Software Systems Design

Version control systems and documentation

Who are we?

Krzysztof Kąkol
Software Developer

Jarosław Świniarski
Software Developer

Presentation based on materials prepared by

Andrzej Ciarkowski, M.Sc., Eng.

Outline

- Version control systems
  - Distributed vs Client-server approach
  - Common vocabulary & idioms
  - The tools
- Documentation generation tools
  - API Documentation Metalanguages
  - The tools
- Issue tracking systems
  - The lifecycle of a bug
  - The tools
- Requirements engineering
  - The process
  - Requirements elicitation
  - Requirements types
- Software design
  - Design levels
  - Design concepts

Version control systems

What is it for?

- Allows to get back to any point of time in the history of the project – e.g. when some code changes introduce new bug, it can be easily traced
- Serves as an additional backup copy of the project files
- Allows to make informative comments on each code revision
- Makes it possible to make concurrent lines of code (branches) with different features developed separately
- Allows to blame responsible persons for the bugs

VCS architectures

Client-server

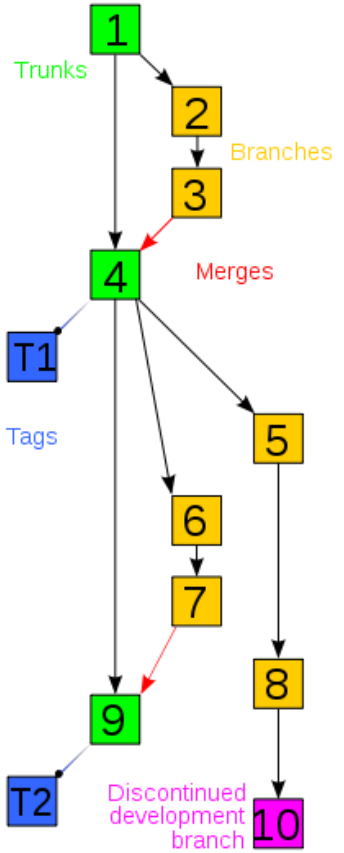
- The “traditional” approach
- Single repository is located on a single VCS server (Master copy)
- Each team member uses VCS client to connect to the server and retrieve or commit back the changes
- The examples: CVS, Subversion (SVN)

Distributed

- The “modern” approach
- There are multiple repositories, each of them may contain full of partial copy of the entire codebase
- The repositories may be synchronized to each other
- The programmers’ “working copy” is also a distributed repository
- The examples: Git, Mercurial

Revision graph

- Code revisions in the repository form a revision graph
- The main development branch spanning the entire project life is a “trunk”
- There may be some additional branches used for e.g. adding experimental features
- When the features developed in a branch are mature enough they may be merged back into the trunk
- Alternatively the branch may be abandoned and discontinued
- Project milestones are labelled with “tags” so that it’s possible to easily return to any chosen milestone/software version
- The users’ working copy may be also considered a branch, which is merged back to the repository when user commits the code (this is exactly how it is done in Git)



Basic VCS operations

- **checkout** – create local working copy from the repository (svn checkout | git checkout)
- **commit** – send back local changes to the repository (svn commit | git commit)
- **merge** – combine changes made in two branches into one branch, possibly causing conflicts (svn merge | git merge)
- **update** – merge changes made in the repository by other people into the local working copy (svn update | git pull)
- **branch/tag** – create a new named branch or tag based on the base revision of working copy (svn branch / tag | git branch / tag)
- **status** – get changed status of the files in the working copy (svn stat | git stat)
- **resolve** – mark versioning conflict as resolved (svn resolve | git resolve)
- **revert** – revert changes done to specific files/folders since last update/checkout (svn revert | git revert)

Basic usage scenario - SVN

```
svn checkout <remote repository url> [local path]
```

creates local working copy from the repo

edit some code, add files, etc.

```
svn commit –m „[project] added file test.c”
```

commit local changes to the remote repo

Basic usage scenario - SVN

```
svn stat
```

query status of local working copy

```
svn update
```

merge remote changes into local working copy

```
svn add test2.c
```

add new file to the working copy

```
svn commit –m „[project] added file test2.c”
```

commit local changes to the remote repo

Basic usage scenario – Git

```
git clone <remote repository url> [local path]
```

