Good practices in Software development

Jan Engels

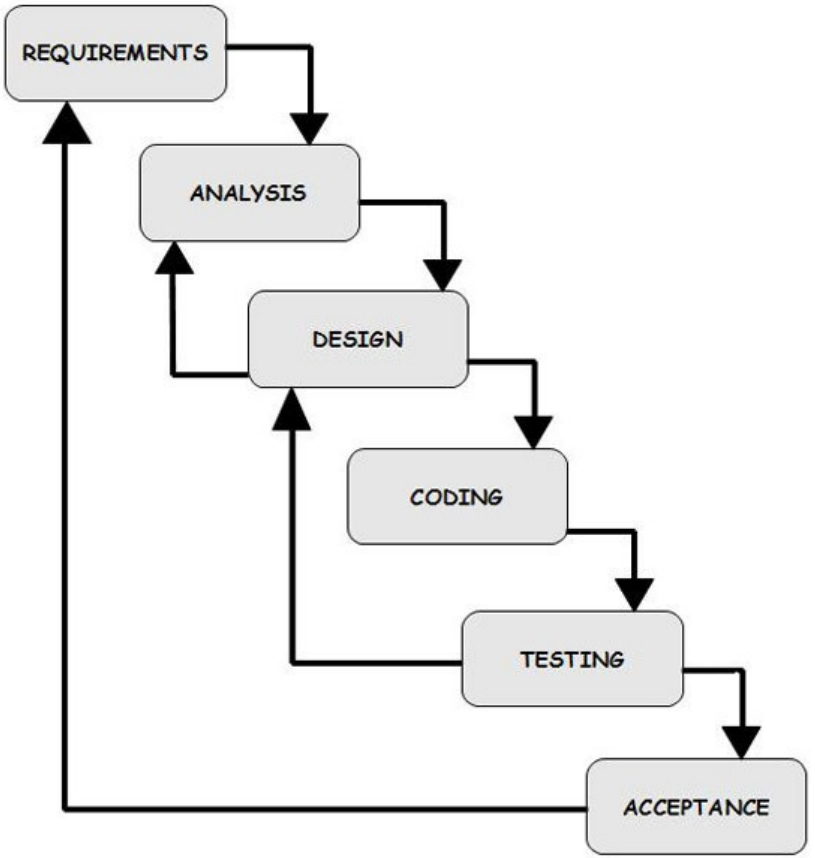
Advanced Programming Concepts Workshop 2012

Desy, 2012-10-11

A few words about this talk

- Software development is sometimes seen as an “Art”
  - Any artist needs to learn some basic techniques before starting to paint
- There is no such thing as “perfect coding styles”
  - All programmers have their own personal preferences and we are all just human beings!
- But...
  - All of us can try to follow very simple rules in order to write better software
- This talk...
  - Will highlight some of the most important “basic rules”
  - Will focus on rules which are programming language independent
  - Hopefully will convince many of us to stick to some of the rules :)

Outline



Requirements, Analysis & Design

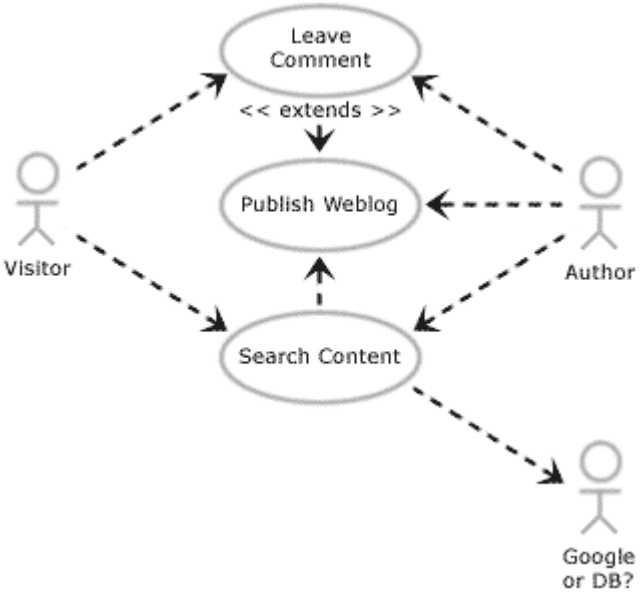
- **Requirements, Analysis & Design**
  - Requirements
  - Analysis
  - Design

