

# Technical Debt

---



Version 1.4 – November 2023

© Antonio Vetrò, 2023



## Licensing Note






This work is licensed under the Creative Commons Attribution–NonCommercial–NoDerivatives 4.0 International License.

To view a copy of this license, visit

<http://creativecommons.org/licenses/by-nc-nd/4.0/>.

**You are free:** to copy, distribute, display, and perform the work

**Under the following conditions:**

-  **Attribution.** You must attribute the work in the manner specified by the author or licensor.
-  **Non-commercial.** You may not use this work for commercial purposes.
-  **No Derivative Works.** You may not alter, transform, or build upon this work.
  - For any reuse or distribution, you must make clear to others the license terms of this work.
  - Any of these conditions can be waived if you get permission from the copyright holder.

**Your fair use and other rights are in no way affected by the above.**

## Context

---

- Development projects are often followed by (or combined with) maintenance projects
- Software maintenance is:
  - Corrective Defects correction
  - Preventive
  - Adaptive Improvement
  - Evolutionary/perfective

## Technical Debt

---

OOPSLA '92  
Experience Report

### **The WyCash Portfolio Management System**

*Ward Cunningham*  
*March 26, 1992*

<http://c2.com/doc/oopsla92.html>

## Ward Cunningham – March 26, 1992

---

*Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise.*

---

**SoftEng**  
<http://softeng.polito.it>

<http://wiki.c2.com/?WardExplainsDebtMetaphor>

---



### Ward Explains Debt Metaphor

WardCunningham posted a [video](#) explaining Debt Metaphor (see [TechnicalDebt](#), for example) himself, clearing some confusion in the blogosphere about it.



The transcript of the video is below.

(Thank you, Ward. We are indebted to you for your inspirational words)

Contributors: [JuneKim](#), [LawrenceWang](#)

---

---

**SoftEng**  
<http://softeng.polito.it>

# Technical Debt (TD)

---

Imperfections in a software system that were caused by lack of time or by intentional choices and that run the risk of causing higher future maintenance cost

Examples:

- Bad organization of classes
- Known defects never fixed
- Requirements only partially implemented
- Partial test coverage
- Not automated tests

## TD as a gap and a trade-off

---

- Tradeoff between making a change perfectly and just make it working
  - What is a perfect change?
    - Documentation up to date, 100% test coverage, compliance with good programming practices, high modularity, etc.
- The higher the gap, the worst the long-term consequences
- Trade-off is necessary

## Everyday Indicators of Technical Debt

---

- Leave documentation to the end
- Not commenting code
- Not reviewing code
- Duplicated code
- Highly nested code
- Iper-specialization of team members
- Change requests from management during sprint
- ...

## Example and key terms

---

- You take out a loan to buy a car:
  - The cost of the car is 20.000 €
  - You can spend now only 5.000 €
  - The principal on the loan is 15.000 €
  - The bank charges 5% interest rate
  - The interest expense is 750 €

# Definition

---

- Principal:
  - the initial amount of money that is borrowed in a loan
- Interest rate:
  - the percent of principal charged by a lender for the use of its money
    - To the borrower, it is the cost of debt
    - To the lender, it will be the rate of return.
- Interest expense:
  - Money paid back minus principal

---

**SoftEng**  
<http://softeng.polito.it>

## The TD metaphor: example

---

- Skipping design is like borrowing money.
- Refactoring is like repaying principal
- Slower development due to complexity is like paying interest

Adapted from <http://wiki.c2.com/?ComplexityAsDebt>

---

**SoftEng**  
<http://softeng.polito.it>

## Stories from the field 1 / 2

---

“A company in Canada developed a good product for its local customers. Based on local success, the company decided to extend the market to the rest of Canada and immediately faced a new challenge: addressing the 20% of Canada that uses the French language in most aspects of life. The developers labored for a week to produce a French version of the product, planting a global flag for French = Yes or No as well as hundreds of if-then-else statements all over the code. A product demo went smoothly, and they got the sale!”

[https://insights.sei.cmu.edu/sei\\_blog/2019/05/managing-the-consequences-of-technical-debt-5-stories-from-the-field.html](https://insights.sei.cmu.edu/sei_blog/2019/05/managing-the-consequences-of-technical-debt-5-stories-from-the-field.html)

---

**SoftEng**  
<http://softeng.polito.it>

## Stories from the field 2 / 2

---

“Then, a month later, on a trip to Japan, a salesperson proudly boasted that the software was multilingual, returned to Canada with a potential order, and assumed that a Japanese version was only one week of work away. Now the decision not to use a more sophisticated strategy--such as externalizing all the text strings and using an internationalization package--was badly hurting the developers. They would not only have to select and implement a scalable and maintainable strategy but also have to undo all the quick-and-dirty if-then-else statements.”

[https://insights.sei.cmu.edu/sei\\_blog/2019/05/managing-the-consequences-of-technical-debt-5-stories-from-the-field.html](https://insights.sei.cmu.edu/sei_blog/2019/05/managing-the-consequences-of-technical-debt-5-stories-from-the-field.html)

---

**SoftEng**  
<http://softeng.polito.it>

# TD: Beyond Software Engineering



Popular

Latest

The Atlantic

Sign In

Subscribe

## TECHNOLOGY

### The Toxic Bubble of Technical Debt Threatening America

Climate change will soon expose a crippling problem embedded in the nation's infrastructure. In fire-ravaged California, it already has.

