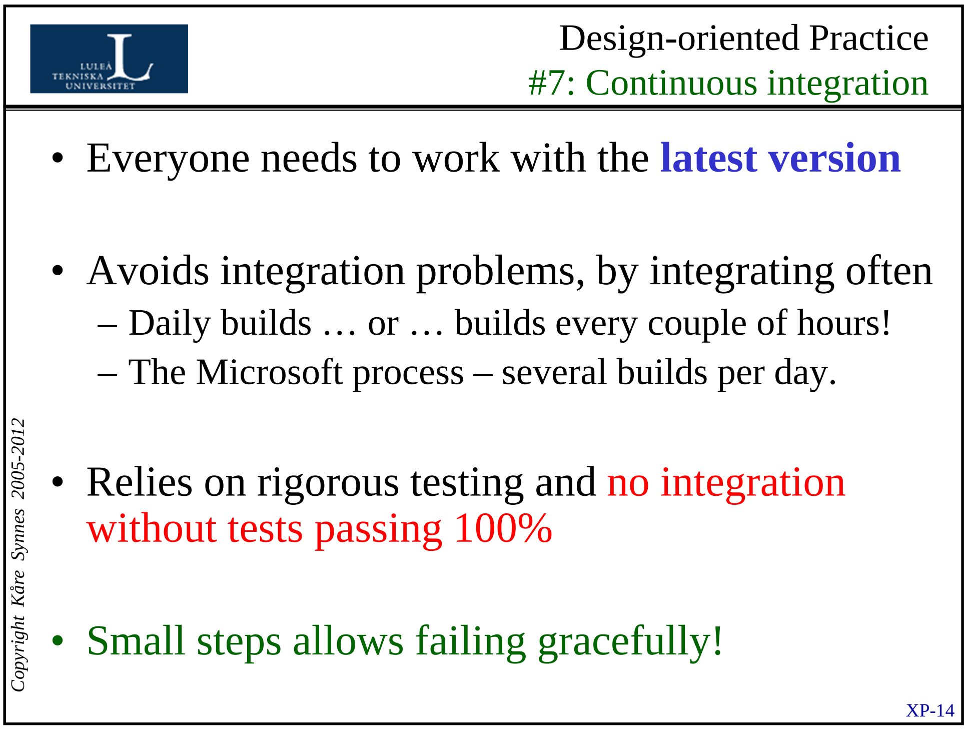
Design-oriented Practice
#5: Simple design

- “Do the simplest thing that could possibly work”
- *“If you believe the future is uncertain, and that you can cheaply change your mind, then putting in functionality on* ***speculation is crazy****”*
  - Beck
- Do not add functionality before it is needed!
- Guesswork leads to developing things we do not need!
- Refactor!!



Design-oriented Practice
#6: Small releases

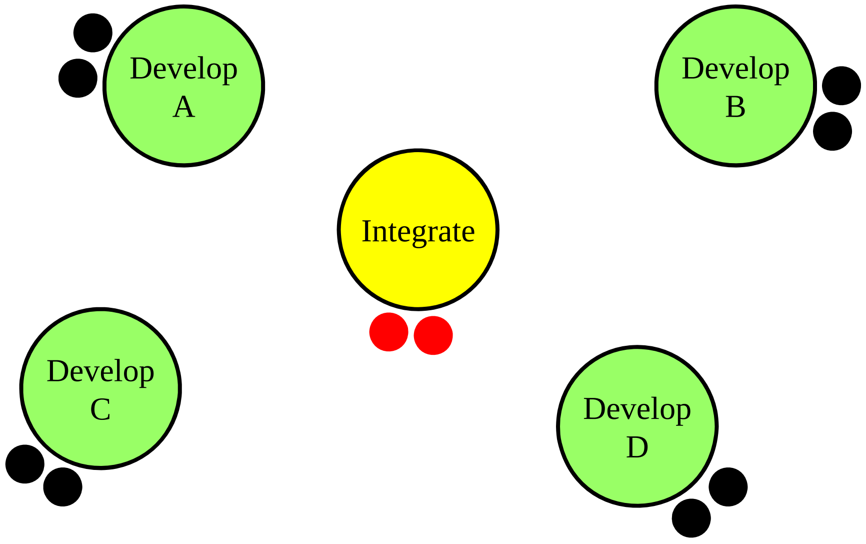
- ***"Every release should be as small as possible, containing the most valuable business requirements"***
  – Beck, XP
- ***"Evolutionary steps should be delivered on the principle of the juiciest one next"***
  – Tom Gilb, Princiles of Software Engineering Management
- ***"Divide and Conquer!"***
  – Ceasar
- Releases relate to features
- Sense of accomplishment and satisfaction while enabling good feedback