Requirements

- Requirements
  - What is the software supposed to do?
  - Who will be the end users?
  - How much development time to be invested?
  - How much manpower?
  - Performance
  - Compatibility
  - Scalability
  - Storage
  - Security
  - Maintenance

Analysis

- Analysis
  - What environment will the software run on?
  - How can the requirements be fulfilled
  - Description of tasks and workflows
  - Design of use cases



Design

- Design
  - Develop a concrete plan to solve the problems defined in the Analysis and Requirements phase
  - Evaluate design patterns
  - Use of modeling languages
  - Technologies, standards, services ...
  - What programming language(s)/tools to choose?
    - Development time Vs. Application performance
  - Security
    - Not something that can be added later on!
  - Dependencies
    - Can I (or do I need to) use existing libraries?
    - Evaluate existing solutions

Implementation

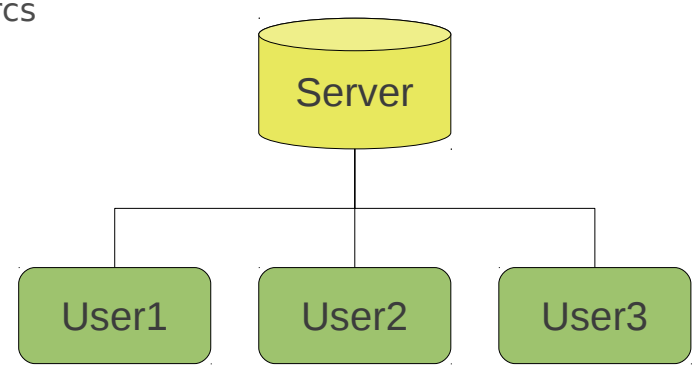
- **Implementation**
  - Source control management tools
  - Libraries
  - Logging
  - Configurability
  - Tips and good programming practices

Implementation

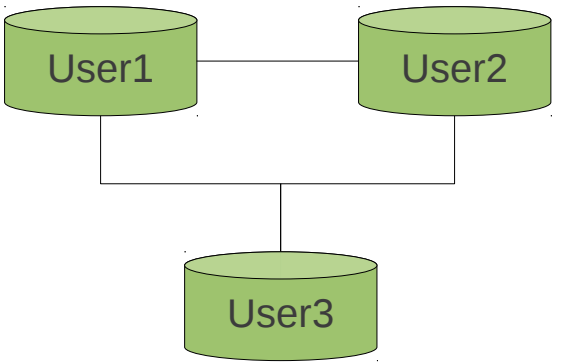
- Source control management tools (SCM)
  - CVS, SVN, Git, Mercurial, Bazaar...
  - Start using SCM as soon as possible in your project!
  - Code is automatically backed up
  - Share code with other people
    - Other people might help fixing bugs or even adding features
    - Only commit/push changes if code is tested and working!
  - Go back to a previous state in time
    - Freedom to experiment without fear of breaking things
  - Branching, Tagging, Patching
  - Code Sign Off
  - Crucial for defining workflows in software development
    - Production vs. Development
    - Releasing

Implementation

- Source control management tools (SCM)
  - Centralized vs Distributed version control systems
    - Centralized systems - Global server where everyone commits their changes into
      - CVS
      - SVN
    - Distributed - Each local working repository is a server as well
      - Git
      - Mercurial (hg)
      - Bazaar (bzr)
      - Darcs



Centralized systems



Distributed systems

Implementation

- Source control management tools (SCM)

  - SVN Introduction (server side)

    - Create a new repository (tested on ubuntu 12.04)
      - sudo -i
      - apt-get install subversion
      - svnadmin create /svn
      - sed -i 's/# password-db/password-db/' /svn/conf/svnserve.conf
      - echo 'calvin = hobbes' >> /svn/conf/passwd # add user calvin with password: hobbes
      - chmod 600 /svn/conf/*