*creates local repo clone of remote **origin** repo*

edit some code, add files, etc.

```
git commit –m „[project] added file test.c”
```

commit local changes to the local repo

```
git push origin
```

synchronize changes in local repo to remote origin

Basic usage scenario – Git

```
git stat
```
query status of local working copy

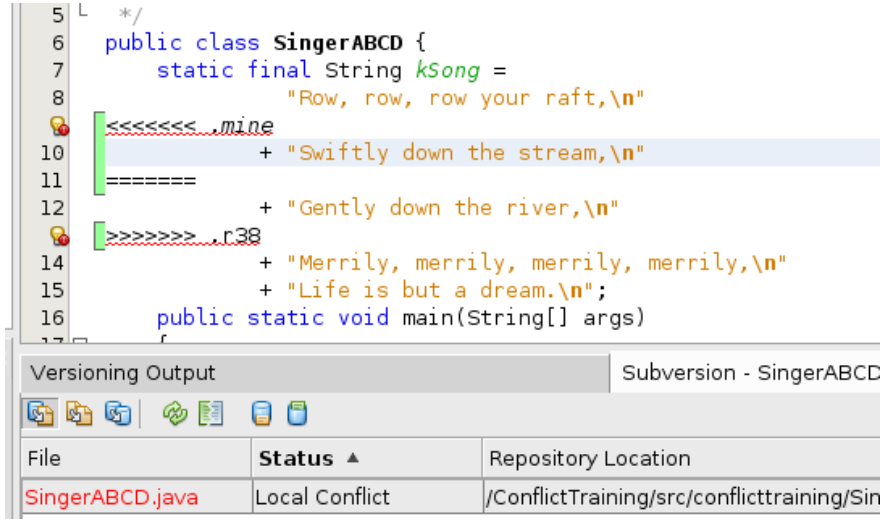
```
git pull
```
merge remote changes into local repository clone & working copy

```
git add test2.c
```
add new file to the working copy

```
git commit –m „[project] added file test2.c”
git push origin
```

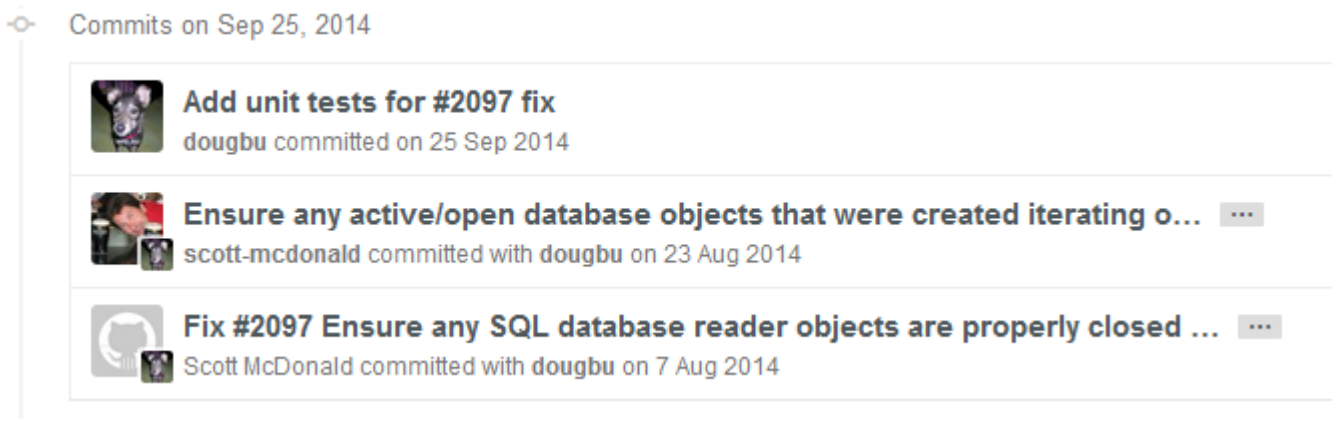
Versioning conflicts

- **Conflict** happens when multiple users make changes to the same file simultaneously
- Conflicts usually indicate bad communication in the development team and should be avoided
- Kinds of conflicts
  - Tree conflict – change in the file/folder structure, renaming files, moving folders etc
  - Editing conflict – multiple edits of the same file region
- Conflict must be resolved before file may be committed



