Modularity Anomaly Types – Part I

Alessandro Garcia

LES | DI |PUC-Rio - Brazil

OPUS Group

Symptoms of quality degradation

- Modularity anomalies
- Robustness anomalies
- Security vulnerabilities

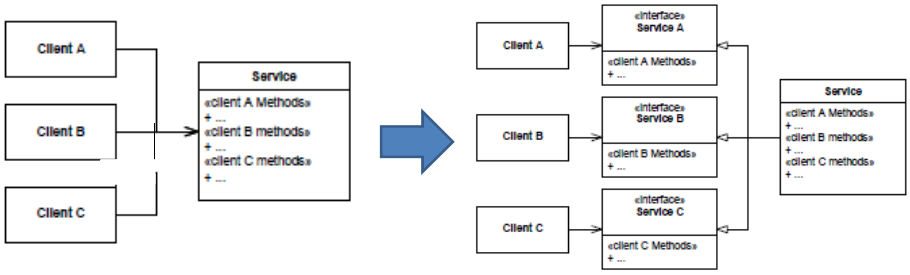
A modularity anomaly is…

- … an indication, observed in the modules of a *software artefact*, that usually corresponds to a deeper *quality* problem [Fowler 99]
  - quality: *maintainability*, comprehensibility, reusability, …
- it represents the violations of one or more *modularity principles* in a system module

Modularity principles

- Each module should satisfy:
  - explicit and simple interface
  - encapsulation
  - high cohesion
  - low coupling
  - single responsibility (~separation of concerns)
  - OCP – the open-closed principle [Meyer 88]
  - ISP – the interface segregation principle [Martin 96]
    - *"Clients should not be forced to depend on methods they do not use"*
    - cohesive interfaces instead of "fat" interfaces

ISP: The Interface Segregation Principle



May 16 5

Types of Design and Code Anomalies

- They manifest in different artifacts produced in specific SE stages
  - code anomaly
    - e.g. class decomposition: *Fowler's catalogue*
  - design anomaly
    - e.g. *Brow's catalogue*, *Riel's catalogue*
  - architecture design anomaly
    - e.g. component-connector decomposition: *Medvidovic's catalogue*
    - architecture anomaly: drift problem

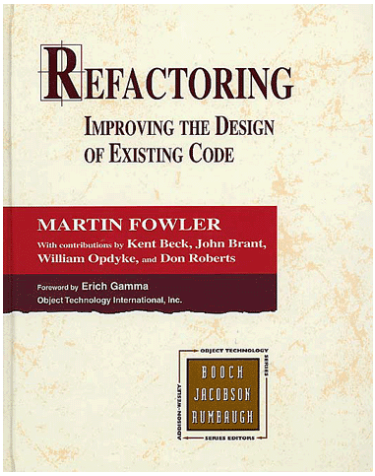
May 16 6

Code Anomalies

Code Anomaly

A code smell is a surface indication observed in the implementation that usually corresponds to a deeper quality problem in the system.

Martin Fowler, 1999



7

Code Anomalies

- popular term: "bad smell" or "code smell"
- may indicate architecture or design problems
  - indicative that refactoring may be appropriate
- arose out of developments in refactoring
  - modifying an existing code to accommodate future changes
- each 'code smell' is associated with a number of possible refactorings

- Examples of Frequent Code Anomalies
  - Feature Envy
  - God Modules
  - Long Method
  - Duplicated Code

Code Anomalies

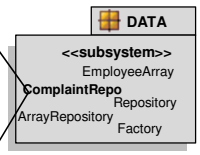
Feature Envy

If a (piece of) method seems more interested in a module other than the module it actually is in *[Fowler 99]*

- Examples of possible principle violations:
  - cohesion and coupling
  - single responsibility
  - information hiding

Feature Envy - Example I

```
public class ComplaintRepo{
   ...
   public int insert(..){..}
   public void update(..){..}
   public int getIndex(..){..}
   public boolean exists(..){..}
   public Complaint search(..){..}
   public void reset(..){..}
   public Object next(..){..}
   public void remove(..){..}
   public List getList(..){..}
   public boolean hasNext(..){..}
   public void updateTimestamp(..){..}
   public int searchTimestamp(..){..}
   ...
}
```



- at least one different axis of change
- no cohesion with the rest
- two feature envies within the module

