Change Tolerant Code

Emmanuel CHENU
emmanuel.chenu@fr.thalesgroup.com
http://emmanuelchenu.blogspot.com

In most of the books I've read concerning software development, we are told to ***accept change.*** *The authors share several practices to write* ***change tolerant code****. This article is a compilation of these various practices and of personal experience.*

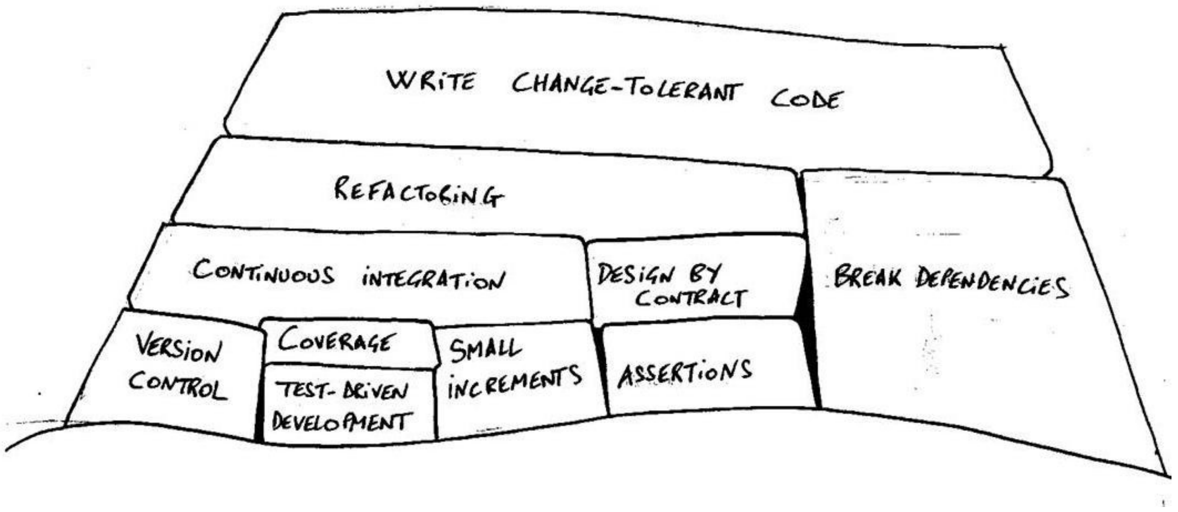
Use a version-control system. With such a tool, you can roll-back from a change which brought regressions into the code.
Work in short iterations and integrate continuously. Always have an operational application, ready to be changed.
Design-by-contract with assertions. The assertions will stop at runtime when a contract is broken by a side-effect of a change.
Test-driven development. The automated self-checking tests will detect any regression brought into the code by a change.
Measure code coverage by tests. You don't a change to affect a non-tested part of the code. Identify the lines of code never exercized by tests, and add some test cases.
Apply the Single Responsiblity Principle (SRP). As Robert Martin says in **AGILE SOFTWARE DEVELOPMENT**:

> *A class should have only one reason to change.*

Apply the Common Closure Principle. As Robert Martin says in **AGILE SOFTWARE DEVELOPMENT**:

> *The classes in a package should be closed together against the same kind of changes. A change that affects a closed package affects all the classes in that package and no other packages.*

Use design-patterns. Many design-patterns organize your design to anticipate change.
Use layering, information hiding and encapsulation. This limits coupling by regrouping and isolating cohesive code which may be affected by change, therefore limiting the impact of change into the code.
Simplicity. It's just easier to change simple code.
Write less code. It's just easier when there is less code to change.
No repetition. Don't do the same change twice.
Stop-the-line. Detect regressions brought by change as soon as possible. Then, stop work, correct the problem and resume work. A continuous-integration tool which detects changes, builds the application, runs the tests and notifies the development team when a failures occurs can really help.
Refactor. Refactor the code and the tests to keep them healthy:eliminate complexity and repetition. The code will be easier to change.


WRITE CHANGE-TOLERANT CODE
REFACTORING
CONTINUOUS INTEGRATION
DESIGN BY CONTRACT
BREAK DEPENDENCIES
VERSION CONTROL
COVERAGE
TEST-DRIVEN DEVELOPMENT
SMALL INCREMENTS
ASSERTIONS