Basics of collaborative software development

Software project
"NLP tools for low-resource languages"

Alexis Palmer & Michaela Regneri

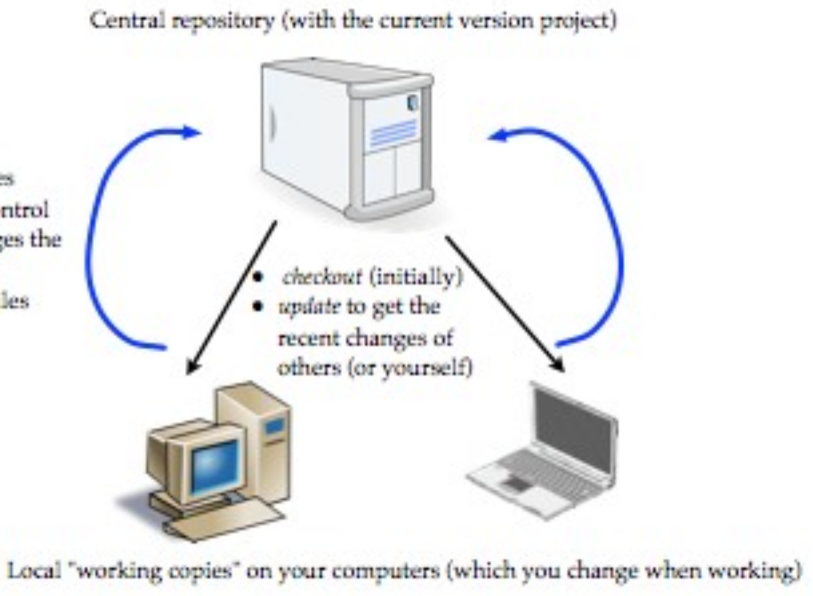
Differences

Let's go for this

Overview

- version control & bug / issue tracking
- core tasks for a (comp. ling.) developer team
  - coding
  - quality assurance
  - data management
  - project management
- your development process(es)

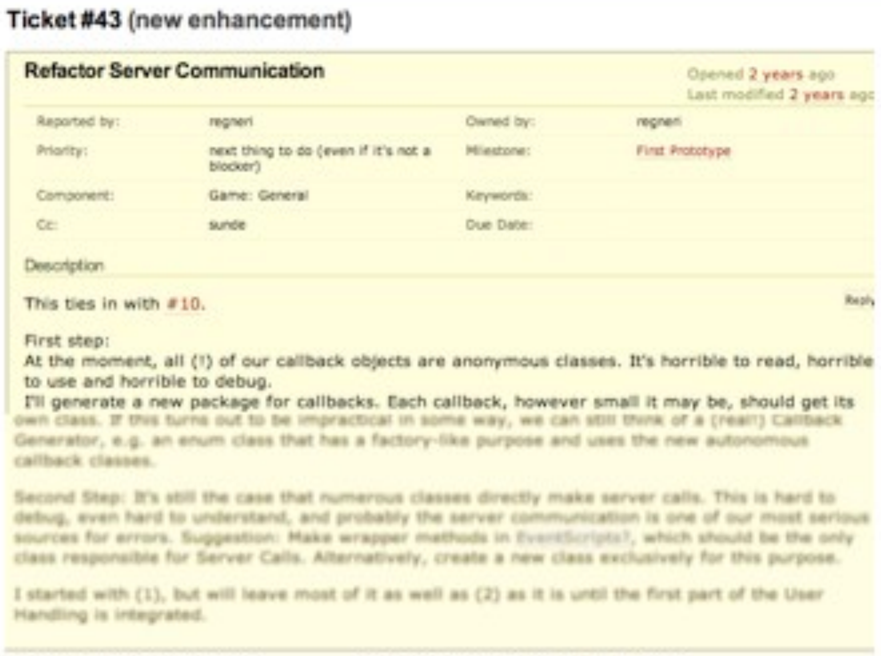
Version Control


Central repository (with the current version project)
• *commit* changes (the version control system manages the merging)
• *add* / *remove* files
• *checkout* (initially)
• *update* to get the recent changes of others (or yourself)
Local "working copies" on your computers (which you change when working)

Version Control

- you can retrieve any previous version from the repository
- if you and your colleagues work on different things (even in the same file), svn merges the results
- you can only commit your changes if you checked out the most recent changes before
- if the recent changes concern something you changed locally, there will be *merge conflicts*

Bug / Issue tracking



Ticket #43 (new enhancement)

Refactor Server Communication

Opened 2 years ago
Last modified 2 years ago

Reported by:	regneri	Owned by:	regneri
Priority:	next thing to do (even if it's not a blocker)	Milestone:	First Prototype
Component:	Game: General	Keywords:	
Cc:	sunde	Due Date:	

Description

Reply

This ties in with #10.

