Refactoring C++, Ruby, etc for Decremental Development
Regaining Simplicity in Software

Prof. Peter Sommerlad
HSR - Hochschule für Technik Rapperswil
IFS Institute for Software
Oberseestraße 10, CH-8640 Rapperswil
peter.sommerlad@hsr.ch
http://ifs.hsr.ch
http://wiki.hsr.ch/PeterSommerlad
Work Areas
- Refactoring Tools (C++, Ruby, Python, ...) for Eclipse
- Decremental Development (make SW 10% its size!)
- Modern Software Engineering
- Patterns
  - Pattern-oriented Software Architecture (POSA)
  - Security Patterns

Background
- Diplom-Informatiker Univ. Frankfurt/M
- Siemens Corporate Research Munich
- itopia corporate information technology, Zurich (Partner)
- Professor for Software HSR Rapperswil, Head Institute for Software

Credo:

- People create Software
  - communication
  - feedback
  - courage

- Experience through Practice
  - programming is a trade
  - Patterns encapsulate practical experience

- Pragmatic Programming
  - test-driven development
  - automated development
  - Simplicity: fight complexity
Inside every large program, there is a small program trying to get out.

There are two ways of constructing a software design:

- one way is to make it so simple that there are obviously no deficiencies, and
- the other way is to make it so complicated that there are no obvious deficiencies.

The first method is far more difficult.
Decremental Development

- **Reduce software size TO 10%**
  - while keeping required functionality
  - while improving its quality
  - while improving its design

- **one means: Refactoring**
  - requires (Test-) Automation
Why we need Decremental Development

- **Problems solved by Software increase**
  - more problems
  - larger & more complex
- **“Good-enough” quality often isn’t**
  - when deployed (Beta-Release)
  - while maintained (updates breaking stuff)
- **Useful Software is used longer than intended**
  - pro-active maintenance often neglected
  - repeated bug-hunt-fix-patch deteriorates quality
  - need tools and methods to sustain software
Research Proposal

Decremental Development

- Create Tools for automated Refactoring
  - for languages lacking support, i.e.,
  - C/C++, PL/I, Ada, COBOL(?)
  - javascript, PHP, groovy (student projects started)

- Develop new Approaches for higher-level Software Simplification
  - beyond Refactoring
  - i.e., detecting potential for simplification
  - i.e., automatic unit test generation for C++ code

- Increase Valuation of Simplicity
  - as a software design goal
  - articles, presentations, case studies
Existing Results
Eclipse Plug-Ins

- Ruby Refactoring for RDT and Eclipse DLTK
  - integrated into official release (RDT)

- Python Refactoring for PyDev
  - student project, results integrated into release

- C++ Refactoring
  - student project for CDT 2.1
  - student project for CDT 3.0
  - CDT’s Europa Release has minimal infrastructure
    - which was provided by IFS
    - C++ Refactoring plug-in as add-on available now!

- C++ Unit Testing plug-in
  - CUTE C++ Unit Testing Easier
Refactoring Plug-ins specialties

- **Comment-augmented Syntax Tree**
  - regenerate code including comments from AST
  - keep comments attached to symbols across refactoring, i.e. move method, based on heuristics

- **“virtual” Preprocessing of C++ code (partially)**
  - keep relation to original source within AST

- **Duplication detection based on AST**
  - structural equivalent code could be replaced

- **Refactoring Preview**
  - diff-viewer with syntax coloring
  - selectable application of refactoring
Refactoring plug-ins

Testing

I teach automated testing and make my students implement a build server with continuous builds and testing.

- I guess, Eclipse projects could benefit from that.
- setting CruiseControl for plug-in testing is a bit of “magic”

(Plug-in) tests requiring the workspace tend to run (too) slow

- but still better than manual testing

We created a text-based test description language, plus a test-editor plug-in

- used for almost all our refactoring plug-ins
- much easier to specify language-processing tests
Testing Refactoring
an example

```c++
// method at end, without parameters and void return value
// #org.eclipse.cdt.hsrefactoring.refactoring.tests.implementmethod.ImplementMethodRefactoringTest
//@A.h
#ifndef A_H
#define A_H

class A
{
 public:
  A();
  // $void foo();$//
};
#endif /*A_H*/
//@A.cpp
#include "A.h"

A::A()
{
}

 //=
#include "A.h"

A::A()
{
}

void A::foo()
{
}

// method at beginning, without parameters, void return value and const
// #org.eclipse.cdt.hsrefactoring.refactoring.tests.implementmethod.ImplementMethodRefactoringTest
```
Challenges in Development Logistics

- **Funding**
  - today “free” student work and Swiss public funding

- **Keeping plug-ins up-to-date with platform and depending projects**
  - i.e. RDT (Aptana) failed to maintain our Refactoring contribution to RDT (even with automated test cases)
  - each year (or multiple times a year) because of moving platform/target

- **Contribution to “official” Eclipse projects as a university is hard**
  - time-lag for legal aspects, long-term funding
CDT’s internal code quality
- many coding/design practices that can be improved (simplified!), i.e.,
  - duplication
  - C-style Java code
- messy/buggy C/C++ parser and AST code
  - C++ parser fails on C++ std-library headers
  - Refactoring requires good AST

Minimal-invasive implementation for comment-insertion into AST uses dynamic proxy objects
- no cost/change to CDT, but implementation very hard to get right, to test and debug.
Current projects

Student projects:
- extending Python Refactoring
  - for PyDev, might use DLTKs type-inference engine
- PHP Refactoring
- Javascript Refactoring
- groovy Refactoring

Institute projects (planned, ongoing):
- Ruby Refactoring port to DLTK-Ruby
  - using DLTKs type-inference engine
- more C++ Refactorings and code generators
  - including CUTE support
- could provide vendor-specific plug-ins
Conclusion/Summary

- Think, where you can simplify your code
- Refactoring automation for non-java languages can be implemented and is useful
- Eclipse plug-ins can be automatically tested in a continuous build process and you benefit from it
- We would appreciate funding for our ongoing work
  - and can provide vendor-(i.e. Nokia, QNX, IBM,...) specific Refactoring features (for money :-)
- CDT’s code base should be a target for Decremental Development - Refactoring
  - continuous build&test server?
Questions? Discussion?

Going forward...

Sorry for the begging for money but sponsoring is appreciated!