ALEXIS C. MADRIGAL OCTOBER 29, 2019



#### MORE STORIES

This Is What Adapting to Climate Change Looks Like

ROBINSON MEYER



Pacific Gas and Electric Charged With 12 Felonies in 2010 San Bruno Explosion

SARA MORRISON



Power Lines Are Burning the West

KENDRA ATLEWORK



<https://www.theatlantic.com/technology/archive/2019/10/california-fires-and-pge-toxic-debt/600979/>

SoftEng

<http://softeng.polito.it>

# TD: Beyond Software Engineering



SoftEng

<http://softeng.polito.it>



## TD: Beyond Software Engineering

---



---

**SoftEng** [https://en.m.wikipedia.org/wiki/McDonnell\\_Douglas\\_DC-10](https://en.m.wikipedia.org/wiki/McDonnell_Douglas_DC-10)  
<http://softeng.polito.it>

## DC-10 cargo door

---



---

**SoftEng** [https://en.m.wikipedia.org/wiki/McDonnell\\_Douglas\\_DC-10](https://en.m.wikipedia.org/wiki/McDonnell_Douglas_DC-10)  
<http://softeng.polito.it>

## 1974, Turkish Airlines Flight 981

---



346 killed

**SoftEng**  
<http://softeng.polito.it>

## Back to software: Ariane 5 crash

---

- On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in an explosion only about 40 seconds after initiation of the flight sequence, at an altitude of about 3700 m.
- The cause were specification and design errors in the software of the inertial reference system, and more in details an execution of a data conversion from 64-bit floating point to 16-bit signed integer value.
- An automatic code analyzer could have easily identified the issue.



**SoftEng**  
<http://softeng.polito.it>

## Process-related causes

---

- Requirements not revised from previous environment (Ariane 4).
- Wrong specifications for error-handling.
- Neither the hardware tests, nor the system integration tests were completely repeated in the new environment.

---

**SoftEng**  
http://softeng.polito.it

## Worst TD interest: disasters

---

### How Boeing 737 MAX's flawed flight control system led to 2 crashes that killed 346

Watch the full story on "20/20"

By [Joseph Rhee](#), [Gerry Wagschal](#), and [Jinsol Jung](#)

**NASA** SOLAR SYSTEM EXPLORATION  
Our Galactic Neighborhood

#### What was Mars Climate Orbiter?

NASA's Mars Climate Orbiter was designed to study Mars from orbit and to serve as a communications relay for the Mars Polar Lander and Deep Space probes. The mission was unsuccessful due to a navigation error caused by a failure to translate English units to metric.

#### Toyota "Unintended Acceleration" Has Killed 89

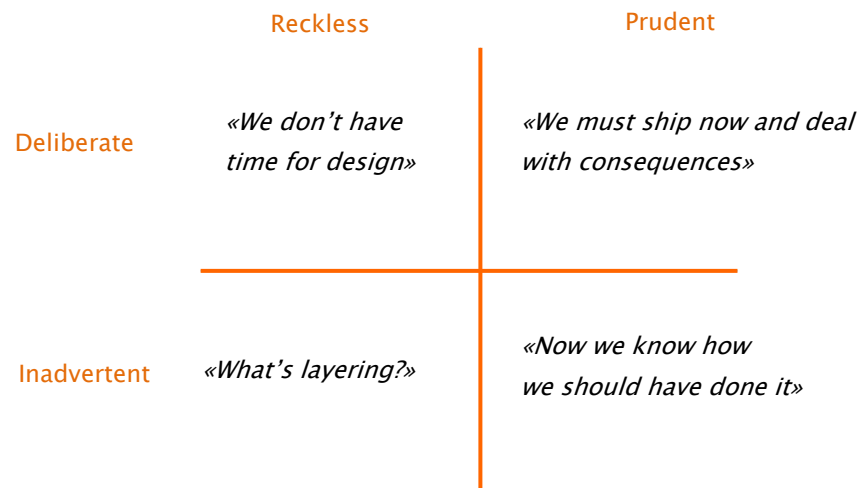
**MONEY** WATCH  
MAY 25, 2010 / 7:08 PM / AP

More at: <https://raygun.com/blog/costly-software-errors-history/>

---

**SoftEng**  
http://softeng.polito.it

## Technical Debt Quadrant



<https://martinfowler.com/bliki/TechnicalDebtQuadrant.html>

SoftEng  
http://softeng.polito.it

## Examples of benefits of TD

- Release on time and within budget
- Higher “writing-code” productivity
- Minimal functionalities working soon
- Fast feedback from stakeholders



SoftEng  
http://softeng.polito.it

## Examples of TD interests

---



- Increased maintenance costs as project grows
- Decreased flexibility of the code
- Productivity decreases

## TD types

---

- Defect debt
  - E.g., latent defects not yet fixed
- Design/architecture debt
  - E.g., bad organization of classes
- Documentation debt
  - E.g., outdated or incomplete documentation
- Testing debt
  - E.g., missing/not executed test cases/plans

## Causes of TD: examples

---

- TECHNOLOGY
  - Technology limitations
  - Legacy code
  - Project maturity
- PROCESS
  - Unclear requirements
  - Little or no history of design decisions
  - Not knowing or adopting best practices
  - Code maintenance not explicitly managed

## Causes of TD

---

- PEOPLE
  - Postpone work until needed
  - Poor communication between developers and stakeholders or within team members
  - Inexperience
  - Aggressive customers
  - Subcontractors
  - Changes in schedule/budget