How to write messages

- Author, timestamp and list of affected files is given automatically (who, when, what)
- Reference of task or bug in bugtracking system (what)
- Short description (why, how)



What should be stored in VCS repository

- Source code (except for runtime-generated files)
- IDE project files (except for site/user-specific configuration)
- Project resources
- Project maintenance/build scripts (e.g. database creation scripts)
- Everything that is needed to get the project to build and is not generated on the fly or common component downloaded from public location

What shouldn’t be stored in the repository

- Source files generated by the build system during the building process
- Project intermediate & output files (e.g. .obj, .o, binary files, debugging symbols…)
- Everything that is automatically generated during the build

Tools

- Command-line clients
  - svn (Subversion); use **svn --help**
  - git (Git); use **git –help**
- GUI clients
  - TortoiseSVN (Windows)
  - TortoiseGIT (Windows)
  - GitHub (Windows/Mac)
  - SourceTree (Windows/Mac)

Tools

- Integrated Development Environment (IDE) Integration
  - AnkhSVN (Subversion for Visual Studio)
  - Subclipse (Subversion for Eclipse)
  - Git is integrated by default both in Visual Studio and Eclipse
- Additional tools used by VCS
  - **diff** – UNIX tool for generating “difference” files between 2 text files in so-called “unified diff” format; diff tool is used by almost all VCS for sending the code changes between revisions
  - **patch** – UNIX tool for integrating diff files back with source files

Tools

- Hosted repositories
  - GitHub – free for open-source projects (public repository)
  - GitLab
  - BitBucket
  - Assembla
  - CodePlex
  - SourceForge

VCS systems - sources

https://en.wikipedia.org/wiki/Revision_control

https://subversion.apache.org/

http://svnbook.red-bean.com/

http://git-scm.com/

http://git-scm.com/book/en/v2

http://pcottle.github.io/learnGitBranching/

Automatic code documentation tools

- Writing code is funny enough to be distracted by writing additional API docs ;)
- Code documentation tools serve for generating documentation automatically based on the code and developers' meta tags
- The code still needs to be amended by the tags and the documentation prose in the comments, but there's no need for context switching between writing the code and the docs

API docs metalanguages

- Some languages have documentation built into the standard and supported by the compiler
  - Java – JavaDoc
  - C# - XML Docs
- There is no “official” standard for C nor C++
  - But “de facto” standard exists – Doxygen
- The good thing is that Doxygen supports also JavaDoc and MS XML Docs, so it’s a “one for all” tool

```
/**
 * Graphics is the abstract base class for all gra
 * and includes:
 * <ul>
 * <li>The Component to draw on
 * <li>A translation origin for rendering and clip
 * <li>The current clip
 * <li>The current color
 * <li>The current font
 * <li>The current logical pixel operation functio
 * <li>The current XOR alternation color
 *     (see <a href="#setXORMode">setXORMode</a>)
 * </ul>
 * <p>
 * Coordinates lie between the pixels of the
 * output device.
 *
 * @author      Sami Shaio
 * @author      Arthur van Hoff
 * @version     %I%, %G%
 * @since       1.0
 */
public class Graphics {
```

Doxygen features

- Generation of HTML, PDF, LaTeX, Windows Help, UNIX man and many other formats from single sources
- Support for embedded math formulas (LaTeX), rich formatting elements, code hyperlinking
- One documentation language to cover all interesting programming languages

Doxygen in action

```
/*!
 * @brief Functor for computing Discrete Fourier Transform (DFT).
 * This class implements DFT of complex sequences. The complex number domain
 * is determined by the template parameter Real. The DFT is calculated
 * through FFT algorithm (Cooley-Tukey) with Danielson-Lanczos recursion.
 * The "canonical" DFT equation is:
 * @f[
 * X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi\frac{k}{N}n}
 * @f]
 * so the sign of the exponent is -1, which is reflected in the va
 * @link dft::sign::forward @endlink constant (for forward DFT). T
 * for inverse DFT is:
 * @f[
 * x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k \cdot e^{i2\pi\frac{k}{N}
 * @f]
 * where the sign of exponent is 1 (and consequently the value of
 * @link dft::sign::backward @endlink). This class implements both
 * through the sign parameter of the constructor, however it doesn
 * the normalization of IDFT results (the @f$\frac{1}{N}@f$ factor
 * To obtain correctly normalized values, the output sequence shou
 * divided by N.
 * @see W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P.Flannery.
 * @tparam Real floating-point type using during the calculations.
 */
template<class Real>
class fft<complex<Real>, complex<Real> > {
public:
```

