Agile software management for research environments

Gema Ramírez-Sánchez
Prompsit Language Engineering

Campus UMH. Quorum III building
Avinguda de la Universitat s/n
03202, Elx (Alacant) Spain

Overview

- Aim: defining best practices for software management in research environments
- 8 keys for success:

Version control	Packages and releases
Regression tests	Licenses and authorship
Standards	Sharing
Documentation	Maintenance and management

Some useful info:

- Guide in PDF and this slides:
- Go to www.abumatran.eu
- Download:
  abumatran-deliverable-2.1-workshop-guide.pdf,
  abumatran-deliverable-2.1-workshop-slides.pdf
- For tasks: use Firefox or Chrome for the exercises.

Version control

- Version control systems:
  - track and provide control over changes in source code.
  - work as stand-alone applications or embedded in other systems (LibreOffice)
- Useful when:
  - you need to collaborate with people on documents
  - you are working on a development team

Version control

- Version control systems provide:
  - A repository (centralised or distributed)
  - A concurrency policy (lock or merge)
- Great choice, basically all the same:
  - Creation of repositories, files management, check-outs and check-ins, branching and tagging.
- See SVN and Git basic commands

(Task 1: try out Git!)

Best practice number 1

Put your software source code and documents under a version control system

Regression tests

- Regression tests
  - detecting bugs in a piece of software after a modification.
  - avoid lost of previous behaviour or functionalities
  - ensure that we've modified our program as we were intending to
- Two main types of regression tests:
  - Functional tests
  - Unit tests

Regression tests

Functional tests	Unit tests
assess user-visible behavior of a system	assess the behaviour of individual pieces of a system source code
black-box examination, what it does	white-box examination, how it does it
written from user's perspective	written from developer's perspective
do not change frequently	change as frequently as code grows
Example: **arrival day field** of a webform should display a list of *possible days of the week* and option *any day*	Example: function **validate_day_of_the_week()** = *false* if *January* = true if *null*

Regression tests

Functional tests	Unit tests
Step 1: identify functionalities	Mantra: one unit test per method, class, function = time consuming!
Step 2: create test set	At least one unit test per bug and specially buggy parts of code
Step 3: create output set	Test for success, failure and sanity
Step 4: execute the test cases	Make tests create their own test data and place them in isolated files
Step 5: evaluate against output set	Each test must: set up the testing condition, call the method, verify

Regression testing

- Task 2: Unit testing a roman numeral conversor:
  - Some thoughts about roman numbers
    - Which roman numbers exist?
    - Which are particular cases of other ones?
    - Are there non-integer or negative roman numbers?
  - Test: http://www.thecalculatorsite.com/misc/romannumerals.php
  - See: http://www.diveintopython.net/unit_testing/testing_for_failure.html

Best practice number 2

Write regression tests to asses performance of your code (unit tests) and the functionalities of your system (functional tests)

Standards

A (*de jure*) standard is...

> *“A document established by consensus and approved by a recognized body that provides for common and repeated use, rules, guidelines or characteristics for activities or their results, aimed at the achievement of the optimum degree of order in a given context.” ISO/IEC Guide 2:1996, definition 3.2*

… an antidote against chaos!

Standards

We want standards for...

> *“Standards define what is needed for interoperability: no more, no less.”* Andrew Tanenbaum.

… maximizing interoperability!

And for...

- ease of understanding and learning
- robustness
- development speed up

Standards

Standards for software? Not a unique answer...

- established languages/technologies: C/C++, JAVA for programming, HTML, PHP for web, MySQL as a database, XML for data, etc.
- modularity
- code and data decoupling

"*The nice thing about standards is that you have so many to choose from.*" Andrew Tanenbaum.

(Task 3: Who did what in software standards?)

Best practice number 3

Use standards to maximize interoperability, to avoid misunderstandings, to save time and to build upon solid foundations

Documentation

- Development phase
  - specifications
  - architecture/design
  - technical: code, algorithms, API
- Release phase
  - manuals for user, administration and support
- Dissemination phase
  - papers, articles, projects
  - a website!!!
  - press releases, marketing material

Documentation

- A non-trivial task
  - Schedule time and do it when feeling like
  - Be empathic: play the role of each reader
  - Don't expect immediate gratification: it will come...

Further reading: *Fifteen Thoughts and Tips on Writing Software Documentation*. Borja Sotomayor. Available at:

https://plus.google.com/106052512842507340767/posts/heH3pT2pczt

Documentation

- Some helping ideas
  - define a time and a goal per document and go!
  - or ask others to help (teaming or outsourcing)
  - use templates as guides
  - automate, when possible and useful

(Task 4: Documenting style: see Git and Subversion examples)

Documentation

- Some helping ideas
  - define a time and a goal per document and go!
  - or ask others to help (teaming or outsourcing)
  - use templates as guides
  - automate, when possible and useful

(Task 4: Documenting style: see Git and Subversion examples)

Best practice number 4

Document your software, for a given purpose, having in mind who is the reader, taking the pleasure to tell others about your work

Packages and releases

Software native form = source code

Package = source code + building, installing, maintaining mechanism

GIMP 2.8.4 is now available at ftp://ftp.gimp.org/pub/gimp/v2.8/. You may want to read the Release Notes for GIMP 2.8.