---

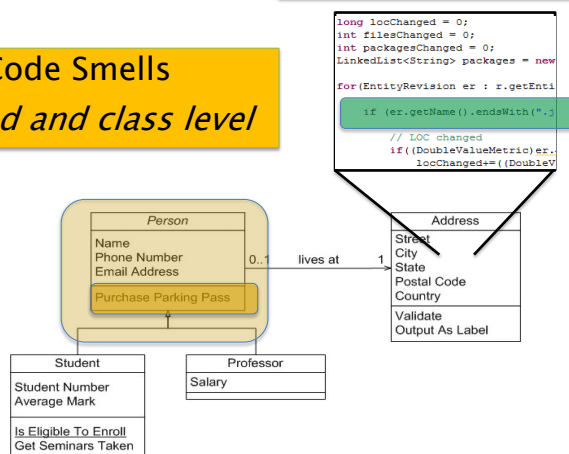
# TD identification

## Automatic TD identification: our focus

---

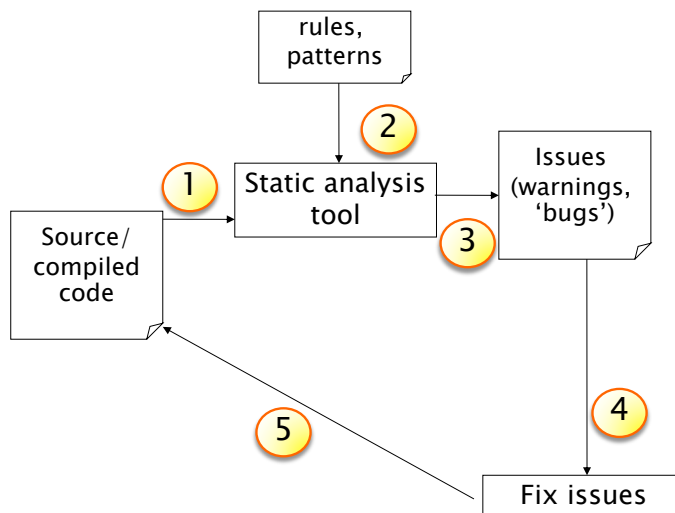
Automatic Static Analysis  
*Line level*

Code Smells  
*Method and class level*



## Automatic Static Analysis (ASA)

---



## ASA issues

---

- Different categories of warnings:
  - Correctness
  - Multithread
  - Security
  - Naming conventions
  - Performance
  - ...



# Example 1

---

```
if (file != null) {  
    if (file.isFile() || file.isDirectory()) {  
        /* ... */  
    }  
}
```

Not compliant

<https://rules.sonarsource.com/java/tag/clumsy/RSPEC-1066>

**SoftEng**  
<http://softeng.polito.it>

# Example 1 (collapsible if)

---

```
if (file != null) {  
    if (file.isFile() || file.isDirectory()) {  
        /* ... */  
    }  
}
```

Not compliant



```
if (file != null && isFileOrDirectory(file)) {  
    /* ... */  
}
```

Compliant

```
private static boolean isFileOrDirectory(File file) {  
    return file.isFile() || file.isDirectory();  
}
```

<https://rules.sonarsource.com/java/tag/clumsy/RSPEC-1066>

**SoftEng**  
<http://softeng.polito.it>

## Example 2

---

```
int k;  
boolean b = true;  
while (b) {  
    k++;  
}
```

Not compliant

<https://rules.sonarsource.com/java/tag/clumsy/RSPEC-2189>

---

**SoftEng**  
<http://softeng.polito.it>

## Example 2 (infinite loop)

---

```
int k;  
boolean b = true;  
while (b) {  
    k++;  
}
```

Not compliant



```
int k;  
boolean b = true;  
while (b) {  
    k++;  
    b = k < Integer.MAX_VALUE;  
}
```

Compliant

<https://rules.sonarsource.com/java/tag/clumsy/RSPEC-2189>

---

**SoftEng**  
<http://softeng.polito.it>

## Example 3

---

```
public class Fruit {  
    protected Season ripe;  
    // ...  
}  
public class Raspberry extends Fruit {  
    private boolean ripe;  
    // ...  
}
```

Not compliant

<https://rules.sonarsource.com/java/tag/confusing/RSPEC-2387>

---


**SoftEng**  
<http://softeng.polito.it>

## Example 3 (shadowing parent)

---

```
public class Fruit {  
    protected Season ripe;  
    // ...  
}  
public class Raspberry extends Fruit {  
    private boolean ripe;  
    // ...  
}
```

Not compliant



```
public class Raspberry extends Fruit {  
    private boolean ripened;  
    // ...  
}
```

Compliant

<https://rules.sonarsource.com/java/tag/confusing/RSPEC-2387>

---

**SoftEng**  
<http://softeng.polito.it>

## Example 4

---

```
boolean result = performAction();  
assertThat(result);
```

Not compliant

<https://rules.sonarsource.com/java/tag/tests/RSPEC-2970>

**SoftEng**  
<http://softeng.polito.it>

## Example 4 (uncomplete assert)

---

```
boolean result = performAction();  
assertThat(result);
```

Not compliant



```
boolean result = performAction();  
assertThat(result).isTrue();
```

Compliant

<https://rules.sonarsource.com/java/tag/tests/RSPEC-2970>

**SoftEng**  
<http://softeng.polito.it>

# Code Smells

---

- Methods and classes that violate the principles of good object-oriented design:
  - Clearly defined single responsibility
  - Encapsulation
  - Information hiding
  - Few and clear interfaces
  - Proper use of inheritance

## Code smells types

---

- Identity disharmonies
  - design flaws that affect single entities
- Collaboration disharmonies
  - design flaws that affect several entities at once
- Classification disharmonies
  - design flaws that affect entities tied by inheritance/abstraction

---

# Identity disharmonies

## Identity disharmonies

---

- Proportion rule:
  - Operations and classes should have a harmonious size
- Presentation rule:
  - Each class should have a set of services, each with a single responsibility and unique behavior
- Implementation rule:
  - Data and operations should collaborate within the class to which they semantically belong

