Software and systems engineering

Paulo Borba
Informatics Center
Federal University of Pernambuco

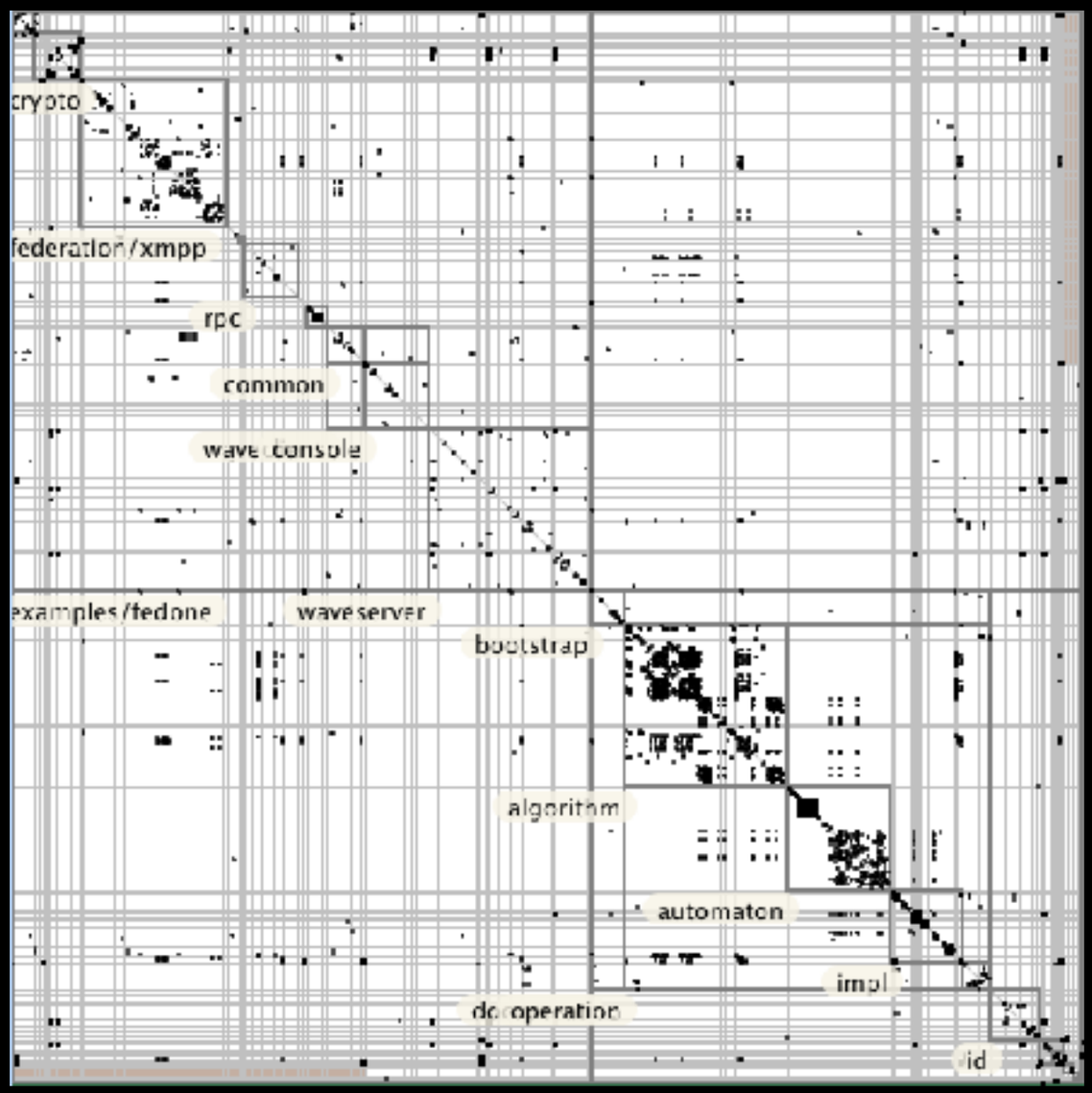
phmb@cin.ufpe.br ◈ twitter.com/pauloborba

