# Software Quality Assurance

## CS 428

Dr. Salma Hamza

salma.hamza@medtech.tn
http://www.smu.tn/medtech

April, 2025

# Table of contents



MedTech
MEDITERRANEAN INSTITUTE OF TECHNOLOGY
SOUTH MEDITERRANEAN UNIVERSITY

# Static Analysis

# Static Analysis



## Definition

Static analysis methods where the software artifact is examined manually, or with a set of tools, but not executed

- Document Review (manual)
  - Different types
- Static Code Analysis (automatic)
  - Metrics

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Reviews - Terminology



- **Static testing** - testing without software execution
- **Review** - meeting to evaluate software artifact
- **Inspection** - formally defined review
- **Walkthrough** - author guided review

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Reviews Complement Testing



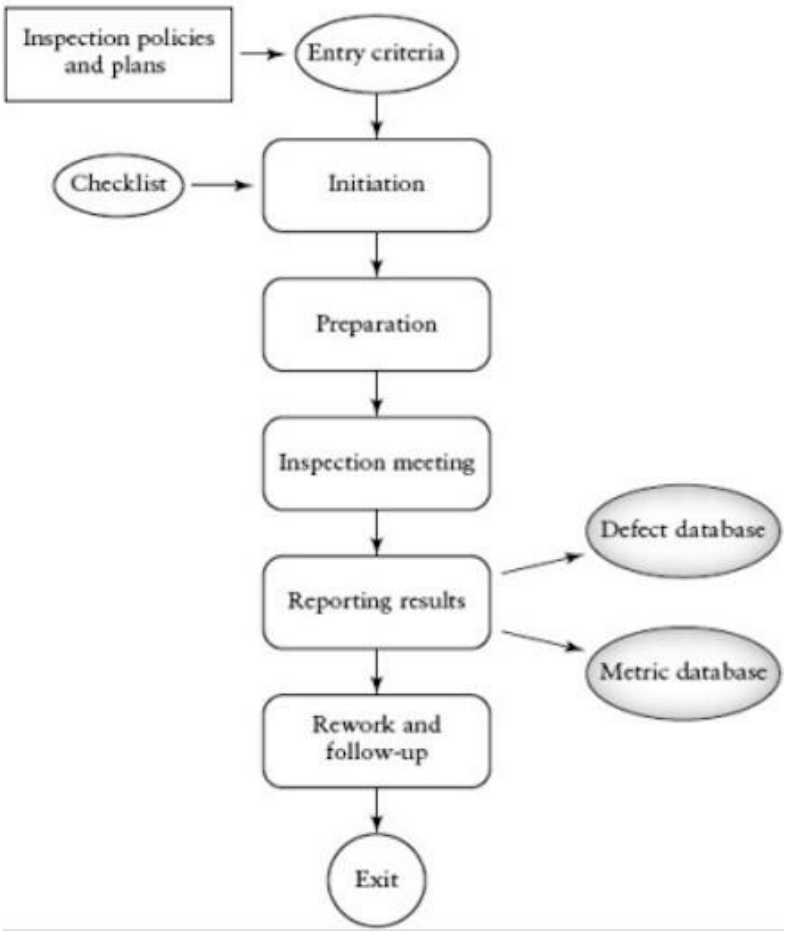



Reviews complement testing

# Inspection Process

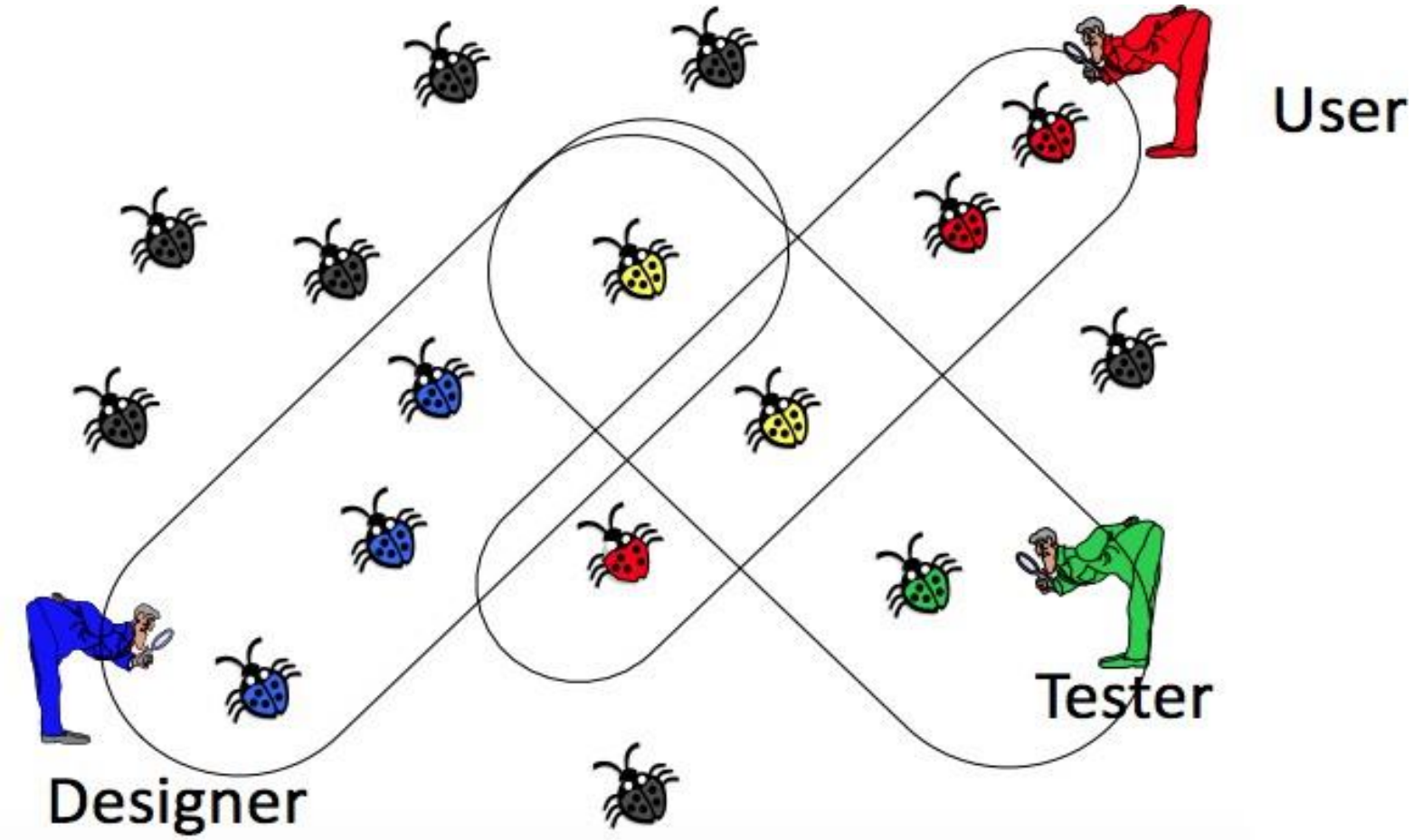


- Origin: Michael Fagan (IBM, early 1970's)
- Approach: Checklist-based
- Phases
  - Overview, Preparation, Meeting, Rework, Follow-up
  - Fault searching at meeting! - Synergy
- Roles
  - Author (designer), reader (coder), tester, moderator
- Classification
  - Major and minor