# Identity disharmonies–Smells

---

- God class
- Brain class
- Brain method
- Feature envy
- Data class
- Significant duplication

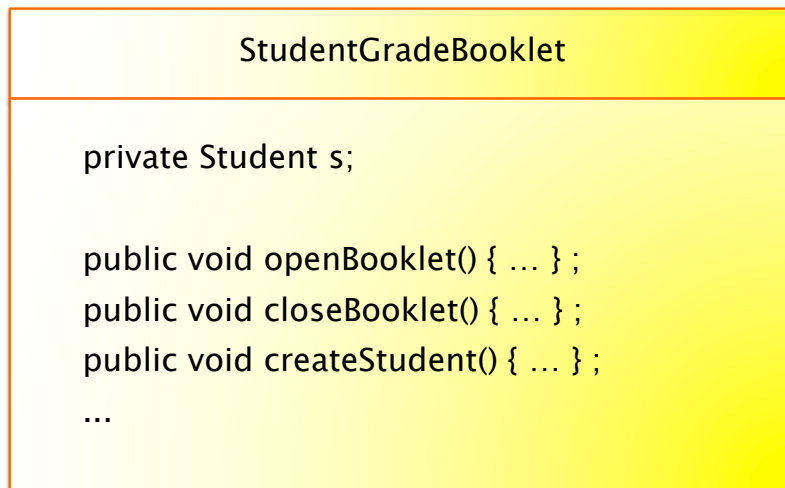
## God Class

---

- It applies to classes
  - Also known as “Large Class” (Fowler)
- Characteristics
  - Centralizes logic
  - Multiple responsibilities
  - Delegates minor details
  - Uses data of other classes

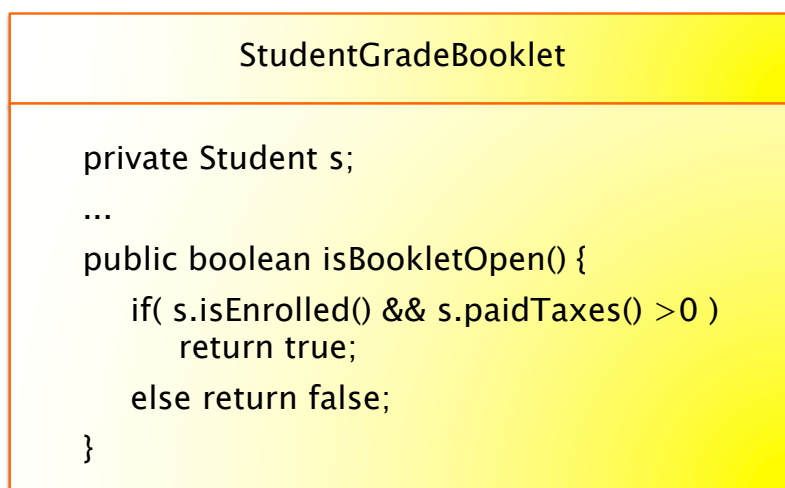
## Example: logic centralization

---



## Example: use external data

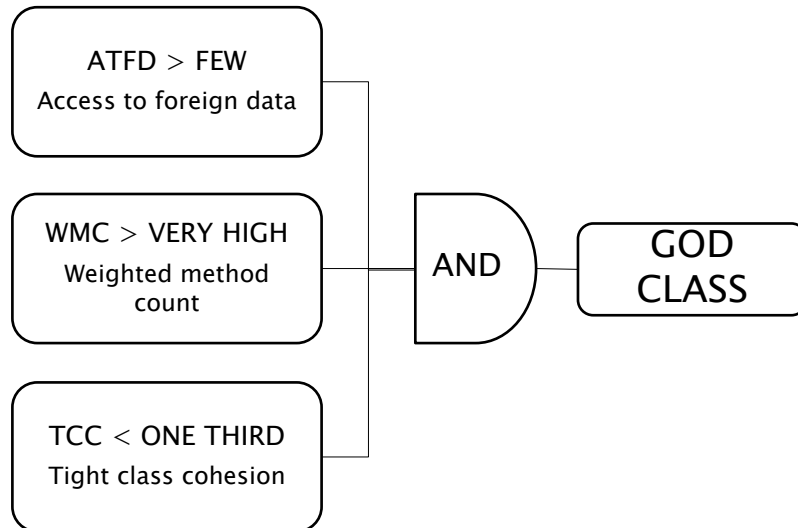
---





## God Class: detection heuristic

---



## Access to foreign data

---

- Number of usages of foreign attributes, both directly and through accessors.

# Weighted Method Count

---

- Each function has a minimum complexity of 1
- Whenever the control flow of a function splits, the complexity counter gets incremented by one.
  - Calculation varies slightly by language because of different keywords and functionalities involved.

## WMC: example in Java

---

```
void highCyclo() { // wmc = 1
    int x = 0, y = 2;
    boolean a = false, b = true;
    if (a && (y == 1 ? b : true)) { // +3
        if (y == x) { // +1
            while (true) { // +1
                if (x++ < 20) { // +1
                    break; // +1
                }
            }
        } else if (y == t && !d) { // +2
            x = a ? y : x; // +1
        } else { x = 2; }
    } } // wmc = 11
```

+1 at each method invocation  
+1 at each control flow statement: if, case, catch, throw, do, while, for, break, continue  
+1 at each boolean operator (&&, ||) in the guard condition of a control flow statement

Example adapted from  
[https://pmd.github.io/latest/pmd\\_java\\_metrics\\_index.html#cycloomatic-complexity-cyclo](https://pmd.github.io/latest/pmd_java_metrics_index.html#cycloomatic-complexity-cyclo)