    - Add a new project (comics)
      - mkdir -p /tmp/comics/{trunk,tags,branches}
      - svn import /tmp/comics file:///svn/comics -m"added initial version of comics"

    - Start server
      - svnserve -d

Implementation

- Source control management tools (SCM)

  - SVN Introduction (client side)
    - Checkout copy of the project
      - svn co svn://localhost/svn/comics/trunk comics
      - cd comics
    - Show some infos
      - svn info
    - Check status of local working copy
      - svn status # no changes were done so far
    - Show differences between local working copy and online repository
      - svn diff # no changes were done so far

Implementation

- Source control management tools (SCM)

  - SVN Introduction (client side)
    - Add a new file
      - echo spiderman > newcomics
      - svn add newcomics
    - Check project status
      - svn status # now shows a new file
    - Show differences in working directory
      - svn diff # now shows the contents of the new file
    - Commit changes to the online repository
      - svn commit -m 'added new spiderman comic'
    - References:
      - http://svnbook.red-bean.com/

Implementation

- Source control management tools (SCM)

  - Mercurial Introduction

    - Create a new repository (tested on ubuntu 12.04)
      - hg init comics
      - cd comics
    - Setup my username (calvin)
      - echo -e "[ui]\nusername = calvin hobbes <calvin@hobbes.com>" >> ~/.hgrc
    - Add a new file
      - echo spiderman > newcomics
      - hg add
    - Commit changes
      - hg commit -m 'added new spiderman comic'
    - Start server
      - hg serve -n "my comics"

Implementation

- Source control management tools (SCM)

  - Mercurial Introduction

    - Checkout copy (clone) of the project
      - hg clone http://localhost:8000/ mycomics
      - cd mycomics
    - Check project status
      - hg status # no changes were done so far
    - Show differences in working directory
      - hg diff # no changes were done so far
    - Change file and push changes into original repository
      - echo 'x-men' >> newcomics
      - hg status # now shows modified file
      - hg diff # now shows changes in modified file
      - hg commit -m 'added new x-men comic'
      - hg push /path/to/comics # push the changes into the original comics repository
      - hg push http://localhost:8000/ # alternatively one can use the url if the server is still running

Implementation

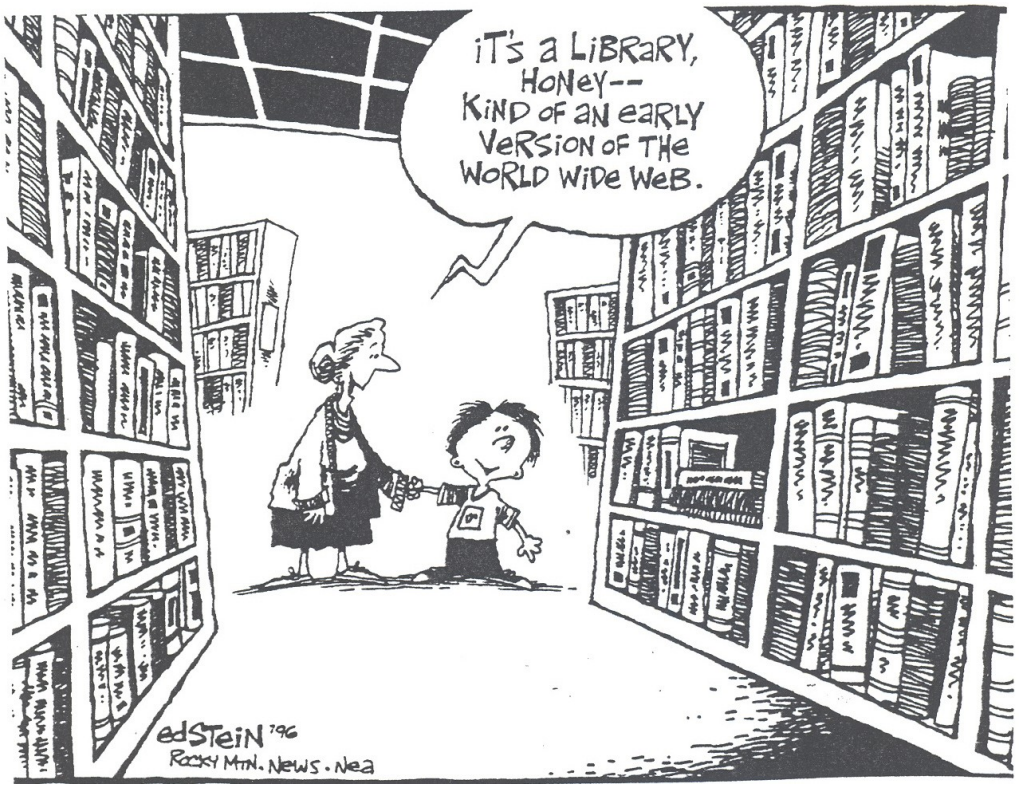
- Source control management tools (SCM)