First step:
At the moment, all (!) of our callback objects are anonymous classes. It's horrible to read, horrible to use and horrible to debug.
I'll generate a new package for callbacks. Each callback, however small it may be, should get its own class. If this turns out to be impractical in some way, we can still think of a (real!) Callback Generator, e.g. an enum class that has a factory-like purpose and uses the new autonomous callback classes.

Second Step: It's still the case that numerous classes directly make server calls. This is hard to debug, even hard to understand, and probably the server communication is one of our most serious sources for errors. Suggestion: Make wrapper methods in EventScripts?, which should be the only class responsible for Server Calls. Alternatively, create a new class exclusively for this purpose.

I started with (1), but will leave most of it as well as (2) as it is until the first part of the User Handling is integrated.

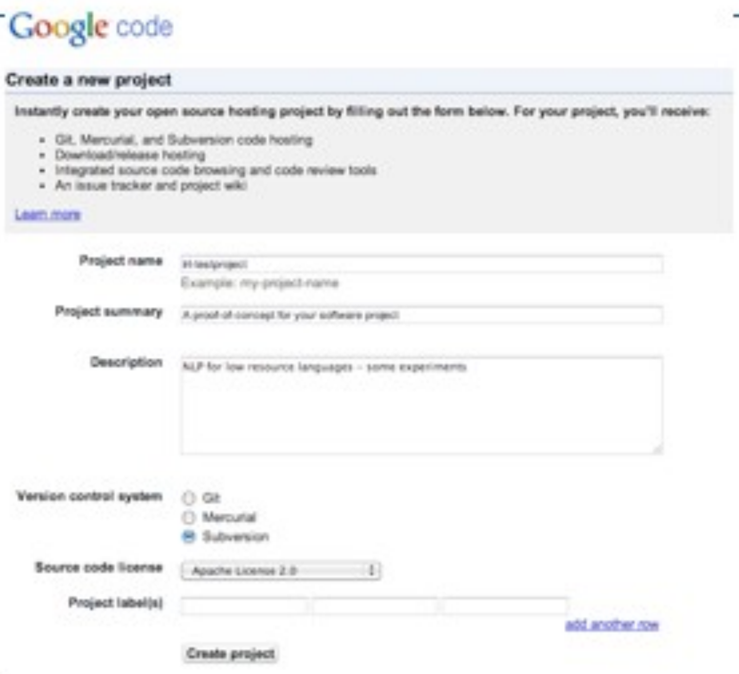
Example: Google Code

- easy to use, mostly self-explaining
- comes with 3 different repository types (SVN, Git, Mercurial)
- has an online bug / issue tracker
- http://code.google.com

Other possibilities

- there are other repository types, other bug tracking tools etc. (e.g. GitHub)
- feel free to use whatever you want
- if you want to install your own svn / trac / whatever, we can give you webspace
- there is nothing particularly extraordinary about the Google solution, but it's convenient and easy to use (and free of charge)

Create a project



Google code

Create a new project

Instantly create your open source hosting project by filling out the form below. For your project, you'll receive:

- Git, Mercurial, and Subversion code hosting
- Download/release hosting
- Integrated source code browsing and code review tools
- An issue tracker and project wiki

Learn more

Project name	nl-testproject Example: my-project-name
Project summary	A proof-of-concept for your software project
Description	NLP for low resource languages – some experiments
Version control system	○ Git ○ Mercurial [✓ Subversion]
Source code license	Apache License 2.0
Project label(s)	