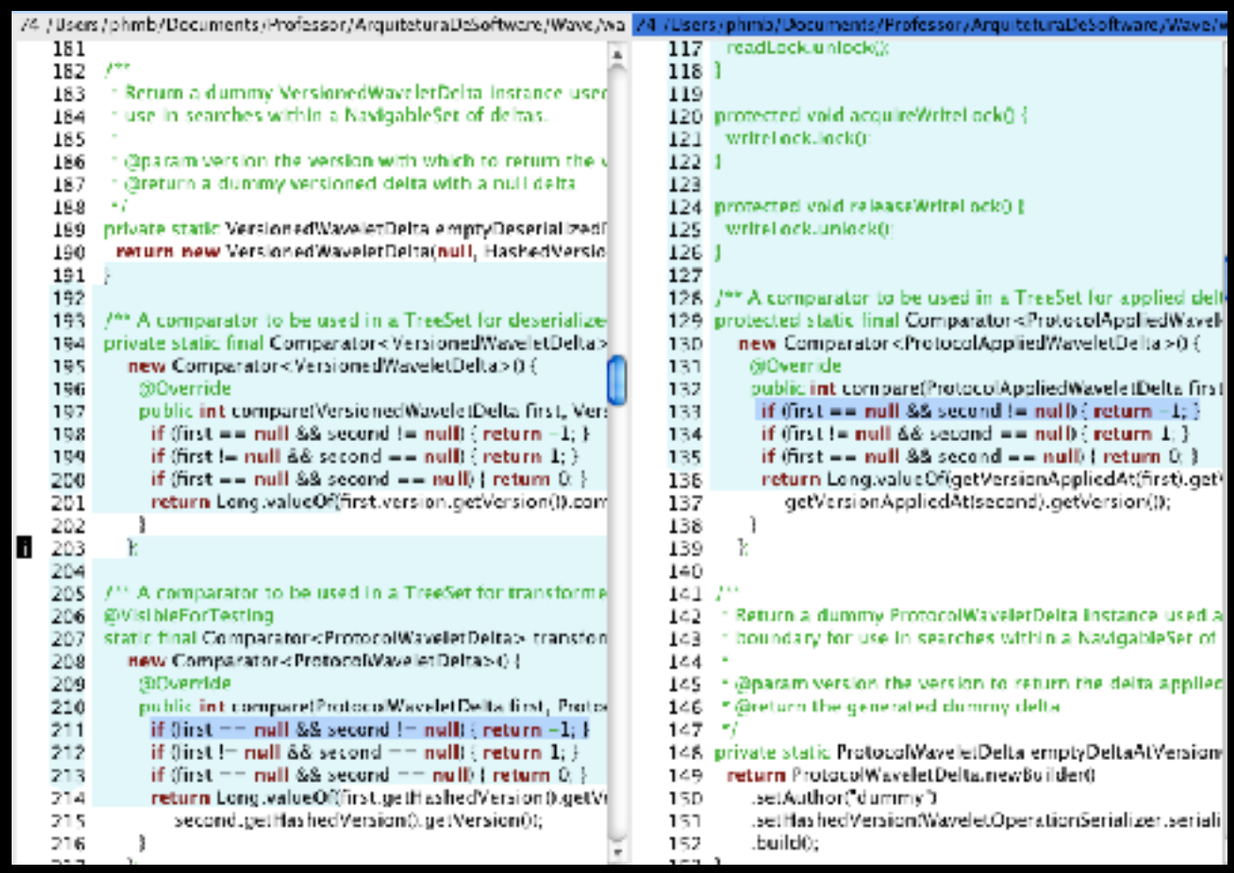
To do before class

- Watch videos
- Read chapter 9 in the textbook
- Send questions and opinions through slack