## Tight Class Cohesion

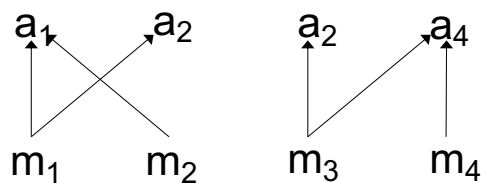
---

- Fraction of directly connected pairs of methods
- Two methods are directly connected if they are directly connected to an attribute.
- A method  $m$  can be directly connected to an attribute :
  - when the attribute appears within the method's body
  - when the attribute is within the body of a method invoked by method  $m$  directly or transitively

## Tight Class Cohesion: example

---

- Number of method pairs: 6
  - $(N \times N - 1) / 2$
- Number of connected methods pairs: 2
  - $m_1 - m_2$  ,  $m_3 - m_4$
- $TCC = 2/6 = 0,33$



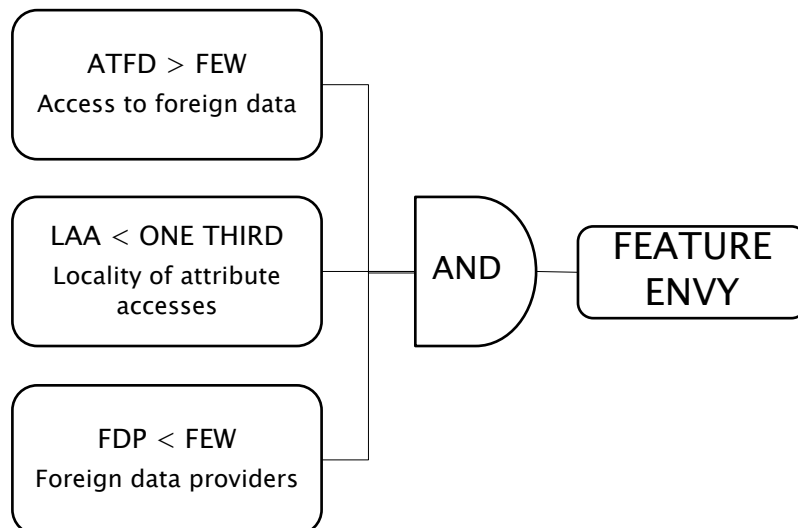
## Feature envy

---

- It applies to methods
- Characteristics
  - Access to data of other classes

## Feature envy: detection heuristic

---



---

# Collaboration disharmonies

# Collaboration disharmonies

---

- Collaboration rule:
  - Collaborations should be only in terms of method invocations and have a limited extent, intensity and dispersion

## Coll. Disharmonies smells

---

- Intensive coupling
- Dispersed coupling
- Shotgun surgery

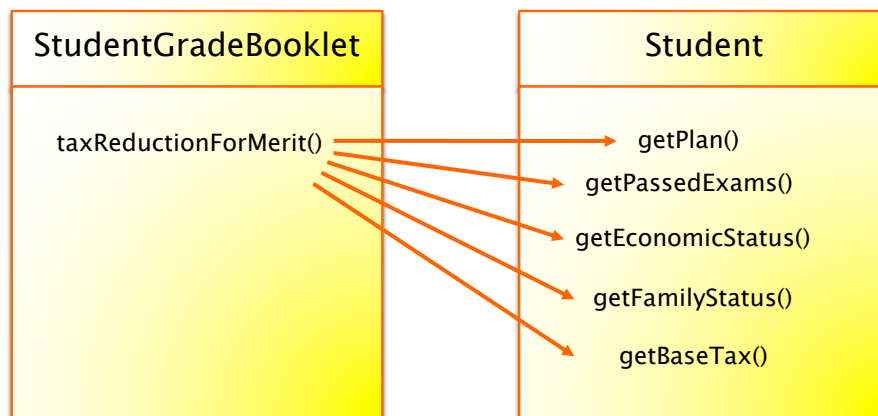
## Intensive coupling

---

- It applies to methods
  - that calls too many methods from few unrelated classes
- Characteristics
  - tied to many other operations in the system
  - provider operations are dispersed only into one or a few classes

