10 Practices to Improve Your CSE Software Project Management

Software Engineering and Scientific Computing

Barbara Paech, Hanna Remmel
Institute of Computer Science
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
http://se.ifi.uni-heidelberg.de
valtokari@informatik.uni-heidelberg.de

software engineering heidelberg

RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

Schedule Second Day

9:00	Best practices for SE in CS Project Management
10:00	Break
10:30	eXtreme Hour
12:00	Lunch
13:00 Incl. a short break	Tools, Exercises Unit test Code Documentation
16.00	End

Programming in a small team

What is Ron doing?

Project management
Issue Tracking

I want to explain my ideas to Hermione

Modeling
Knowledge Management

I want to change Ginnys code

Version management,
Build management

I want to check Harrys changes

Quality assurance
Testing

[Kay 2011]

10 Practices to Improve Your CSE Software

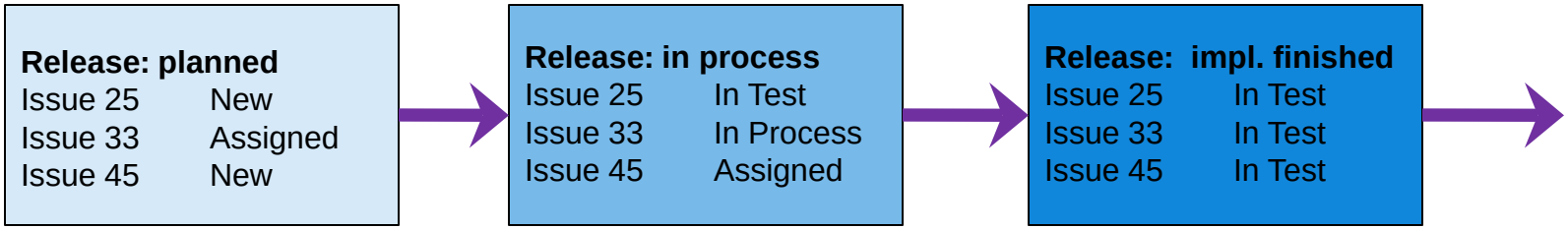
[Heroux 2009]

10 Practices to Improve Your CSE Software

- **Practice 1:** Use issue-tracking for requirements, features and bugs.
- **Practice 2:** Manage source with a version control tool
- **Practice 3:** Use configuration management tools
- **Practice 4:** Use a formal release process
- **Practice 5:** Create source-centric documentation
- **Practice 6:** Write tests first, run them often
- **Practice 7:** Use mail lists to communicate
- **Practice 8:** Use checklists for repeated processes
- **Practice 9:** Program tough stuff together
- **Practice 10:** Perform continual process improvement

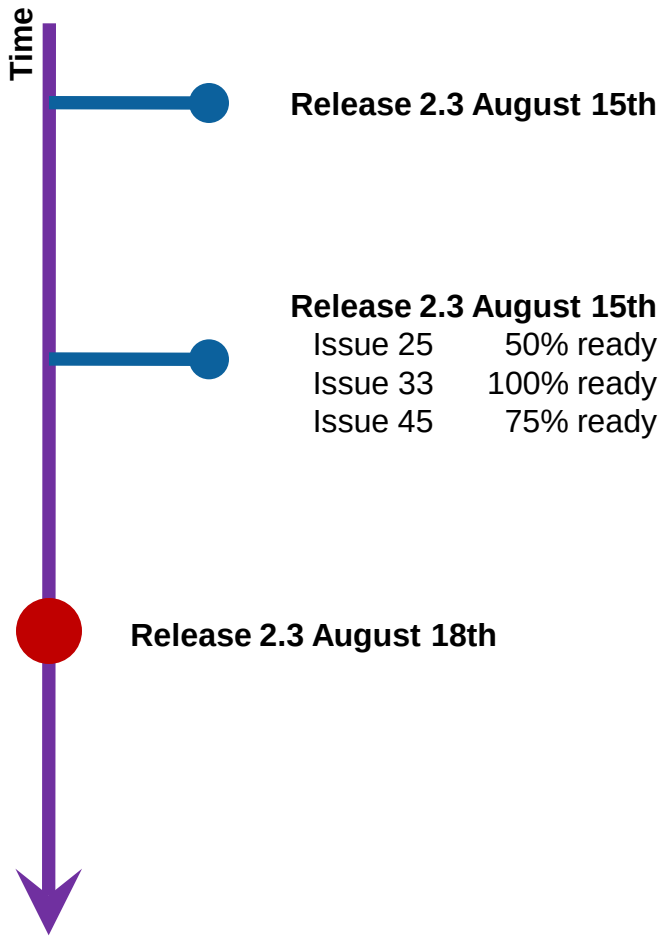
Practice 4: Use a formal release process

- Release = a configuration of the software, that is delivered to a „customer“
  - Customer does not have to be extern
  - Test release, „stable version“ of the software
- Assign issues (enhancements and bugs) to a release
- Trace the status of a release
  - Planned, in Process, implementation finished, tested, approved
  - Status depends on the status of assigned issues
- Plan
  - Purpose of a release
  - Timeline