Design-oriented Practice
#7: Continuous integration

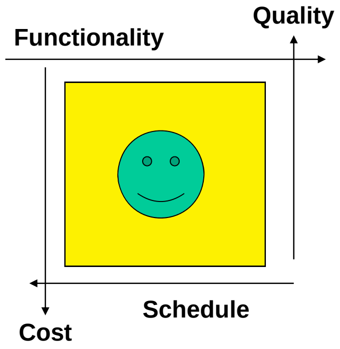
- Everyone needs to work with the **latest version**
- Avoids integration problems, by integrating often
  - Daily builds … or … builds every couple of hours!
  - The Microsoft process – several builds per day.
- Relies on rigorous testing and no integration without tests passing 100%
- Small steps allows failing gracefully!

The Integration Station



Social, Psychological and Organizational Practice
#8: The planning game

- **Short, 3-4 week cycles, frequent updates**
- *Splitting business and technological priorities*
- Stories defines feature requirements, in card format
- Involves designers and customers in choosing features and estimating time

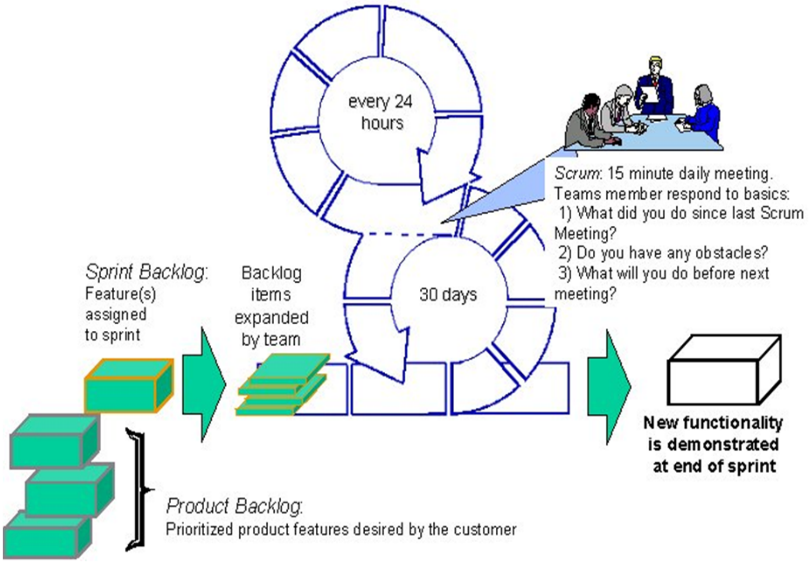


On time,
within budget and
meet requirements!

XP Activities
Planning

- User stories are written.
  - Use Cases
  - Storyboarding
- Release planning creates the schedule.
- Make frequent small releases.
- The Project Velocity is measured.
- The project is divided into iterations.
- Iteration planning starts each iteration.
- Move people around.
- A stand-up meeting starts each day.
- Fix XP when it breaks.

SCRUM Planning



every 24 hours

Scrum: 15 minute daily meeting.
Teams member respond to basics:
1) What did you do since last Scrum Meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?

30 days

Sprint Backlog: Feature(s) assigned to sprint

Backlog items expanded by team

New functionality is demonstrated at end of sprint

Product Backlog: Prioritized product features desired by the customer

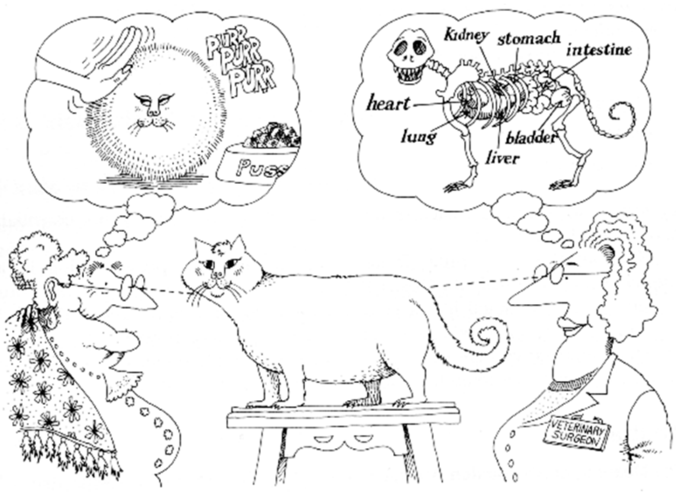
”Failing gracefully”