  - Mercurial Introduction

    - Push vs Pull
      - In previous slide we used the push command to “push” changes into a different repository
      - It is also possible to “pull” changes from another repository, e.g.
        - cd mycomics
        - hg pull /path/to/comics # pull the newest changes from the original comics repository
        - hg update # apply the newest changes into your working directory
    - Don't forget: in distributed systems there are no “master servers”
      - All clones are “master servers” themselves
      - When you **clone** a repository you get an **exact copy** of the original repository!
    - References:
      - http://mercurial.selenic.com/

Implementation

- Libraries
  - Why libraries?
  - Share code/functionality in and/or between applications
  - Help prevent the “spaghetti-code” phenomena



Implementation

- Libraries
  - **Difference between private and public!**
  - Every method exposed in a public API involves documentation and can be responsible for breaking backwards compatibility of a library!
    - Example1:
      - python 2: 5/2 = 2
      - python 3: 5/2 = 2.5
    - Example2:
      - myprog v1.0: addUser(name, surname, age)
      - myprog v1.1: addUser(name, surname, age, job)
      - myprog v2.0: addUser(name, surname, birthdate, job) # backwards compatibility broken!
  - Versioning
    - Increase major version when API changes and backwards compatibility is broken
    - Increase minor version when changes are made but API is still backwards compatible
    - Increase patch version when only bugfixes/patches are made
  - Building a good library increases the overall development time but code becomes usually well documented and tested

Implementation

- Logging
  - Start using a logging library from the very beginning in your project
    - Saves you time in the long term..
  - Some programming languages have a logging library “built-in”
  - Easily add an option to run applications “quietly” or in debug mode
  - Using a logging library makes debugging applications easier
    - Splitting different logging levels into different files
    - Configurable logging for different libraries/classes
  - Logging across the network
  - Log file rotation
    - One of sysadmin's favorite problems are disks getting full due to log files!
    - Either provided by logging library or linux standard logging facility
  - Linux standard logging facility: syslog, logger, logrotate

Implementation

- Logging
  - Example using python's logging module:

```
#!/usr/bin/python

import sys, logging

# create a logger
mylog = logging.getLogger('mylog')

# set the logging level for 'mylog'
mylog.setLevel(logging.DEBUG)

# define a handler for writting messages to console
ch = logging.StreamHandler(sys.stdout)

# set the logging level for the console handler
ch.setLevel(logging.INFO)

# bind the console handler to 'mylog'
mylog.addHandler(ch)

mylog.debug('debug message')
mylog.info('info message')
mylog.warning('warning message')
mylog.error('error message')
mylog.critical('SYSTEM FAILURE!!!')
```

Output:
info message
warning message
error message
SYSTEM FAILURE!!!