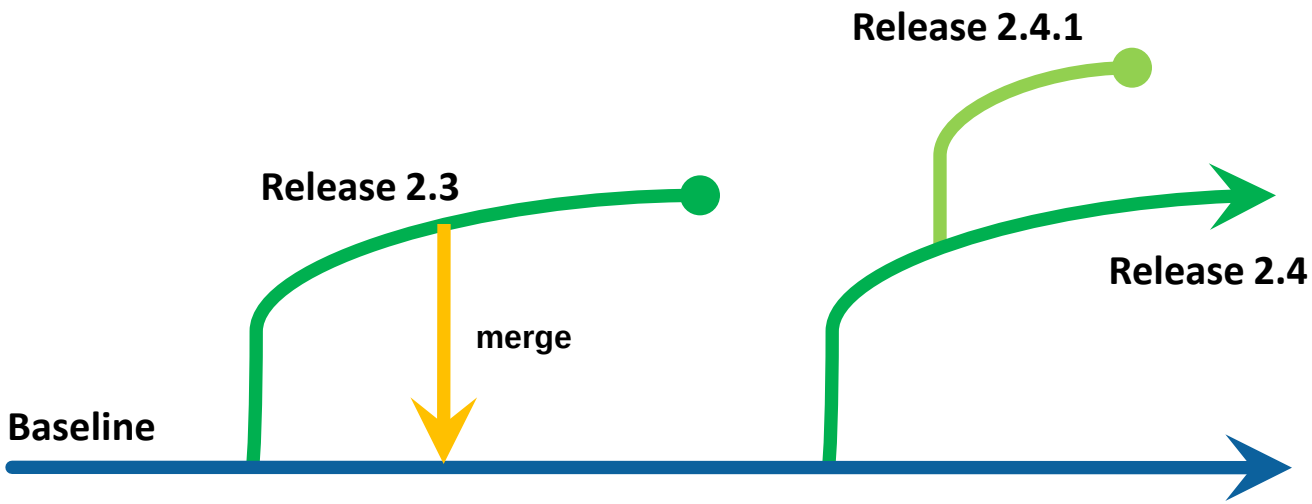


Planing Release

- **In the long term:** Project plan includes planing for releases (which kind of release and when)
- **In the medium term:** planing includes the issues (bugs and enhanchements) for one release
  - Example: issue can have different kind of release information:
    - Release as desired by the customer
    - Release as agreed with the development
    - Delivered release
- If there are dependencies between different software modules that are released separately, there must be overall release planing (integration plan)



Releases and source files



- Brances for releases 2.3, 2.4 and 2.4.1
- The release 2.3 branch is closed, since 2.3 is no longer in production and won't be maintained.

Baseline / Checkpoint and Labeln

- Baseline / tag
  - For a new release
  - To be able to reproduce the state of source files at this point of time
- Label sources in Subversion
  - Single Files,
  - Intermediate results,
  - …

Release Quality Assurance

- In a simple case
  - Run some reasonable set of tests on defined set of platforms
- When all necessary processes have been completed, a release can be completed with greater confidence
- For minor releases, a carefully chosen subset of the major release process could be used

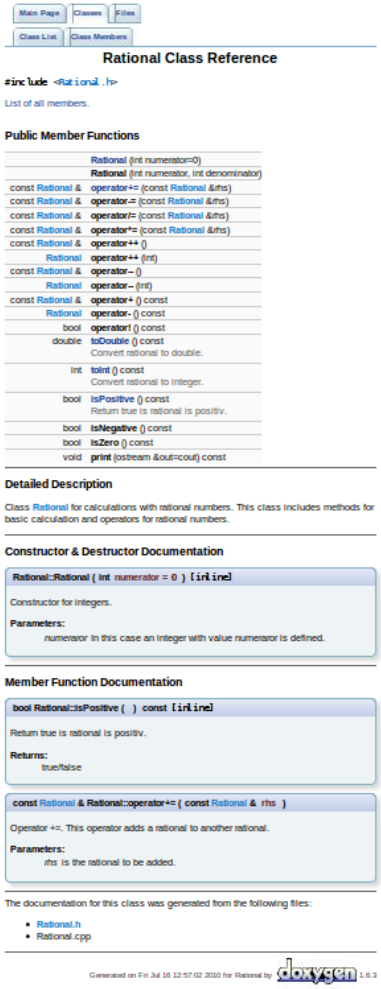
Practice 5: Create source-centric documentation

Point of view in Trilinos – project

- in scientific software engineering
  - documentation should be sufficient but minimal
  - No large-scale formal document generation
- A combination of near-to-the-source and in-source documentation
  - Functions and executable in source code (Doxygen)
  - Conceptual documentation near-to-source
  - Requirements, analysis and design in issue trackin tool
    - Bugzilla, Trac, Flyspray,…
    - UML graphics tools (e.g. Microsoft Vision, Doxygen)

Doxygen

- Why use an automated system?
  - Documentation is up-to-date
  - Reuse of your own comments
  - automatic formatting, and crosslinking
  - In-code comments carry important meta information
- Why doxygen?
  - It's free
  - OpenSource with installer
    - It's fairly comfortable to use
  - Configurable
    - With a basic style sheet, and
    - twiddling the options you can
    - customize many aspects of the documentation



Main Page | Classes | Files

Class List | Class Members

Rational Class Reference

#include <Rational.h>

List of all members.

Public Member Functions

	Rational (int numerator=0)
	Rational (int numerator, int denominator)
const Rational &	**operator+=** (const Rational &rhs)
const Rational &	**operator-=** (const Rational &rhs)
const Rational &	**operator/=** (const Rational &rhs)
const Rational &	**operator*=** (const Rational &rhs)
const Rational &	**operator++** ()
Rational	**operator++** (int)
const Rational &	**operator--** ()
Rational	**operator--** (int)
const Rational &	**operator+** () const
Rational	**operator-** () const
bool	**operator!** () const
double	**toDouble** () const Convert rational to double.
int	**toInt** () const Convert rational to integer.
bool	**isPositive** () const Return true is rational is positiv.
bool	**isNegative** () const
bool	**isZero** () const
void	**print** (ostream &out=cout) const

Detailed Description

Class Rational for calculations with rational numbers. This class includes methods for basic calculation and operators for rational numbers.

Constructor & Destructor Documentation

Rational::Rational (int numerator = 0) [inline]

Constructor for integers.

Parameters:
numerator In this case an integer with value numeraor is defined.

Member Function Documentation

bool Rational::isPositive ( ) const [inline]

Return true is rational is positiv.

Returns:
true/false

const Rational & Rational::operator+= (const Rational & rhs)

Operator +=. This operator adds a rational to another rational.

Parameters:
rhs is the rational to be added.

The documentation for this class was generated from the following files:

- Rational.h
- Rational.cpp

Generated on Fri Jul 16 12:57:02 2010 for Rational by doxygen 1.6.3

Coding Style

- common code layout style of the source code
  - A good best practice when developing in groups
  - Makes communication easier
  - Reading Code gets faster
  - Training for new developers is easier
- Talk about coding style at the start of a project
  - If you get in a running project, adapt yourself to their coding style

[http://www.equalizergraphics.com/doc_developer.html]

Coding Style Example

Indentation: 4 spaces

const-correctness

No tabs in source files

Opening brace on new line

Space after opening and before closing brace(s)