# Perspective-based Reading






MedTech
MEDITERRANEAN
INSTITUTE OF TECHNOLOGY
SOUTH MEDITERRANEAN UNIVERSITY

# Test Planning

# Test Planning

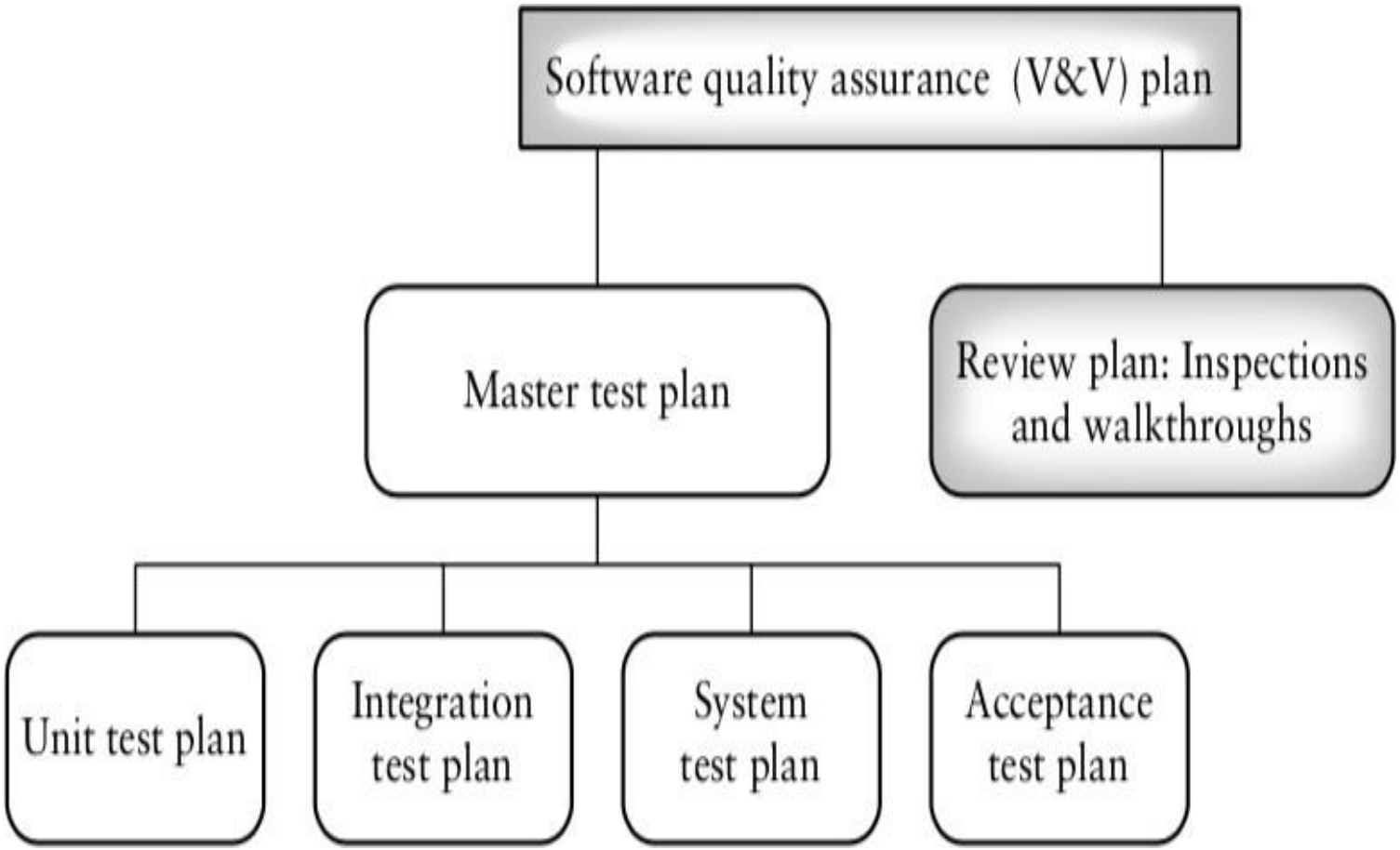


- Objectives
- What to test
- Who will test
- When to test
- How to test
- When to stop

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Hierarchy of Test Plans





# Test Plan Component



| Test Plan Components |
|---|
| 1. Test plan identifier |
| 2. Introduction |
| 3. Items to be tested |
| 4. Features to be tested |
| 5. Approach |
| 6. Pass/fail criteria |
| 7. Suspension and resumption criteria |
| 8. Test deliverables |
| 9. Testing Tasks |
| 10. Test environment |
| 11. Responsibilities |
| 12. Staffing and training needs |
| 13. Scheduling |
| 14. Risks and contingencies |
| 15. Testing costs |
| 16. Approvals |