template<class Real>
class dsp::dft::fft< complex< Real >, complex< Real > >

Functor for computing Discrete Fourier Transform (DFT). This class implements DFT of complex sequences. The complex number domain is determined by the template parameter Real. The DFT is calculated through FFT algorithm (Cooley-Tukey) with Danielson-Lanczos recursion. The "canonical" DFT equation is:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi\frac{k}{N}n}$$

so the sign of the exponent is -1, which is reflected in the value of **dft::sign::forward** constant (for forward DFT). The equation for inverse DFT is:

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k \cdot e^{i2\pi\frac{k}{N}n}$$

where the sign of exponent is 1 (and consequently the value of constant **dft::sign::backward**). This class implements both forward and inverse-DFT through the sign parameter of the constructor, however it doesn't perform the normalization of IDFT results (the $\frac{1}{N}$ factor in the formula). To obtain correctly normalized values, the output sequence should be divided by N.

See also

W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P.Flannery. *Numerical Recipes in C++*. Cambridge university press, 2002.

Template Parameters

Real floating-point type using during the calculations.

Definition at line **70** of file **fft.h**.

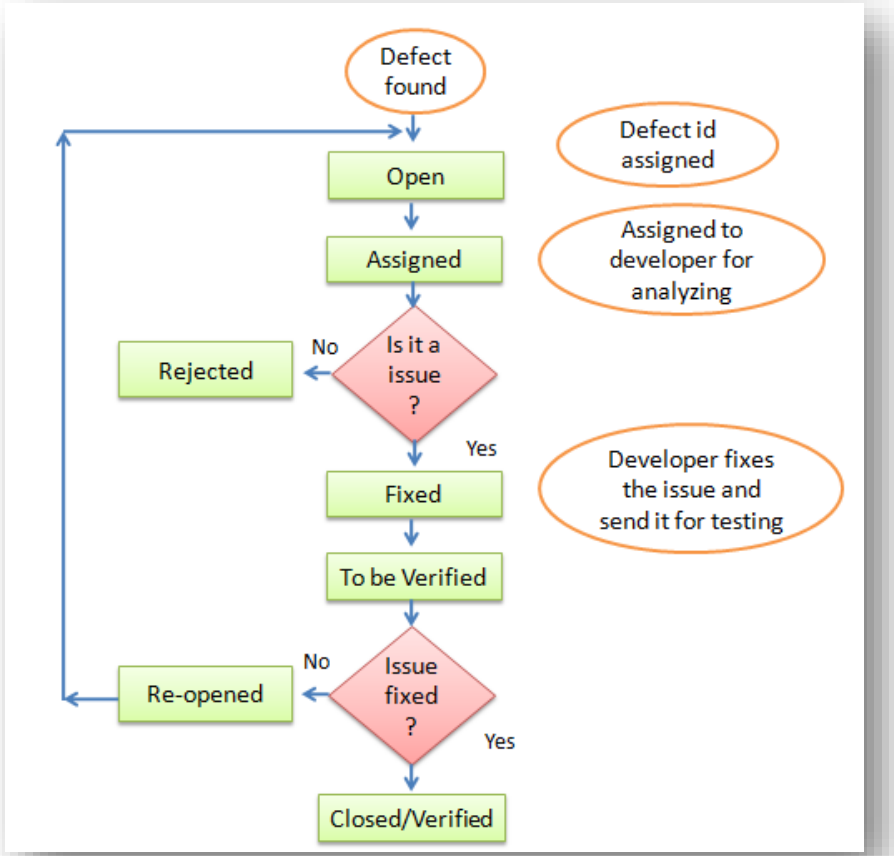
Additional tools

GraphViz – Graph Visualization Toolbox – integrated with Doxygen, allows for automatic generation of UML class diagrams and many others