10

Code Anomalies

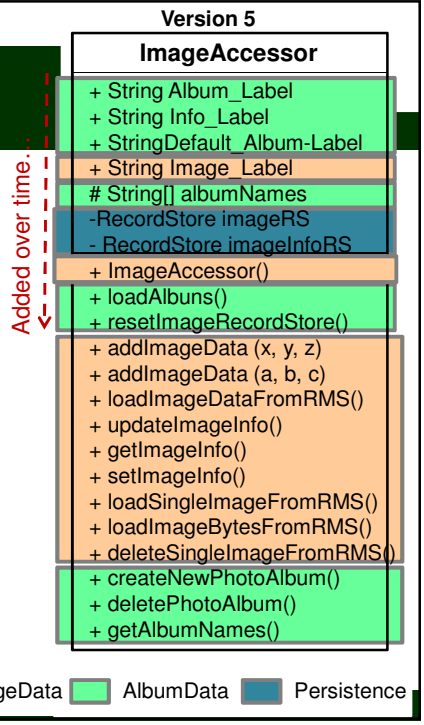
God Class

is a module that *knows too much* or *does too much* *[Fowler 99]*

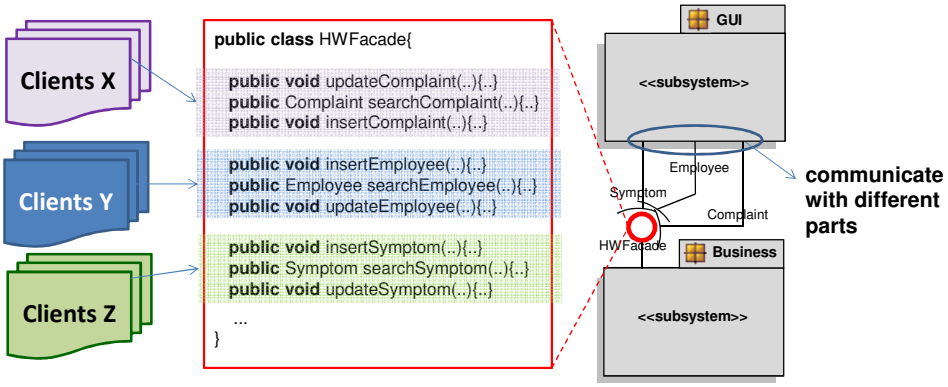
- Examples of possible principle violations:
  - single responsibility
  - cohesion and coupling
  - information hiding
  - interface segregation

Example of a God Class…

- … accumulating too many responsibilities
  - three concerns
  - most of the **module changes** are due to the presence of AlbumData and Persistence



God Class - Another Example



- Three responsibilities: three axes of change (violation of SRP and ISP)

13

Code Anomalies

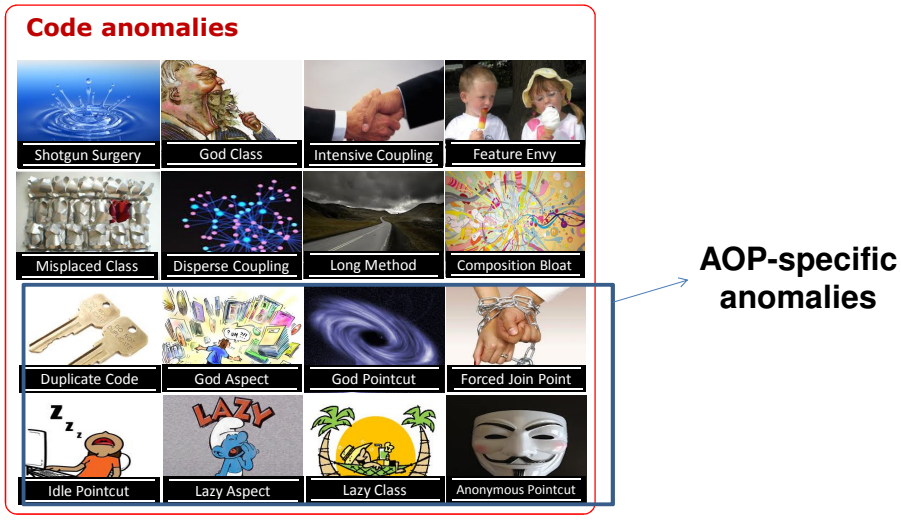
God Class

is a module that *knows too much* or *does too much* *[Fowler 99]*

- Split into Modules – decompose it into two or more classes
- Remove Feature Envies – apply refactorings for each instance of feature envy

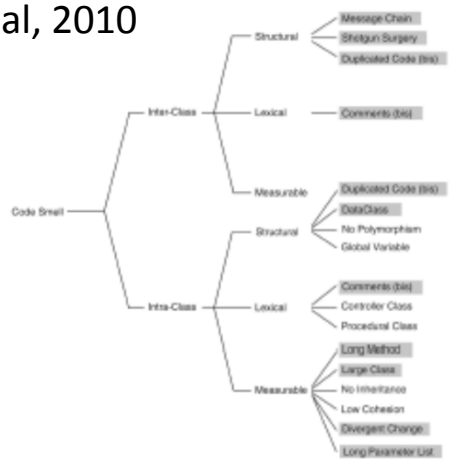
Other Code Anomalies

Code Anomalies also depend on the underlying programming technique



Classification of Code Anomalies

- By Moha et al, 2010



Naouel Moha, Yann-Gaël Guéhéneuc, Laurence Duchien, Anne-Françoise Le Meur: **DECOR: A Method for the Specification and Detection of Code and Design Smells.** IEEE Trans. Software Eng. 36(1): 20-36 (2010)