**IEEE Std 829-1983**

***IEEE Standard for Software Test Documentation***

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Software Quality Assurance

# SQA Definition



## Software Quality Assurance – The IEEE definition

1. A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.

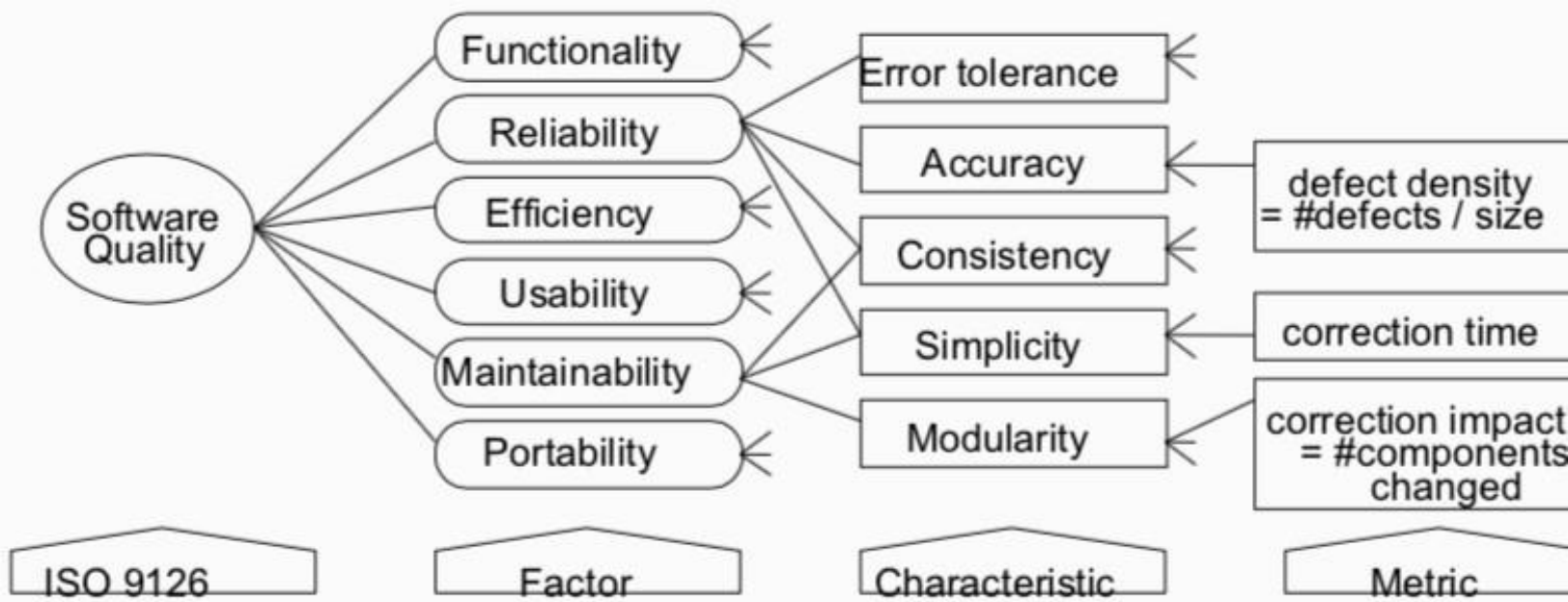
2. A set of activities designed to evaluate the process by which the products are developed or manufactured. Contrast with quality control.

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Quantitative Quality Model



## Quality according to ISO 9126 standard

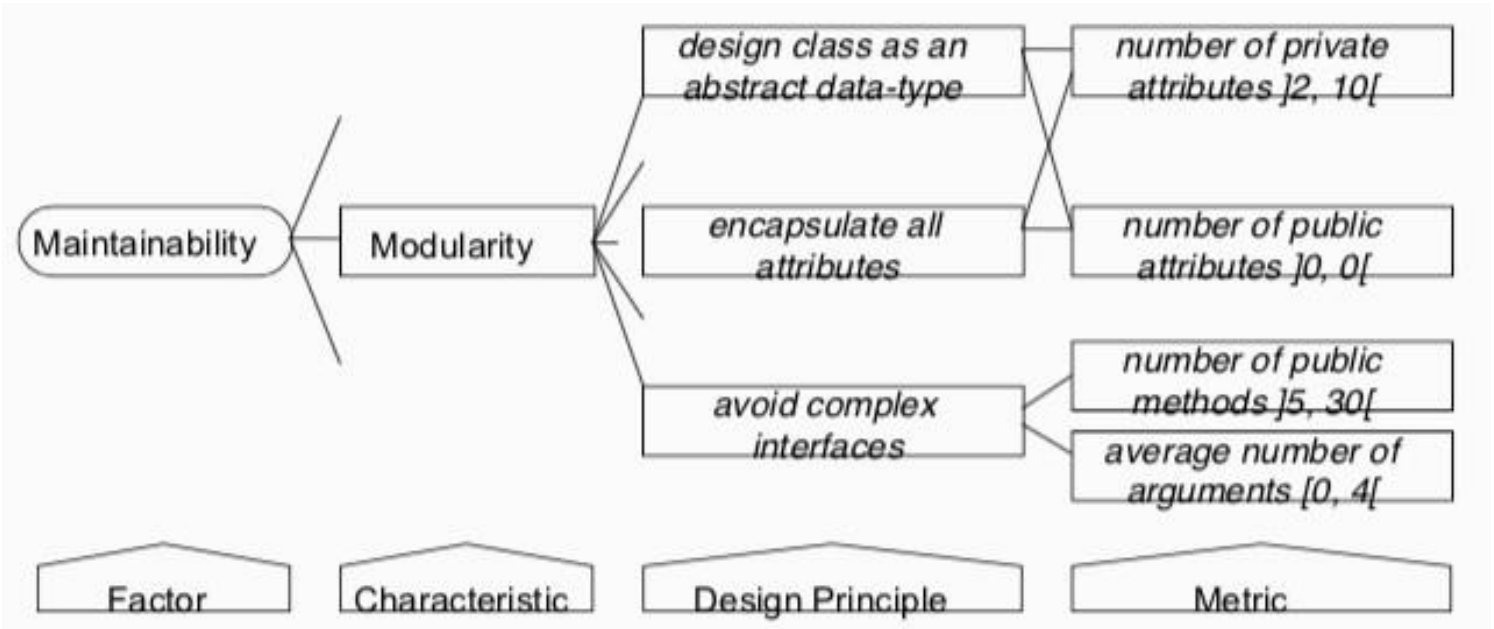


# "Define your own" Quality Model



## Define the quality model with the development team

- Team chooses the characteristics, design principles, metrics … and the thresholds



# Software Metrics

# Motivations



- How big is the program?
  - Huge!!
- How close are you to finishing?
  - We are almost there!!
- Can you, as a manager, make any useful decisions from such subjective information?
  - Need information like, cost, effort, size of project.

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Metrics



- Quantifiable measures that could be used to measure characteristics of a software system or the software development process
- Required in all phases
- Required for effective management
- Managers need quantifiable information, and not subjective information
  - Subjective information goes against the fundamental goal of engineering

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Direct and Indirect Measures/Metrics



- Direct Measures
  - Measured directly in terms of the observed aNribute (usually by counting)
    - Length of source-code, Duration of process, Number of defects discovered
- Indirect Measures
  - Calculated from other direct and indirect measures
    - Module Defect Density = Number of defects discovered / Length of source

# Challenges



- **Cost of collecting metrics**
  - Automation is less costly than manual method
  - Interpretation of metrics consumes resources
- **Validity of metrics**
  - Does the metric really measure what it should?
  - What exactly should be measured?
- **Selection of metrics for measurement**
  - Hundreds available

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Metrics: requirements phase



- Number of requirements that change during the rest of the software development process
  - if a large number changed during specification, design, …, something is wrong in the requirements phase
- **Example**
  - Cost
  - Duration
  - Effort
  - Quality
    - number of faults found during inspection
    - rate at which faults are found (efficiency of inspection)

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Metrics: design phase



- Number of modules (measure of size of target product)
- Fault statistics
- Module cohesion
- Module coupling

# OO design Metrics



- Assumption: The effort in developing a class is determined by the number of methods.
- Hence the overall complexity of a class can be measured as a function of the complexity of its methods.