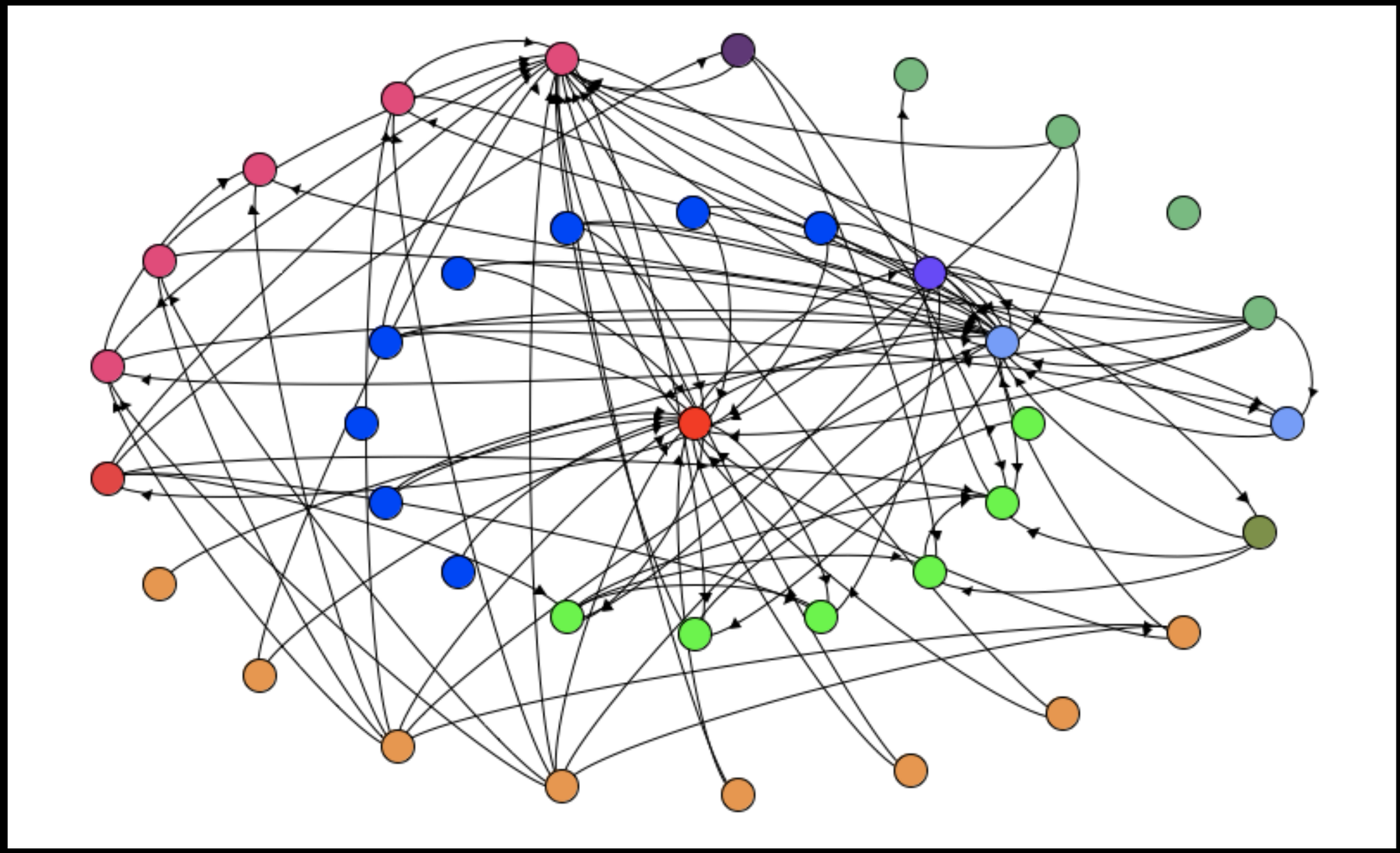
Refactoring

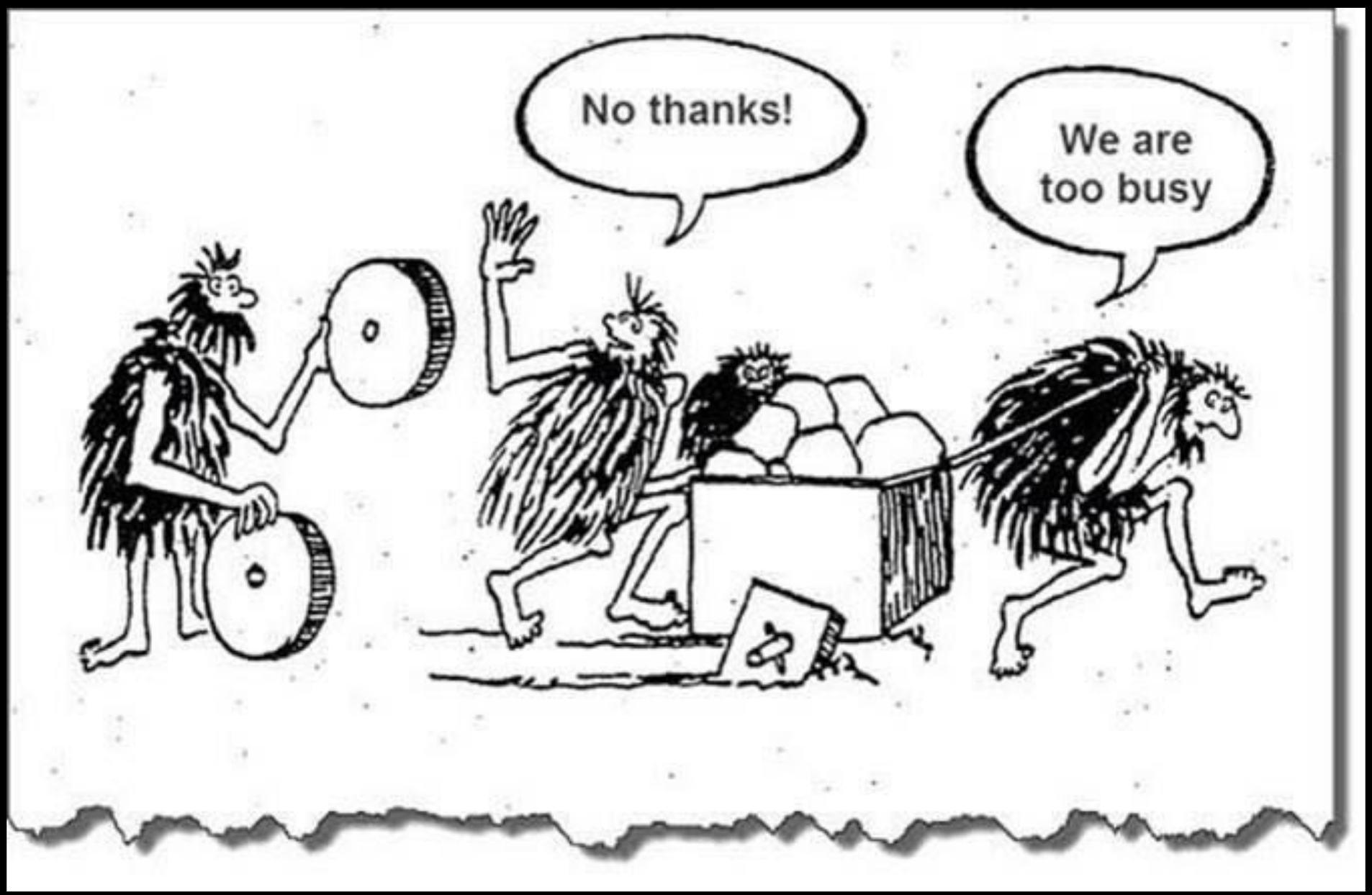
Reuse problems





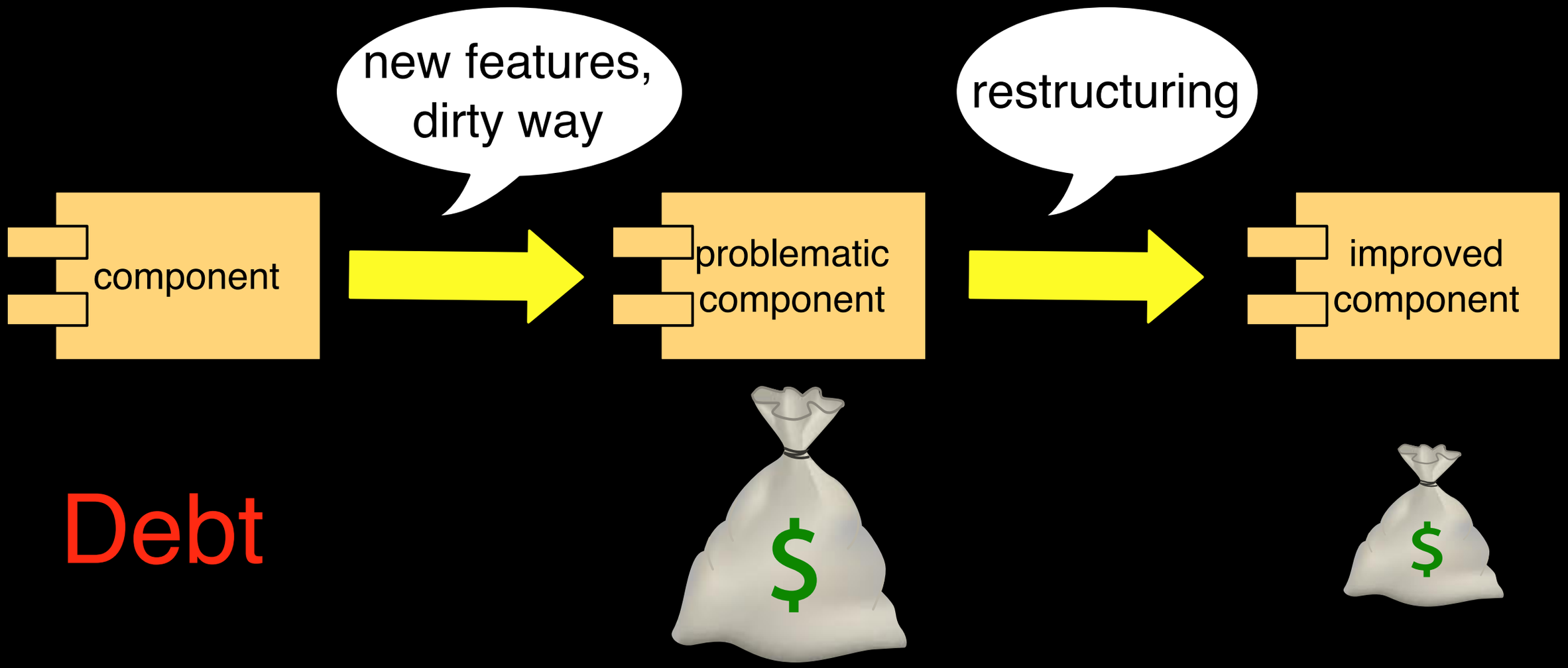
Modularity problems




No thanks!
We are too busy

How and
when to
solve them?

Technical debt



Debt

Maintenance costs more

Much of the early work in program restructuring was inspired by the need to reduce the cost of maintaining programs. It has been shown that the principle cost of any software development is the maintenance after the software is "done." A study of one Air Force system revealed that it cost $30 per line to develop and $4,000 per line to maintain over its lifetime [Boe75]. By analyzing the IBM OS/360 project, Belady and Lehman determined that the