Sonar
Structure101
Understand
SAVE
SCOOP

Ferramenta	Plataforma	Observações
ReSharper	.NET, JavaScript	Detecção de anomalias estruturais, como código duplicado e heuristicamente inalcançável; possibilidade de personalizar inspeções de código
Ndepend	.NET	Efetua o cálculo de métricas, indicando possíveis pontos de melhoria; possibilidade de criar novas métricas baseadas em linguagens de consulta de código (CQL)
FxCop	.NET	Analisa a aderência do código a padrões de codificação próprios; possui um conjunto fixo de regras
Reek	Ruby	Detecção de anomalias estruturais tais como métodos longos, nomes inadequados, código estrangeiro.
Saikuro	Ruby	Analisa a complexidade ciclomática de cada método
Flay	Ruby	Detecção de código duplicado
Jdepend	Java	Coleta métricas para pacotes, como número de classes, acoplamento, e dependências entre pacotes
DECOR	Java	Detecção de anomalias estruturais; permite a criação de estratégias de detecção personalizadas através de uma linguagem própria
inCode	Java	Detecção de uma pequena variedade de anomalias estruturais, como métodos longos
Together	Java	Coleta de métricas e detecção de várias anomalias estruturais
Hint	Java	Baseia-se em métricas coletadas por outra ferramenta para a detecção de anomalias sensíveis a história
ArchJava	Java	Extensão de Java para dar suporte à definição de componentes e portas destinadas à verificação de conformidade arquitetural
Semmle	Java	Coleta de métricas, como número de linhas de código; possibilidade de criar novas regras baseadas em linguagens de consulta de código (CQL)
PMD	Java	Ferramenta de análise estática que permite criar regras para análise e verificação estrutural de código
DCL	Java	Linguagem específica que permite definir restrições sobre dependências entre módulos, bem como verificar a conformidade a estas restrições
Sonar	Java, C, PHP, Groovy	Detecção de várias anomalias estruturais, padrões de codificação e coleta de métricas. Permite recuperação do projeto arquitetural
CodeAssurance	Java	Análise do código para identificar re-ocorrência de falhas já conhecidas
CloneDetections / Clever	Java	Detecção de código duplicado
FindBugs	Java	Análise estática de código para detecção de falhas
JSLint	JavaScript	Detecção de anomalias e más práticas de programação, tais como o uso de operadores ineficazes

Underlying technique: detection strategies

- Techniques to detect code smells are **based on module attributes**
- Detection strategy is a logical expression based the combination of metric values - example

$$GC = ((ATFD > 1)\ E\ (WMC,\ TopValues(25\%))\ E\ (TCC,\ BottomValues(25\%))$$

access data from more than one class with few methods

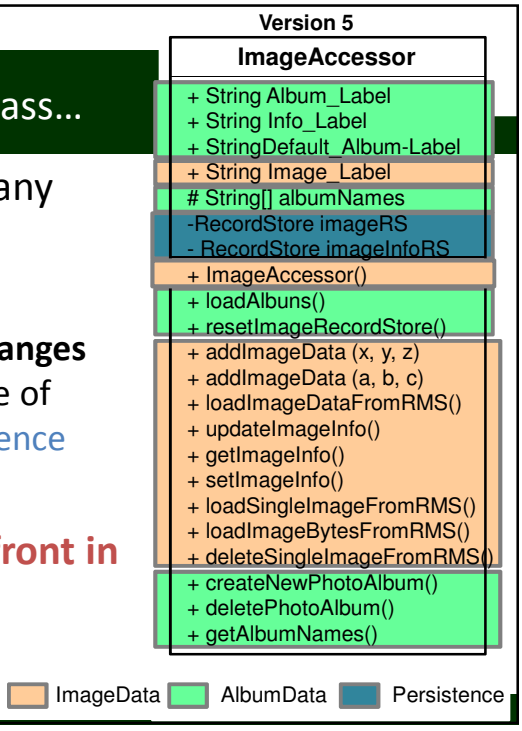
low cohesion (not many shared attributes)

one of the classes with higher number of methods

Code Smell	Description
Class data should be private	A class having at least one public field.
Complex class	A class having at least one method for which McCabe cyclomatic complexity is higher than 10.
Feature envy	All methods having more calls with another class than the one they are implemented.
Blob class	All classes having (i) cohesion lower than the average of the system AND (ii) LOCs > 500.
Lazy class	All classes having LOCs lower than the first quartile of the distribution of LOCs for all systems classes.
Long method	All methods having LOCs higher than the average of the system.
Long parameter list	All methods having a number of parameters higher than the average of the system.
Message chain	All chains of methods calls longer than three.
Refused bequest	All classes overriding more than half of the methods inherited by a superclass.
Spaghetti code	A class implementing at least two long methods interacting between them through method calls or shared fields.
Speculative generality	A class declared as abstract having less than three children classes using its methods.