Implementation

- Configurability
  - Command line options
    - Make your application more portable and easier to maintain
    - There are many standards and libraries out there: e.g. getopt
  - Configuration files
    - Useful for storing profiles or different settings of configurations
    - Some languages include standard libraries for this purpose
  - Environment variables
    - Useful for sharing configuration settings across applications
    - Use only for settings which must be common at any time between all applications
  - Dependencies between configuration settings
  - Use of a database?
    - Consider using object-relational mapping:
    - http://en.wikipedia.org/wiki/List_of_object-relational_mapping_software

Implementation

- Configurability
  - Example using python's ConfigParser module:

example.cfg

```
[DEFAULT]
server = myserver.foo.com
port = 12345


[DE]
server = myserver.foo.de
port = 54321


[EN]
server = myserver.foo.en
```

example.py:

```
#!/usr/bin/python

import ConfigParser

config = ConfigParser.ConfigParser()
config.read('example.cfg')

print config.get( 'DE', 'server' )
print config.get( 'DE', 'port' )

print config.get( 'EN', 'server' )
print config.get( 'EN', 'port' )
```

Output:
myserver.foo.de
54321
myserver.foo.en
12345

Implementation

- Configurability
  - Example using python's OptParse module:

```
#! /usr/bin/python

from optparse import OptionParser
parser = OptionParser( usage='%prog [options] ARG1 ARG2', version="%prog 1.0" )
parser.add_option('--log-level', help='set the level of verbosity [%default]', default='INFO')
parser.add_option('-t', '--timeout', help='set the timeout value [%default]', type='int', default=300)
parser.add_option('-v', '--verbose', '--debug', action='store_true', dest='verbose', help='run in debug mode')

(options, args) = parser.parse_args()

if len(args) < 2:
    parser.error('incorrect number of arguments (-h for help)')


print 'options:'
print options.timeout
print options.verbose
print options.log_level
```

Run:
./optparse-example.py

Output:
Usage: optparse-example.py [options] ARG1 ARG2

optparse-example.py: error: incorrect number of arguments (-h for help)

Run:
./optparse-example.py -h

Output:
Usage: optparse-example.py [options] ARG1 ARG2

Options:
  --version show program's version number and exit
  -h, --help show this help message and exit
  --log-level=LOG_LEVEL
  set the level of verbosity [INFO]
  -t TIMEOUT, --timeout=TIMEOUT
  set the timeout value [300]
  -v, --verbose, --debug
  run in debug mode

Implementation

- Tips and good programming practices
  - Lazy programmers are good programmers ;)
    - DRY principle: Only change things in one single place in code
    - In other words: Don't duplicate code!
  - Readability counts!
    - pol=$(echo "scale=3 ;$([[$pol =~ L$|R$]] && pol=${pol}100 ; echo $pol | tr "LR" "- ") / 100.0" | bc)
    - Numerous conventions exist for different programming languages
    - What do you think is easier to read?
    - `NumberValves = NumberValvesPerCylinder * NumberCylinders`
    - `nv=nvpc*nc`
  - Comments
    - Imagine looking at your code in 2 years from now on :)
    - Be able to hand over your code to someone else

Implementation

- Tips and good programming practices
  - Recursion
    - Be very careful with recursion!!
    - Performance...
    - Memory consumption...
    - No control over the calling sequence (harder do debug..)

Fibonacci (python)	Recursive	Iterative
N = 35	10 sec	0.05 sec
N = 40	1 min 30 sec	0.05 sec
N = 45	20 min	0.05 sec
N = 100.000	ZzzZzZz...	0.5 sec

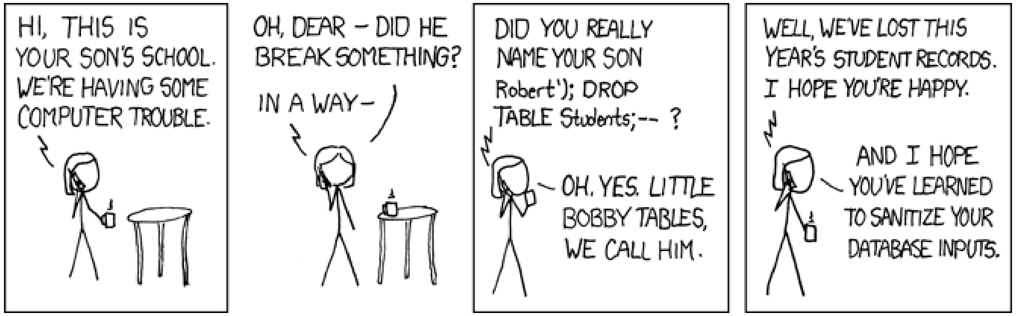
Implementation

- Tips and good programming practices
  - Learn to program defensively!
    - Always check your function/method arguments
    - Whenever possible check min/max for any numeric inputs
    - Don't expose more than required in an API
    - C++ only:
      - Use compiler flags: `-Wall -ansi -pedantic`
      - For shared libraries use linker flag: `-Wl,--no-undefined`
      - Remove all warnings in code
      - DANGER: Unitialized pointers: after deleting set to NULL!
        - `delete p;`
        - `p = NULL;`