- Experience is won through an equal amount of successes and failures.
- One way to become truly successful is to know how to ”fail gracefully”!
- Start small and simple, then evolve in small steps!
  - A failure does not mean that too much is lost. It’s small!
- Manage risks by:
  - Identifying risks early, then weigh value against risk to prioritize work.
  - Doing the parts of the system with least value/risks ratio last!
  - Starting with studying critical risks! (The hardest parts)

Social, Psychological and Organizational Practice
#9: Pair programming

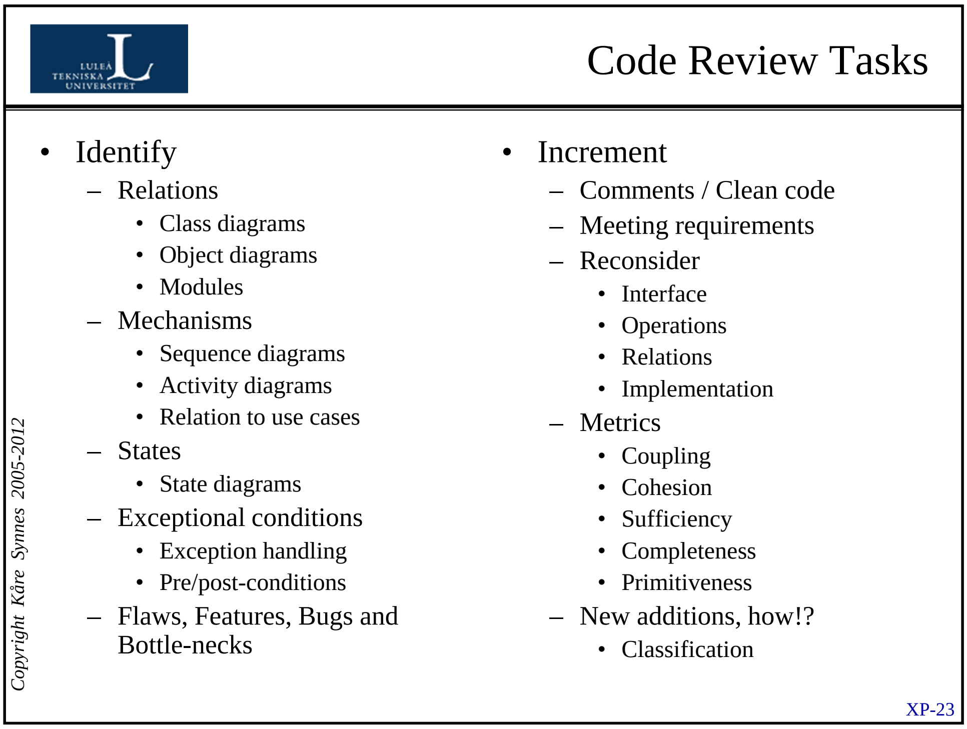
- All code is written by 2 people at one machine
- One person tactical (writing code and tests), the other strategic (reviewing and thinking)
- Time to isolate defect:
  - 15 hours per defect testing
  - 2-3 hours per defect using inspection
  - 15 minutes per defect before inspection!
  - Few minutes with pair programming!!
- Pairs change often
- Quality is a big win
- People stay more focused and ‘on target’
- Inspection! Code reviews and Walkthroughs.
  - Collaborative interaction
  - Speed learning, better programming practices
  - Uncover and prevent defects, cost-efficiently

Who is right?