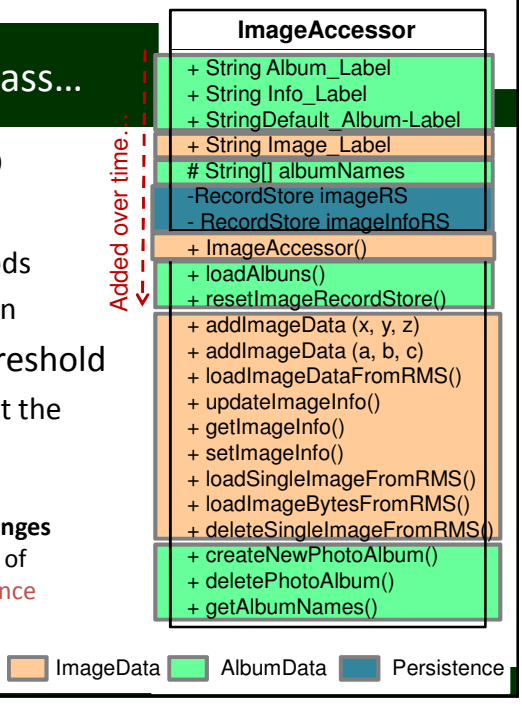
Example of a God Class…

- … accumulating too many responsibilities
  - three concerns
  - most of the **module changes** are due to the presence of AlbumData and Persistence
- **Not easy to detect upfront in large systems**

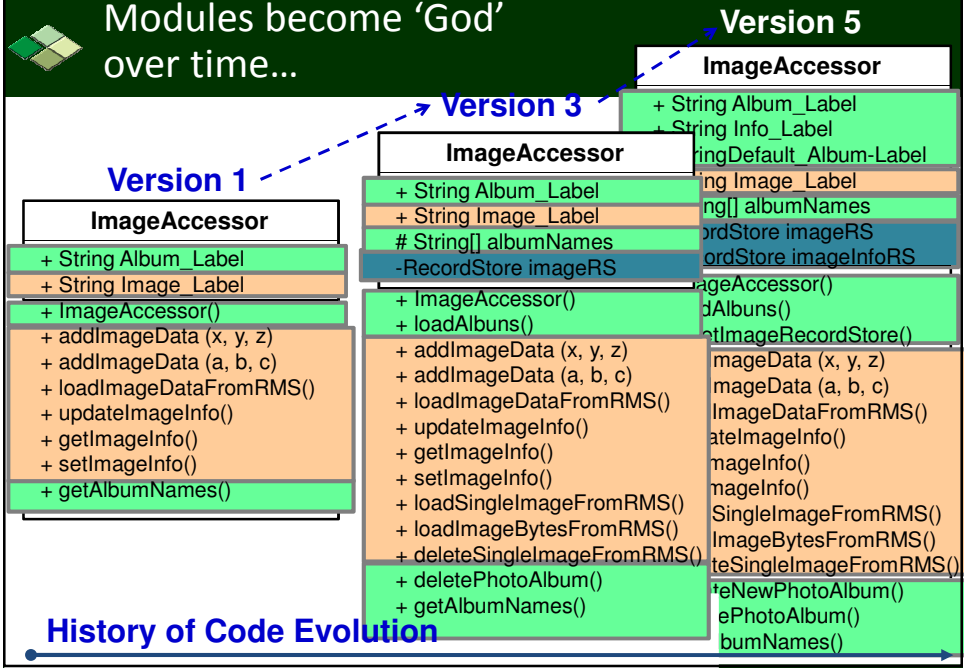


Example of a God Class…

- … It isn't not in the top list (25%) of modules:
  - with highest # of methods
  - with the lowest cohesion
  - …even changing the threshold
    - is not able to detect that the module addresses:
      - three implicit concerns
      - most of the **module changes** are due to the presence of AlbumData and Persistence

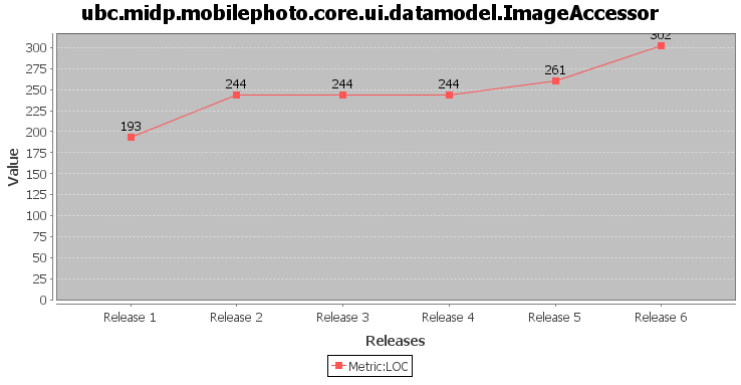


Modules become 'God' over time…



History-Sensitive Analysis: the Solution?

- 'Actual' God Classes tend to show increasing bad modularity properties *over time*: increasing lines of code (LOC), growing coupling, high cohesion, etc…



Challenge

- History-sensitive analysis – disadvantage:
  - may encourage late detection of code anomalies
  - not an adequate solution for congentinal or early-introduced code anomalies
- How to support continuous detection of code anomalies?

