ModularityCheck

A Tool for Assessing Modularity using Co-Change Clusters

Luciana Silva, Daniel Félix, Marco Tulio Valente, and Marcelo Maia

http://aserg.labsoft.dcc.ufmg.br
Motivation

What is a well-modularized system?
On the Criteria To Be Used in Decomposing Systems into Modules

D.L. Parnas
Carnegie-Mellon University

This paper discusses modularization as a mechanism for improving the flexibility and comprehensibility of a system while allowing the shortening of its development time. The effectiveness of a "modularization" is dependent upon the criteria used in dividing the system into modules. A system design problem is presented and both a conventional and unconventional decomposition are described. It is shown that the unconventional decompositions have distinct advantages for the goals outlined. The criteria used in arriving at the decompositions are discussed. The unconventional decomposition, if implemented with the conventional assumption that a module consists of one or more subroutines, will be less efficient in most cases. An alternative approach to implementation which does not have this effect is sketched.

Key Words and Phrases: software, modules, modularity, software engineering, KWIC index, software design

Introduction

A lucid statement of the philosophy of modular programming can be found in a 1970 textbook on the design of system programs by Gouthier and Pont [1, ¶10.23], which we quote below:¹

A well-defined segmentation of the project effort ensures system modularity. Each task forms a separate, distinct program module. At implementation time each module and its inputs and outputs are well-defined, there is no confusion in the intended interface with other system modules. At checkout time the integrity of the module is tested independently; there are few scheduling problems in synchronizing the completion of several tasks before checkout can begin. Finally, the system is maintained in modular fashion; system errors and deficiencies can be traced to specific system modules, thus limiting the scope of detailed error searching.

Usually nothing is said about the criteria to be used in dividing the system into modules. This paper will discuss that issue and, by means of examples, suggest
Conclusion

We have tried to demonstrate by these examples that it is almost always incorrect to begin the decomposition of a system into modules on the basis of a flowchart. We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others. Since, in most cases, design decisions transcend time of execution, modules will not correspond to steps in the processing. To
ModularityCheck

• A tool for **assessing** and **exploring** modularity

• Using **change data** (commits)
How it Works

• **Step #1**: Co-Change Graph
Co-change Graphs

- Geronimo
  - Density = 0.01
Co-change Graphs

• Geronimo
  – Density = 0.01 ➔ sparse
Co-change Graphs

- Geronimo
  - Density = 0.01 \(\Rightarrow\) sparse, but **dense subgraphs**
Co-change Clustering

- **Step #2**: graph clustering algorithm
  - Chamaleon
  - Designed for sparse graphs

- Co-change clusters:
  - Groups of classes that frequently change together

- Geronimo:
  - 21 co-change clusters
  - Density: 0.80 / cluster
Step #3: Visualization using distribution maps
Co-change Clusters Patterns

• **Step #4**: Classification in patterns:
  – Well-encapsulated
  – Partially encapsulated
  – Crosscutting
Well-encapsulated clusters
Well-encapsulated clusters
Well-encapsulated clusters

Nice modules!
Well-encapsulated clusters
Well-encapsulated clusters
Well-encapsulated clusters

Package Split refactoring?
Partially encapsulated clusters
Partially encapsulated clusters
Partially encapsulated clusters

Move Class refactoring?
Crosscutting Clusters
Crosscutting Clusters
Crosscutting Clusters

Crosscutting concern? Modularization Flaw? Major refactoring? Re-architecting?
serg.labsoft.dcc.ufmg.br/modularitycheck

Live Demo
Tomorrow – 11:00 AM