Break lines to 80 characters

Align continuing argument lists

```
void GLXEventThread::_handleEvent( RefPtr<X11Connection> connection )
{
    Display*    display = connection->getDisplay();
    const Pipe* pipe    = connection->getUserdata();

    while( XPending( display ))
    {
        WindowEvent event;
        XEvent&     xEvent = event.xEvent;
        XNextEvent( display, &xEvent );

        XID            drawable = xEvent.xany.window;
        const uint32_t nWindows = pipe->nWindows();

        event.window   = NULL;
        for( uint32_t i=0; i<nWindows; ++i )
        {
            if( pipe->getWindow(i)->getXDrawable() == drawable )
            {
                event.window = pipe->getWindow(i);
                break;
            }
        }

        switch( xEvent.type )
        {
            case ConfigureNotify:
            {
                // Get window coordinates from X11, the event data is relative
                // to window parent, but we report pvp relative to root window.
                XWindowAttributes windowAttrs;

                XGetWindowAttributes( display, drawable, &windowAttrs );

                XID child;
                XTranslateCoordinates( display, drawable,
                                       RootWindowOfScreen( windowAttrs.screen ),
                                       windowAttrs.x, windowAttrs.y,
                                       &event.resize.x, &event.resize.y,
                                       &child );
```

Static Code Analysis Tools

- Java
  - CheckStyle (http://checkstyle.sourceforge.net/)
- C#
  - StyleCop (http://stylecop.codeplex.com/)
- C++
  - No standard tool for checking code layout style
  - Uncrustify, Astyle, Make pretty,…

Practice 6: Write tests first, run them often

- Many developers think tests should be developed late in the develpment process
- test-driven development (TDD)
  - Write tests first
  - Provide a full covarage of the expected functionality
- Benefits of TDD
  - Test programs debug your design
  - You can measure the progress on passing test cases

Catching Errors with Exceptions

- Using exceptions for error handling
  - Separates normal operation from error handling
  - Makes both easier to read
- Structured like if/else
  - Code for healthy case goes in a try block
  - Error handling code goes in a matching except block
- When something goes wrong in the try block, raise an exception
  - This is caught by the matching except

```
const Number& Number::operator/=( const Number & rhx )
{
  if (rhx == 0) {
    throw Number::DivideByZero();
  }
  int newNumber = m_number / rhs;
  return *this;
}
```

```
void calculate(Number x, Number y)
{
  try {
    Number sum  = x + y;
    Number quot = x / y;
  }
  catch (Number::Overflow& exception) {
    ...code that handles overflow...
  }
  catch (Number::Underflow& exception) {
    ...code that handles underflow...
  }
  catch (Number::DivideByZero& exception) {
    ...code that handles divide-by-zero...
  }
}
```

Exceptional Style

- Always use exceptions to report errors instead of returning None, -1, False, or some other value
  - Allows callers to separate normal code from error handling
  - And sooner or later, your function will probably actually *want* to return that "special" value
- Throw low, catch high
  - I.e., throw lots of very specific exceptions...
  - ...but only catch them where you can actually take corrective action
  - Because every application handles errors differently
    - If someone is using your library in a GUI, you don't want to be printing to stderr

Practice 7: Use mail lists to communicate

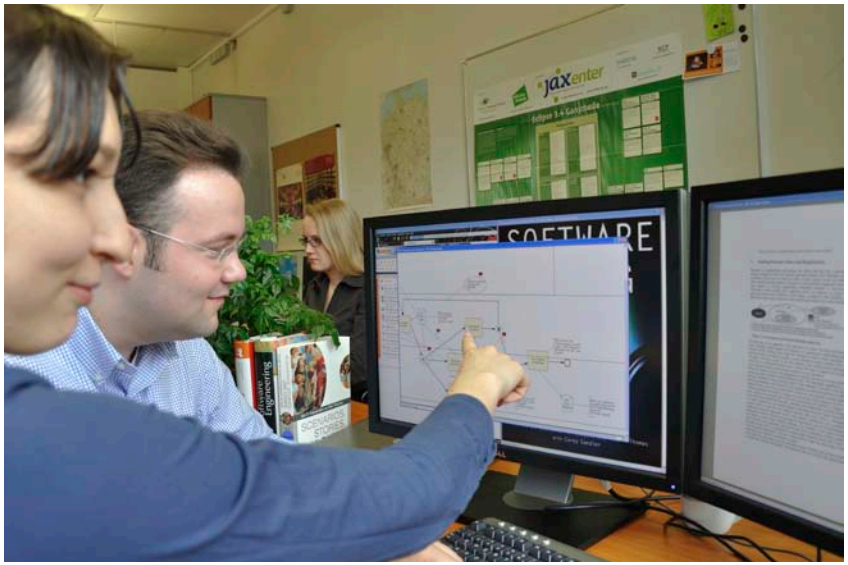
- Why use mailing lists instead of private email accounts?
  - Information is available for everyone
    - Also when someone is sick or in vacation
  - No more CC
  - Changes in responsibilities don‘t lead to a chaos
    - New developers have access to all mails, formar developers don‘t keep getting mail
- Several mailing lists
  - Users
  - Developers
  - Leaders
  - Check-In (automatically generated from commit logs, i.e. Subversion)
  - Announce
- Tool: Mailman
- Also: Wikis

Practice 8: Use checklists for repeated processes

- Checklists are a valuable tool for
  - Making easily repeatable processes
  - For training purposes
  - Documenting workflows that
    - could get lost otherwise
    - Are performed slightly different by different developers
    - Include simple steps that get forgotten
- Examples
  - Relese checklist
  - Version control commit checklist

Practice 9: Program tough stuff together

- Pair programming is a concept formalized by Extreme Programming
- For development of complex software functions, working with a partner side-by-side is very valuable



Practice 10: Perform continual process improvement

Heroux :

"Any software process, no matter how poorly defined, can be written down and improved upon, and any process, no matter how mature, can be made better."

[http://www.softdistrict.com/free-project-management-software/]