cost of a change rose exponentially with respect to a system's age. They attributed this rising cost to the decay of the software's structure [BL71, BL76]. Gerald Weinberg maintains a private list of the world's most expensive program errors. The top three errors involved the change of exactly one line of code [Wei83]. His theory as to why these errors happened is that since the actual change was so small, the programmers did not take the time to fully test the code or consider the ramifications of the change.

Practical Analysis for Refactoring, Don Roberts, 1999.

Lehman's first law

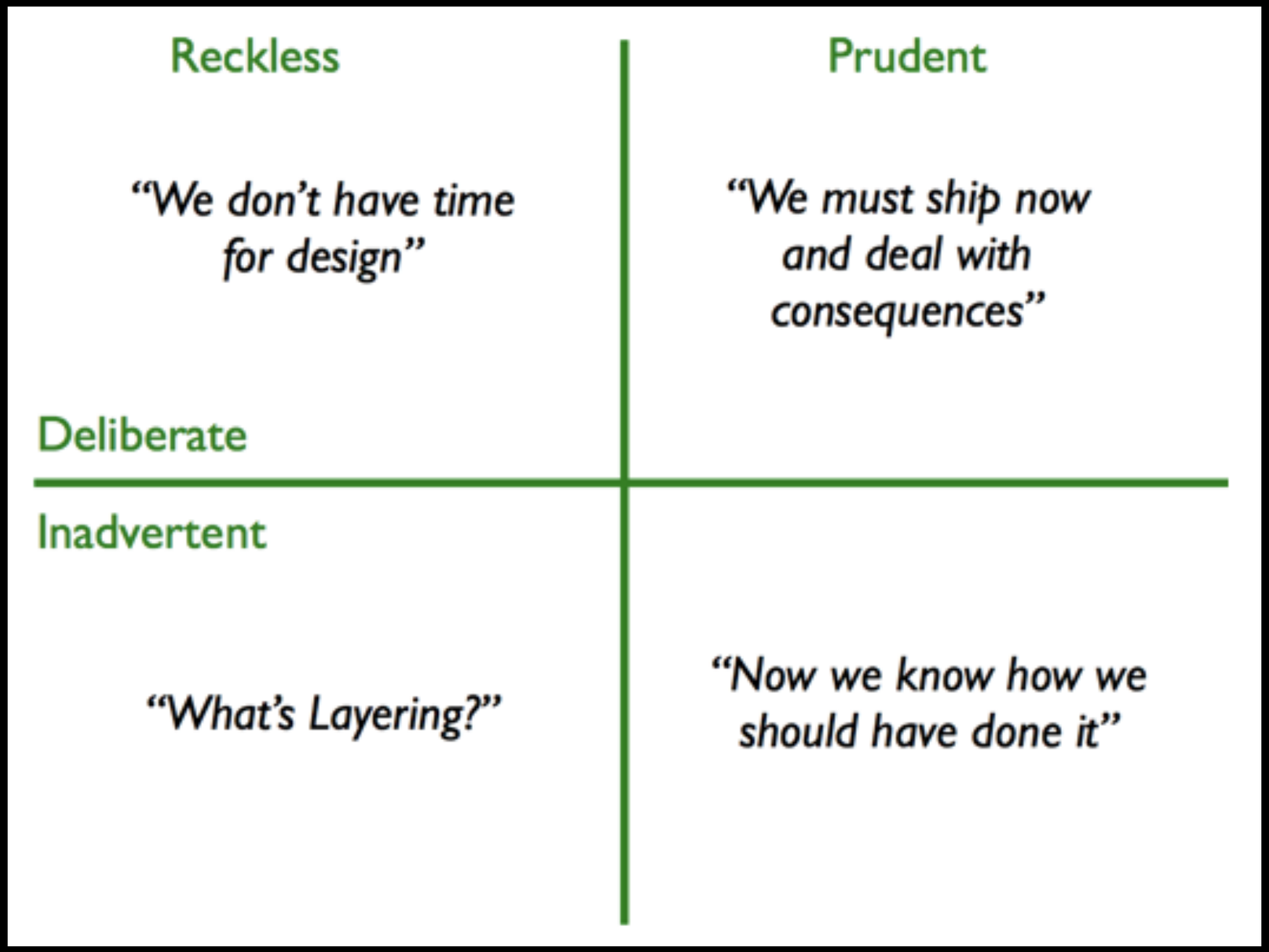
"A large program that is used undergoes continuing change or becomes progressively less useful"