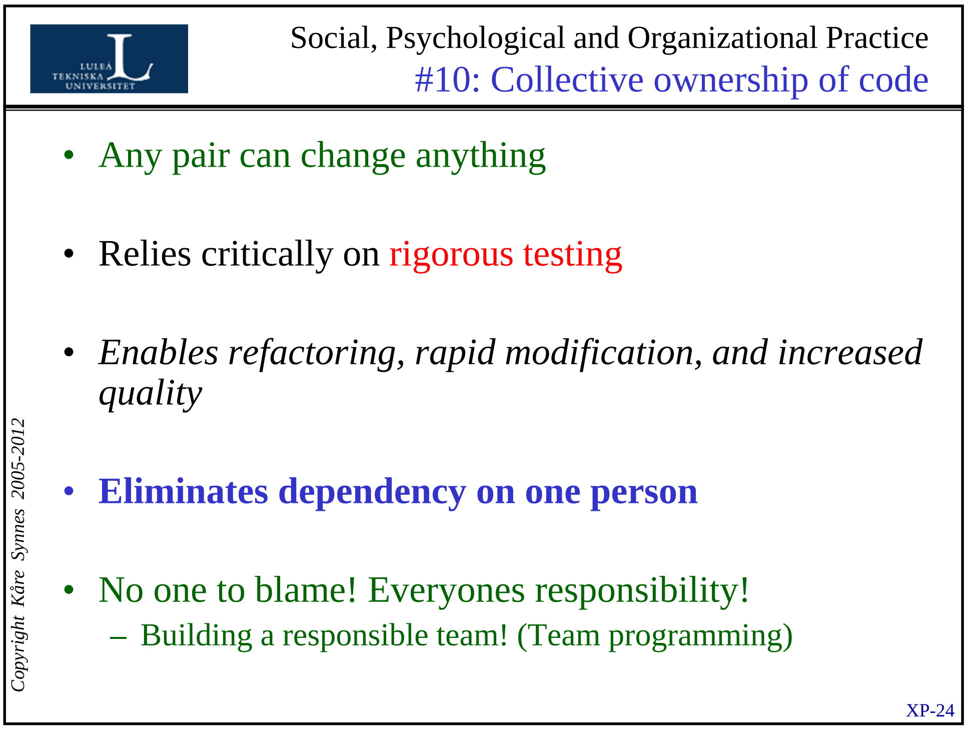
Code Review

- Weekly for a team of designers/developers
- Share information about the system
  - Redundancy!
- Create a common view of the system!
- Documents the code design
  - Diagrams in UML
- Find flaws, bugs, features and bottlenecks
- Classification!



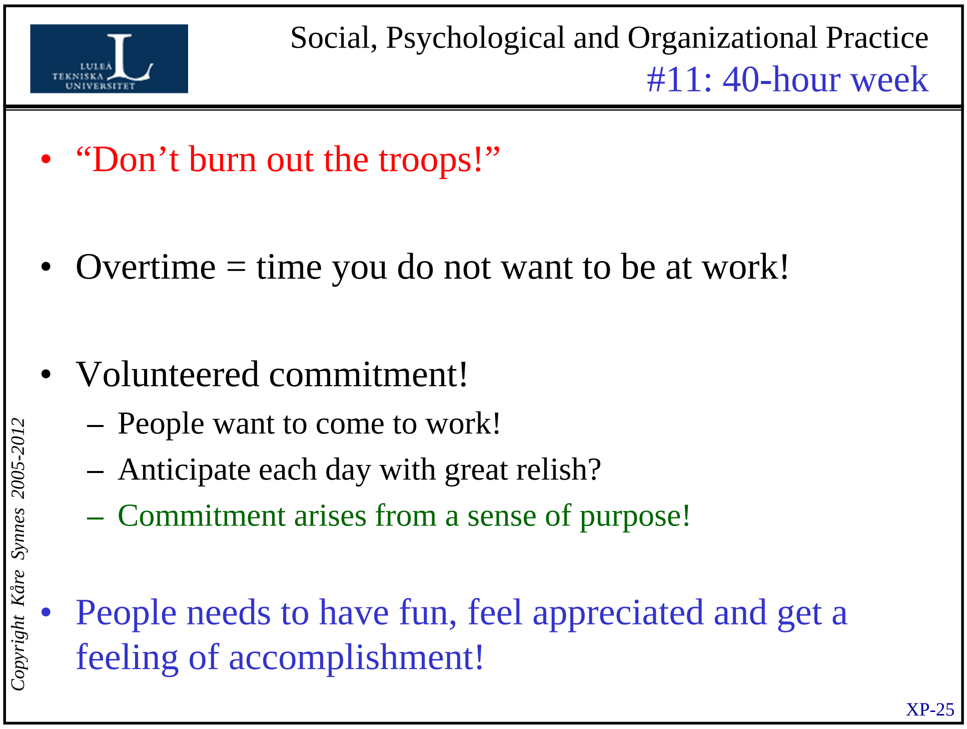
Code Review Tasks

- Identify
  - Relations
    - Class diagrams
    - Object diagrams
    - Modules
  - Mechanisms
    - Sequence diagrams
    - Activity diagrams
    - Relation to use cases
  - States
    - State diagrams
  - Exceptional conditions
    - Exception handling
    - Pre/post-conditions
  - Flaws, Features, Bugs and Bottle-necks
- Increment
  - Comments / Clean code
  - Meeting requirements
  - Reconsider
    - Interface
    - Operations
    - Relations
    - Implementation
  - Metrics
    - Coupling
    - Cohesion
    - Sufficiency
    - Completeness
    - Primitiveness
  - New additions, how!?
    - Classification