Issue tracking systems

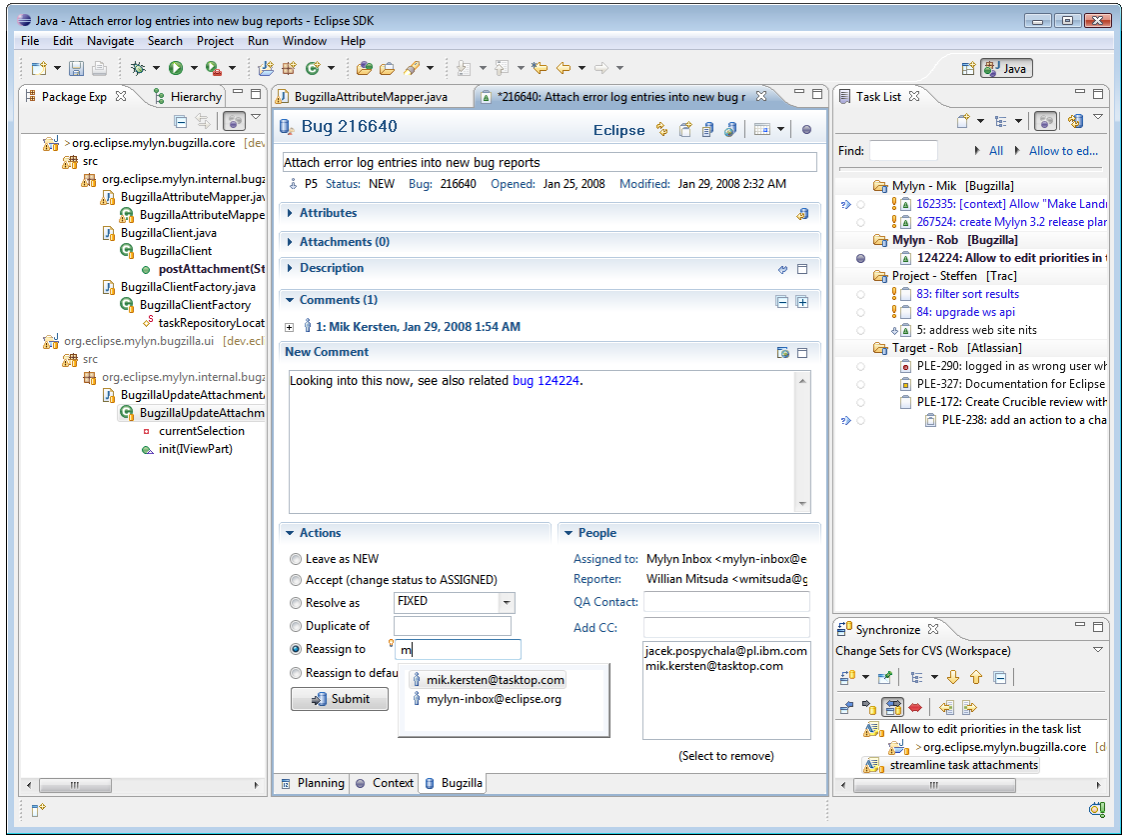
- Every piece of software has **bugs**
- Developers tend to forget about discovered bugs as soon as they start coding again (or even sooner)
- Issue tracking systems are for managing the information about the bugs and not letting the lazy developers never get back to them ;)

Typical lifecycle of a bug



Issue tracking system features

- Web-based user interface for filing & managing bug reports
- Integration with VCS repositories
  - Hyperlinking bug reports in VCS commit comments (#3345)
  - Hyperlinking revision numbers in bug reports (r109)
  - Browsing VCS code changes through issue tracking UI
- Sending email notifications on bug status changes (notifying project maintainer on open bugs, notifying reporter on closing, etc.)
- IDE Integration – task-focused interfaces