**Proposal: Weighted Methods per class (WMC)**

# WMC



- Let class C have methods M1, M2, .....Mn.
- Let Ci denote the complexity of method

$$WMC = \sum_{i=1}^{n} c_i$$

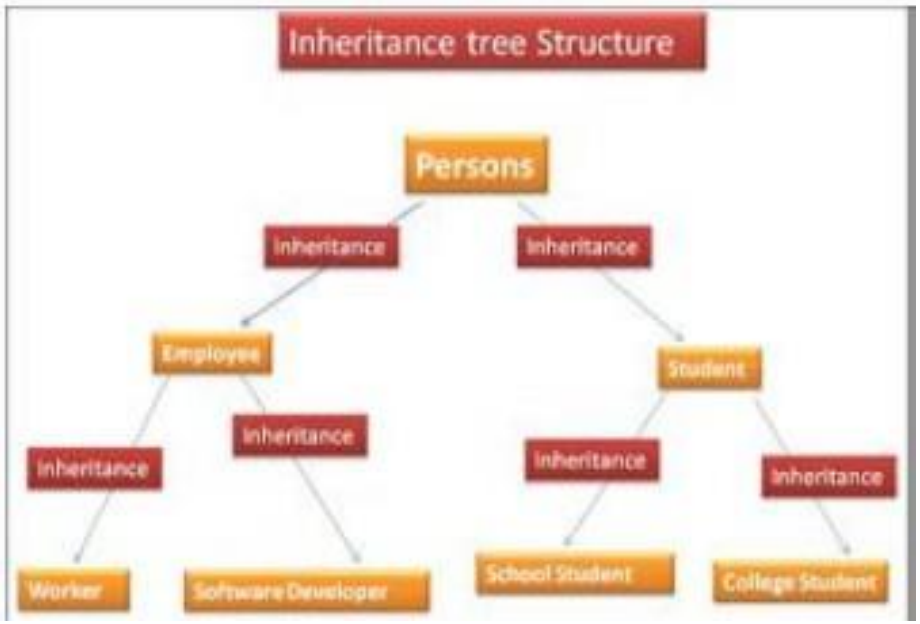
- How to measure Ci?
  - Number of attributes
  - Number of calls
  - ....

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Depth of Inheritance Tree



- Depth of a class in a class hierarchy determines potential for reuse. Deeper classes have higher potential for re-use.
- Depth of Inheritance (DIT) of class C is the length of the shortest path from the root of the inheritance tree to C.




MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Metrics: Implementation phase



- Intuition: more complex modules are more likely to contain faults
- Redesigning complex modules may be cheaper than debugging complex faulty modules
- Measures of complexity:
  - LOC
    - assume constant probability of fault per LOC
    - empirical evidence: number of faults related to the size of the product

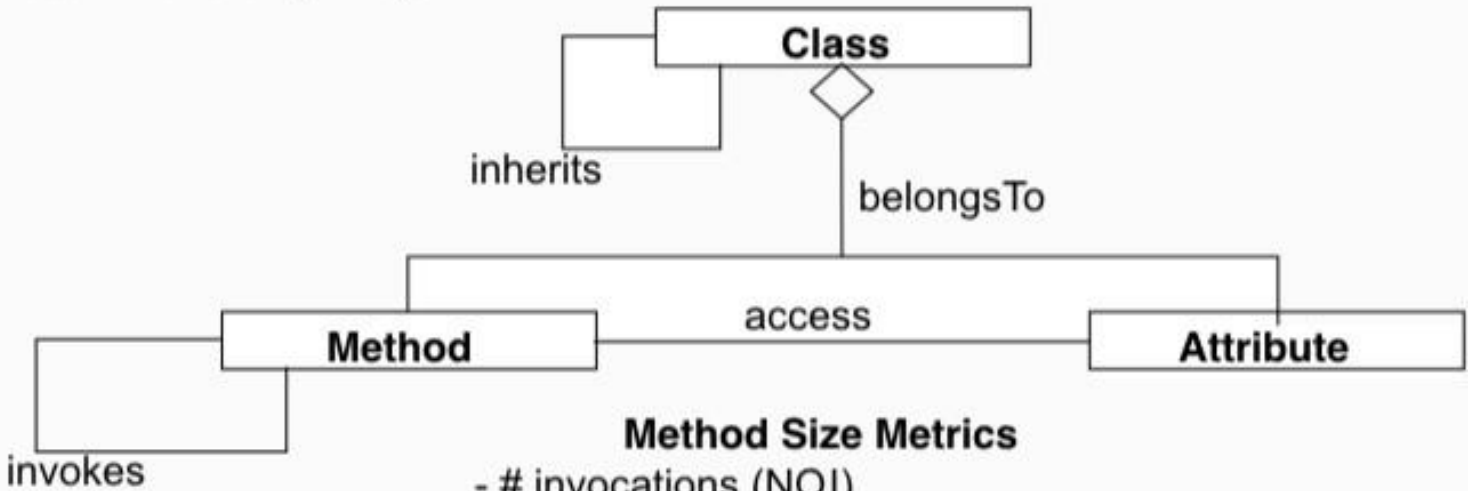
MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Sample Size (and Inheritance) Metrics



**Inheritance Metrics**
- hierarchy nesting level (HNL)
- # immediate children (NOC)
- # inherited methods, unmodified (NMI)
- #overridden methods (NMO)

**Class Size Metrics**
- # methods (NOM)
- # attributes, instance/class (NIA, NCA)
- # S of method size (WMC




**Method Size Metrics**
- # invocations (NOI)
- # statements (NOS)
- # lines of code (LOC)
- # arguments (NOA)

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Sample Coupling & Cohesion Metrics



**Coupling Between Objects (CBO)** CBO = number of other classes to which given class is coupled Interpret as “number of other classes a class requires to compile”

**Lack of Cohesion in Methods (LCOM)** LCOM = number of disjoint sets (= not accessing same attribute) of local methods

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Metrics Implementation Phase



- McCabe's cyclomatic complexity
  - Essentially the number of branches in a module
  - Number of tests needed for branch coverage of a module
  - Easily computed
  - In some cases, good for predicting faults