Social, Psychological and Organizational Practice
#10: Collective ownership of code

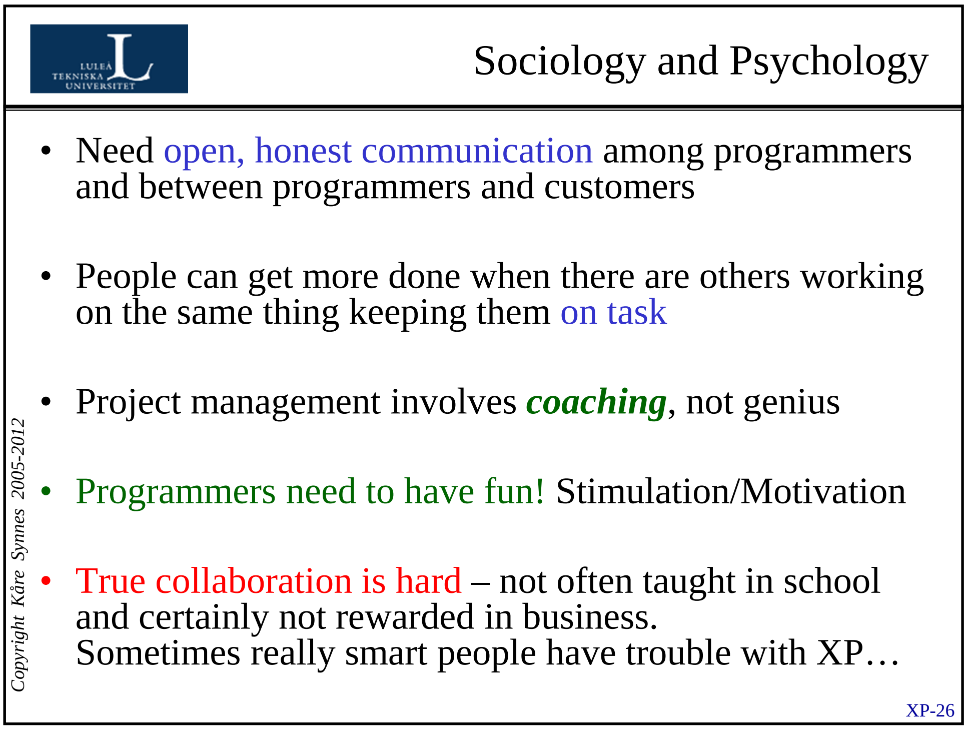
- Any pair can change anything
- Relies critically on rigorous testing
- *Enables refactoring, rapid modification, and increased quality*
- **Eliminates dependency on one person**
- No one to blame! Everyones responsibility!
  - Building a responsible team! (Team programming)



Social, Psychological and Organizational Practice

#11: 40-hour week

- “Don’t burn out the troops!”
- Overtime = time you do not want to be at work!
- Volunteered commitment!
  - People want to come to work!
  - Anticipate each day with great relish?
  - Commitment arises from a sense of purpose!
- People needs to have fun, feel appreciated and get a feeling of accomplishment!



Sociology and Psychology

- Need open, honest communication among programmers and between programmers and customers
- People can get more done when there are others working on the same thing keeping them on task
- Project management involves ***coaching***, not genius
- Programmers need to have fun! Stimulation/Motivation
- True collaboration is hard – not often taught in school and certainly not rewarded in business. Sometimes really smart people have trouble with XP…

Social, Psychological and Organizational Practice
#12: On-site customer

- The oldest cry in software development – user involvement!
- Feedback is very important!
  - Save resources
  - Meet requirements
  - Feel accomplishment and satisfaction

XP Activities
Coding

- **The customer is always available.**
- **Code must be written to agreed standards.**
- **Code the unit test first.**
- **All production code is pair programmed.**
- **Only one pair integrates code at a time.**
- **Integrate often.**
- **Use collective code ownership.**
- **Leave optimization till last.**

Conclusions

- *Attracted to **elegance***
- The 12 Practices make sense on their own, and are **synergistic** when used **collectively**

Non-Extreme *versus* Extreme Programming

Limited customer contact	Customer on team
Central up-front design	*Open evolving design*
Build for the future too	Evolve; just in time
Complex implementation	*Radical simplicity*
Tasks assigned	Tasks self-chosen
Developers in isolation	*Pair programming*
Infrequent integration	Continuous integration
Limited communication	*Continual communication*

Do not use eXtreme Programming, If...