"The change process continues until it is judged more cost effective to replace the system with a recreated version"

Lehman's second law

"As a large program is continuously changed, its complexity, which reflects deteriorating structure, increases unless work is done to maintain or reduce it"

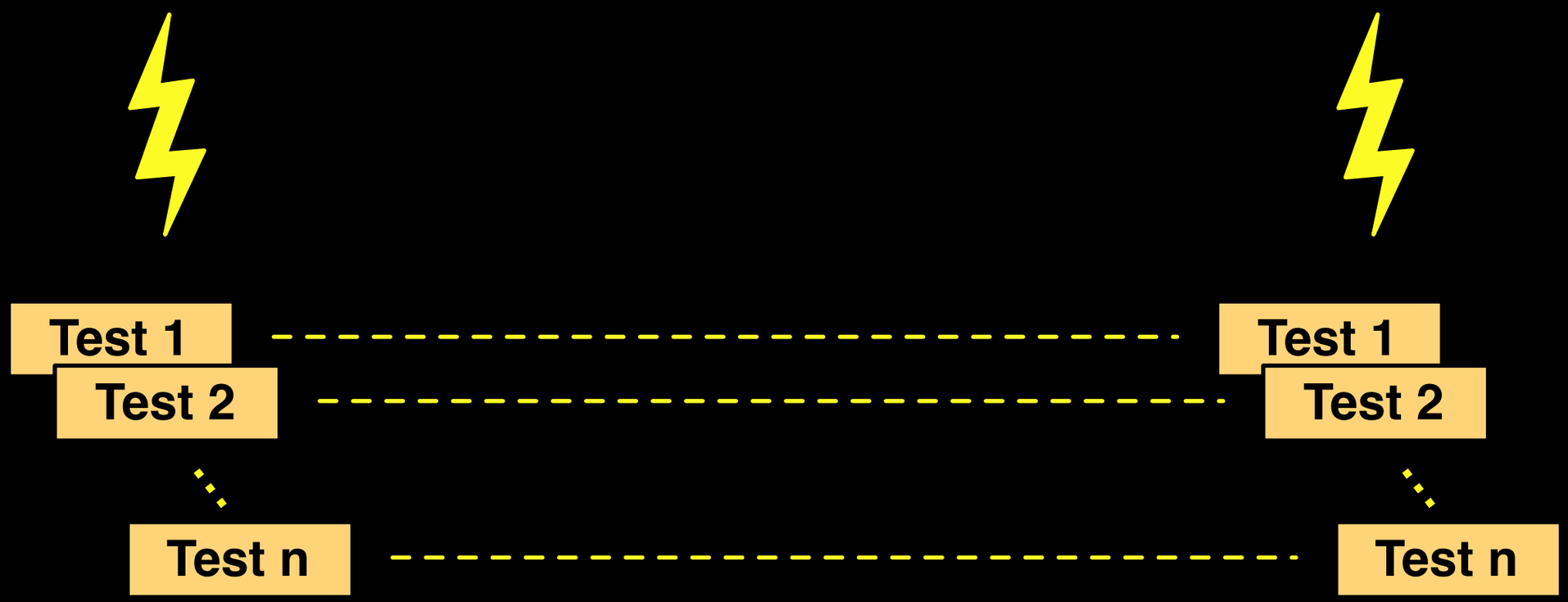
Technical debt quadrant



http://martinfowler.com/bliki/TechnicalDebtQuadrant.html

How to solve the problems and reduce the debt?

Preserving behavior



Refactorings are...

behavior-preserving source-to-source transformations that improve some internal quality factors

Reuse and modularity problem

```
if(objeto.getNome() == null ||
    objeto.getNome().equals("") ||
    objeto.getSobrenome() == null ||
    objeto.getSobrenome().equals("") ||
    objeto.getTipo() == null ||
    objeto.getTipo().equals("") ||
    ...
```

Extract method refactoring

```
boolean nullOrEmpty(Membro s) {
    return s.getNome() == null ||
           s.getNome().equals("");
}
```

```
boolean nullOrEmpty(String s) {
    return s == null ||
           s.equals("");
}
```

Debt reduced

```
if(nullOrEmpty(objeto.getNome()) ||
    nullOrEmpty(objeto.getSobrenome()) ||
    nullOrEmpty(objeto.getTipo()) ||
    ...
```