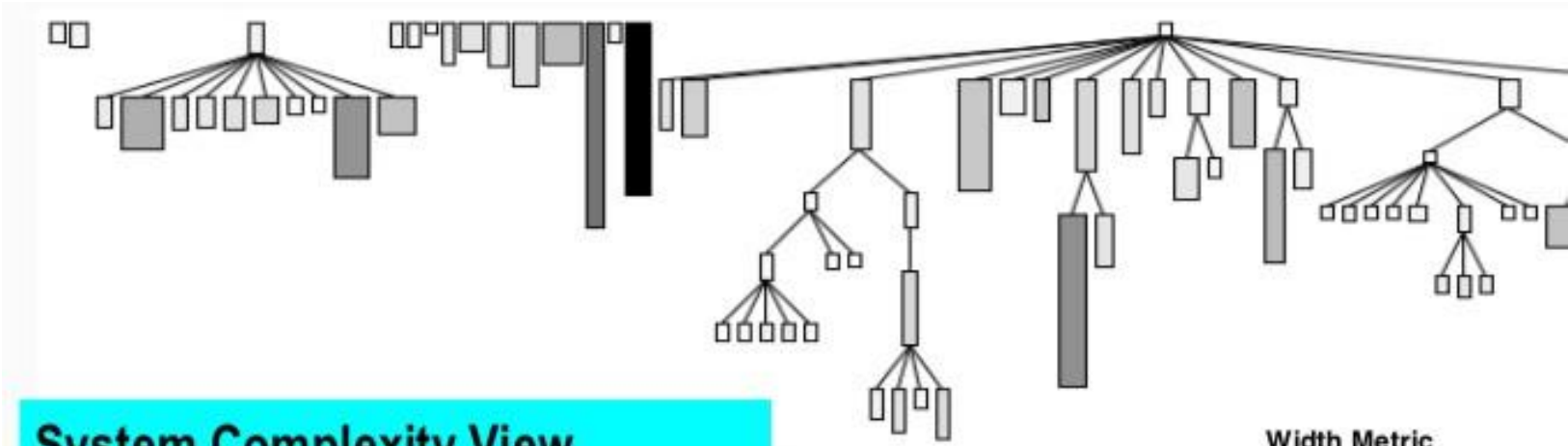
# Metrics: Maintenance Phase



- Example:
  - total number of faults reported
  - classifications by severity, fault type
  - status of fault reports (reported/fixed)

# Visualization tools for Software Metrics

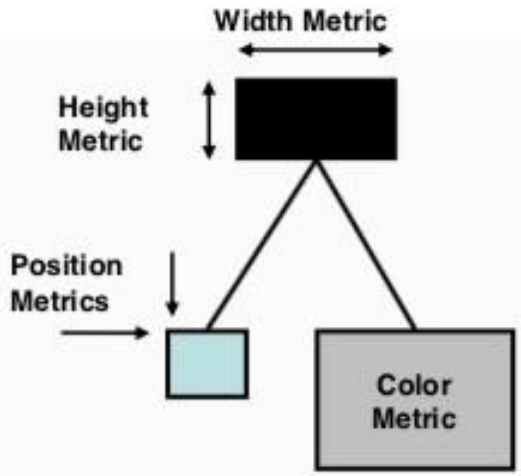




**System Complexity View**

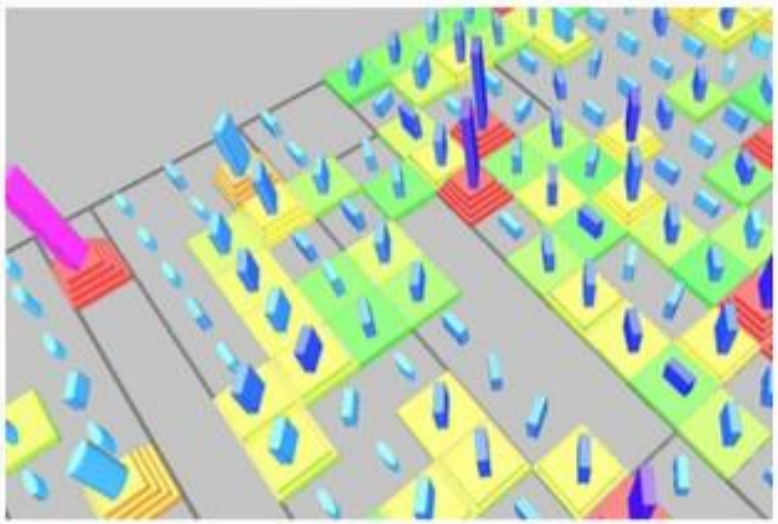
Nodes = Classes
Edges = Inheritance Relationships

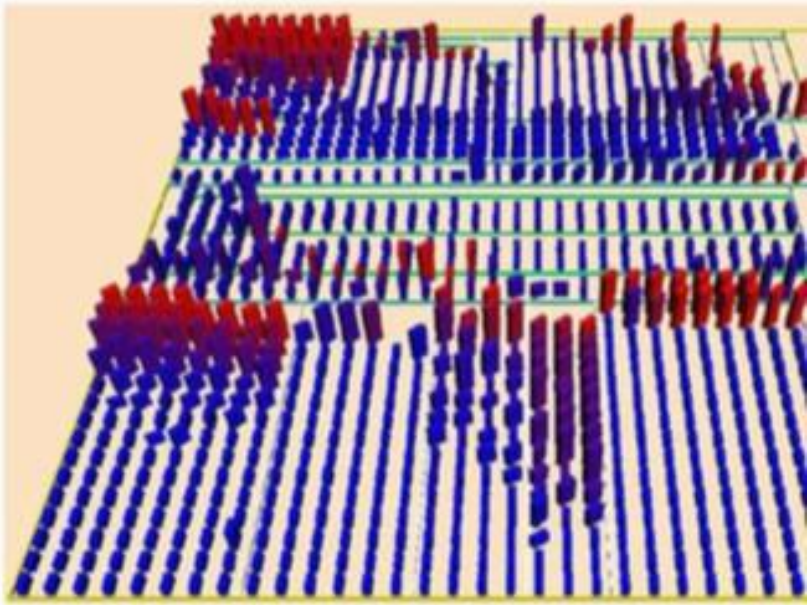
Width = Number of Attributes
Height = Number of Methods
Color = Number of Lines of Code