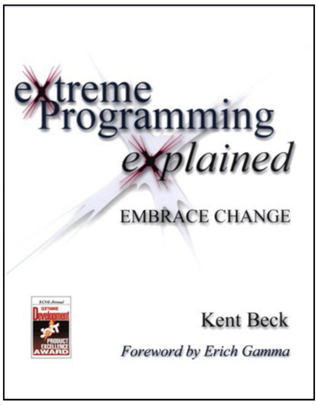
- You're using a process and developers and customers are happy with it!
- Your requirements are **truly** fixed
- You cannot keep the cost of change low in your environment

- Known bad spots:
  - "Dilbertesque" companies
  - More than about 20 programmers (unless teams)
  - Commitment to existing code to maintain existing applications
  - Long time required for feedback
  - Programmers separated in space

Teams that adopt XP
frequently find they are delivering
vastly higher quality software faster
than they could before.

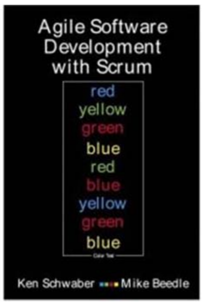
More on eXtreme Programming

- Cutter White Paper (good intro!): http://www.cutter.com/freestuff/ead0002.pdf
- ”XP Explained – Embrace Change”, Beck 2000
- The XP series by Addison&Wesley
- http://www.extremeprogramming.org/



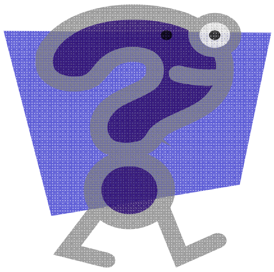
More on SCRUM

- Jeff Sutherland’s site http://www.jeffsutherland.org/scrum/
- “Scrum and XP from the Trenches”, by Henrik Kniberg *(Free to download!)* http://www.infoq.com/minibooks/scrum-xp-from-the-trenches
- “Agile Software Development with Scrum”, by Ken Schwaber and Mike Beedle
- http://www.controlchaos.com/



WE'RE GOING TO TRY SOMETHING CALLED EXTREME PROGRAMMING.
www.dilbert.com scottadams@aol.com
FIRST, PICK A PARTNER. THE TWO OF YOU WILL WORK AT ONE COMPUTER FOR FORTY HOURS A WEEK.
1/9/03 © 2002 United Feature Syndicate, Inc.
THE NEW SYSTEM IS A MINUTE OLD AND I ALREADY HATE EVERYONE.

unicorn@ltu.se



Conclusions

- The name of the game is ***agility***
- Traditional methodologies were developed to build software for low levels of change and reasonably predictable desired outcomes.
- But, the business world is no longer very predictable, and software requirements change at extremely high rates.

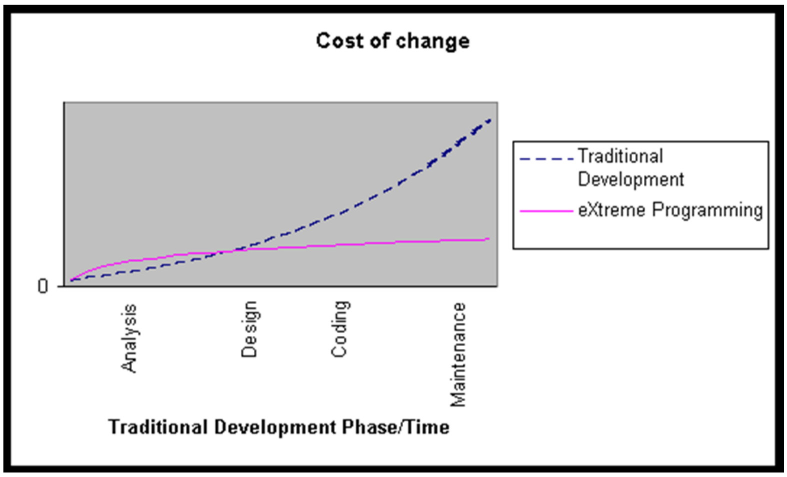
Four Critical Ideas

- Most of the practices from XP is not new, **they are as old as programming!**

- However, the conceptual foundation and how the practices are melded together is greatly enhancing older practices!