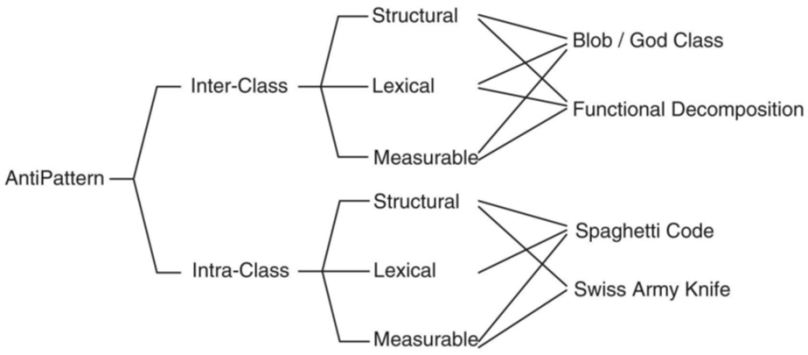
Code vs. Design Anomalies

- The difference is subtle
- Difference of design and code anomalies
  - code anomalies are localized in one or a few modules
  - design anomalies are the generalized *misuse of design* mechanisms in the system: e.g. inheritance
    - Functional Decomposition,

W.J. Brown, R.C. Malveau, W.H. Brown, H.W. McCormick III, and T.J. Mowbray, **Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis**, first ed. John Wiley and Sons, Mar. 1998.

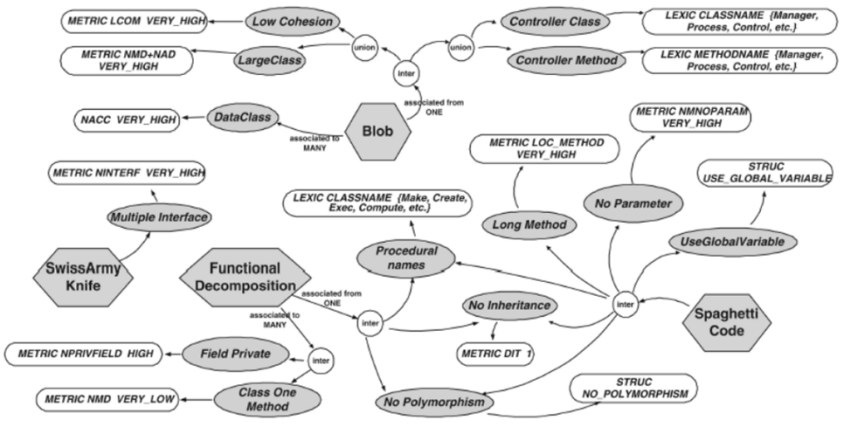
Classification of Design Anomalies

- By Moha et al, 2010



Naouel Moha, Yann-Gaël Guéhéneuc, Laurence Duchien, Anne-Françoise Le Meur: **DECOR: A Method for the Specification and Detection of Code and Design Smells.** IEEE Trans. Software Eng. 36(1): 20-36 (2010)

Relationships



Naouel Moha, Yann-Gaël Guéhéneuc, Laurence Duchien, Anne-Françoise Le Meur: **DECOR: A Method for the Specification and Detection of Code and Design Smells.** IEEE Trans. Software Eng. 36(1): 20-36 (2010)

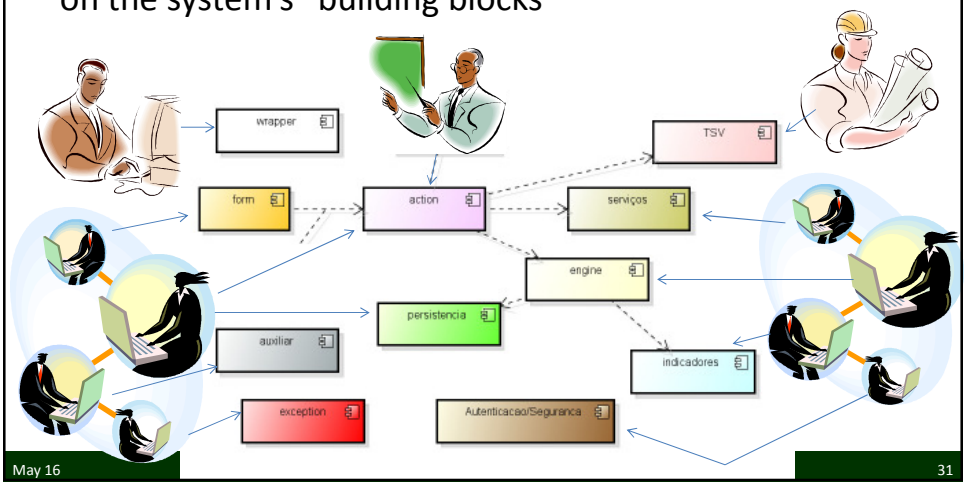
Types of Design and Code Anomalies

- They manifest in different artifacts produced in specific SE stages
  - code anomaly
    - e.g. class decomposition: *Fowler's catalogue*
  - design anomaly
    - e.g. *Brow's catalogue*, *Riel's catalogue*
  - architecture design anomaly
    - e.g. component-connector decomposition: *Medvidovic's catalogue*
    - architecture anomaly: drift problem – violation of a modularity principle
      - drift vs. erosion problems

Software has an "architecture" too!