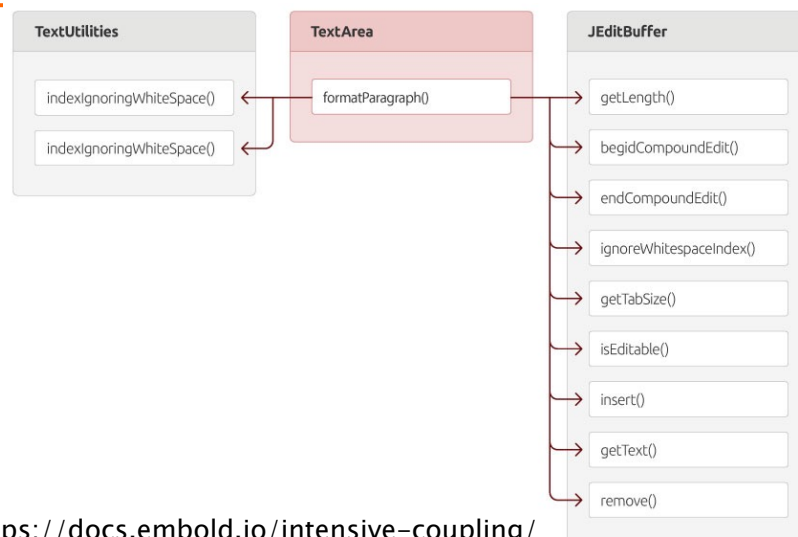
# Example 1

---



# Example 2

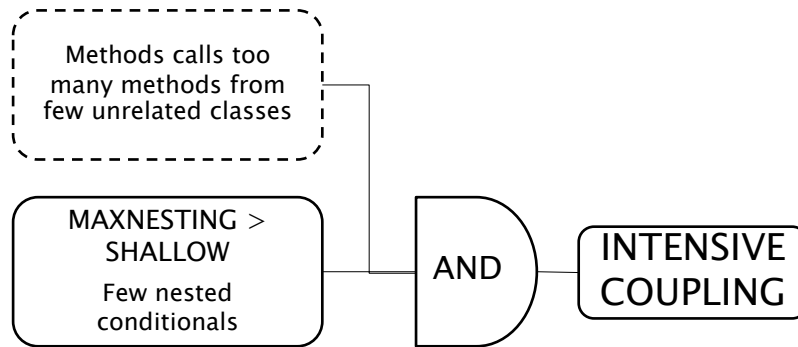
---



<https://docs.embold.io/intensive-coupling/>

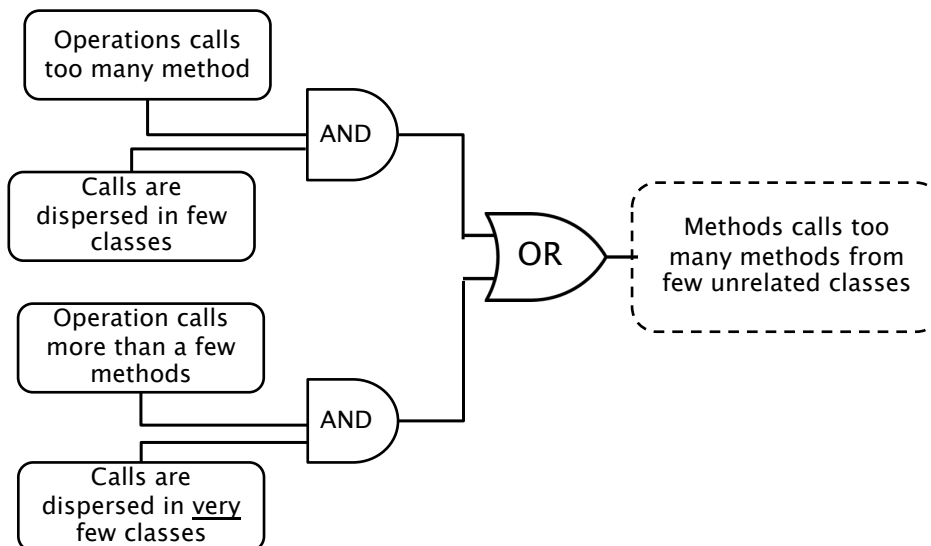
## Intensive coupling – detection

---



## Intensive coupling – detection

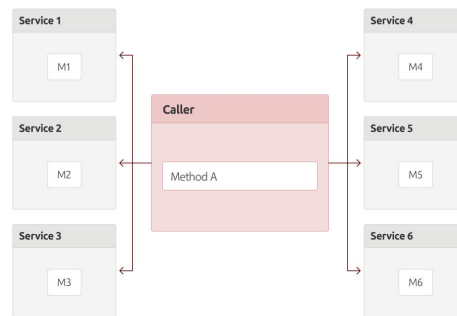
---





# Dispersed coupling

- A method suffers from Dispersed Coupling when it calls many other methods that are dispersed among many classes.

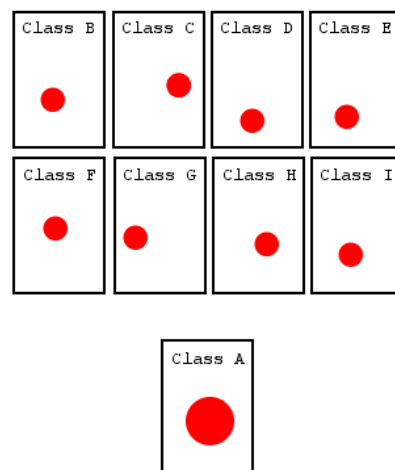


<https://docs.embold.io/dispersed-coupling/>

SoftEng  
<http://softeng.polito.it>

# Shotgun surgery

- A method suffers from Shotgun Surgery if it is called many times from many other classes
- Making any modifications requires that you make many small changes to many different classes.



[https://en.wikipedia.org/wiki/Shotgun\\_surgery](https://en.wikipedia.org/wiki/Shotgun_surgery)

SoftEng  
<http://softeng.polito.it>

---

# Classification disharmonies

## Classification disharmonies

---

- Proportion rule:
  - classes should be organized in hierarchies having harmonious shapes
- Presentation rule:
  - the identity of an abstraction should be harmonious with respect to its ancestors
- Implementation rule:
  - harmonious collaborations within a hierarchy are directed only towards ancestors, and serve mainly the refinement of the inherited identity

## Class. Disharmonies – smells

---

- Refused parent bequest
- Tradition braker

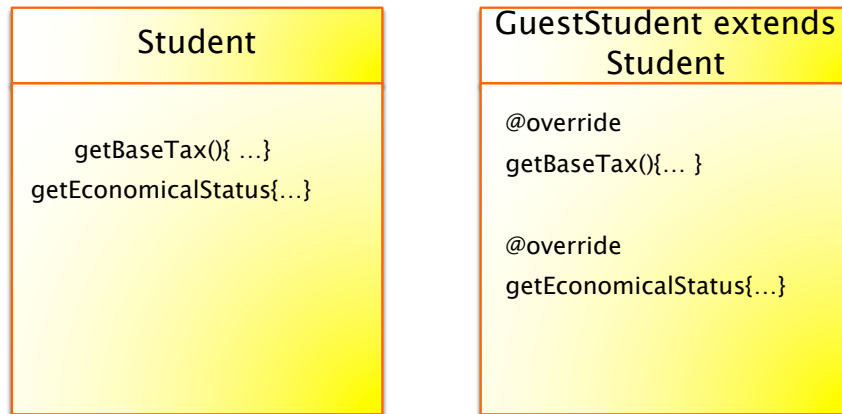
## Refused parent bequest

---

- It applies to classes
- Characteristics
  - low usage of inheritance– specific members from the base class
  - bequest refusal is intentional
  - class has its own identity