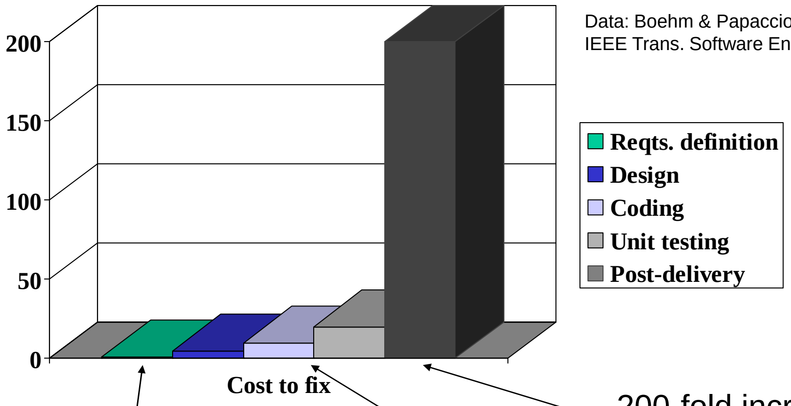
- The Cost of Change
- ReFactoring
- Collaboration
- Simplicity

Cost of Change



*” **XP keeps the cost of change low**, so that it's not much more expensive to implement a feature later than it is to implement it now, and then leverages this cost-of-change environment to **produce software faster**. ”*

The Cost of Delay in Fixing Requirements Errors



Data: Boehm & Papaccio (1988)
IEEE Trans. Software Eng.

Nominal
unit cost

20-fold increase
during development

200-fold increase
after delivery

eXtreme Programming Values

- Open, honest communication
- Rapid feedback at all levels
- Quality Work
- Assume Simplicity
- Incremental Change

- Small initial investment
- Embrace Change
- Travel light
- Teach learning
- Courage - play to win
- Local adaptation

eXtreme Philosophy

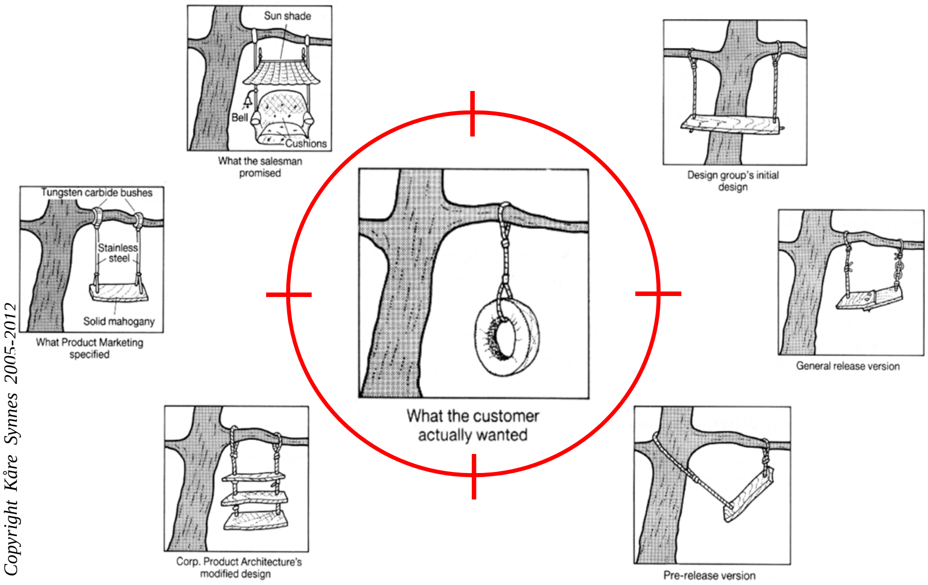
- Must have extremely **rapid** system development
- Must have extreme **customer involvement**
- Must be extreme in **avoiding defects**
- Must not be afraid to change code – in the extreme, one can “**embrace change**”
- Must have extreme involvement of coders – **everyone owns all the code**
- Must be extremely **respectful** of people and their personal, social, and psychological needs

eXtreme Programming

What is XP?