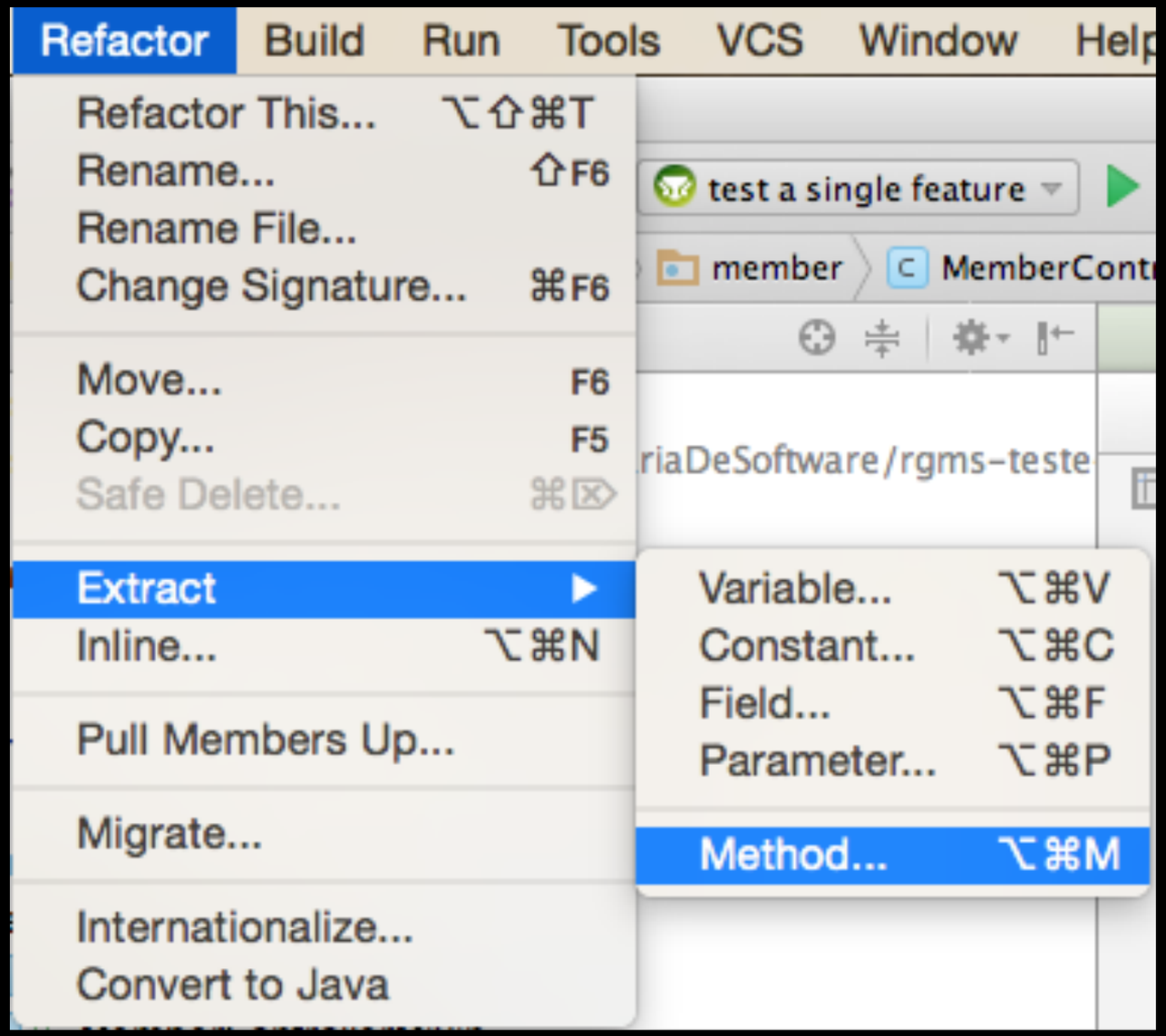
```
if(objeto.getNome() == null ||
   objeto.getNome().equals("") ||
   objeto.getSobrenome() == null ||
   objeto.getSobrenome().equals("") ||
   objeto.getTipo() == null ||
   objeto.getTipo().equals("") ||
   ...
```

