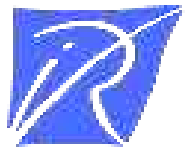
Howto avoid bugs
(Refactoring)

Mathieu Lacage
<lacage@sophia.inria.fr>



INRIA
SOPHIA ANTIPOLIS

DREAM: http://www-sop.inria.fr/dream

The problem(s)

- Software Quality. i.e., Bugs

- Cost of Software Evolution:
  - bugs,
  - new features.

Refactoring is a solution

- To decrease the number of Bugs.
- To decrease the cost of software evolution.

Outline

- Requirements to Refactoring
- What Refactoring is

Refactoring is not a silver bullet

- The most important factor for the success of your project is your Team, not the techniques you force upon them.
- Refactoring requires everyone to work together to be really useful.

Requirements to refactoring

- Spirit: If it is broken, *you* must fix it.
  - every developer must feel responsible for everything
- Use the right tools:
  - version management system (CVS)
  - ChangeLog
  - regression tests
- Process:
  - release schedule
  - roadmap

What is refactoring ?

- Refactoring is a method which can be used to modify code to:
  - fix a bug
  - add a new feature
  - remove a feature
  - cleanup code

What is refactoring ? (2)

- Each modification of the code:
  - it must be easy to realize
  - it must be easy to understand
  - it must be easy to verify through code inspection if it does what it should do
  - it must be easy to verify through regression tests if it does not break anything

What is refactoring ? (3)

- If you have any doubt about any such small modification, throw it away:
  - because it is easy to re-do: it is usually easier to re-do than to debug
  - because you can revert to the previous state: you use a version management system, don't you ?

An example

- You can layout text in a rectangle box:

```
class TextLayout {
  int getHeight (String text, int width);
}
```

- You want to layout text in an arbitrary shape:

```
class TextLayout {
  int getVerticalExtent (String text,
     Shape shape);
}
```

An example (2)

- There are many ways to add this feature:
  1. break everything, trying to make it work again. Weeks later, finally get it to half-work.
  2. break the work in small incremental changes, perform each step, one after the other, verify that the system still works after each step.
- Refactoring is about solution 2.

An example (3)

- Create a new Shape interface:
  1. create a RectangleShape implementation
  2. make sure it builds, runs
  3. add regression test for this Shape object
  4. checkin

An example (4)

- Add the getVerticalExtents method
  1. implement it at least if Shape == RectangleShape
  2. make sure it builds, runs
  3. copy the regression test for getHeight and modify it to work for getVerticalExtents
  4. make sure this new regression test passes
  5. checkin

An example (5)

- Implement getHeight by using getVerticalExtent
  1. make sure the code still builds, runs
  2. make sure it passes regression tests
  3. checkin

An example (6)

- Remove getHeight
  1. change every user of getHeight to create a RectangleShape object and call getVerticalExtent
  2. make sure the code still builds, runs, passes regression tests
  3. checkin

Example summary

- The version management system is mandatory.
- Regression tests are very very important.
- Compiler warnings are very very important:
  - always use all the compiler warnings
  - make all warnings errors to make sure you cannot compile the code if errors are left

What if it is an old codebase ?

- It compiles with lots of warnings:
  you must fix all compiler warnings first once and for all
- There is no version management system:
  use one. NOW !
- It has no regression tests:
  you can write regression tests on a needed-basis

A second nature

- This process must become a second nature: whenever you see something wrong, fix it
  - variables with meaningless names: rename them now
  - functions/methods with meaningless names: rename them now
  - redundant/dead/unused code: remove it now

➔This is an incremental process

Tools

- There exist a lot of tools to help you perform refactoring:
  - Java: eclipse has a lot of automatic refactoring tools and a powerful search tool
  - C++: Visual C++ has a powerful search tool

References

- The bible, a *must read*: “Refactoring: Improving The Design Of Existing Code” by Martin Fowler.
- “Design Patterns: Elements of Reusable Object-Oriented Software” by “The Gang of Four”.