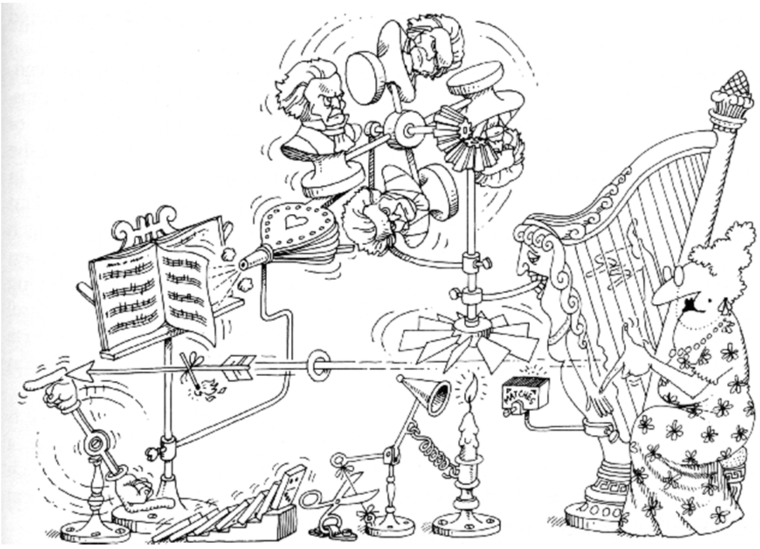
- A software development process aligned with “developer nature”
- It sounds very constraining, but it’s not!
- Targeted at dynamic projects developed with small co-located teams
- Based on social & collaborative values as well as technical practices
- Minimal! (At least concerning anything other than code and test cases…)

Customer focus


Sun shade
Bell
Cushions
What the salesman promised
Design group's initial design
Tungsten carbide bushes
Stainless steel
Solid mahogany
What Product Marketing specified
What the customer actually wanted
General release version
Corp. Product Architecture's modified design
Pre-release version

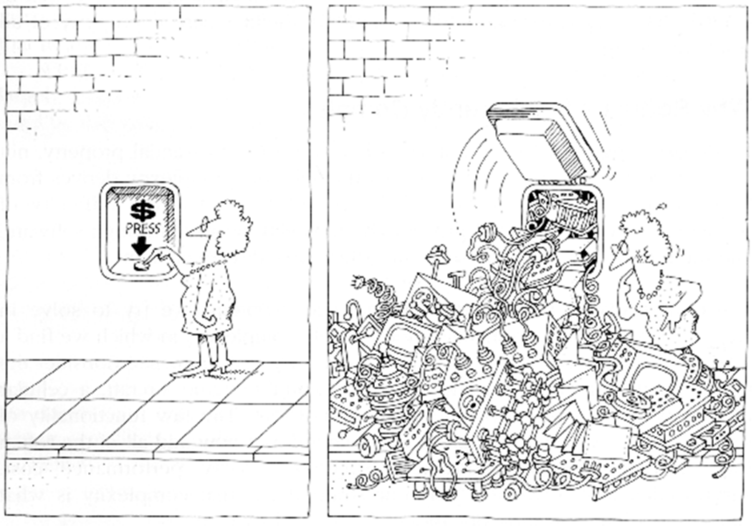
Copyright Kåre Synnes 2005-2012

Design of Software Systems = Control Chaos



Copyright Kåre Synnes 2005-2012

The Ultimate Aim = Simplicity!



KISS – Keep It Simple, Stupid!

SCRUM

The story:

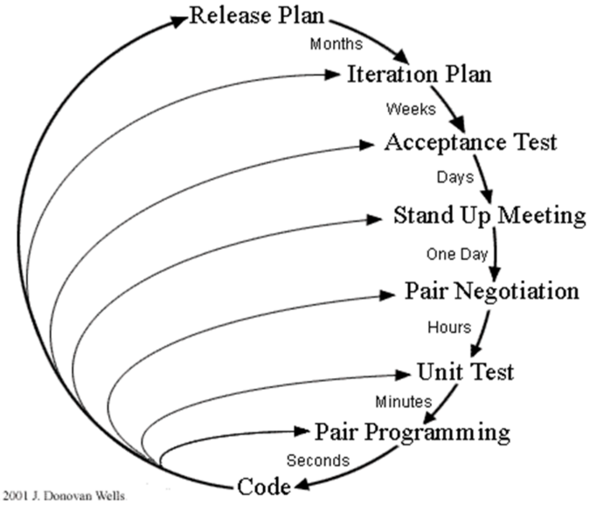
- Agile programming methodology
  - Light-weight
  - Rapid
- Jeff Sutherland and Ken Schwaber, 1995
  - Ideas from ’lean development’ in Smalltalk
- ”A simple framework for project management on complex projects”
- ”Extremely simple, but exceptionally hard”

eXtreme Programming

The story:

- Chrysler Comprehensive Compensation system, payrolls
  - OO/Smalltalk project
  - Started in the mid-1990s, stuck in 1997
  - Piloted eXtreme Programming practices and was ready within time and budget during the spring 1999
- ”The Three Extremoes”
  - Beck, Cunningham and Jeffries

Don Wells' feedback loop



Copyright 2001 J. Donovan Wells