# Visualization tools for Software Metrics







MedTech
MEDITERRANEAN
INSTITUTE OF TECHNOLOGY
SOUTH MEDITERRANEAN UNIVERSITY

# Anti-patterns

# What Are Antipatterns

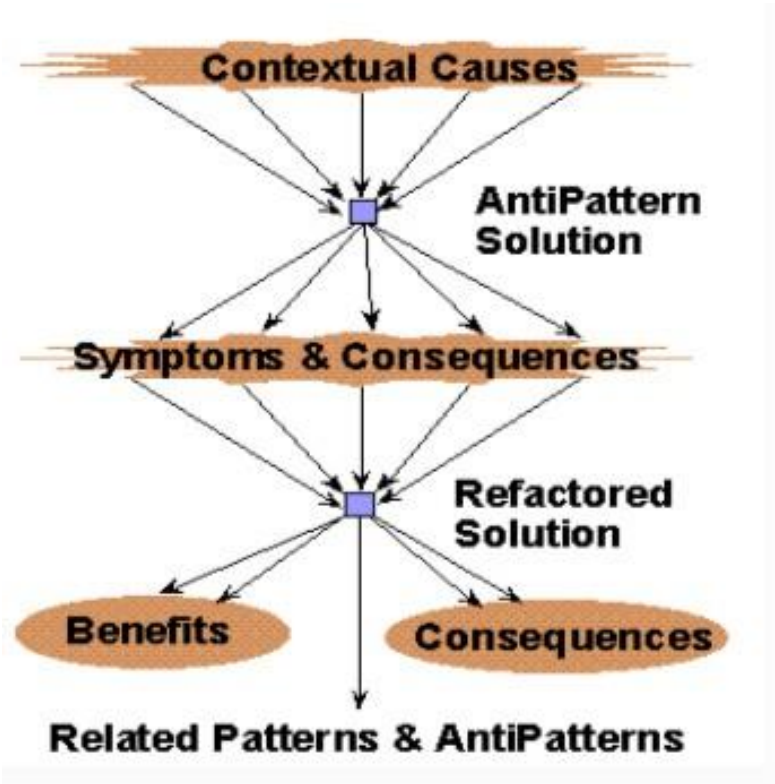


- "Negative Solutions," or solutions that present more problems than they address.
- Provide Knowledge to prevent and recover from common Mistakes.

MedTech
MEDITERRANEAN INSTITUTE OF TECHNOLOGY
SOUTH MEDITERRANEAN UNIVERSITY

# Antipatterns Vs. Design Patterns






- Design Pattern- a general repeatable solution to a commonly occurring problem
- Antipattern- Such a solution which is recognized as a poor way to solve the problem, and a refactored solution

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Terminology



- Antipatterns are also called
  - Code smells
  - Design flaws
  - Defects
  - Bad smells
  - …

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Antipatterns Examples



- Duplicated code
- Long method
- Large class
- Long parameter list
- Message chain
- ...

- Switch statements
- Data class
- Functional decomposition
- ...

MedTech
MEDITERRANEAN INSTITUTE OF TECHNOLOGY
SOUTH MEDITERRANEAN UNIVERSITY

# Large class/Blob/God Class



- Synopsis - Style design leads to one object with numerous responsibilities and most other objects only holding data.
- One class monopolizes the processing and other classes primarily encapsulate data.
- More than a couple dozen methods, or half a dozen variables
- Problem
  - The majority of the responsibility are allocated to a single class.
- Solution
  - Split into component classes

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Large class/Blob/God Class



*Development AntiPattern:*

## The Blob

- **Symptoms**
  - Single class with many attributes & operations
  - Controller class with simple, data-object classes.
  - Lack of OO design.
  - A migrated legacy design
- **Consequences**
  - Lost OO advantage
  - Too complex to reuse or test.
  - Expensive to load

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Large class/Blob/God Class

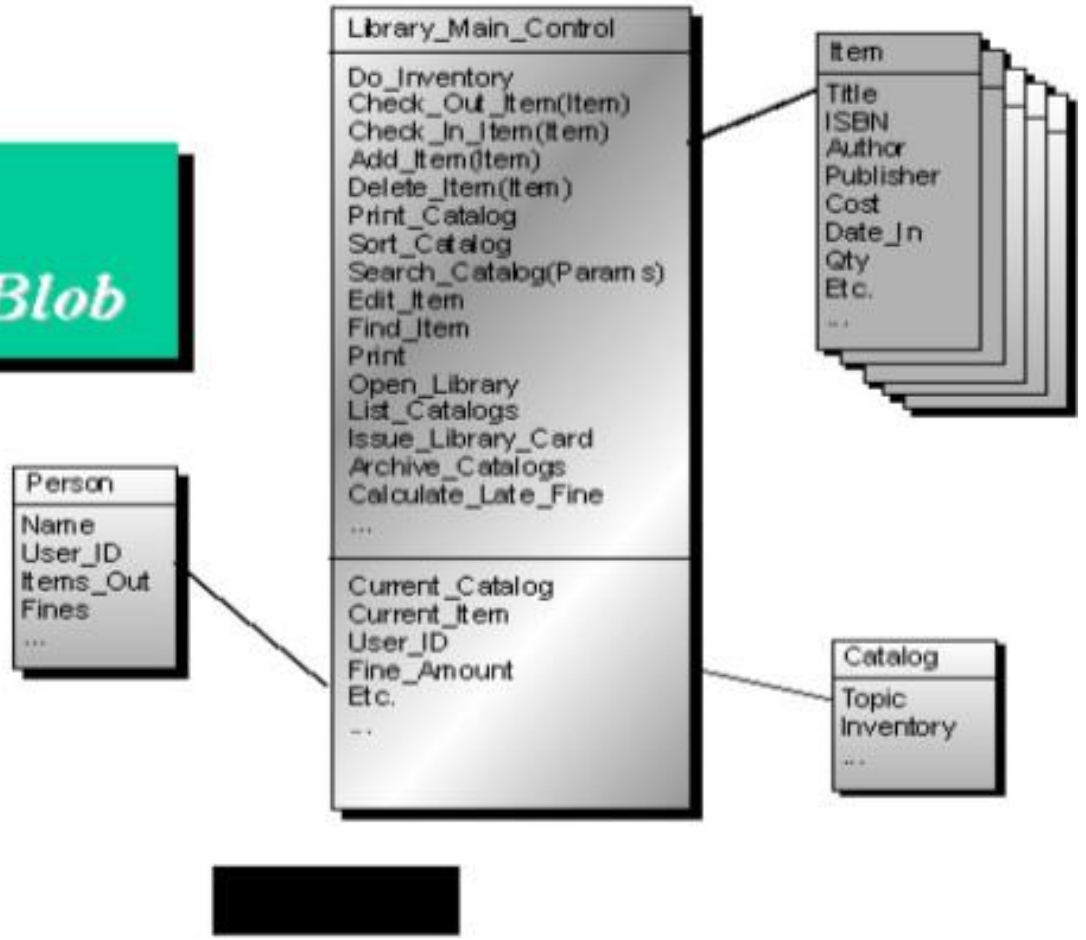


*Development AntiPattern:*

## The Blob

***Example:***

***The Library Blob***



# Duplicate code or Cut and Paste Programming



- Duplicate methods in subclasses
  - Move to superclass, possibly create superclass
- Duplicate expressions in same class
  - Extract method
- Duplicate expressions in different classes
  - Extract method, move to common component
- Code reused by copying source statements leads to significant maintenance problems.