Project Management

Project Management

- Has to balance cost, time and quality
- Has to organize the project

Four Essentials of Good Management

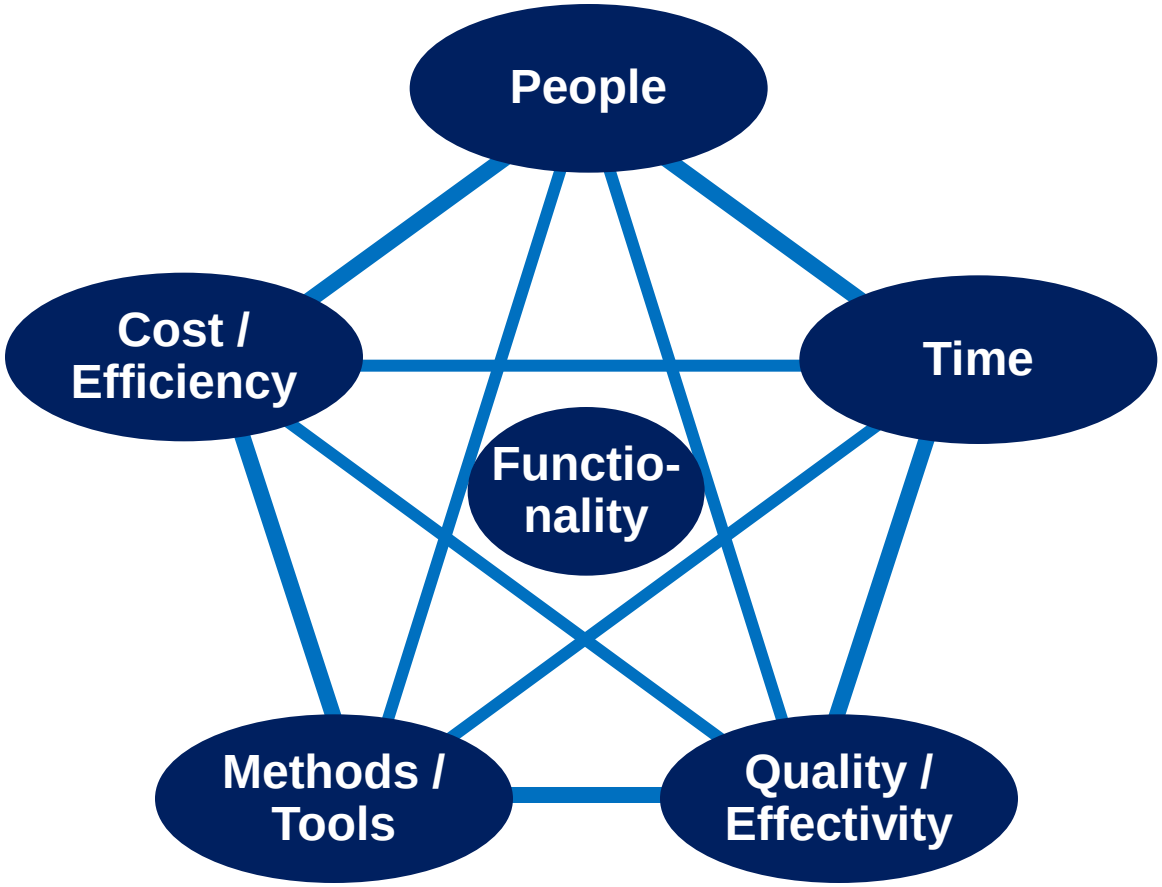
- Get the right people
- Match them to the right jobs
- Keep them motivated
- Help their teams to jell and stay jelled

(all the rest is Administrativa)

Tom DeMarco, The Deadline

Dorset House, 1997

Magic Pentagon



Project phases

•Project-Kick-Off

•Process, Team, Tools

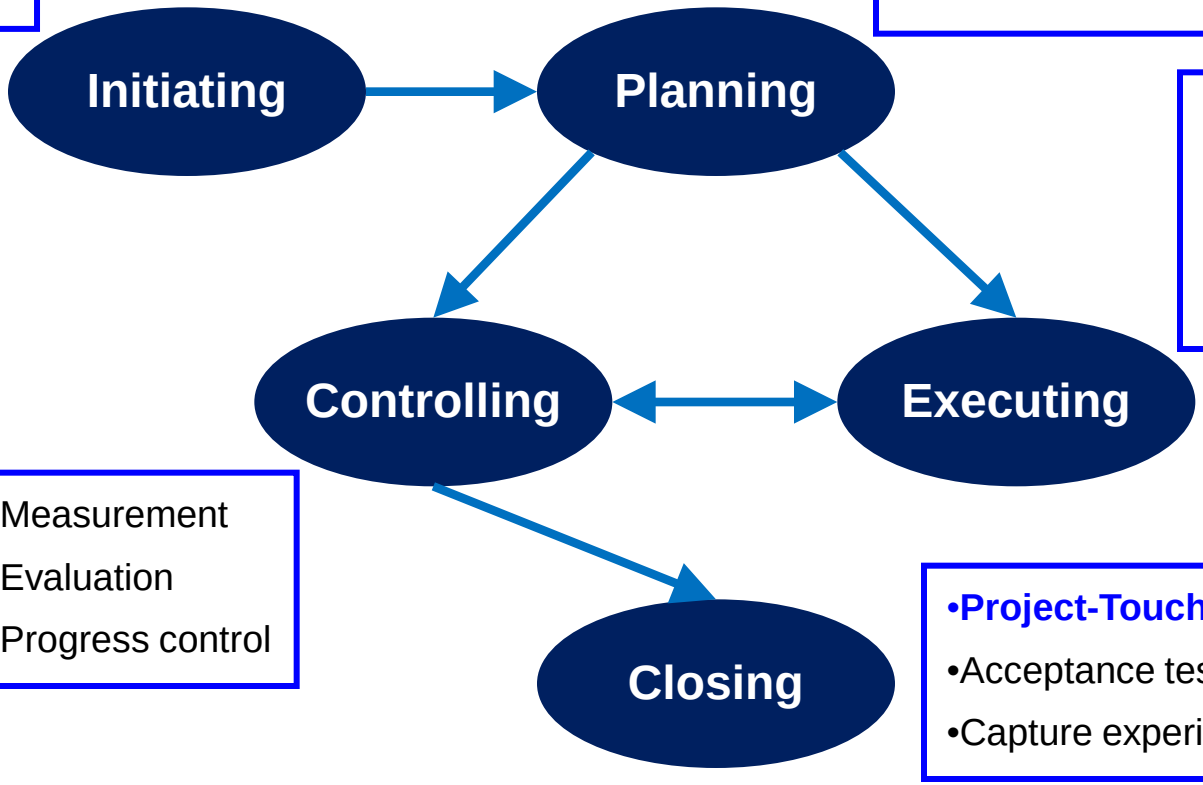
•Project goals

•Phases and milestones

•Work structure

•Effort, estimation

•Time schedule



•Risk management

•Quality assurance

•Team leadership

•Information

•Measurement

•Evaluation

•Progress control

•Project-Touch-Down

•Acceptance test

•Capture experiences

[http://semanticstudios.com/publications/semantics/000228.php]