Java - Attach error log entries into new bug reports - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Java
Package Exp
Hierarchy
BugzillaAttributeMapper.java
*216640: Attach error log entries into new bug r
Bug 216640
Eclipse
Attach error log entries into new bug reports
P5 Status: NEW Bug: 216640 Opened: Jan 25, 2008 Modified: Jan 29, 2008 2:32 AM
Attributes
Attachments (0)
Description
Comments (1)
1: Mik Kersten, Jan 29, 2008 1:54 AM
New Comment
Looking into this now, see also related bug 124224.
Actions
Leave as NEW
Accept (change status to ASSIGNED)
Resolve as
FIXED
Duplicate of
Reassign to
Reassign to defau
Submit
mik.kersten@tasktop.com
mylyn-inbox@eclipse.org
People
Assigned to: Mylyn Inbox <mylyn-inbox@e
Reporter: Willian Mitsuda <wmitsuda@g
QA Contact:
Add CC:
jacek.pospychala@pl.ibm.com
mik.kersten@tasktop.com
(Select to remove)
Planning
Context
Bugzilla
Task List
Find:
All
Allow to ed...
Mylyn - Mik [Bugzilla]
162335: [context] Allow "Make Landi
267524: create Mylyn 3.2 release plar
Mylyn - Rob [Bugzilla]
124224: Allow to edit priorities in
Project - Steffen [Trac]
83: filter sort results
84: upgrade ws api
5: address web site nits
Target - Rob [Atlassian]
PLE-290: logged in as wrong user wh
PLE-327: Documentation for Eclipse
PLE-172: Create Crucible review with
PLE-238: add an action to a cha
Synchronize
Change Sets for CVS (Workspace)
Allow to edit priorities in the task list
streamline task attachments
org.eclipse.mylyn.bugzilla.core
org.eclipse.mylyn.bugzilla.ui
src
BugzillaClient.java
BugzillaClient
BugzillaClientFactory.java
BugzillaClientFactory
BugzillaUpdateAttachment
currentSelection
init(IViewPart)

How to properly report a bug

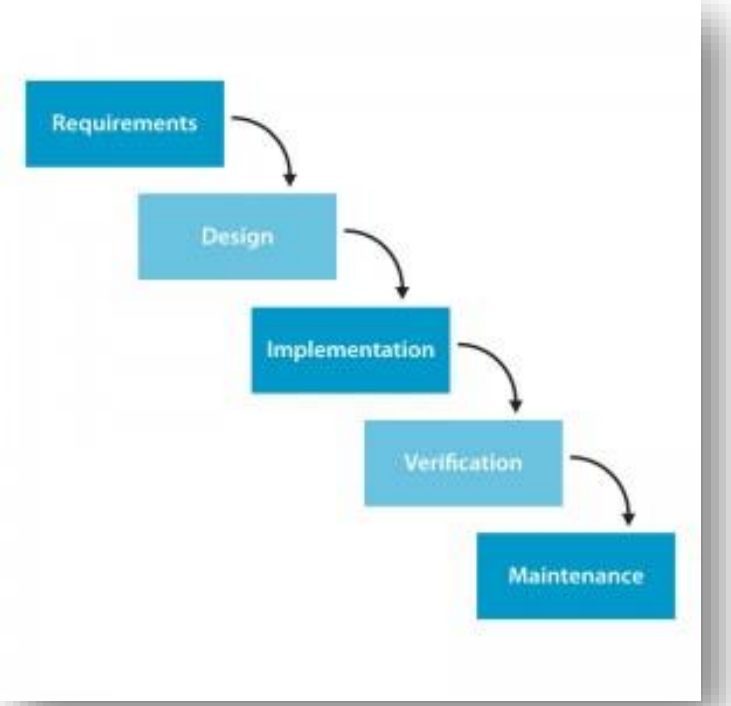
- Brief summary i.e. „Module XYZ throws exception when no data provided”
- Detailed description
- Steps to reproduce the problem
- Description of environment (OS, software version, platform etc.)
- Screenshots, video screen captures
- Logs
- Data needed to reproduce the problem i.e. account names

Tools

- Bugzilla – issue tracking system originally developed for Mozilla web browser
- Trac – issue tracking, project management & documentation Wiki in python
- JIRA – large-scale software project management & bug tracking (commercial)
- Mantis, Launchpad, GNATS and many many others…