To allow you to check the integrity of the tarballs, here are the MD5 sums of the latest releases:

File	MD5 sum
gimp-2.8.4.tar.bz2	392592e8755d046317878d226145900f
gimp-2.8.2.tar.bz2	b542138820ca3a41cbd63fc331907955
gimp-2.8.0.tar.bz2	28997d14055f15db063eb92e1c8a7ebb

Which source code is included in a package? The one defined as a *release*

Packages and releases

What it's a release? It is a version of the source code where maybe...

- known bugs have been fixed
- new features have been added
- new configuration options have been added

Who decides what and when?

- schedule / circumstances
- project manager or a project committee

Packages and releases

What it's a release? It is a version of the source code where maybe...

- known bugs have been fixed
- new features have been added
- new configuration options have been added

Who decides what and when?

- schedule / circumstances
- project manager or a project committee

Packages and releases

When making a release we should think about:

- Stabilising: make the release in a branch
- Numbering: decide a scheme which reflects release ordering, deep and nature of changes
- Scheduling: decide strategy (*release early, release often*[1] vs. polished bug-free releases)

1. "*Release early, release often. And listen to your customers!*" Eric Raymond.

Packages and releases

To get a package ready:

- Format: TAR – tgz (Linux) – zip (Windows)
- Name: software name + release version + file extension
- Layout: software name directory + README, INSTALL + COPYING + CHANGES + source code + config + compilation + third-party software
- Compilation and installation: configure && make && make install, for C/C++ under Linux, ant for JAVA, Visual Studio or VS.NET environments for Windows
- Binary package: .deb, .rpm, .msi, .exe

Packages and releases

To release the package:

- pass regression and unit tests
- make users install and try it out
- digitally sign it
- compute MD5/SHA1 checksum
- publish and announce it!

(Task 5: a hot topic: numbering a release.)

(Task 6: agree or disagree with some releasing policies)

Best practice number 5

Package clean and easy-to-use versions of your software and release them

Licenses and authorship

- A software license is a way to tell others what are they allowed to do with a software
- By default, in many countries, *copyright* (*all rights reserved*) applies
- Which are the alternatives?
  - Public domain, uncopyrighted
  - Registry offices
  - Use other software licenses

Licenses and authorship

- Software licenses:
  - Late 80's: first *copyleft* licenses appear. Currently: more than 60 different licenses
  - Proprietary software: the most used is the end-user license agreement (EULA), *some rights reserved*
  - Free software: the most common ones are
    - GNU General Public License (copyleft)
    - BSD-Like Licenses (non-copyleft)
    - Creative Commons Licences (non-)copyleft

(Task 7: is copyleft a freedom or a limitation?)

Authorship

To tell the world that you are the author of a software:

- distribute software with an AUTHORS file
- put in it the name of the author/authors: can be you or the organisation you are working for

In collaborative environments: authorship is shared and AUTHORS file will be extended to all contributors

Best practice number 6

Choose a license for your software, indicate the authorship

Sharing

- Sharing is all about:
  - **giving something back to society**: end-user tools, opportunities for further research or business, etc.
  - **contributing to science**: by radically guaranteeing the reproducibility of experiments. By avoiding starting from scratch or reinventing
  - **contributing to the creation of *de facto* standards**
  - **creating an opportunity for collaboration**
  - **increasing your work visibility**

Sharing

- Web-based hosting services ease sharing:
  - SourceForge.net (1999), hosts open-source software projects. 320.000 projects, 3.4 million users, 4,000,000 downloads a day. Provides source-code browser, wikis, ticketing, mailing, forum, branching, etc.
  - Github (2008), hosts open-source (free) and private (paid) software. 6 million repositories, 3.5 million users. Provides source-code browser, in-line editing, wikis, and ticketing.

(Task 8: shelf-hosting advantages/disadvantages?)

Best practice number 7

Share your software, let it succeed!

Maintenance and management

Development is... where all it starts...

- Maintenance and management are long-lasting activities but crucial to any software project!
- Problem: research environments usually work with projects limited in time.
- Sharing brings a possible solution and a real workforce: the community

Maintenance and management

A maintenance policy should define :

- how will users report bugs or make suggestions
- what will be maintained
- how frequently maintenance activities are expected
- who is the maintainer of what parts of the software
- who is the project manager
- what is planned for future work

Maintenance and management

A management policy should define:

- who is who in the project: decision-makers, contributors, users, etc.
- which are the governing by-laws of the project
- the general lines of interest, activities and aims of the project

(Task 9: community managing. How to solve a conflict among contributors?)

Best practice number 8

Ensure and organise support for your software project, let everybody know how things will work

Agile software management for research environments

THANKS!!!

Visit: www.prompsit.com - www.abumatran.eu

Write to: gramirez@prompsit.com

Acknowledgments: Abu-MaTran project funded by EU FP7/2007-2013 under grant agreement PIAP-GA-2012-324414