

dotEASY

The “dotEASY” project

Abstract: *“dotEASY” is a Visual Studio .Net Add-in that performs “advices” over C# source code. Its objective is to improve software quality. The configuration and programming of the “advices” is invisible to the developer, tool’s final user, who only requests for code evaluation. A new “advice” can be created defining metrics, thresholds and optionally programming its validation “DLL” and execution “DLL”. The export and import capabilities allows one person to create an “advice”, which can be configured and used by many other people.*

Introduction

The “dotEASY” project was initially conceived as a requirement for the graduation to the Software Engineering degree, ORT University, Uruguay. For this reason, the project’s team is formed by five students, all of them coursing the last year of Software Engineering.

As an initial objective for the project, we decided to make a research of the state-of-the-art in many subjects, such as testing, refactoring, AOP. Using all of our resources for this research, we finally found an interesting matter by crossing some of these subjects, like metrics with quality attributes, and metrics with patterns.

Finally we found the main objective for the project: the development of a tool that uses the source code to provide hints and advices to improve software’s quality.

Motivation

After the study of different subjects based on the Object Oriented Design, as metrics, refactoring, patterns, and different tools, we found out that we could mix these matters to develop an interesting product that improves the quality of OO designed source code.

The product should be used by developers who know about metrics and refactoring, and also by those who don’t. How is this? To provide the usability of this tool to all the OO software developers, we suggest that the metrics taken from the code thresholds, should not be configured by the final user of the tool, as they could be configured and saved by some other person, probably the responsible for the software quality.

On the other hand, we have the need to develop the refactoring function needed to improve the software quality. How can we avoid programming a change in the code structure more than once?. We, or the person that configures the tool for his own software project, could import it from other source. Someone could program a change to improve the software quality, and it could be re-used, through import an personalization.

Description

What is “dotEASY”?

“dotEASY” is a code-smell finder, which evaluates a code, searching for violations to the configured thresholds and programmed validations. If it finds any violation, it provides an advice, and offers the possibility (if it is programmed) to make an automatic change to the code, improving its quality towards the defined standards.

How does it work?

How does “dotEASY” evaluate a code? The evaluation of the code is made through configurable metrics and the execution of validation libraries (DLL’s).

A developer, an architect or the responsible for the software quality, can define and configure the evaluation parameters and the “advices” that the tool should give.

With these parameters configured, the software developer (final user) can evaluate his code when he thinks it is adequate. The evaluation could be executed in different levels. A whole

package (namespace) could be evaluated, or a class, or only a single method.

What is an "Advice"?

An "advice" is quality violation alert that let us know that a portion of code could be improved.

To explain what is and "advice" we need to define the concept of "code smell".

A "code smell" is composed by a set of metrics thresholds, and optionally a validation library. The metrics thresholds allows customization.

The "code smell" evaluation is made through the comparison of the real metrics (calculated from the code) with the pre-configured thresholds, and the approbation of it's validation library.

Then, an "advice" is associated with a set of "code smells", and it is validated through the validation of any of them.

An "advice" can be associated to an extra validation library, and to an execution library.

If an "advice" is validated, the user gets a notification, and if it has an execution library, the user is asked for the automated execution.

An "Advice" example

Name: Reduce method parameters number using "*Replace parameter with Method*"

Metric: Number of method parameters
Threshold: > 4

Validation DLL: A parameter could be obtained by a class method.

Execution DLL:

1. Create a method temporal variable with the parameter 's name and type
2. Assign the method of the class to the variable
3. Remove the parameter
4. Update the reference to the method

Details

Implementation

The initial implementation will be developed as a Visual Studio.Net Add-in, and it will work on C# source code.

Extension

"dotEASY", as can be inferred from previous description, is a Framework. The framework

will be extended through the creation of new "advices". The "advices" will be formed by metrics configuration, a validation library, and an execution library. These libraries will be extensions of the "dotEASY" Framework, and for each of them there is going to be a defined subset of interfaces.

The "advices", as the libraries, could be imported and exported, giving ways to perform only one "advice" programming, sharing this "advice" with thousands of users.

The configuration allows that any user could use an "advice" in a customized way for each one of his projects.

Refactoring execution

After an "advice" validation, there would be, if there is an execution library associated, a conditional execution of a source code change. This change could be undone after its execution.

"dotEASY" will also provide a refactoring Wizard, that will allow the final user to perform simple refactoring changes to his source code structure. After an execution of a change using this Wizard the user will be able to undo the lasts changes performed.

Contact

For any questions, suggestions, hints, ideas, advices or "advices" please contact us.

Pablo Bertón. pberton@adinet.com.uy
Juan Suárez. jesuarez@adinet.com.uy

Your feedback will be kindly appreciated.

References

Pressman, Roger: 1998, Ingeniería del Software, Cuarta Edición, Madrid.

Rosenberg, Linda – Hyatt, Larry: Applying and Interpreting OO Metrics, April de 1998, http://satc.gsfc.nasa.gov/support/STC_APR98/apply_oo/apply_oo.html. (ooply.pdf)

Robert Martin, 1994 OO Design Quality Metrics, oodmetrc.pdf.

SATC (Software Assurance Technology Center) , june 1995, SOFTWARE QUALITY METRICS For Object Oriented System Environments Ootech.pdf .

Together 6.0, Together Soft.

Basili, Briand, Melo, A Validation of OO Metrics as Quality Indicators, 1996

SATC (Software Assurance Technology Center) , august 2002, Object Oriented Metric Definitions.

<http://satc.gsfc.nasa.gov/metrics/codemetrics/oo/definitions/index.html>

William Wake – October 2001 - Refactoring Workbook – Measured Smells - [William.Wake@acm.org](http://www.williamwake.com). - Refactoring Workbook (10-11-01) – Chapter 3

William Wake – October 2001 - Refactoring Workbook – Measured Smells - Xplorations - [William.Wake@acm.org](http://www.williamwake.com). - Refactoring Workbook (10-11-01) – Chapter 5

Roberts, Brant, Jonson. A Refactoring Tool for Smalltalk. Supporting Software Evolution with Automated Refactorings.htm.

Martin Fowler. Refactoring Home. <http://www.refactoring.com/index.html>.

IntelliJ .IntelliJ IDEA. <http://www.intellij.com/idea/>.

JRefactory (Java). http://emw.inf.tu-dresden.de/de/pdai/Forschung/refactoring/refactoring_html/node71.html#jrefactory.

RefactorIT. RefactorIT. <http://www.refactorit.com/>.

IntegriSoft. iComprehend. <http://www.integrisoft.com/isight.htm>

William Wake – October 2001 - Refactoring Workbook – Measured Smells – Xplorations - [William.Wake@acm.org](http://www.williamwake.com). - Refactoring Workbook (10-11-01) – Chapter 7

SATC Code Metrics – Recommended Thresholds for OO Languages

William F. Opdyke – 1992 – Refactoring object oriented frameworks – Thesis degree of doctor of philosophy in computer science – University of Illinois at Urbana-Champaign – 1992

Brian Huston – 2001 – The effects of design pattern application on metrics scores -Systems Engineering faculty, Southampton Institute, East Park Terrace, Southampton UK - The journal of systems and software no. 58 (2001) pags. 261-269