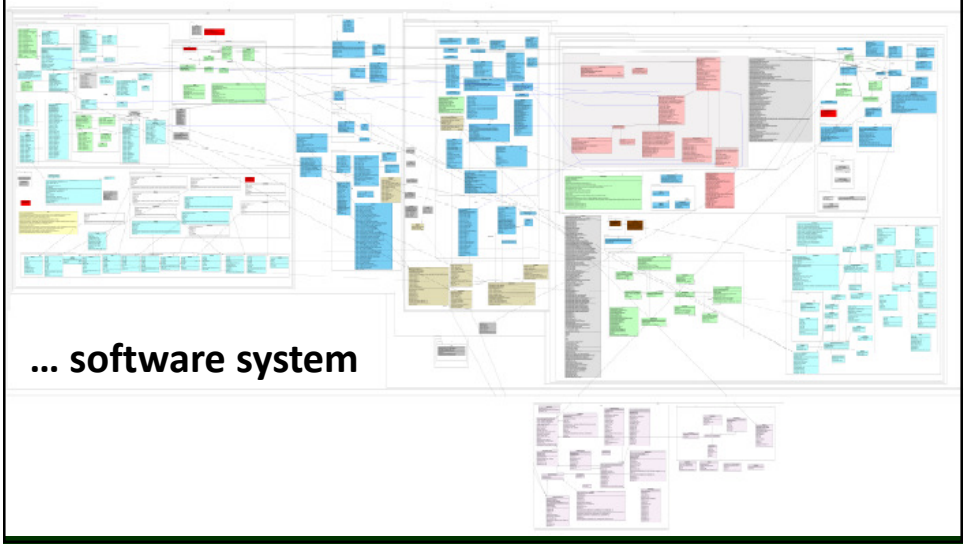
Intended architecture of a software system

- … defines how developers actually communicate and work on the system's "building blocks"