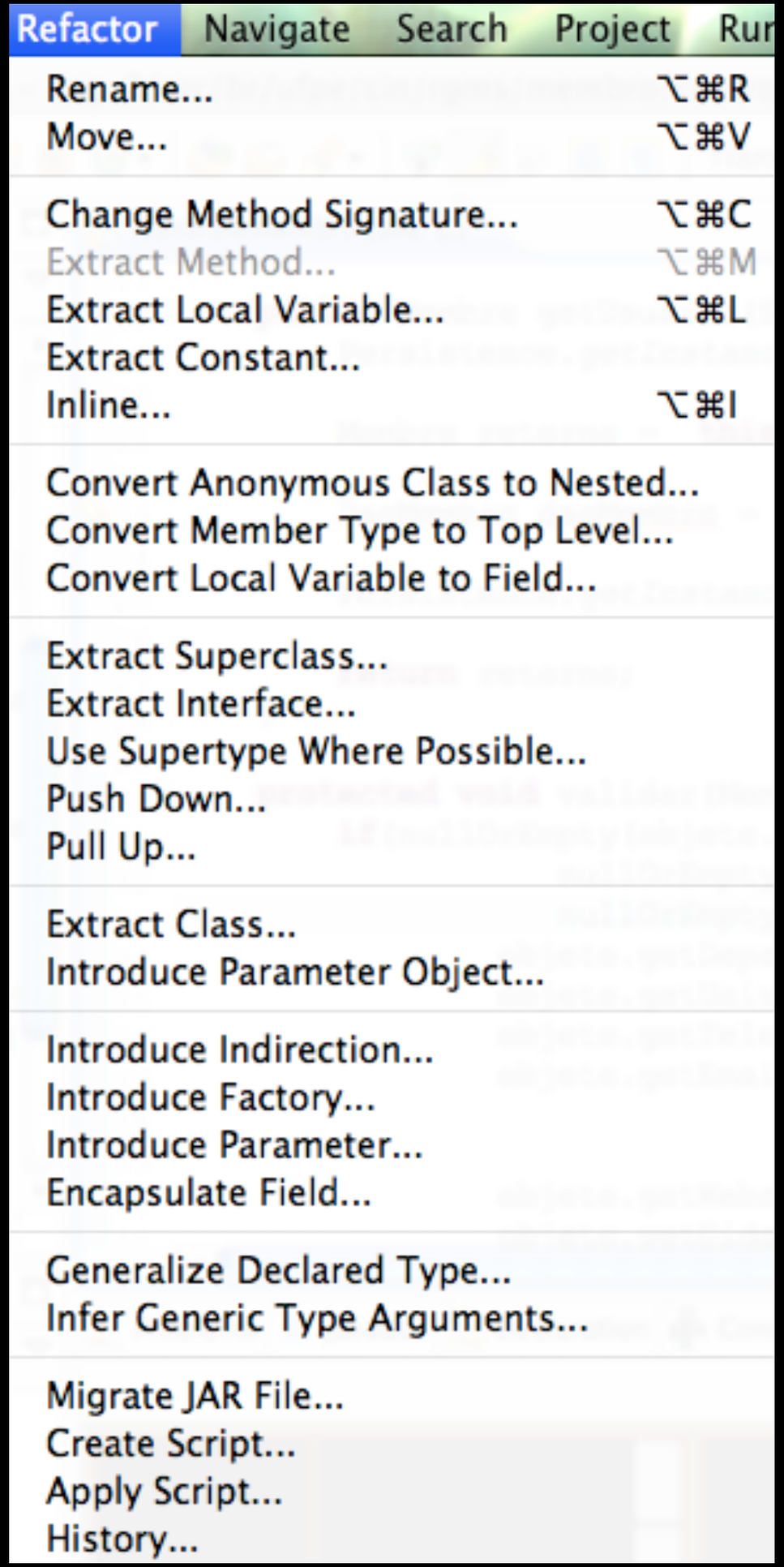
Debt eliminated

```
if(invalidStringFields(objeto)
    ...
```

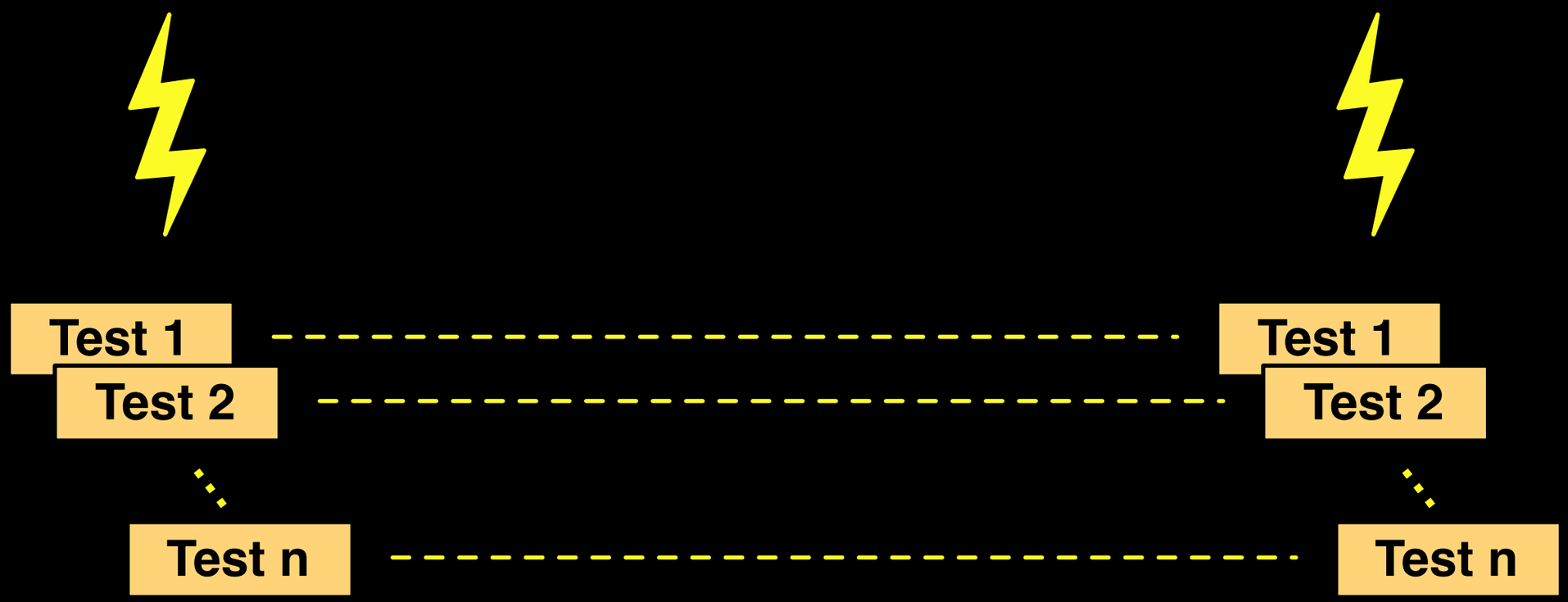
```
if(nullOrEmpty(objeto.getNome()) ||
   nullOrEmpty(objeto.getSobrenome()) ||
   nullOrEmpty(objeto.getTipo()) ||
   ...
```

