CMSE-520 Problem session 27.12.2024

Topics covered:  1) [Lecture 3. Reengineering](https://staff.emu.edu.tr/alexanderchefranov/Documents/CMSE520/CMSE520%20Fall%202024/Lecture%203.%20Reengineering%20.docx) (Sections 5 and 6); 2) [Lecture 4. Impact Analysis](https://staff.emu.edu.tr/alexanderchefranov/Documents/CMSE520/CMSE520%20Fall%202024/Lecture%204.%20Impact%20Analysis.docx); and 3) [Lecture 5 Refactoring](https://staff.emu.edu.tr/alexanderchefranov/Documents/CMSE520/CMSE520%20Fall%202024/Lecture%205%20Refactoring-1.docx)

Contents

[1. Refactoring 2](#_Toc186126426)

[1.1. Restructuring technique 2](#_Toc186126427)

[1.1.1. Elimination-of-goto Approach from code 2](#_Toc186126428)

[1.1.1.1. Forward elimination 2](#_Toc186126429)

[1.1.1.2. Backward elimination 2](#_Toc186126430)

[1.1.1.3. Rewrite 1.1.1.2 using just exit (not exit label) 3](#_Toc186126431)

[1.1.2. Localization and Information Hiding Approach 3](#_Toc186126432)

[***1.1.2.*** System Sandwich Approach 5](#_Toc186126433)

[***1.1.3.*** Clustering Approach 5](#_Toc186126434)

[**1.1.3.1.** https://en.wikipedia.org/wiki/K-means\_clustering 5](#_Toc186126435)

[**1.1.3.2.** https://en.wikipedia.org/wiki/Hierarchical\_clustering 5](#_Toc186126436)

[***1.1.4.*** Program Slicing Approach 5](#_Toc186126437)

[***1.1.5.*** Approaches not involving code changes 5](#_Toc186126438)

[***1.2.*** Factors Influencing Software Structure 5](#_Toc186126439)

[1.3. Examples of refactorings 5](#_Toc186126440)

[1.3.1. Substitute Algorithm 5](#_Toc186126441)

[1.3.2. Replace Parameter with Method 5](#_Toc186126442)

[1.3.3. Push-down Method 5](#_Toc186126443)

[1.3.4. Parameterize Methods 5](#_Toc186126444)

[1.4. Refactorings using metrics 5](#_Toc186126445)

[1.4.1. Cohesion-based refactoring 5](#_Toc186126446)

[1.4.2. Coupling-based refactoring 5](#_Toc186126447)

[1.5. Rule-based refactoring 5](#_Toc186126448)

[1.6. Assertions 5](#_Toc186126449)

[1.7. Refactorings of the class diagram of a LAN simulator 5](#_Toc186126450)

[1.8. Code smells 5](#_Toc186126451)

[1.8.1. Duplicate Code 5](#_Toc186126452)

[1.8.2. Long Parameter List 5](#_Toc186126453)

[1.8.3. Long Methods: 5](#_Toc186126454)

[1.8.4. Message Chain 5](#_Toc186126455)

[1.8.5. Comments 5](#_Toc186126456)

[2. Impact analysis 6](#_Toc186126457)

[3. Reengineering 6](#_Toc186126458)

# Refactoring

##  Restructuring technique

### Elimination-of-goto Approach from code

@article{Ramshaw1988EliminatingGT,

 title={Eliminating go to's while preserving program structure},

 author={Lyle Ramshaw},

 journal={J. ACM},

 year={1988},

 volume={35},

 pages={893-920},

 url={https://api.semanticscholar.org/CorpusID:31001665}]

#### Forward elimination

|  |  |
| --- | --- |
| actionl;action2;if test3 then go to G endthen fi;action4;if test5 then go to G endthen fi;action6;G: action7; | actionl;action2;repeat if test3 then exit L endthen fi; action4; if test5 then exit L endthen fi; action6; exit;endloop : Laction7; |

FIG. 2. An example of the Forward Elimination Rule.

#### 1.1.1.2. Backward elimination

|  |  |
| --- | --- |
| actionlG: action2;if tests3 then go to G endthen fi;action4;if test5 then go to G endthen fi;action6 ;action7; | actionl;repeat action2; Repeat if tests then exit L endthen fi; action4; if test5 then exit L endthen fi; exit m; endloop : Lendloop : m;action6;action7; |

FIG. 3. An example of the Backward Elimination Rule.

#### 1.1.1.3. Rewrite 1.1.1.2 using just exit (not exit label)

### 1.1.2. Localization and Information Hiding Approach

From W. C. Chu and S. Patel, "Software restructuring by enforcing localization and information hiding," Proceedings Conference on Software Maintenance 1992, Orlando, FL, USA, 1992, pp. 165-172, doi: 10.1109/ICSM.1992.242546.

******

******

******







### System Sandwich Approach

### Clustering Approach

#### <https://en.wikipedia.org/wiki/K-means_clustering>

#### <https://en.wikipedia.org/wiki/Hierarchical_clustering>

### Program Slicing Approach

### Approaches not involving code changes

## Factors Influencing Software Structure

## Examples of refactorings

### Substitute Algorithm

### Replace Parameter with Method

### Push-down Method

### Parameterize Methods

## Refactorings using metrics

###  Cohesion-based refactoring

### Coupling-based refactoring

## Rule-based refactoring

## Assertions

## Refactorings of the class diagram of a LAN simulator

## Code smells

### Duplicate Code

### Long Parameter List

### Long Methods:

### Message Chain

### Comments

From **Refactoring: Improving the Design of Existing Code**

by Martin Fowler, Kent Beck (Contributor), John Brant (Contributor), William

Opdyke, don Roberts

<https://www.oreilly.com/library/view/refactoring-improving-the/9780134757681/>

Don't worry, we aren't saying that people shouldn't write comments. In our olfactory analogy, comments aren't a bad smell; indeed they are a sweet smell. The reason we mention comments here is that comments often are used as a deodorant. It's surprising how often you look at thickly commented code and notice that the comments are there because the code is bad.

Comments lead us to bad code that has all the rotten whiffs we've discussed in the rest of this chapter. Our first action is to remove the bad smells by refactoring. When we're finished, we often find that the comments are superfluous.

If you need a comment to explain what a block of code does, try Extract Method. If the method is already extracted but you still need a comment to explain what it does, use Rename Method.

If you need to state some rules about the required state of the system, use Introduce Assertion.

**Tip**

*When you feel the need to write a comment, first try to refactor the code so that any comment becomes superfluous.*

A good time to use a comment is when you don't know what to do. In addition to describing what is going on, comments can indicate areas in which you aren't sure. A comment is a good place to say *why* you did something. This kind of information helps future modifiers, especially forgetful ones.

# Impact analysis

* 1. Starting impact set, candidate impact set, discovered impact set, actual impact set, false positive impact set, recall, precision, adequacy, inclusiveness, effectiveness, ripple sensitivity, amplification, sharpness, change rate, adherence, S-ratio
	2. Identifying SIS
	3. Traceability graph
	4. SLO (software life-cycle objects) dependency graph, connectivity matrix, reachability matrix, distance matrix
	5. Program dependency graph, reaching definitions, static and dynamic program slices
	6. Intra-module and inter-module change propagation. Matrices Vm, Zm, Xm, ripple effect calculation
	7. Recall and precision calculation for change propagation heuristic
	8. Heuristics for change propagation

# Reengineering

* 1. Decompilation versus reverse engineering
	2. Data reverse engineering
	3. Reverse engineering tools