# Example

---



## Tradition braker

---

- It applies to classes
- Characteristics
  - provides a large set of services which are unrelated to those provided by its base class
  - does not specialize inherited services

---

## TD management

## Managing TD

---

- TD often managed implicitly
  - E.g.: a telephone call triggers refactoring
- Benefits of explicitly managing TD :
  - Better planning
  - Lower maintenance costs
  - Less “bad surprises”

## To pay or not to pay ?

---

- Deciding to pay off debt should rely on :
  - Value of debt
    - how much is it going to cost to fix it?
  - Interest rate (how much does it slow down development?)
  - Probability (what is the chance that the debt affects productivity?)

## Prioritizing TD items: cost/benefit

---

- Rational:
  - Select the most profitable opportunities
  - Ignore non-profitable ones.
- Profitable (good cost/benefit ratio) is:
  - Low principal
  - High interest rate

## Prioritizing TD items

---

More impact /  
higher interest



More effort /  
higher principal

**SoftEng**  
<http://softeng.polito.it>

## Prioritizing TD items: example

---

More impact /  
higher interest

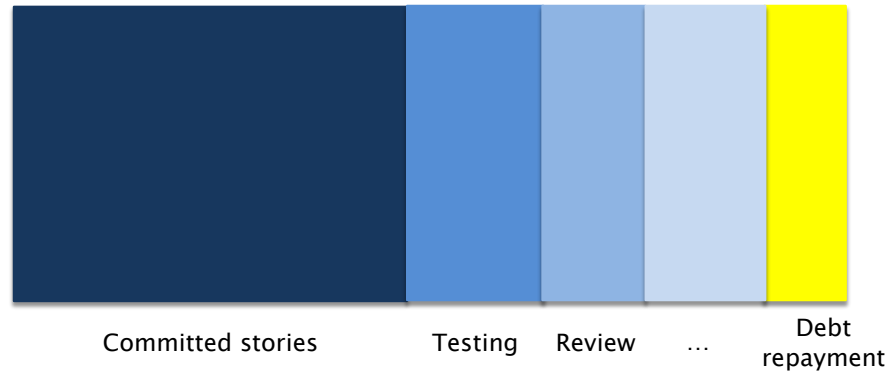


More effort /  
higher principal

**SoftEng**  
<http://softeng.polito.it>

# TD and AGILE

---

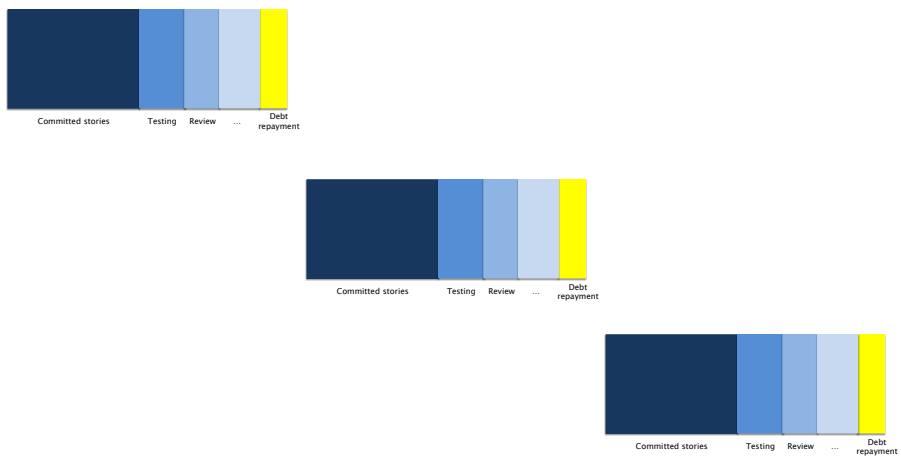


---

**SoftEng**  
<http://softeng.polito.it>

# TD and AGILE

---



---

**SoftEng**  
<http://softeng.polito.it>



# ACM Code of ethics

---

[Home](#) > [Code Of Ethics](#)

ACM Code of Ethics and Professional Conduct

<https://www.acm.org/code-of-ethics>

---

**SoftEng**  
<http://softeng.polito.it>

# ACM Code of ethics

---

## 2. PROFESSIONAL RESPONSIBILITIES.

A computing professional should...

### 2.1 Strive to achieve high quality in both the processes and products of professional work.

Computing professionals should insist on and support high quality work from themselves and from colleagues. The dignity of employers, employees, colleagues, clients, users, and anyone else affected either directly or indirectly by the work should be respected throughout the process. Computing professionals should respect the right of those involved to transparent communication about the project. Professionals should be cognizant of any serious negative consequences affecting any stakeholder that may result from poor quality work and should resist inducements to neglect this responsibility.

