Good practices in Software development

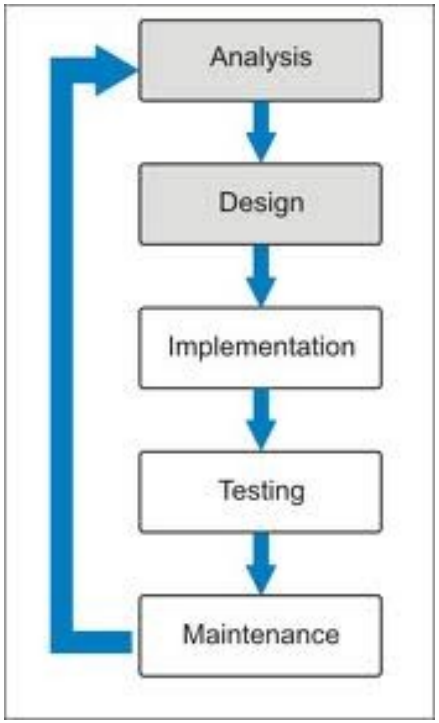
Jan Engels

9th. July 2012

A few words about this talk

- Software development is sometimes considered 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 <u>some</u> 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

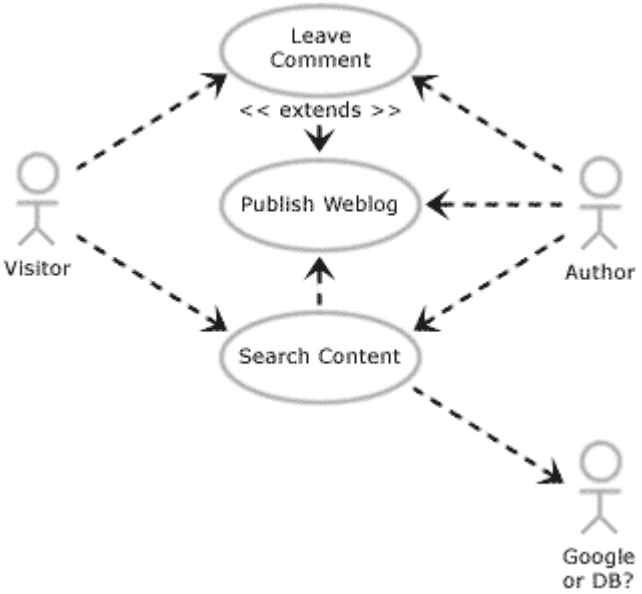


Analysis & Design

- **Analysis & Design**
  - Analysis
  - Use cases
  - Requirements
  - Specification

Analysis

- Analysis
  - What is the software supposed to do?
  - Who will be the end users?
  - How much time to invest in development?
  - Is security relevant?
  - Scalability
  - Compatibility
  - Performance
  - What will be the maintenance costs?
  - ...



- Defining use cases
  - Types of users
  - Description of tasks and workflows

Analysis

- Requirements
  - What must the software be able to do?
  - What environment will the software need to run on?
  - Performance requirements
  - Storage requirements
  - Security requirements
  - Scalability requirements
  - Compatibility
- Specification
  - How should the requirements be fulfilled
  - Technologies, standards, services ...

Design

- Design
  - Develop a concrete plan to solve the problem defined in the Analysis phase
  - Design patterns, e.g. Model View Controler Pattern (MVC)
  - Use of modeling languages
  - 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
  - Coding guidelines
  - Tips and good programming practices

