Coding during a Research Project

By Christof Lutteroth

During a Computer Science research project, you will most likely create a prototypical implementation in order to demonstrate the feasibility and value of your work. This document helps you to get a good start for the coding part of your project.

Since acedemic research projects are typically small, they can benefit from the priciples of agile software development processes such as eXtreme Programming (XP). Some of the principles, and how they should be applied, are described in the following.

Requirements

All implementation efforts are guided by the requirements you are trying to satisfy. Therefore it is very important that you know what these requirements are. This needs to be discussed with your supervisor.

Start with a Design and Test Cases

To beging with, try to develop a simple and elegant design for your system, based on your requirements. This is typically done together with your supervisor. A design means that you create types (e.g. classes and interfaces) with fields and empty methods in them. The types, fileds and methods should have suitable names that reflect the requirements your are trying to satisfy. So someone who understands the requirements should also be able to understand what the different parts of your design are supposed to do.

It is good to create test cases for the methods you specify, even if the methods are empty to begin with. This will help you and others to understand how the methods are used, and what they are supposed to do. There is information in the WWW about how to write good test cases, e.g. using the JUnit tool for Java that is part of the Eclipse IDE.

Start Simple

It is a good idea to create a very simple running prototype early on in your project, and improve it incrementally. Having a running system will give you confidence and the possibility to collect data about it. In particular, do not implement optimizations in your code in early versions, unless they make your implementation easier. If your system is simple and small, this will make development much easier. Complexity should be added as the need arises. A indication of a good design is that it is simple.

Document your Code

Right from the beginning you should put comments into your code, making it easy for outsiders to understand what the different parts are for. Use a convention such as JavaDoc for Java, and make sure that every class, interface, field and method has an appropriate comment.

Refactor your Code

Refactoring means improving the design of your code safely, without changing what the code actually does. Refactoring should make your code easier to understand and easier to maintain. Most refactoring is simply common sense. Just ask yourself how you can make it easier for outsiders to understand and use your code. Examples of refactorings include:

- **Remove dead code**, i.e. code that has been commented out or is never called. This also means that old files should be deleted. Note that anything you delete can be recovered later if you are using a version-controlled repository (see below).
- **Remove unnecessary code**, i.e. old debugging code. Better use a debugger to look at variable values during execution, rather than cluttering the code with `print`'s.
- **Change names** (of types, methods, variable) so that they are more meaningful

IDEs such Eclipse have functions for automatic refactoring such as renaming, which you should get familiar with.

Get your Code Reviewed

At some stage you should someone else look at your code. Sit together and go throught the code type by type, method by method. You should explain the code, and ask the other person for critical feedback. If the other person has difficulties understanding your code, this can be a sign of design problems. Do this also with your supervisor.

Use your Repository

After completing a significant change, you should commit the change to your repository. Once you have reached a stable state where your tests pass , you should push your changes to the central repository. This helps to prevent data loss and allows others to use your work. Do not push changes tha introduce errors into the system, unless you are working on your own branch.

All your commits should have a brief and informative log message, which describes what you have done and why. This will help other developers to work with you more easily as a team.