# Long method



- A method contains too many lines of code.
- Can’t think of whole thing at once

# Long parameter list



- Introduce parameter object
- Only worthwhile if there are several methods with same parameter list, and they call each other

# Message Chain



- Long list of method calls:
- customer.getAddress().getState()
- window.getBoundingbox().getOrigin().getX()

MedTech
MEDITERRANEAN
INSTITUTE OF TECHNOLOGY
SOUTH MEDITERRANEAN UNIVERSITY

# Data class/Lazy class



- Class has no methods except for getter and setters
- What to do:
  - Look for missing methods (feature envy?) and move them to the class
  - Merge with another class

MedTech
MEDITERRANEAN INSTITUTE OF TECHNOLOGY
SOUTH MEDITERRANEAN UNIVERSITY

# Switch statement



- Switch statements are very rare in properly designed object-oriented code
  - Therefore, a switch statement is a simple and easily detected “bad smell”
  - Of course, not all uses of switch are bad
  - A switch statement should not be used to distinguish between various kinds of object

# Example 1



```
class Animal {
   final int MAMMAL = 0, BIRD = 1, REPTILE = 2;
   int myKind;  // set in constructor
   ...
   String getSkin() {
      switch (myKind) {
         case MAMMAL: return "hair";
         case BIRD: return "feathers";
         case REPTILE: return "scales";
         default: return “skin";
      }
   }
}
```

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Example 1, improved



```
class Animal {
    String getSkin() { return “skin"; }
}
class Mammal extends Animal {
    String getSkin() { return "hair"; }
}
class Bird extends Animal {
    String getSkin() { return "feathers"; }
}
class Reptile extends Animal {
    String getSkin() { return "scales"; }
}
```

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Old Baggage



- Description
  - system contains many classes whose purpose is not known
    - Lava Flow, Dead Code
  - much of the code is left over from previous ideas and no longer has a purpose
    - was once fluid and useful, now is solid lava that you are afraid to remove
- Consequences
  - difficult to maintain, just gets worse

# Refactoring

# Where are we?



- The overall goal of software engineering is to create high quality software efficiently.
- What if you don't though? There are always pressures and reasons that software isn't great : Code-Smells
- Today: Refactoring
  - How to fix code smells to improve software quality?

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Refactoring



- **Refactoring is:**
  - restructuring (rearranging) code in a series of small, semantics-preserving transformations (i.e. the code keeps working)
- **Refactoring is not just arbitrary restructuring**
  - Code must still work
  - Semantics are preserved (i.e. not a major re-write)
  - Unit tests to prove the code still works
  - Code is
    - More loosely coupled
    - More cohesive modules
    - More comprehensible

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Refactoring



- The goal of refactoring is NOT to add new functionality
- The goal is refactoring is to make code easier to maintain in the future

MedTech
MEDITERRANEAN INSTITUTE OF TECHNOLOGY
SOUTH MEDITERRANEAN UNIVERSITY

# Refactoring



- **How do I Identify Code to Refactor?**
  - Martin Fowler uses “code smells” to identify when to refactor.
- There are numerous well-known refactoring techniques
  - You should be at least somewhat familiar with these before inventing your own
  - Refactoring “catalog”

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Refactoring Process

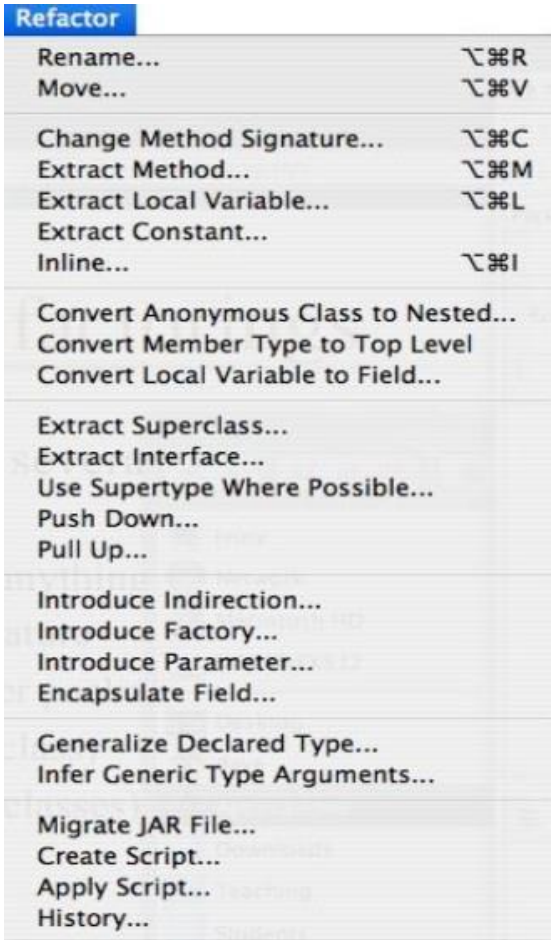


- Make a small change
  - a single refactoring
- Run all the tests to ensure everything still works
- If everything works, move on to the next refactoring
- If not, fix the problem, or undo the change, so you still have a working system

# Eclipse



- Eclipse (and some other IDEs) provide significant support for refactoring



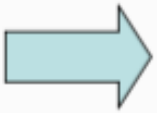

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Rename Variable or Method



- *Motivation:* One of the simplest, but one of the most useful that bears repeating: If the name of a method or variable does not reveal its purpose then change the name of the method or variable.

```
public class Customer
{
   public double getinvcdtlmt();
}
```