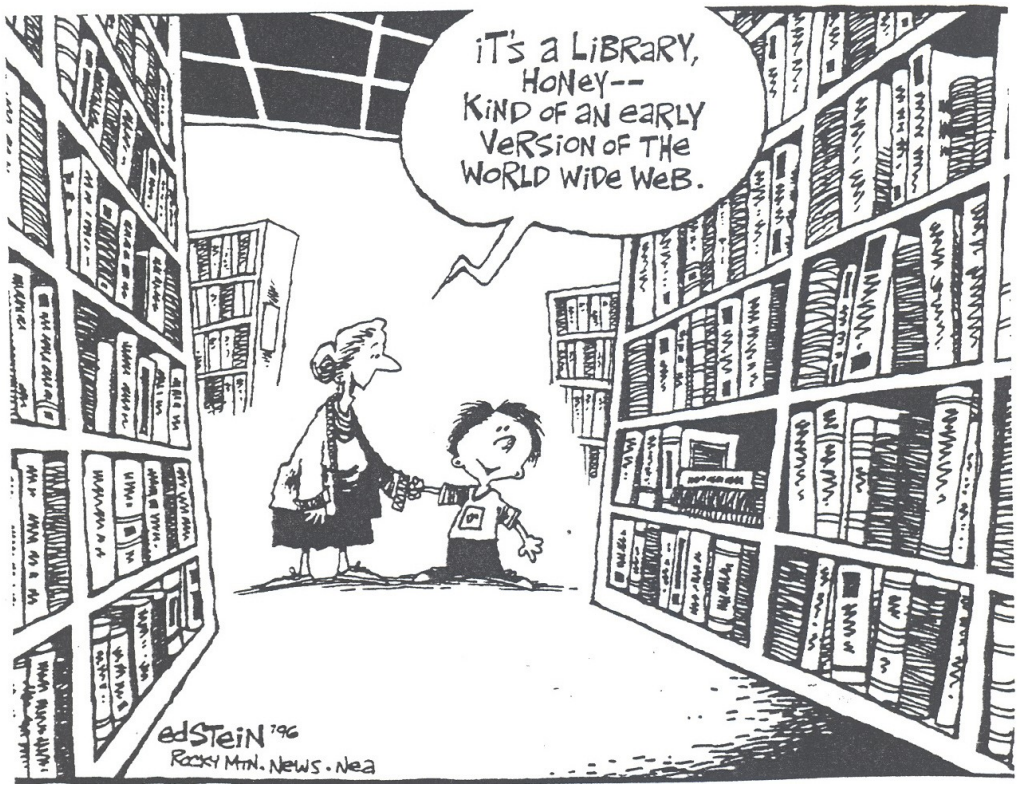
Implementation

- Source code management tools (SCM)
  - CVS, SVN, Git, Mercurial...
    - https://svnsrv.desy.de
  - 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
  - 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
    - production vs. development branches
    - Releasing

  - Request tracker
    - https://rt-system.desy.de

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 a bit maintenance
  - Building a good library generally increases the overall development time but code becomes usually well documented and tested
  - 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

Implementation

- Libraries (C++ only)
  - Static libraries
    - Includes all dependencies (good for shipping pre-compiled binaries)
    - Slightly faster when loading binary into memory
  - Shared libraries
    - As name suggests the library is shared (in disk and in memory)
    - Applications do not need to be recompiled for using a newer library
      - unless major version changes
    - Can be loaded dynamically at run-time (plugins)
    - Smaller binaries
    - RPATH vs RUNPATH vs LD_LIBRARY_PATH and LD_PRELOAD
      - LD_LIBRARY_PATH: environment variable to specify additional search paths for libraries
      - LD_PRELOAD: preload a library before executing any application (DANGER!)
      - RPATH: hardcoded path NOT overwritten by LD_LIBRARY_PATH
      - RUNPATH: same as RPATH but overwritten by LD_LIBRARY_PATH (--enable-new-dtags)
      - readelf -d <bin> # display hard-coded rpaths in libraries/binaries
    - Difference between putting code in source or in header files

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

- 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 a database?
    - Object-relational mapping

Implementation

- Tips and good programming practices
  - Lazy programmers are good programmers ;)
    - DRY principle: Only change things in one single place in 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
  - **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, always make sure to check user input on server side!

http://www.doheth.co.uk/

Implementation

- Tips and good programming practices
  - Recursion
    - Try to avoid recursion unless you are programming in “recursive-friendly” languages ;)
    - Performance...
    - Memory consumption...
    - No control over the calling sequence

	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

Testing

- **Testing**
  - Different types of testing
  - Automated testing
  - Writing tests
  - Test driven development

Testing

- Different types of testing
  - Unit tests
    - Very useful to test fundamental building blocks in your application
    - Many standard libraries available
    - Generally requires writing many tests
  - 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 realize