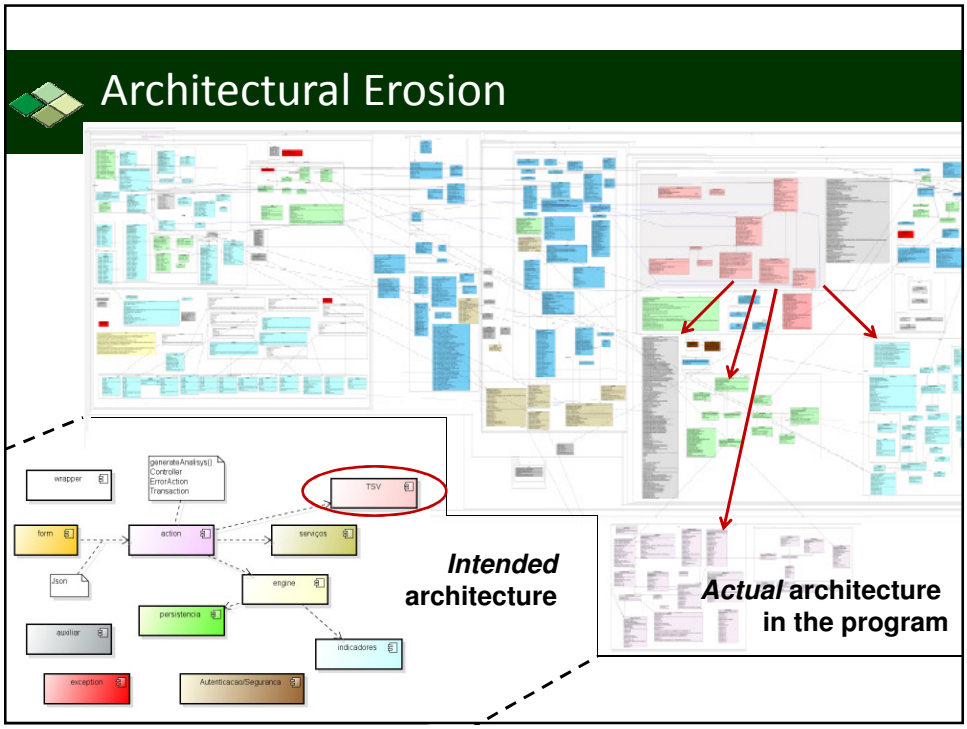


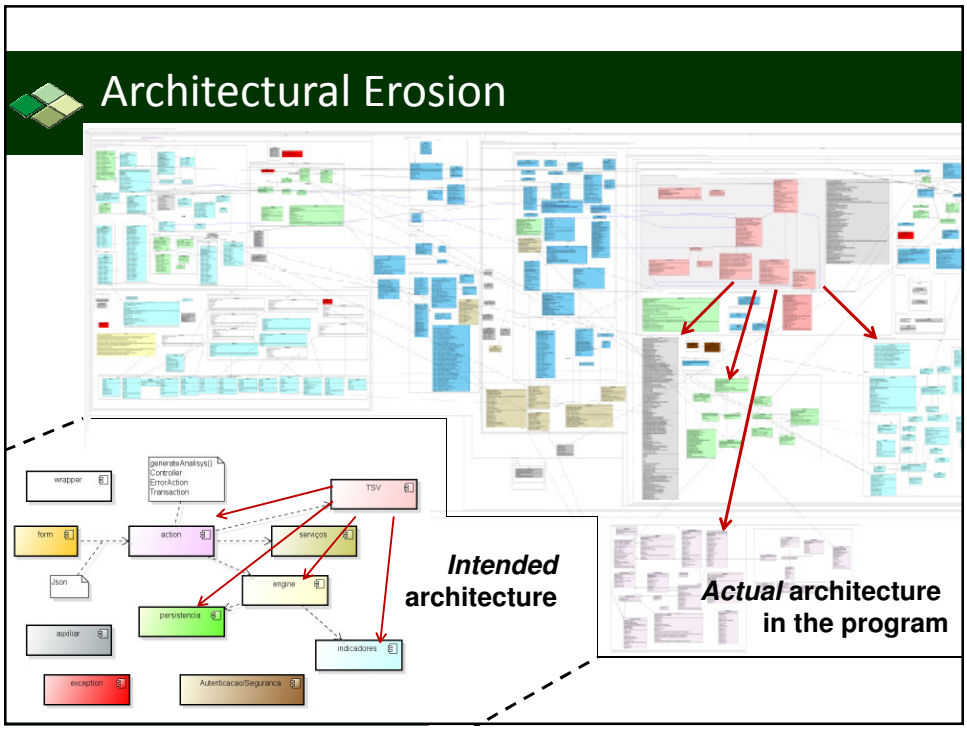
May 16 31

Actual architecture of the same…

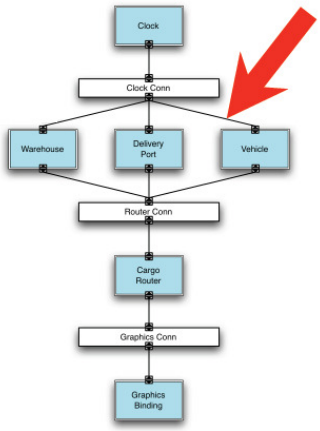


… software system

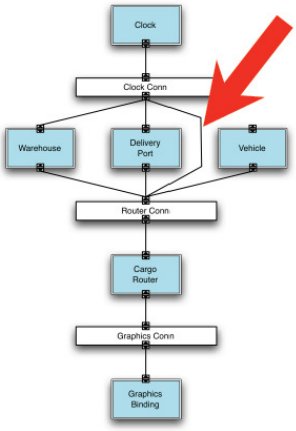

Architectural Erosion
wrapper
generateAnalisys()
Controller
ErrorAction
Transaction
TSV
form
action
serviços
Json
engine
persistencia
auxiliar
indicadores
exception
Autenticacao/Seguranca
Intended architecture
Actual architecture in the program


Architectural Erosion
wrapper
generateAnalisys()
Controller
ErrorAction
Transaction
TSV
form
action
serviços
Json
engine
persistencia
auxiliar
indicadores
exception
Autenticacao/Seguranca
Intended architecture
Actual architecture in the program