Requirements engineering

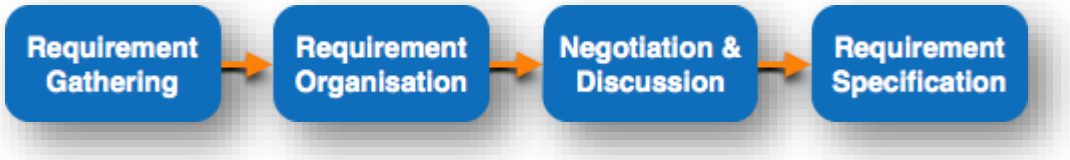
- The “classical” waterfall model lists 2 phases before the actual implementation:
  - **Requirements engineering**
    The requirements define the required functionality of the final package. These are driven by what the end-user expects the system to provide.
  - **Design**
    Given a set of user and system requirements design often takes two steps: an overall system architecture followed by detailed design of the system’s modules and interfaces
- No matter what development model is used these phases take place in some form almost always



Requirements engineering stages

- Feasibility study
  - **The client** approaches the organization for getting the desired product developed, comes up with rough idea about what all functions the software must perform and which all features are expected from the software.
  - **The analyst** performs a study whether the system and functionality are feasible to develop.
  - **The output** is a decision whether project should be undertaken.
- Requirements gathering & analysis
  - **The analyst** and **engineers** communicate with the **client** and **end-users** to know their ideas on what the software should provide and which features they want the software to include.
- Requirements specification
  - **The document** created by the **analyst** after the requirements are collected from the **stakeholders**
- Requirement validation
  - The specification is validated if it is legal, practical, complete, unambiguous etc.

Requirement elicitation process



- Requirement elicitation is performed through
  - Interviews (oral, written, one-on-one, group, etc…)
  - Surveys
  - Questionnaires
  - Task analysis – team of engineers and developers analyses the operation and possibly the existing solutions
  - Domain analysis – experts from the given domain are questioned regarding the requirements
  - Brainstorming – informal debate among stakeholders
  - Prototyping – building UI model and gathering the client's feedback
  - Observation – team of experts visits client's workplace and observe current practices & workflow

Requirement types

- Functional requirements
  - Define function of a system
  - Described as a set of inputs, the behavior and outputs
  - Define **WHAT** a system is supposed to accomplish
  - Expressed in the form “system must do …”
- Non-functional requirements
  - Impose constraints on the design or implementation with regard to performance, security, reliability…
  - Influence system architecture
  - Define how a system is supposed to be
  - Expressed in the form “system shall be …”

Functional requirements

- Business processes and workflow are explored in terms of how they would be mirrored in a software system.
- An analyst with deep understanding of both the business processes and a computer system's capabilities is required to gather functional requirements
- Functional requirements are captured in the Use Cases
- The example
  - Client asks for something simple: "Registered users may enter their address."
  - The functional requirements could be stated as: "Registered users may optionally add or edit one or more addresses associated with their account. Addresses are listed on the account detail screen. The edit form is a popup with fields for street, city, ...".
  - The functional specifications would then state how a user becomes registered, where in the application they may enter an address, the input fields that comprise the address, input validation logic, and the data model and systems used for storage.

Functional requirements examples

- Interface requirements
  - Field 1 accepts numeric data entry
  - Field 2 only accepts dates before the current date
  - Screen 1 can print on-screen data to the printer
- Business requirements
  - Data must be entered before a request can be approved.
  - Clicking the Approve button moves the request to the Approval Workflow.
- Regulatory/compliance requirements
  - The database will have a functional audit trail.
  - The system will limit access to authorized users.
  - The spreadsheet can secure data with electronic signatures.
- Security requirements
  - Members of the Data Entry group can enter requests but can not approve or delete requests.
  - Members of the Managers group can enter or approve a request but can not delete requests.
  - Members of the Administrators group cannot enter or approve requests but can delete requests.

Non-functional requirements

- Called “qualities”, “constraints”, “quality attributes”, “non-behavioral requirements”, “-ilities”

  - Accessibility
  - Availability
  - Capacity
  - Compliance
  - Deployment
  - Effectiveness
  - Environmental protection
  - Exploitability
  - Extensibility
  - Interoperability
  - Maintainability
  - Modifiability
  - Operability
  - Performance
  - Portability
  - Reliability
  - Reusability
  - Response time
  - Scalability
  - Stability
  - Supportability
  - Testability
  - Usability
  - ……