Testing

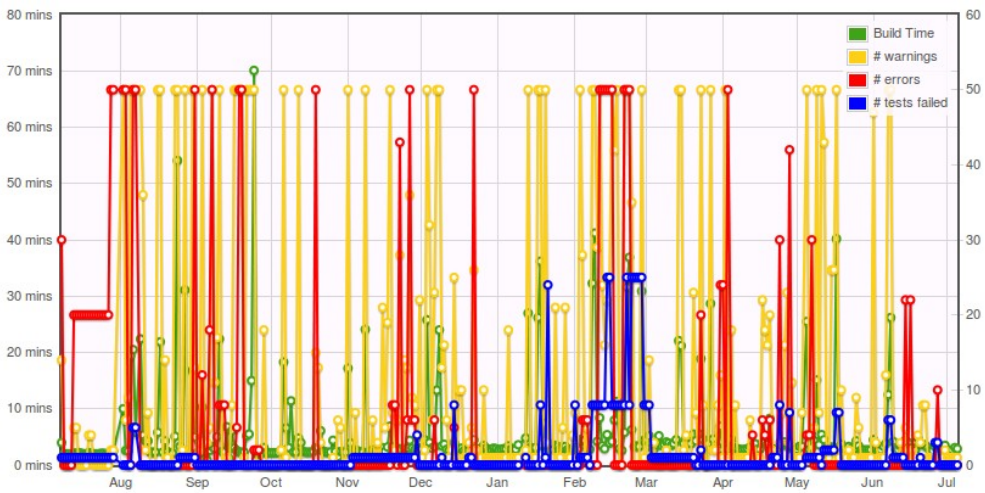
- Writing tests
  - Test suites provide you some guidelines and tools
  - Try to avoid parsing logfiles and matching regular expressions
  - Program examples can often serve as test cases
  - Test definitions vary greatly
    - Test suites generally provide flexible tools to allow writing all kind of tests
    - When writing test macros be careful not to restrict too much!

Testing

- Automated testing
  - If possible, run tests on many different platforms
  - Nightly/Commit-tests / Nightly/Commit-builds
    - Crucial to spot errors as soon as possible
    - Reduces debugging time dramatically
  - Test suites
    - ctest, hudson, ...
  - Valgrind
    - `valgrind -v --tool=memcheck --leak-check=full <bin>`
  - Virtual machines
    - Snapshots
  - `/dev/null`
    - very useful for some types of tests!

Testing

- Test Suites (e.g. CTest)



		Warnings	Configure Errors	Configure Warnings	Build Errors	Build Warnings	Tests Failed
		0	0	0	0	1	0
		0	0	0	0	1	0
		0	0	0	0	1	0
		0	0	0	0	1	0
		0	0	0	0	1	0
		0	0	0	0	2	0
		0	0	0	0	1	0
		0	0	0	0	1	0
		0	0	0	10	1	3
		0	0	0	0	1	3
		0	0	0	0	1	0
		0	0	0	0	1	0
2012-06-23 02:10:23	0	0	0	0	0	1	0
2012-06-22 02:10:22	0	0	0	0	0	8	0
2012-06-21 02:10:21	0	0	0	0	0	8	0
2012-06-20 02:10:20	0	0	0	0	0	3	1
2012-06-19 02:10:19	0	0	0	0	0	4	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	22	1	0
2012-06-15 02:10:15	0	0	0	0	0	2	0
2012-06-14 02:10:14	0	0	0	0	22	5	0
2012-06-13 02:10:13	0	0	0	0	0	1	1
2012-06-12 02:10:12	0	0	0	0	0	3	1
2012-06-11 02:10:11	0	0	0	0	0	1	1
2012-06-10 02:10:10	0	0	0	0	0	1	1
2012-06-09 02:10:09	0	0	0	0	0	1	1
2012-06-08 02:10:08	0	0	0	0	0	50	6
2012-06-07 02:10:07	1	0	0	0	0	50	6
2012-06-06 02:10:06	0	0	0	0	0	12	0
2012-06-05 04:24:05	0	0	0	0	0	1	0

Testing

- Test Suites (e.g. CTest)

Nightly

Site	Build Name	Update	Configure		Build		Test			Build Time
		Files	Error	Warn	Error	Warn	Not Run	Fail	Pass	
it-xenvm027	Linux-c++	0	0	0	0	1	0	1	28	13 hours ago
grid-ilc-pa0	Linux-c++	0	0	0	0	1	0	0	29	13 hours ago

My Projects

Project Name	Actions	Builds	Builds per day	Success Last 24h	Errors Last 24h	Warnings Last 24h
CED		1134	2	2	0	0
CEDViewer		1122	2	2	0	0
ForwardTracking		263	2	2	0	0
GEAR		1127	2	2	0	0
iLCTest		1700	2	0	0	2
KalDet		1118	2	2	0	0
KalTest		1122	2	2	0	0
LCCD		1126	2	2	0	0
LCIO		1122	2	2	0	0
Marlin		1126	2	2	0	0
MarlinReco		1110	2	2	0	0
MarlinTPC		984	2	2	0	0
MarlinTrk		928	2	2	0	0
MarlinUtil		1123	2	2	0	0
Overlay		1110	2	2	0	0
RAIDA		1130	2	2	0	0