The first contents

- Create a new Java / Python project on your machine
- init the repository by checking the folder into the “trunk” folder of your svn repository
- there are different ways for that, here are two:
    - in eclipse, right-click on the project and use *“team” -> “share project”*
    - on command line, check out the “trunk” folder, create the project folder in “trunk”, and use *svn add [project folder name]*

Workflow

- add new files with “svn *add*” (or the IDE menu)
- *update* before you start working
- *commit* often, but never something that does not compile or breaks other people’s code
- commit everything needed to compile, that means source code files, test files, libraries
- *never* commit compiled (*.class / *.pyc) files!
- try to always commit / update the top-level folder

Workflow

- If you commit changes, include a meaningful comment (and the issue number your commit is related to)
- make new issues in the issue tracker
  - for bugs you find
  - for new chunks of code you plan to implement
  - for other tasks (data collection, ...)
- close issues if you're finished (maybe check with another team member)

A NLP development team

programmer(s)

data specialist(s)

project manager

quality manager(s)

software architects, technical writers, different kinds of developers, ...

Coding

programmer(s)

quality manager(s)

data specialist(s)

project manager

- programming (you guessed so.)
- system architecture
- document your code
- write tests (more later...)
- further specialization possible (GUI, DB management, ...)

Data Management

data specialist(s)

programmer(s)

project manager

quality manager(s)

- look for appropriate data
- analyze: quantify, qualify
- clarify things like costs, copyright, availability,...
- make it accessible: find existing libraries, or write a parser, or define i/o format
- cleaning, preprocessing
- design output format(s)

Quality Assurance

- quality manager(s)
- programmer(s)
- data specialist(s)
- project manager

- write (higher-level) tests
- good practice: review & test *other people's code*
- assess general code quality (-> coding conventions)
- make proposals for refactoring / design changes
- make sure that the current version compiles & runs

Project Management

- project manager
- quality manager(s)
- data specialist(s)
- programmer(s)

- plan order of development
- specialist for setting goals
- motivate people to meet the goals
- surveillance of bug / issue tracking
- order pizza for late-night coding sessions
- communication (inner & outer)
- task adjustment, progress monitoring

But we're only 3 people!

- agile software development means that everybody does everything (at least w.r.t. coding)
- most of you might want to do more of one thing and less of the other
- of course there are even more specializations (-> programming), but you'll figure that out
- try to find your favorite role(s), and divide the rest mind least (or everybody)

Some words on testing

- you can use Unit-Tests, or any other framework
- each "meaningful" method should have a test
- test about *correctness*, and about *robustness*
- think about special / difficult cases for testing (null input? ...)
- now you: how do you test a method that is meant to return each string *in upper case* (CASE)?

Some words on testing

- tests get their own package
- test whether you broke your colleague's code before you submit!
- one of you can specialize in writing higher-level tests for larger components - or circulate that duty (write tests for other people's components)
- first homework: 3 methods + tests (more later)

Your development plan

- First: fix your project, create the infrastructure
- each Tuesday, you will set 3 goals for the upcoming week
- write the goals on your wiki page, + an estimation how much time you need for each of them
- in the next meeting, we will review your goals & your time estimations
- try to maintain a running demo (even if it's tiny)

Goals

- goals consist of one bug fix, or implementing some bigger new chunk, or no coding at all
- it's not a problem if you fail a goal (or need much more time) - but think about what went wrong
- learn to set better goals
- incomplete goals >>> untouched tasks!!
- we'll have group discussions with all teams on both, last week's goals, and your proposals for new goals

First Tasks

- set up the repository & issue tracker; create accounts for Alexis & Michaela
- finish your project description (~ 1 page) and put it into the repo (in an appropriate folder)
- individually: figure out where your "team expertise" is:
  - what can you do best?
  - what do you want to do most?
  - what else will you / can you do if necessary?

First Tasks

- first coding tasks: 3 methods + their tests
  - a method *splitting a string* into a list of words (=separated by space or punctuation)
  - a method that reads file#1 and prints file#2 containing words + counts from file#1 (how do you test that?)
  - a method simulating a *dummy version of your project* (e.g. for a POS tagger, the method takes one certain string and outputs pairs of words + hard-coded POS tags)

First goals

1. finish the project proposal
2. set up the repository
3. finish the coding tasks

Now: have fun. :)