Functional vs. non-functional

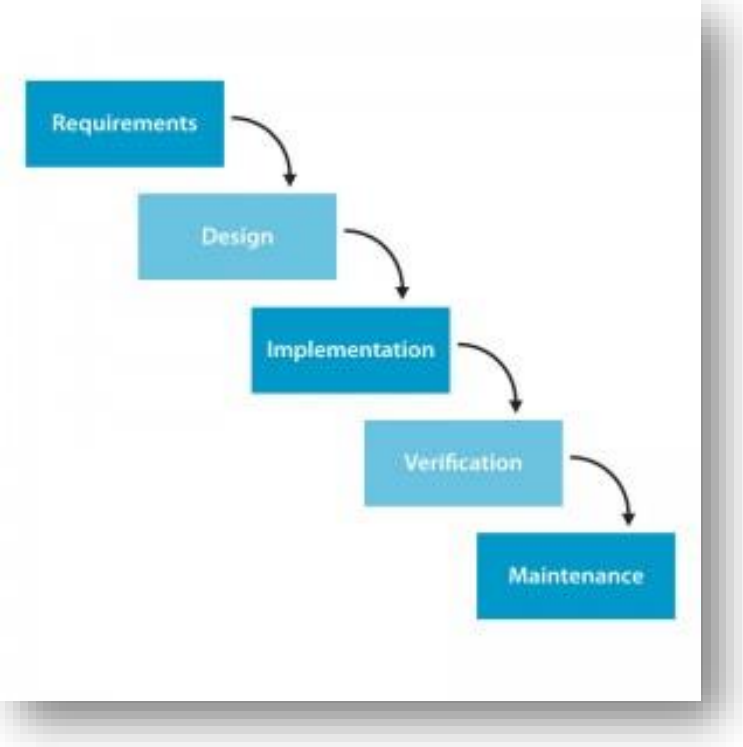
- Sometimes the difference between functional and non-functional requirements is hard to draw
- Sometimes it's impossible to separate the issues "what system does" from "how it is done"
- Depending on the context, similar requirements may be considered functional or non-functional
- It's better to list the requirement under wrong category than to not list it at all
- It takes a great deal of experience to get the distinction right, so don't worry ☺

Other interesting requirement categories

- Data model and data requirements
  - Without technical details like DBMS system, record structure etc.
  - Dependencies or 3rd party data sources, expectations towards user data input
  - Data model explained only at level which solves business needs, e.g. list of the user's address fields.
- User Interface requirements
  - The look and feel of the system
  - Sample UI forms
- Performance requirements
- Hardware platform
- Software dependencies
- Security

Software design

- Ok, so we have our requirements specced out, what next?
- Let’s move to the design phase and switch focus from problem domain to solution domain
- Now is the time to get technical with our requirements
- “software design takes the user requirements as challenges and tries to find optimum solution”



Design levels

- Architectural design – highest abstract version of the system, identifies the software as a system with many components or modules interacting with each other
- High-level design – breaks the components into modules or sub-modules – sometimes needed in large systems
- Detailed design – deals with the implementation of modules, defines logical structure of each module and their interfaces to communicate with other modules

Design strategies

- Structured design
  - Conceptualization of problem into several well-organized elements of solution, based on “divide-and-conquer” rule
  - Small pieces of problem become modules
  - Modules are arranged in hierarchy and communicate with each other
- Function-oriented design
  - System is comprised of many smaller sub-systems (**functions**), which perform some significant tasks
  - Functions are similar to modules, which share information among themselves by means of information passing and **globals**
  - When a program calls a function, the function **changes state of the program**
  - Works well where system state does not matter and program work on input rather than on state

Design strategies

- Object-oriented design
  - Focuses on entities (**objects**) and their characteristics instead of functions
  - All entities involved in the solution are known as objects; objects have some attributes and some methods to perform on attributes
  - All objects associated a generalized description known as **class**. An object is an instance of a class. Class defines the attributes and methods forming the functionality of the objects.
  - Encapsulation – the attributes and methods are bundled together, access to the data and methods from the outside world is restricted (**information hiding**) – decreases coupling
  - Inheritance – classes may be organized in a hierarchical manner where lower sub-classes can import, implement and re-use parts of their super-classes
  - Polymorphism – a mechanism allowing different tasks to be performed on different types (classes) using common interface shared among them

Questions?

Krzysiek kkakol@pgs-soft.com
Jarek jswiniarski@pgs-soft.com

www.pgs-soft.com