---

**SoftEng**  
<http://softeng.polito.it>

# ACM Code of ethics

---

## 3. PROFESSIONAL LEADERSHIP PRINCIPLES.

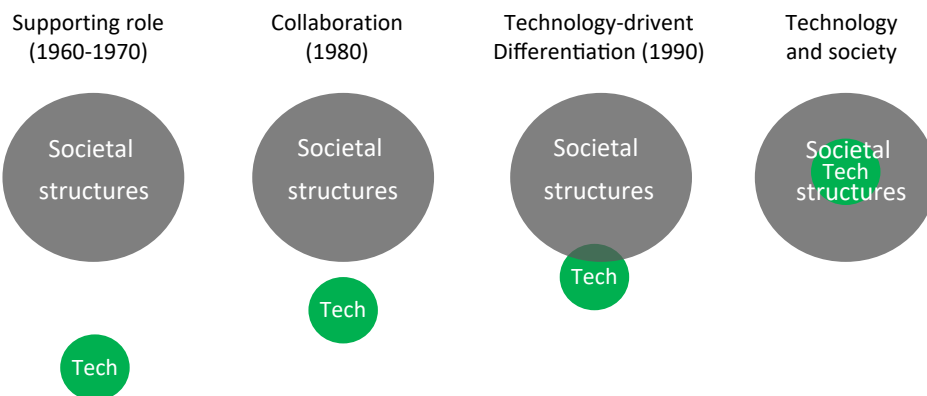
A computing professional, especially one acting as a leader, should...

### 3.1 Ensure that the public good is the central concern during all professional computing work.

People—including users, customers, colleagues, and others affected directly or indirectly—should always be the central concern in computing. The public good should always be an explicit consideration when evaluating tasks associated with research, requirements analysis, design, implementation, testing, validation, deployment, maintenance, retirement, and disposal. Computing professionals should keep this focus no matter which methodologies or techniques they use in their practice.

## Context

---



Adapted from: ERIK DÖRNENBURG, «The Path to DevOps», IEEE Software, Sep/Oct 2018 – Copyright © 2018 IEEE

# ACM Code of ethics

---

## 3. PROFESSIONAL LEADERSHIP PRINCIPLES.

A computing professional, especially one acting as a leader, should...

### 3.7 Recognize and take special care of systems that become integrated into the infrastructure of society.

Even the simplest computer systems have the potential to impact all aspects of society when integrated with everyday activities such as commerce, travel, government, healthcare, and education. When organizations and groups develop systems that become an important part of the infrastructure of society, their leaders have an added responsibility to be good stewards of these systems. Part of that stewardship requires establishing policies for fair system access, including for those who may have been excluded. [...]

# ACM Code of ethics

---

## 3. PROFESSIONAL LEADERSHIP PRINCIPLES.

A computing professional, especially one acting as a leader, should...

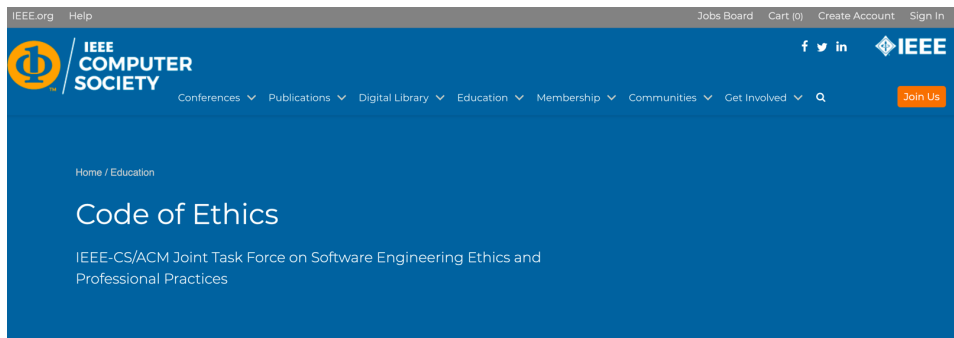
### 3.7 Recognize and take special care of systems that become integrated into the infrastructure of society.

[...] That stewardship also requires that computing professionals monitor the level of integration of their systems into the infrastructure of society. As the level of adoption changes, the ethical responsibilities of the organization or group are likely to change as well. Continual monitoring of how society is using a system will allow the organization or group to remain consistent with their ethical obligations outlined in the Code. When appropriate standards of care do not exist, computing professionals have a duty to ensure they are developed.

# IEEE Software engineering

---

## Code of Ethics



<https://www.computer.org/education/code-of-ethics>

---

**SoftEng**  
<http://softeng.polito.it>

# IEEE Software engineering

---

## Code of Ethics

- 3.01. Strive for **high quality**, acceptable cost and a reasonable schedule, ensuring significant tradeoffs are clear to and accepted by the employer and the client, and are available for consideration by the user and the public.

---

**SoftEng**  
<http://softeng.polito.it>

## TD and ethics

---

Developing high quality software has an ethical aspect, too



**Grady Booch** ✓  
@Grady\_Booch

Every line of code has a moral and ethical implication.

<https://modeling-languages.com/grady-booch-on-the-future-of-software-engineering-video-and-highlights/>

---

**SoftEng**  
<http://softeng.polito.it>

## A wider perspective

---



**Grady Booch** ✓  
@Grady\_Booch

It's a privilege and responsibility to be a software engineer because we can change the world

---

**SoftEng**  
<http://softeng.polito.it>

# At PoliTO

---



+ courses

- Responsible Artificial Intelligence
- Data Ethics and Data Protection

---

**SoftEng**  
<http://softeng.polito.it>

## References

---

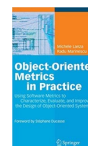
Kruchten, P., Nord, R., & Ozkaya, I. (2019). *Managing Technical Debt: Reducing Friction in Software Development*. Addison-Wesley Professional.



Fowler, M. (2018). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.



Lanza, M., & Marinescu, R. (2007). *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media.



Unger, S. H. (2017). *Controlling technology: Ethics and the responsible engineer*. John Wiley & Sons.



---

**SoftEng**  
<http://softeng.polito.it>