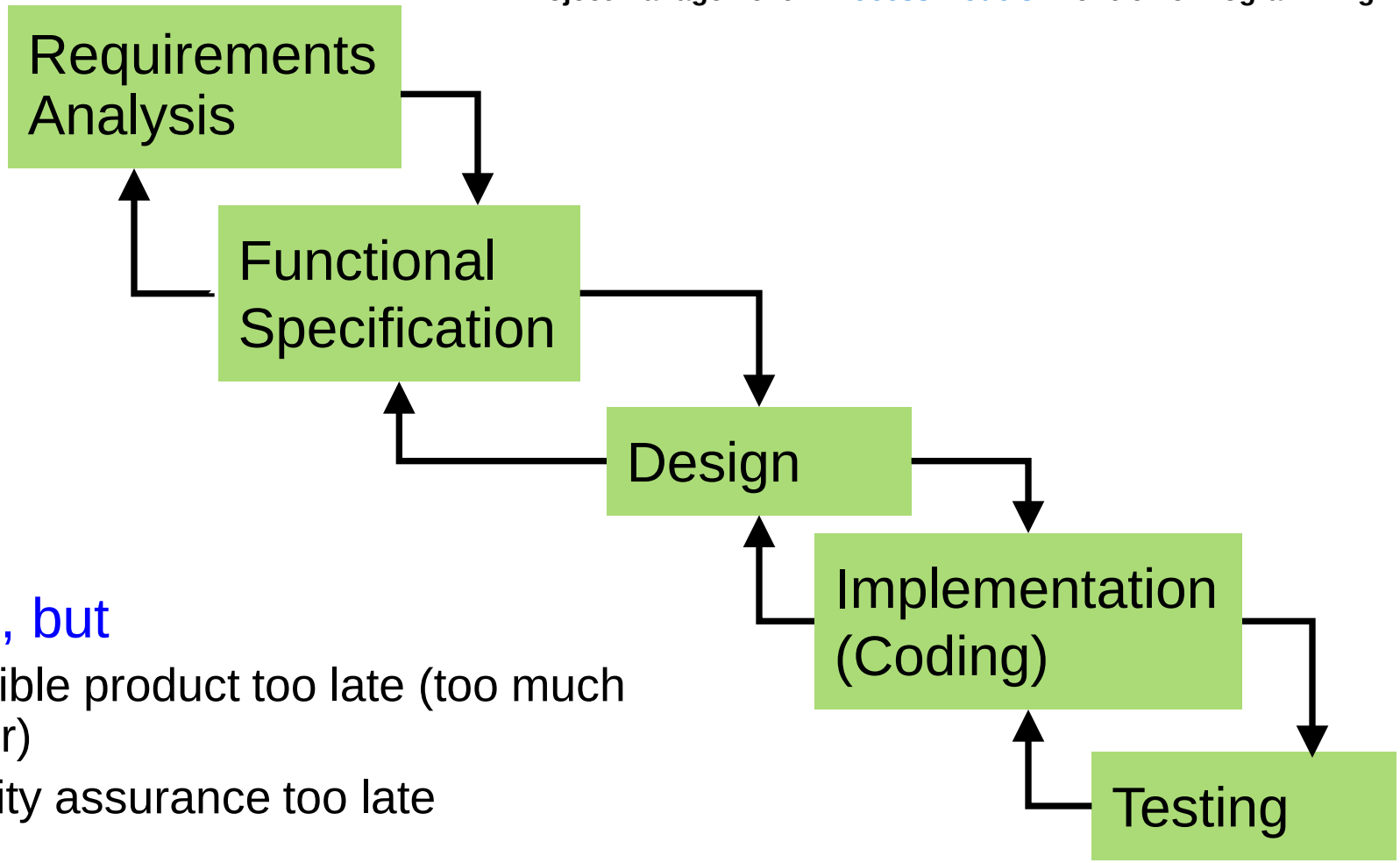
Process models

What to do when?

- Software project organization adapts general software process models to the project context

- There are many possible process models
  - Waterfall model (~1970)
  - V-model (~1980)
  - Rational Unified Process (~1990)
  - Agile methods (~2000)
    - XP
    - Scrum

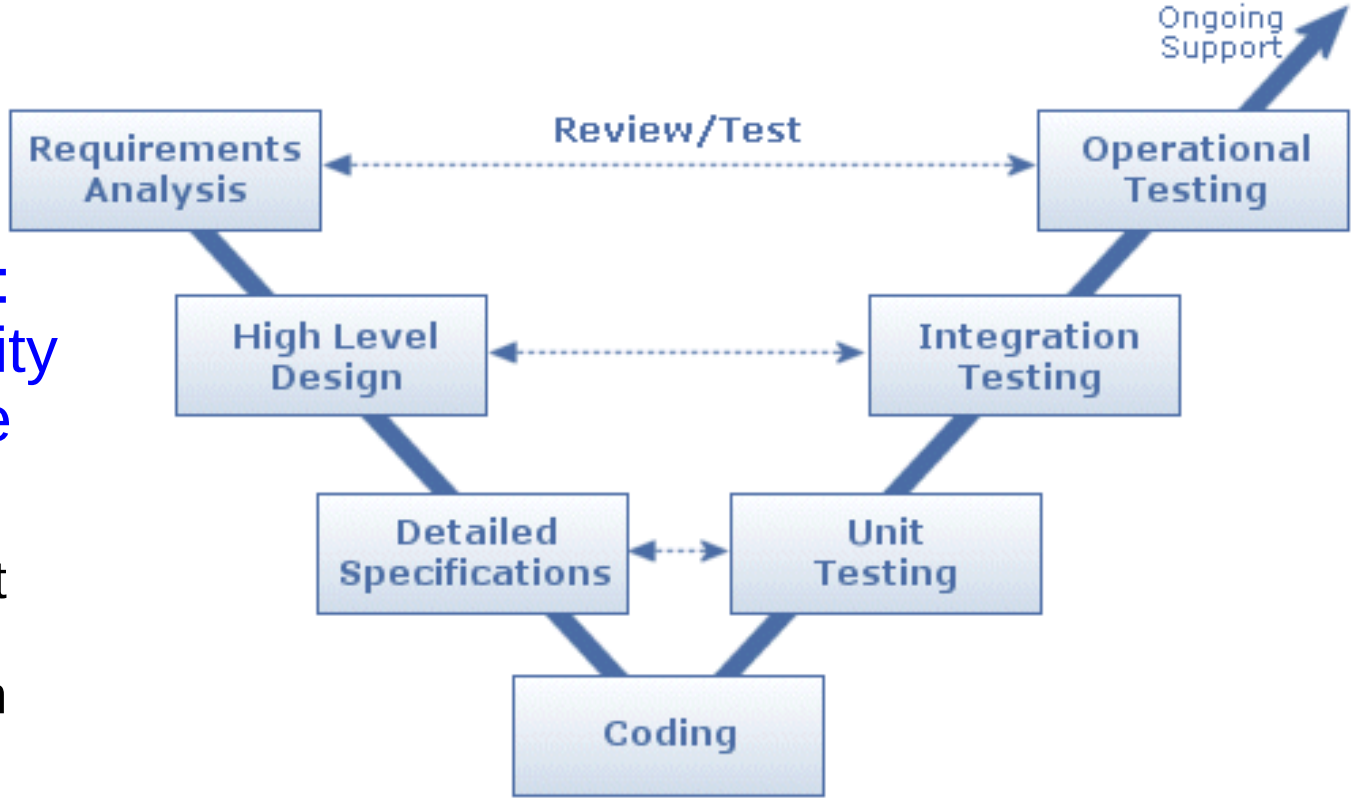
Waterfall model



- Simple, but
  - Tangible product too late (too much paper)
  - Quality assurance too late