Automatic refactorings





Tools give no behavior preservation guarantee

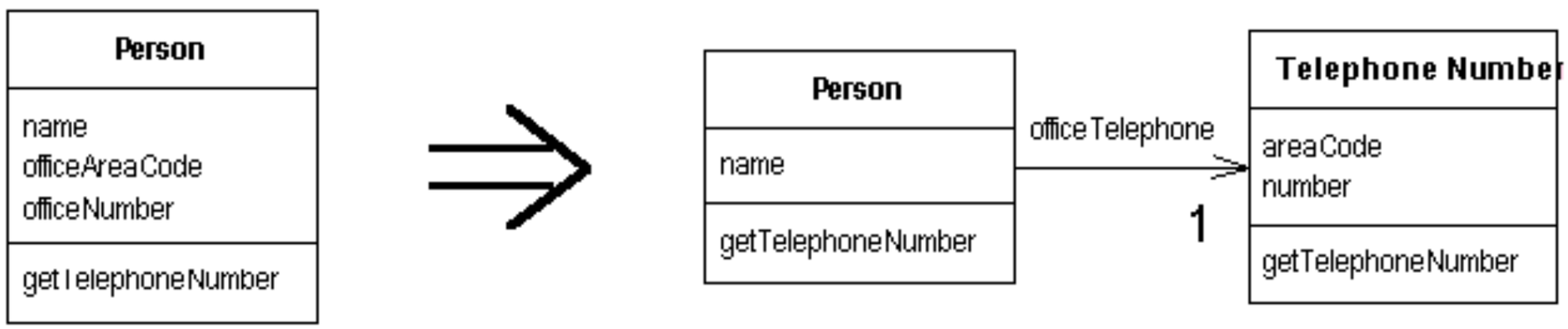


Refactoring catalogue

Extract Class

You have one class doing work that should be done by two.

Create a new class and move the relevant fields and methods from the old class into the new class.



http://www.refactoring.com/catalog/index.html

Black belt parametrization!

Smells good?

functions as parameters

```
artigos.each{|a|
    map = {}
    a.elements.each("DADOS-BASICOS-DO-ARTIGO"){|d|
        atts = d.attributes
        map["TITULO-DO-ARTIGO"] = atts["TITULO-DO-ARTIGO"]
        map["ANO-DO-ARTIGO"] = atts["ANO-DO-ARTIGO"]
    }
    a.elements.each("DETALHAMENTO-DO-ARTIGO"){|d|
       atts = d.attributes
       map["TITULO-DO-MEIO"] = atts["TITULO-DO-MEIO"]
       map["VOLUME"] = atts["VOLUME"]
       map["FASCICULO"] = atts["FASCICULO"]
        map["PAGINA-INICIAL"]  = atts["PAGINA-INICIAL"]
        map["PAGINA-FINAL"]  = atts["PAGINA-FINAL"]
    }
...
```

Not only for articles...

```
...
a.elements.each("AUTORES"){|author|
      map["AUTORES"] = (if map["AUTORES"] then
                          map["AUTORES"]
                        else [] end) +
             [author.attributes["NOME-PARA-CITACAO"]]
    }
}
```

XML structure as parameter

```
structure = {
    "DADOS-BASICOS-DO-ARTIGO" =>
      ["TITULO-DO-ARTIGO", "ANO-DO-ARTIGO"],
   "DETALHAMENTO-DO-ARTIGO" =>
      ["TITULO-DO-MEIO", "VOLUME", "FASCICULO",
       "PAGINA-INICIAL", "PAGINA-FINAL"],
   "AUTORES" =>
      ["*", "NOME-PARA-CITACAO"]}
   }
```

Abstracting structure details

```
structure.keys.each {|e|
   a.elements.each(e) {|d|
      atts = d.attributes
      if structure[e].include?("*") then
         (structure[e] - ["*"]).each {|att|
            map[e] = (if map[e] then map[e] else [] end) +
                     [atts[att]]
          }
      else
         structure[e].each {|att|
            map[att] = atts[att]
         }
      end
   }
}
```

Improvement is more often needed than not, so abstract to see it!

How to choose refactoring targets?

Bad smells based on...

- Values
- Principles
- Patterns

Strategy

- Identify problem
- Any pattern for problem in the given context?
- New solution based on values and principles?
- Apply appropriate refactorings
- Test

Checklist

- Design and implementation conforms to discussed principles and patterns
- Most well known refactoring have been applied

Take notes,

now!

Refactoring research at CIn

- Refactoring of software product lines: Paulo, Leopoldo
- Formal refactoring: Márcio, Augusto

Hands-on!

Check assignment

To do after class

- Answer questionnaire (check classroom assignment), study correct answers
- Finish exercise (check classroom assignment), study correct answers
- Read, again, chapter 9 in the textbook
- Evaluate classes (check classroom assignment)
- Study questions from previous exams

To do after class, optional

- estudar material (definição, quadrante) sobre débito técnico
- estudar catálogo de refactorings, e assistir vídeo se você não é familiar com a noção de refactoring
- ler resumos ou assistir vídeos do debate Is TDD dead?

Questions from previous exams

- Explique (a) o que é teste de regressão e (b) porque eles são úteis para atividades de refatoração.
- Cite (a) um ``mau cheiro" associado à refatoração “extract class”, e (b) explique qual a mudança sugerida por essa refatoração.
- Explique brevemente quais as vantagens de refatorar o código e porque algumas empresas não realizam esta atividade rotineiramente.
- Cite um ``mau cheiro" associado à refatoração extract method, e explique qual a mudança sugerida por essa refatoração.

Software and systems engineering

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ◈ twitter.com/pauloborba