Implementation

- Tips and good programming practices
  - Never trust user input under any circumstances.
  - Never trust user input under any circumstances.
  - Never trust user input under any circumstances.

  - On client-server applications, make sure to always check user input on server side
    - And on client side as well! (whenever possible)



http://bobby-tables.com

Testing

- **Testing**
  - Different types of testing
  - Automated testing

Testing

- Different types of testing
  - Unit tests
    - Very useful to test fundamental building blocks in your application
    - Many standard libraries available for this kind of testing
  - Smoke tests
    - Does software compile?
    - Memory coverage
    - Does some test/example run without crashing?
  - White/Black-Box tests
    - White-Box tests aim at stressing potential failure points in code
    - Black-Box tests ensure the API works as defined

Testing

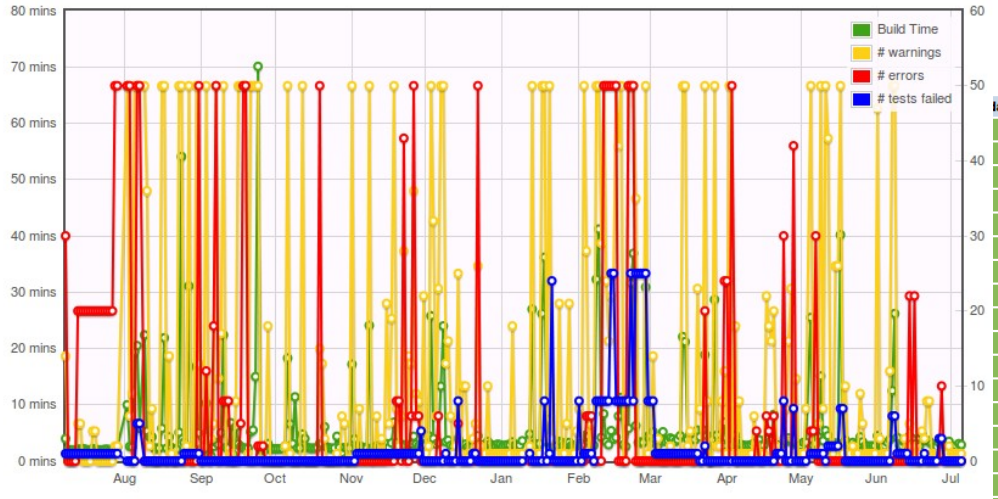
- Different types of testing
  - Functional tests
    - Concept similar to unit tests
    - Tests functionality
  - Regression/integrity tests
    - Ensure test results do not change over time or platform
    - Good for testing overall interaction of components in your project
    - Sometimes it's harder to find exactly what went wrong in this kind of tests
  - Scalability tests
    - Useful if you expect your application to deal with very large quantities
    - Often hard to realise

Testing

- Automated testing
  - Nightly/Commit-tests / Nightly/Commit-builds
    - Crucial for spotting errors as soon as possible
    - Reduces debugging time dramatically
  - Continuous integration systems
    - hudson: http://hudson-ci.org/
    - jenkins: http://jenkins-ci.org/
    - ctest+cdash: http://www.cmake.org/
  - **If possible, run tests on many different platforms!**