Architectural Erosion

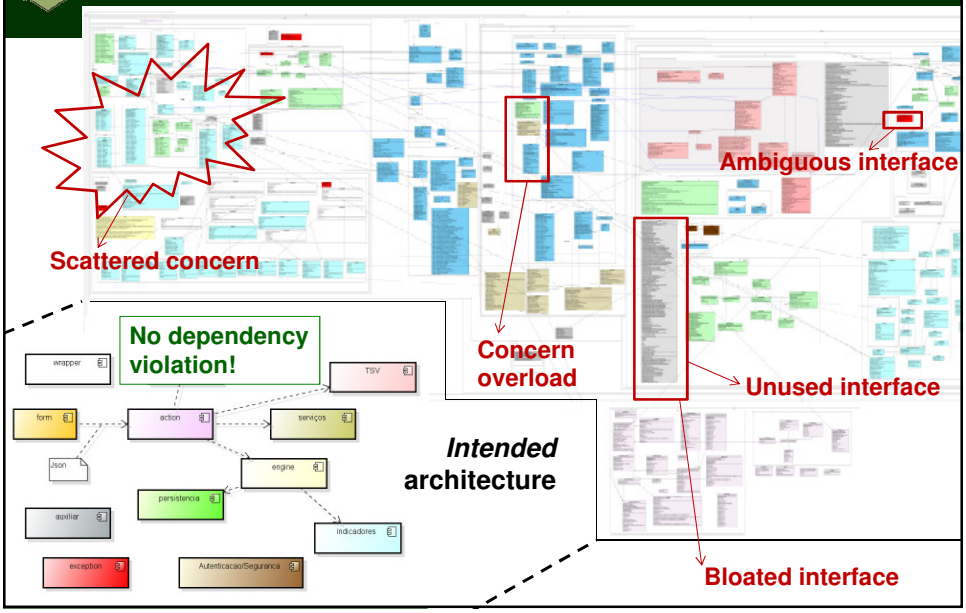


Intended architecture

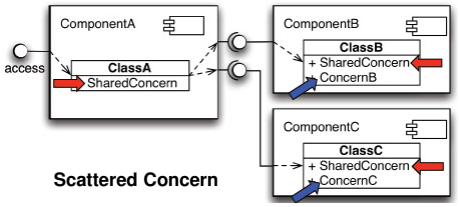


Actual architecture

Architectural Drift



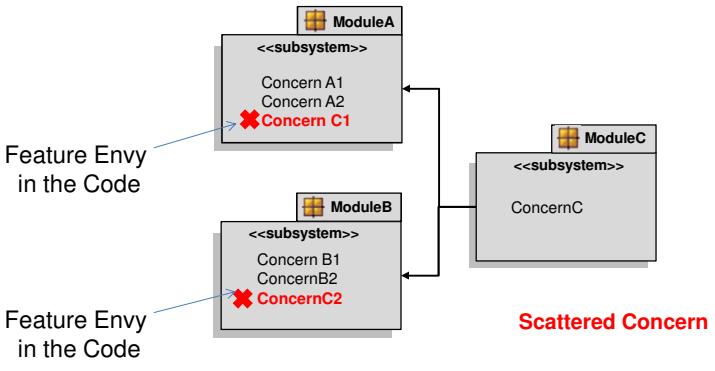
Architectural Drift – Scattered Concern



Violates:
- Separation of concerns
- SRP
- Low Cohesion
- High Coupling

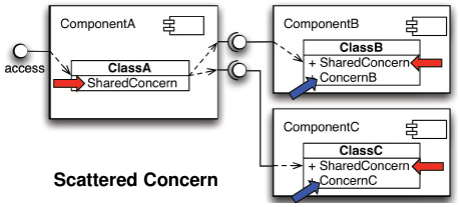
May 16 37

Scattered Concern vs. Code Anomalies



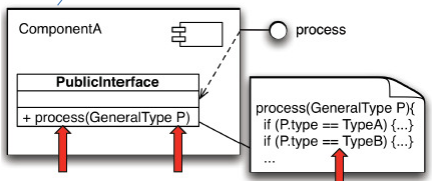
38

Architectural Drift – Ambiguous Interface



Scattered Concern

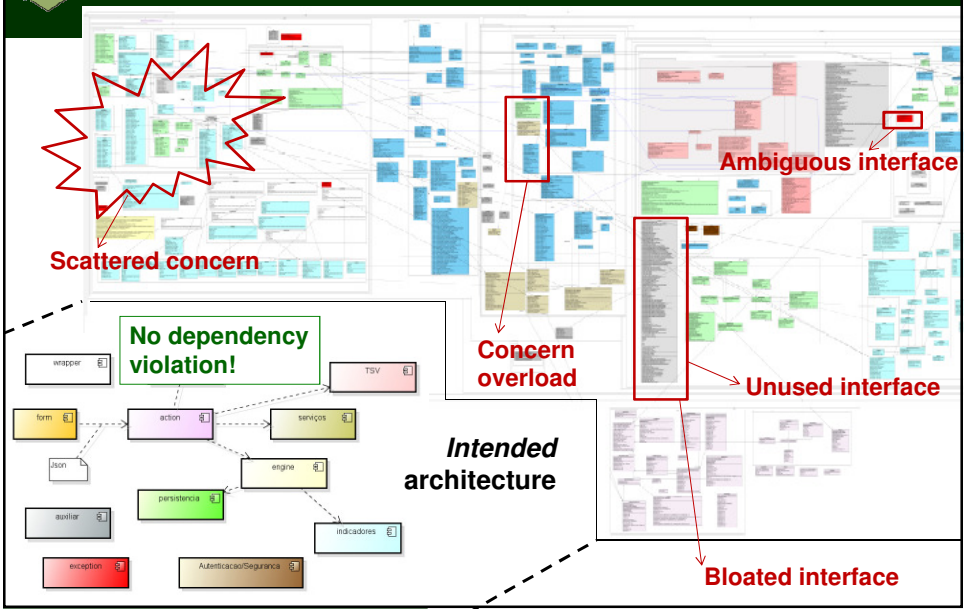
Violates:
- Explicit interface (which services?)
- OCP



Ambiguous Interface

May 16 39

Other Architectural Anomalies



Architecture Anomalies...

- ... are hard to detect since:
  - one need to rely on detailed architectural models, which are not available
  - architecture recovery techniques do not help
    - they retrieve only candidate names of components
    - detailed information about component interfaces is not retrieved
- Source code is often the only artefact available
  - are code anomalies indicators of architectural anomalies? do they serve as hints?

What is the relationship between them? Less understood...

- Architectural models are often not available or updated
- Can we detect architectural anomalies in the source code by observing code anomalies?

Architectural problems

Code anomalies