[http://www.softtutorial.com/software-testing/v-model.html]

V-Model

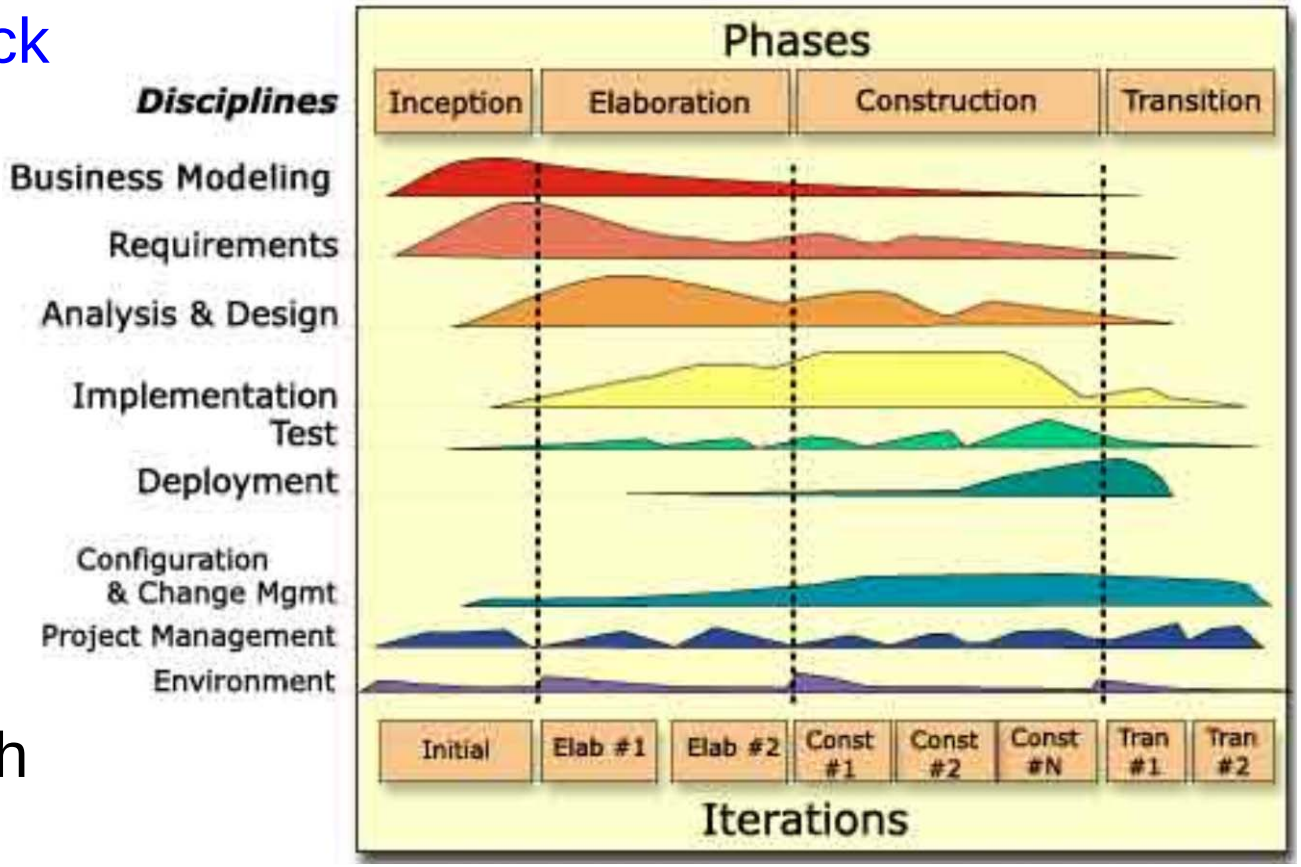


- Core idea: early quality assurance
- but
  - Does not support evolution

[http://www-01.ibm.com/software/awdtools/rup/]