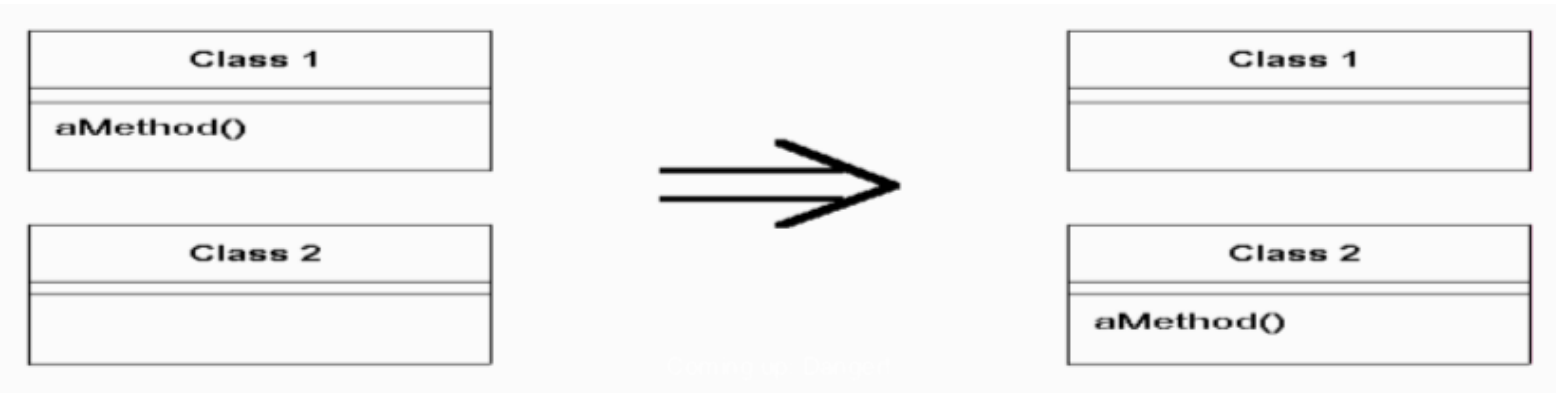


```
public class Customer
{
   public double getInvoiceCreditLimit();
}
```

# Move Method



- *Motivation:* A method will be used by more features of another class than the class on which it is defined.



# Move Method-Before



- *Motivation:* A method on one class uses (or is used by) another class more than the class on which its defined, move it to the other class

```
public class Student
{
  public boolean isTaking(Course course)
  {
    return (course.getStudents().contains(this));
  }
}

public class Course
{
  private List students;
  public List getStudents()
  {
    return students;
  }
}
```

# Move Method - Refactored



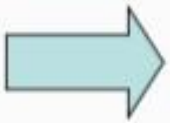
```
public class Student
{
}

public class Course
{
  private List students;
  public boolean isTaking(Student student)
  {
    return students.contains(student);
  }
}
```

# Extract Class



- Break one class into two classes

```
public class Customer
{
 private String name;
 private String workPhoneAreaCode;
 private String workPhoneNumber;
}
```



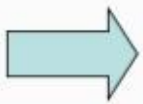
```
public class Customer
{
private String name;
private Phone workPhone;
}

public class Phone
{
private String areaCode;
private String number;
}
```

# Extract Method



- *Motivation:* A method that do too much.

```
public class Customer
{
  void int foo()
  {
    ...
    // Compute score
    score = a*b+c;
    score *= xfactor;
  }
}
```



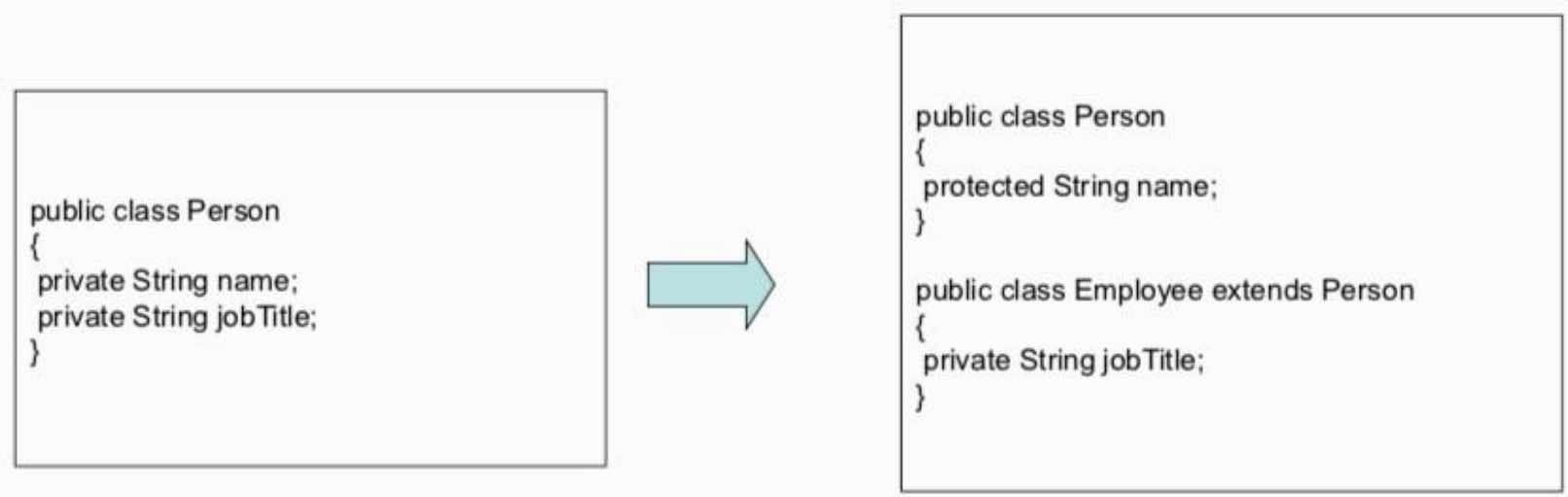
```
public class Customer
{
  void int foo()
  {
    ...
    score = ComputeScore(a,b,c,xfactor);
  }

  int ComputeScore(int a, int b, int c, int x)
  {
    return (a*b+c)*x;
  }
}
```

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Extract Subclass



- *Motivation:* When a class has features (attributes and methods) that would only be useful in specialized instances.

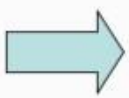


# Extract Super class



- *Motivation:* Two or more classes that share common features.

```
public class Employee
{
  private String name;
  private String jobTitle;
}

public class Student
{
  private String name;
  private Course course;
}
```



```
public abstract class Person
{
  protected String name;
}

public class Employee extends Person
{
  private String jobTitle;
}

public class Student extends Person
{
  private Course course;
}
```

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY

# Summary

# Summary



- Most code should be human-readable first, machine readable second.
- Refactoring improves code comprehension, thus making maintenance easier.
- Although identifying what to refactor is an art, the process of refactoring is very algorithmic.
- ALWAYS, ALWAYS TEST YOUR CODE BEFORE AND AFTER YOU REFACTOR!

MedTech MEDITERRANEAN INSTITUTE OF TECHNOLOGY SOUTH MEDITERRANEAN UNIVERSITY