Testing

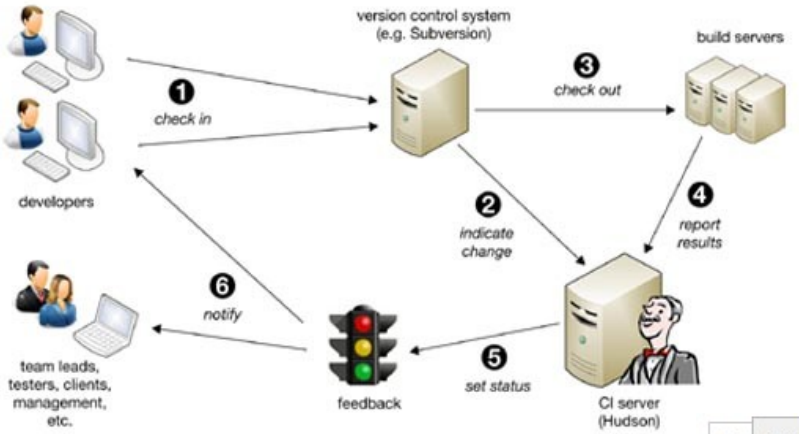
- Automated testing
  - CDash



		late Warnings	Configure Errors	Configure Warnings	Build Errors	Build Warnings	Tests Failed
2012-06-19 02:10:19	0	0	0	0	0	1	0
2012-06-18 02:10:18	0	0	0	0	0	1	0
2012-06-17 02:10:17	0	0	0	0	0	1	0
2012-06-16 02:10:16	0	0	0	0	0	1	0
2012-06-15 02:10:15	0	0	0	0	0	1	0
2012-06-14 02:10:14	0	0	0	0	0	2	0
2012-06-13 02:10:13	0	0	0	0	0	1	0
2012-06-12 02:10:12	0	0	0	0	0	1	0
2012-06-11 02:10:11	0	0	0	0	10	1	3
2012-06-10 02:10:10	0	0	0	0	0	1	3
2012-06-09 02:10:09	0	0	0	0	0	1	0
2012-06-08 02:10:08	0	0	0	0	0	1	0
2012-06-07 02:10:07	1	0	0	0	0	1	0
2012-06-06 02:10:06	0	0	0	0	0	1	0
2012-06-05 04:24:05	0	0	0	0	0	8	0
		0	0	0	0	8	0
		0	0	0	0	3	1
		0	0	0	0	4	0
		0	0	0	0	1	0
		0	0	0	0	1	0
		0	0	0	22	1	0
		0	0	0	0	2	0
		0	0	0	22	5	0
		0	0	0	0	1	1
		0	0	0	0	3	1
		0	0	0	0	1	1
		0	0	0	0	1	1
		0	0	0	0	1	1
		0	0	0	0	50	6
		0	0	0	0	50	6
		0	0	0	0	12	0
		0	0	0	0	1	0

Testing

- Automated testing
  - Continuous integration systems



http://www.methodsandtools.com/tools/tools.php?hudson

Hudson

All | **Grails Plugins** | +

S	W	Job ↓	Last Success	Last Failure	Last Duration	
		grails-joda-time	55 min (#10)	7 hr 44 min (#8)	36 sec	
		grails-selenium-rc	11 hr (#23)	23 hr (#18)	3 min 54 sec	
		grails-session-temp-files	11 hr (#6)	23 hr (#1)	23 sec	
		grails-springcache	11 hr (#36)	1 day 0 hr (#31)	3 min 18 sec	
		grails-tellurium	N/A	11 hr (#8)	2 min 16 sec	

source: http://adhockery.blogspot.de/2010/03/grails-plugins-on-hudson.html

Maintenance

- **Maintenance**
  - Documentation

Maintenance

- Documentation
  - Use auto-generating doctools in your project, such as doxygen, javadoc...
  - Try to find someone else to read your documentation
  - Importance of good documentation is usually underestimated
  - Documentation generally increases maintenance but also reduces the overall support costs

Summary

- Try to do some analysis and design before starting to code
- Evaluate what libraries/tools might be helpful to use in your project
- Use a version control system for backing up your code
- Use a logging library
- Always keep configurability in mind
  - Split configuration and settings from source code
  - Use standard command line argument parsing tools
- Program defensively
  - Never trust user input under any circumstances!
  - Don't expose more functionality than required in public API's
- Testing is as important as writing code!
- Don't forget the documentation
- Try to keep it simple!

Thank you for your attention!