Rational Unified Process

- Early Feedback through Iterations



- But:
  often too much paper work,
  to little flexibility

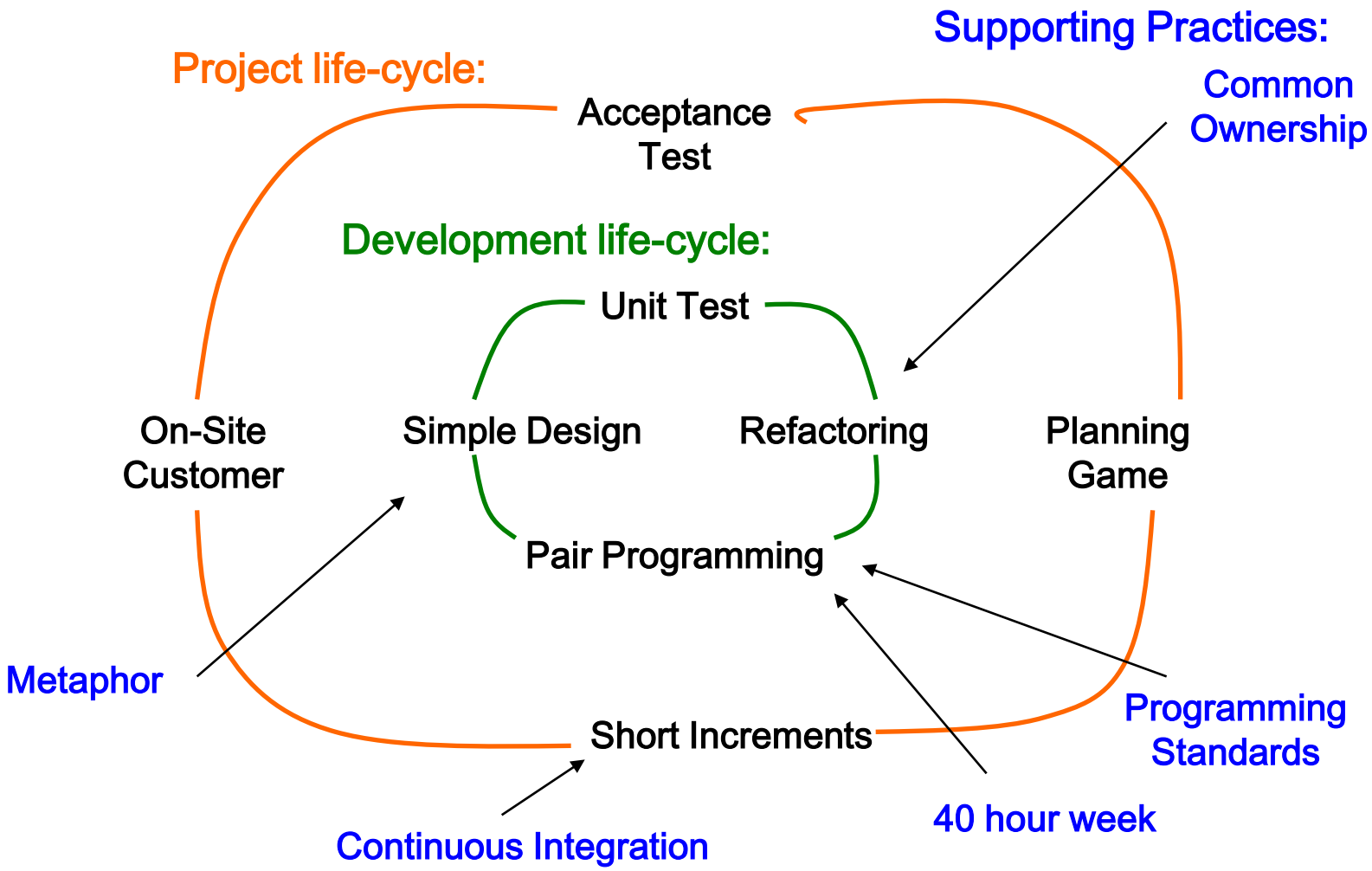
Extreme Programming Principles

Agile Method

Difficult for big projects

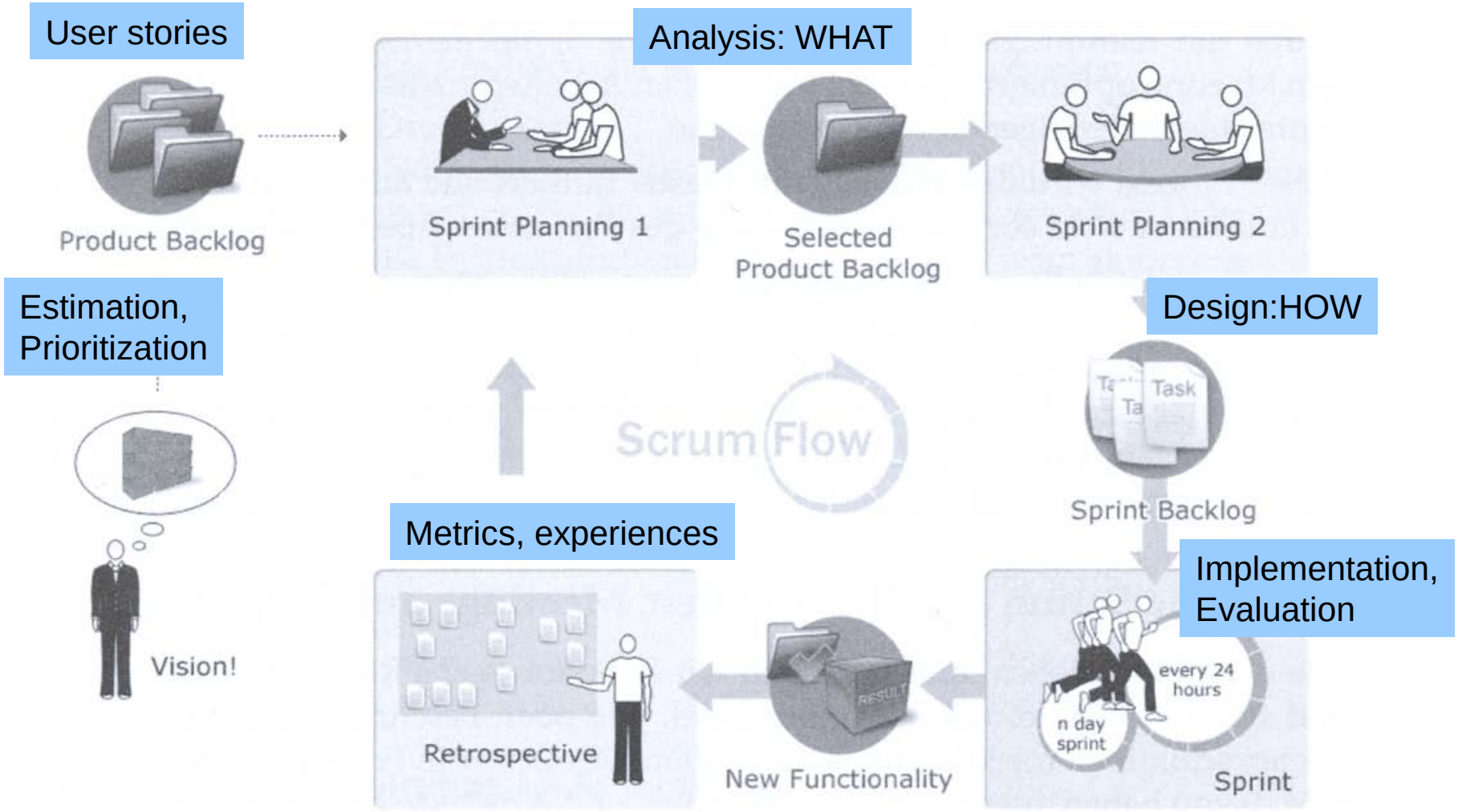
1. Fast Feedback
   - The learning process depends on the time between the activities.
2. Straightforward Thinking
   - Simple solutions are often sufficient
3. Incremental changes
   - Do not change everything at once, but instead in small steps
4. Embrace change
   - Do not fear changes – it induces more costs to postpone changes
5. Quality focus
   - Quality supports the flexibility to react to changes

XP Practices



[Gloger]

Scrum



[Gloger]