Testing

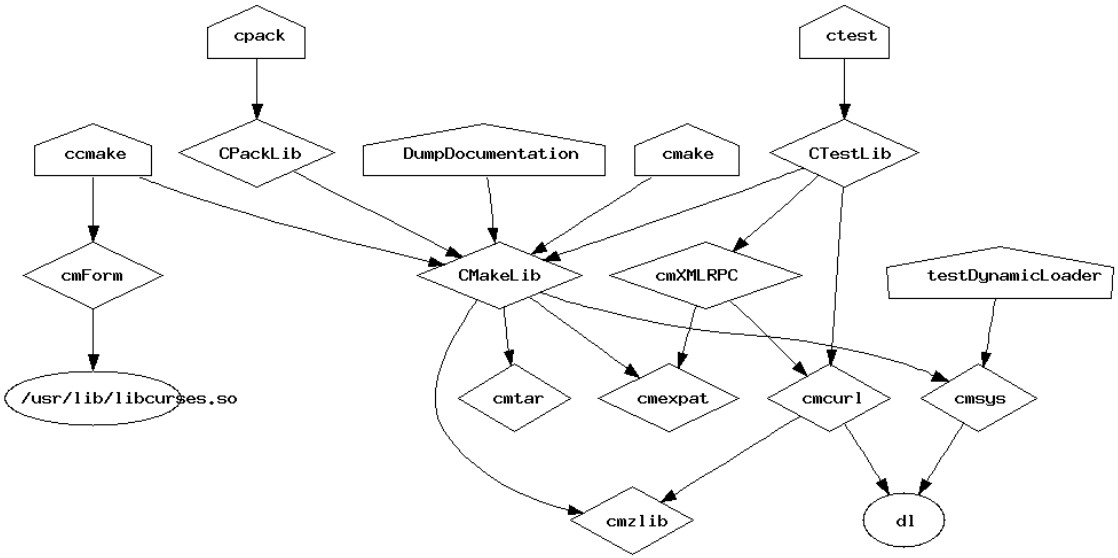
- Test driven development
  - Write tests before starting the implementation
  - Enforces testing from the very beginning
  - Good for projects having many algorithm libraries
  - Improves documentation
  - Helpful if more than one person is involved in the project
  - Also helpful for establishing priorities

Maintenance

- **Maintenance**
  - Dependencies
  - Releasing
  - Deployment
  - Documentation

Maintenance

- Dependencies
  - Internal dependencies
    - Create subpackages or rather re-organize internal code/package directory structure?
    - Each subpackage requires additional maintenance when releasing
    - Some tools can generate dependency graphs (e.g. cmake)



Maintenance

- External dependencies
  - Adding new dependency is easy, removing can be quite painful
    - Wrappers around API's / typedefs
      - Requires substantial amount of additional development time and maintenance!
  - Platform provided packages
    - Usually support is good
    - Must use system version... Not always possible :(
  - Build external packages yourself
    - Requires additional maintenance
  - Don't copy external code into your project...
    - Only hides the real dependency!
    - In the end can cost you more work than it saves you in the beginning
    - You become responsible for updates and security
  - **Support provided? Is it good? For how long?**

Maintenance

- Releasing
  - Create a release policy
    - Once release branch is announced only patches and bugfixes allowed!
    - Distinguish between development and production releases
    - Debug vs stripped binaries
    - Packaging (see next slide)

Maintenance

- Deployment
  - Who are the end users?
  - What environment will the software need to run on?
  - Need to pack dependencies?
  - Tarball Vs. System packages
  - Some build tools provide nice tools for packaging, e.g. cpack
  - Debug, Release, Source packages

Maintenance

- Documentation
  - Undocumented code is unwritten code
  - 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 increases maintenance but reduces the overall support costs

Summary

- Do some analysis and design before starting your project
- Evaluate what libraries/tools might be helpful to use in your project
- Start using source code management tools as soon as possible
- Testing is as important as code!
- Use logging
- Split configuration and settings from source code
- Use standard cmd line argument parsing tools
- Don't neglet configurability
- Don't trust user input under any circumstances!
- Don't give end users more than they need
- Don't forget the documentation
- Try to keep it simple!

Thanks for listening, your feedback is welcome!