Scrum ideas

- Change Management, Focus on Team
- Main idea:
  - Develop software in sprints
  - Daily meetings: Daily Scrum
  - Team is responsible for planning and results
- Roles:
  - Product Owner (from the customer organization)
    - Vision, Prioritization
  - Team
  - ScrumMaster (not Project manager!)
    - Supports Team
    - Moderates between Product-Owner and Team

[http://software-carpentry.org/]

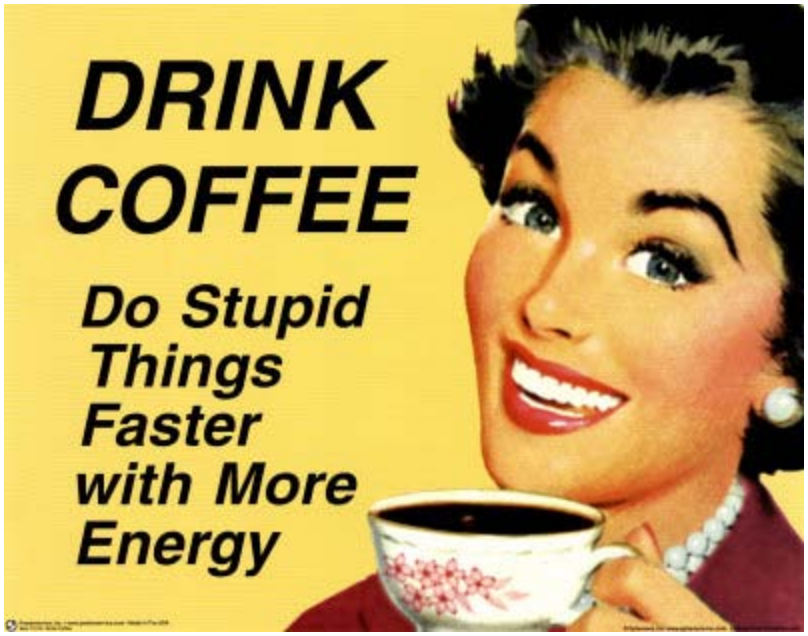
Scrum Key Practices

- Sprint planning meeting held at the beginning of each iteration
  - Analyze and prioritize current product backlog
  - Select overall goal for sprint , Decide how to achieve the goal (design)
  - Create a sprint backlog from the product backlog
  - Estimate backlog in hours
    - Nothing should be longer than a couple of working days
    - Anything that is should be broken into smaller testable/deliverable chunks
- Daily scrum meeting
  - Every morning, 15 minutes long, standing up (to make sure it stays 15 minutes long)
  - Everyone says:
    - What they did yesterday , What they are going to do today , What stands in their way
  - *Not* a status update, but rather making commitments to colleagues
- Sprint review held at the end of the iteration
  - Team presents what it accomplished
    - Demo, not slides , And yes, everything *can* be demo'd
- Sprint retrospective also held at the end of the iteration
  - What do we want to start doing?
  - What do we want to stop doing?
  - What do we want to keep doing?

- Which process model do you use?
- Which process model could you use?

Extreme Hour

Extreme Hour Vision: A better coffee machine



- Accept this new vision. Plan, Schedule, Develop and Quality Assure our initial release.
- Project timeframe: 1 Hour

60 Minutes Project

- **10 Minutes** User Stories & Spike Architecture
- **10 Minutes** Priority & Scope and 1st Commitment Schedule
- **10 Minutes** Iteration 1
- **10 Minutes** 2nd Commitment Schedule
- **10 Minutes** Iteration 2
- **10 Minutes Release!**

Project Roles

- Customers specify and estimate
- Quality assurance (QA) runs acceptance tests
- Developers estimate and implement

- Rules:
  - If It Ain't Drawn, It Ain't Delivered.
  - If It Ain't Written, It Ain't Required.
  - QA can't see what developers do till iteration's end
    - NB: In real XP, QA communicates with Developers & Customers.

10 Minutes: User Stories, Test Fixtures, Spikes

- Customers write Stories.
  - E.g: I want to choose between 3 different kinds of coffee
- QA details quality requirements per story
  - E.g.: It shall be possible to get coffee in 3 seconds
- Developers define Architecture.

10 Minutes: 1st Commitment Schedule

- Customers sort Stories into 3 piles:
  - Must Have
  - Market Advantage
  - Really Cool
- Then rank relative priorities within each pile.
- Then schedule stories for 2 Iterations.
  - Use “Load Factor 2” for Project Velocity

- Developers assign Ideal Minute costs to Stories based on Spike (and quality requirements).
- Max story size 3 ideal minutes, or else split/clarify.
- If developer estimates disagree, optimist wins.

- QA specifies Acceptance Tests for all stories

10 Minutes: Iteration 1

- QA can't see what developers draw until end of Iteration.
- QA finishes Acceptance Tests
- Customers modify (also add) and reprioritize Iteration 2 Stories.

- Developers pair. Each pair picks 1 User Story & 1 pen.
- First draw simplest thing that could possibly work.
- Then Refactor drawing to make simplest system.

10 Minutes: 2nd Commitment Schedule

- Customers reveal new Stories & Developers estimate them.
- QA “run” tests and note bugs as stories

- Customers prioritize bug vs. new stories
- Then schedule Second Iteration
  - Use Measured Project Velocity

10 Minutes: Iteration 2

- QA writes down Acceptance Tests for each Story.
- QA can't see what developers draw until end of Iteration.
- Customers modify and reprioritize Iteration 3 Stories.
- Developers pair. Each pair picks 1 User Story & 1 pen.
- First draw simplest thing that could possibly work.
- Then Refactor drawing to make simplest system.

10 Minutes: Release!

- QA “run” tests and note bugs as stories

- Joint Decision whether release is possible

Retrospective

- How did you like it?
- What did you learn about the 3 roles?
- What did you learn about software project organization?

References

- Heroux, M.A.W., James M., *Barely sufficient software engineering: 10 practices to improve your CSE software*, in *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*. 2009, IEEE Computer Society.
- http://www.suodenjoki.dk/us/archive/2010/cpp-checkstyle.htm
- http://www.dune-project.org/doc/devel/codingstyle.html
- http://www.flipcode.com/archives/C_Coding_Style-My_Code_Style.shtml
- Dr. Frank Houdek, Michael Stupperich, Vorlesung „Management von Softwareprojekten“
- http://www.infoq.com/articles/agile-version-control
- http://www.equalizergraphics.com

Hanna Remmel
Institute of Computer Science
Chair of Software Engineering
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
http://se.ifi.uni-heidelberg.de
valtokari@informatik.uni-heidelberg.de

RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG