Comprendre un Logiciel en regardant son histoire

Apprendre du passé pour répondre aux questions du présent et du futur

Pr. Xavier Blanc – Bordeaux Université – Institut Universitaire de France

Plan

- Problématique du présent et du futur
- Analyser plusieurs présents pour mieux comprendre
- Apprendre du passé, observer les propriétés dynamiques
- Analyse de dépôt, quelle méthode et quels outils

Problématique du présent et du futur

Coder bien, coder mieux, améliorer sans cesse le code …

Coder (seulement) ou Coder Bien

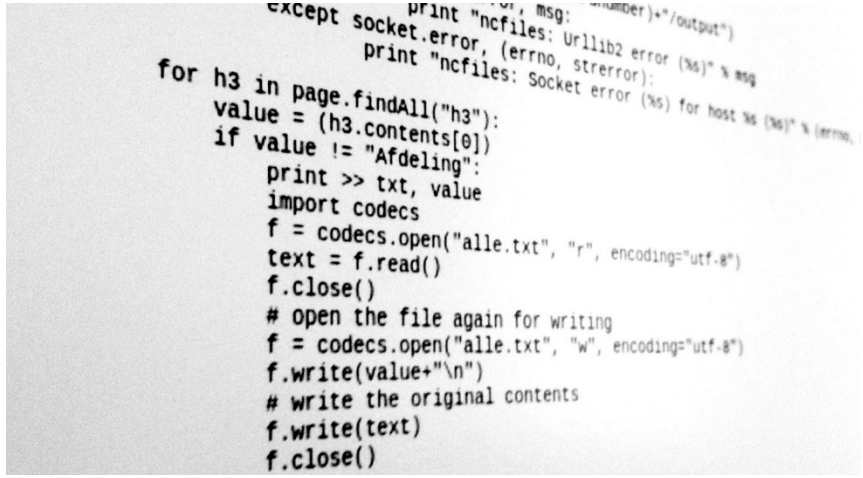
- Coder (seulement) pour répondre aux exigences
  - Coder jusqu'à ce que ça marche !
- Coder bien
  - Rapide, pas de bug, ...
  - Maintenable, portable

Coder Bien, à quoi ca sert ?

‘‘ Le code n’est pas utile en soi ’’

- Des Mensonges
  - Code Généré à 100%
  - Tout le monde peut coder
- La Vérité
  - Le code est ce qui reste
  - Les détails sont dans le code
  - Les DSL, les outils, c’est du code

=> Bien coder pour durer !



```
except socket.error, (errno, strerror):
        print "ncfiles: Urllib2 error (%s)" % msg
        print "ncfiles: Socket error (%s) for host %s (%s)" % (errno,
for h3 in page.findAll("h3"):
    value = (h3.contents[0])
    if value != "Afdeling":
        print >> txt, value
        import codecs
        f = codecs.open("alle.txt", "r", encoding="utf-8")
        text = f.read()
        f.close()
        # open the file again for writing
        f = codecs.open("alle.txt", "w", encoding="utf-8")
        f.write(value+"\n")
        # write the original contents
        f.write(text)
        f.close()
```

Il était une fois …

- Une société met sur le marché une nouvelle application qui fait sensation. Les utilisateurs sont ravis. Le marché est en pleine expansion.
- Pourtant, plus les années avancent, plus il est difficile de suivre le rythme des évolutions. La deuxième version était un succès. La troisième était plus mitigée. La quatrième arrive en retard et contient plusieurs bugs. L'ambiance interne se dégrade. L'image de marque disparait.
- Le marché pousse mais les nouvelles versions ne sortent plus. La société dépose le bilan.

Que s'est-il passé? La qualité du code a été négligé !

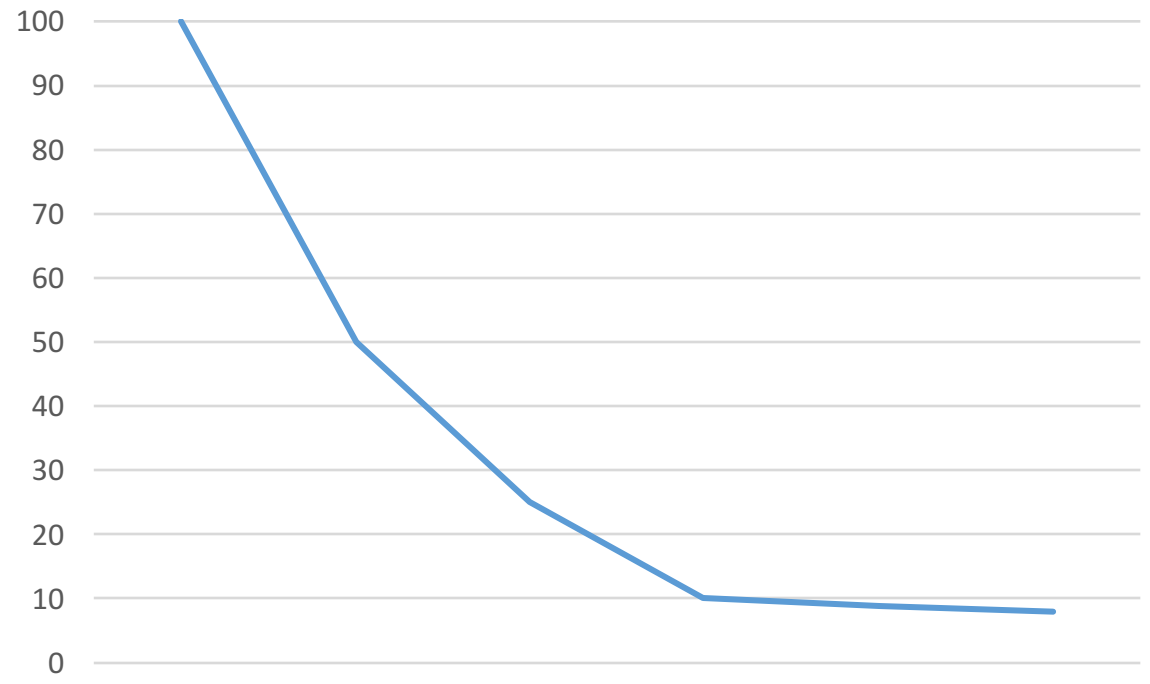
Pourquoi néglige-t-on le code?

- Tout le monde savait que le code était négligé. Pourquoi n'a-t-on rien fait ?
  - Il fallait aller vite. Il fallait livrer tôt.
  - Si on n'avait pas livré, le client aurait été mécontent *et le boss furieux*.
  - Améliorer le code aurait pris trop de temps, pour aucun bénéfice
  - Franchement, ce n'était plus notre problème

Et maintenant que la société n'existe plus, est-ce de bonnes raisons ?

Productivité et code négligé

- La productivité chute drastiquement si la qualité du code est négligée
  - Le nombre de service / version
  - Coût de développement
  - …
- Ecart important avec ceux qui ne négligent pas le code (McKinsey)
  - +230% de services
  - -83% de bug



Et si …

- On avait embauché une équipe de ranger/coder pour tout nettoyer et rendre notre application parfaite ?
- On avait passé un mois à tout réparer, à enlever toutes les erreurs ?
- On avait utilisé la nouvelle architecture 2.0 avec le nouveau framework ?

Ces solutions ne sont que temporaires et ne résolvent pas le problème

Faute Professionnelle ?

- Pourquoi le code se degrade si rapidement ?
- Pourquoi n'est-il pas possible de le réparer rapidement ?
- Faut-il blamer le rythme, les exigences ?
- Faut-il blamer les managers qui ne veulent pas de bon code ?

Non, la faute vient de nous, nous ne sommes pas professionnels.

Apprendre à Coder Bien

- Le problème c’est que coder bien est un apprentissage long et difficile
- Même si des principes existent, il ne suffit pas de les lire
- Il faut coder, appliquer ces principes et apprendre à les dompter
- Pour coder bien il faut déjà être capable d’identifier du mauvais code
- Puis il faut sans arrêt essayer d’améliorer le code, sans arrêt !

Coder Bien

- Les grands développeurs parlent tous de bon et de mauvais code
- Pour la plupart,
  - un bon code est lisible, facilement compréhensible
  - Alors qu'un mauvais code est difficile à lire, à comprendre
- La qualité d'une application (pas de bug, rapide, belle, etc) sera décuplée si le code et lisible et compréhensible

De ses enseignements, il existe des principes

Questions

- Connaissez vous des principes de programmation ?
- Doutez vous de certains principes ?
- Pensez-vous que les industriels respectent ces principes ?
- Savez vous juger de l'importance de ces principes ?

Analyser plusieurs présents pour mieux comprendre

Ou comment vérifier des principes de programmation

Principe de taille : horizontal

- Small, Smaller
  - 60 lignes au max pour une classe
- Aéré
  - Donner de l'air entre les méthodes, les morceaux de code, etc.
- Dense
  - Les concepts liés doivent être rapprochés
- Distance
  - Les fonctions liées doivent être proches. L'appelant avant l'appelé.
- Variable
  - Les variables doivent être déclarées au plus proche de leur utilisation
  - Les membres de la classe doivent être déclarés en haut
- Ordre
  - Le sens de la lecture

Principe de taille : vertical

- Small, Smaller
  - 80, 100, 120 caractères
- Aéré
  - Donner peu d'air (conventions)
- Alignement
  - Ne pas aligner
- Indentation
  - Important. Préciser les conventions

Apprendre à supprimer les défauts

Coder bien nécessite

- d'identifier les défauts dans le code
- … et de savoir comment y faire face

=> Amélioration continue

Défaut vs Bug

- Un défaut rend le code moins lisible
- Le code sera donc plus difficile à maintenir, à optimiser, à améliorer, à corriger, etc.
- Un défaut n'est pas forcément la source d'un bug

- Un bug dans le code est la cause d'une erreur dans l'application
- Les conséquences d'un bug sont donc mesurables
- La sévérité du bug est calculée en fonction de l'impact qu'il cause

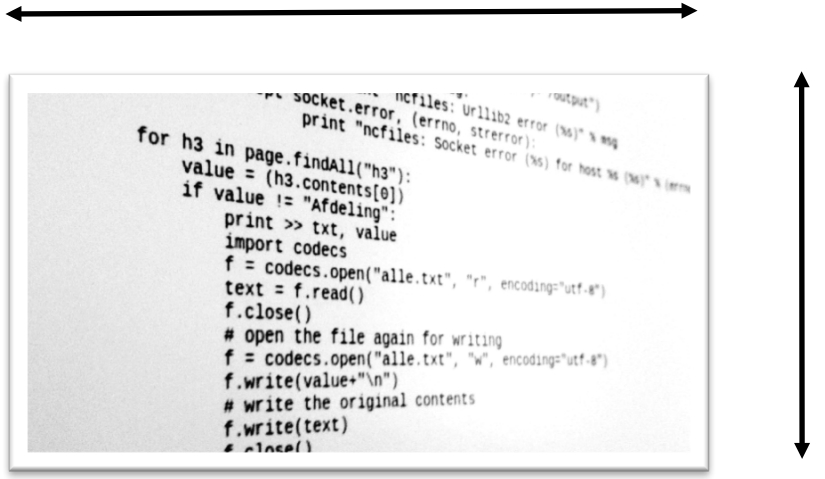
=> plus il y a de défauts, plus il y a de bugs

Bad Smell : identifier les défauts

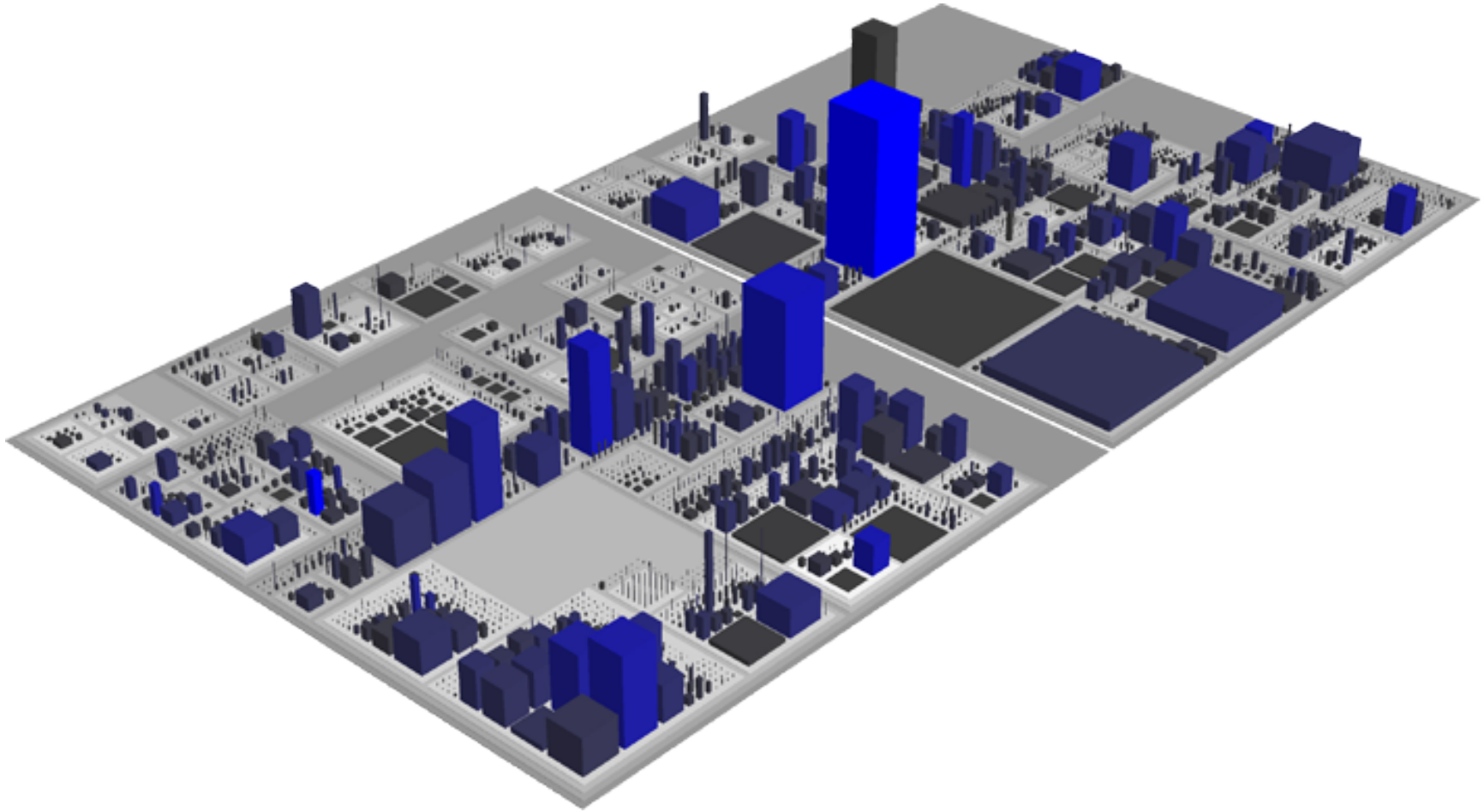
- Les Bad Smells sont des indices pour identifier des défauts potentiels
- Calculables automatiquement les Bad Smells pointent les éléments du code à surveiller
- L'objectif des Bad Smells est d'offrir une estimation quantitative du nombre de défauts
- Attention, l'absence de Bad Smell ne veut pas dire qu'il n'y a pas de défaut

Taille

- La taille du code est un très bon indicateur des défauts
- Le code propre doit être petit, donc un grand code contient probablement des défauts
- Les Bad Smell de taille
  - LOC (Ligne de Code)
  - Largeur des lignes
  - Nombre de classes,
  - Nombre de fonctions,
  - etc.



Wettel Lanza 07



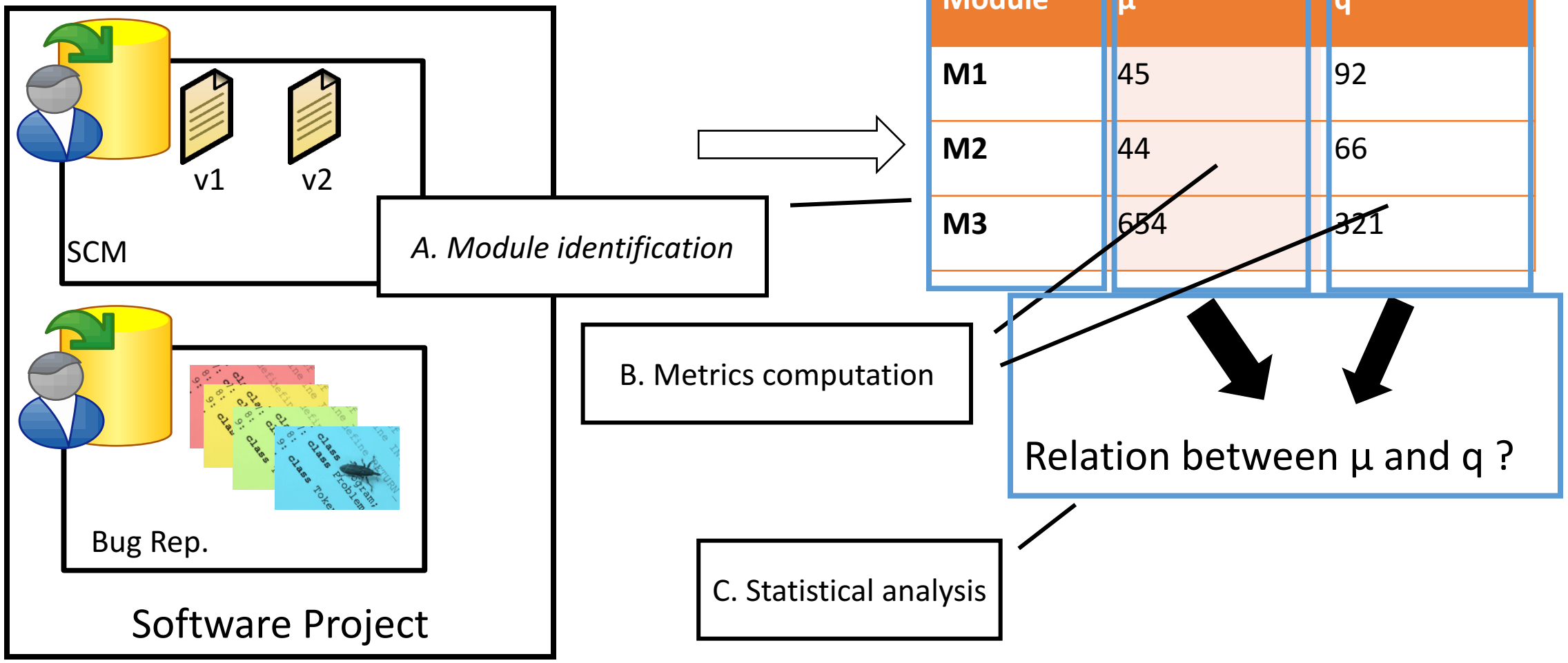
Cas concret

- Mesurer le nombre de lignes de code de chaque classe (fichier)
- Réalisez cette mesure sur plusieurs projets (10K projets)
- Identifiez les seuils communs

Application avec Diggit

- Préparation:
  - Construisez un dossier de travail « work_dir »
  - Construisez un dossier pour stocker les analyses
    - work_dir/plugins/analysis
  - Ajouter y les analyses qui sont dans http://www.labri.fr/perso/xblanc/nice/
- Docker
  - docker run --rm -v work_dir:/diggit xblanc/dgit init
  - docker run --rm -v work_dir :/diggit xblanc/dgit so add https://github.com/<votre_projet>
  - docker run --rm -v work_dir:/diggit xblanc/dgit cl p
  - docker run --rm -v work_dir:/diggit xblanc/dgit an a test_cloc
  - docker run --rm -v work_dir:/diggit xblanc/dgit an p

Metrics validation



Nagappan Ball Zeller 06

- Corrélation Métriques / Bug pour identifier les composants critiques
- L'objectif étant de focaliser la maintenance sur ces composants

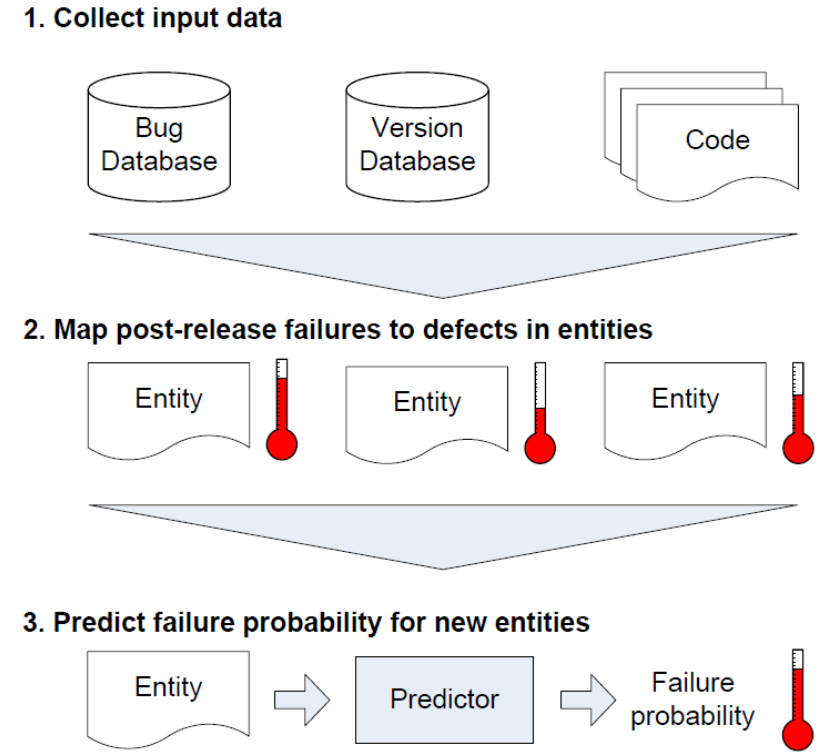


Figure 1. After mapping historical failures to entities, we can use their complexity metrics to predict failures of new entities.

Corrélation

Metric	Description		Correlation with post-release defects of *M*				
			A	B	C	D	E
Module metrics — correlation with metric in a module *M*							
Classes	# Classes in *M*		**0.531**	**0.612**	**0.713**	0.066	0.438
Function	# Functions in *M*		0.131	**0.699**	**0.761**	0.104	**0.531**
GlobalVariables	# global variables in *M*		0.023	**0.664**	**0.695**	0.108	0.460
Per-function metrics — correlation with maximum and sum of metric across all functions *f*() in a module *M*							
Lines	# executable lines in *f*()	Max	-0.236	**0.514**	**0.585**	**0.496**	**0.509**
		Total	0.131	**0.709**	**0.797**	0.187	**0.506**
Parameters	# parameters in *f*()	Max	-0.344	0.372	**0.547**	0.015	0.346
		Total	0.116	**0.689**	**0.790**	0.152	0.478
Arcs	# arcs in *f*()'s control flow graph	Max	-0.209	0.376	**0.587**	0.527	0.444
		Total	0.127	**0.679**	**0.803**	0.158	**0.484**
Blocks	# basic blocks in *f*()'s control flow graph	Max	-0.245	0.347	**0.585**	0.546	0.462
		Total	0.128	**0.707**	**0.787**	0.158	0.472
ReadCoupling	# global variables read in *f*()	Max	-0.005	**0.582**	**0.633**	0.362	0.229
		Total	-0.172	**0.676**	**0.756**	0.277	0.445
WriteCoupling	# global variables written in *f*()	Max	0.043	**0.618**	**0.392**	0.011	0.450
		Total	-0.128	**0.629**	**0.629**	0.230	0.406
AddrTakenCoupling	# global variables whose address is taken in *f*()	Max	0.237	**0.491**	**0.412**	0.016	0.263
		Total	0.182	**0.593**	**0.667**	0.175	0.145
ProcCoupling	# functions that access a global variable written in *f*()	Max	-0.063	**0.614**	**0.496**	0.024	0.357
		Total	0.043	**0.562**	**0.579**	0.000	0.443
FanIn	# functions calling *f*()	Max	0.034	**0.578**	**0.846**	0.037	**0.530**
		Total	0.066	**0.676**	**0.814**	0.074	**0.537**
FanOut	# functions called by *f*()	Max	0.197	0.360	**0.613**	0.345	0.465

Dépend des métriques mais aussi du logiciel

Combiner les métriques

Table 5. Regression models and their explanative power

Project	Number of principal components	% cumulative variance explained	R^2	Adjusted R^2	F - test
A	9	95.33	**0.741**	0.612	5.731, $p < 0.001$
B	6	96.13	**0.779**	0.684	8.215, $p < 0.001$
C	7	95.34	**0.579**	0.416	3.541, $p < 0.005$
D	7	96.44	**0.684**	0.440	2.794, $p < 0.077$
E	5	96.33	**0.919**	0.882	24.823, $p < 0.0005$

Mais …

Table 6. Predictive power of the regression models in random split experiments

Project	Correlation type	Random split 1	Random split 2	Random split 3	Random split 4	Random split 5
A	Pearson	**0.480**	**0.327**	**0.725**	-0.381	**0.637**
	Spearman	**0.238**	**0.185**	**0.693**	-0.602	**0.422**
B	Pearson	-0.173	**0.410**	**0.181**	**0.939**	**0.227**
	Spearman	-0.055	**0.054**	**0.318**	**0.906**	**0.218**
C	Pearson	**0.559**	-0.539	-0.190	**0.495**	-0.060
	Spearman	**0.445**	-0.165	**0.050**	**0.190**	**0.082**
D	Pearson	**0.572**	**0.845**	**0.522**	**0.266**	**0.419**
	Spearman	**0.617**	**0.828**	**0.494**	**0.494**	**0.494**
E	Pearson	-0.711	**0.976**	-0.818	**0.418**	**0.007**
	Spearman	-0.759	**0.577**	-0.883	**0.120**	**0.152**

Predictors are accurate only when obtained from the same or similar projects.

Apprendre du passé, observer les propriétés dynamiques

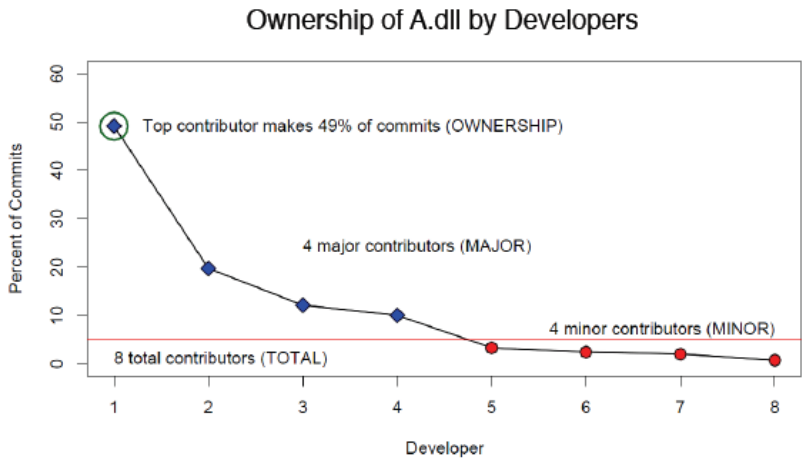
Observer l’historique des projets

Bird Nagappan Murphy Gall Devanbu 2011

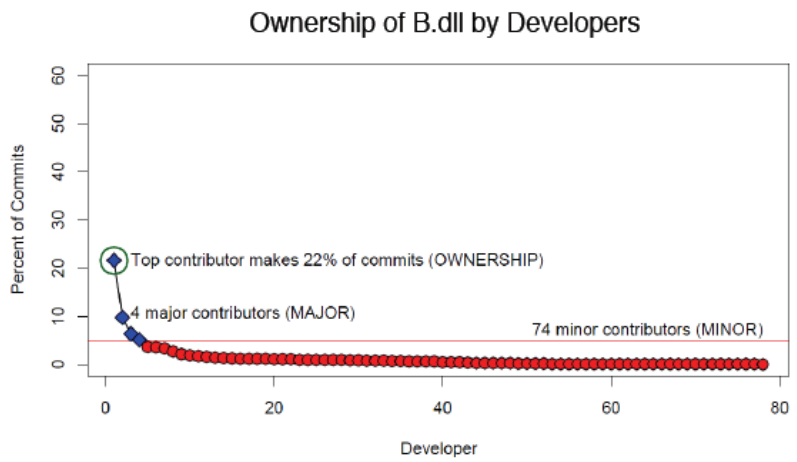
- Définir des métriques sur les contributions des développeurs
- Corréler avec les bug
- L'objectif étant d'évaluer les pratiques des développeurs (process) plutôt qu'exclusivement leur production

Propriété

- Propriété = pourcentage de commit
- Contributeur mineur = Propriété < 5%
- Contributeur majeur = Propriété > 5%



(a) A.dll



(b) B.dll

Propriété et Bug

- Un composant avec essentiellement des contributeurs majeurs a moins de bug
- Un composant avec beaucoup de contributeurs mineurs a plus de bug
- Les composants aux interfaces sont plus souvent modifiés par des contributeurs mineurs

Cas concret

- Utiliser dgit pour mesurer l’ownership de chaque fichier (fichier)

Two kinds of metrics

Source code metrics

Complexity

Size

Coupling

Process metrics

Churn

Frequency

Developers organisation

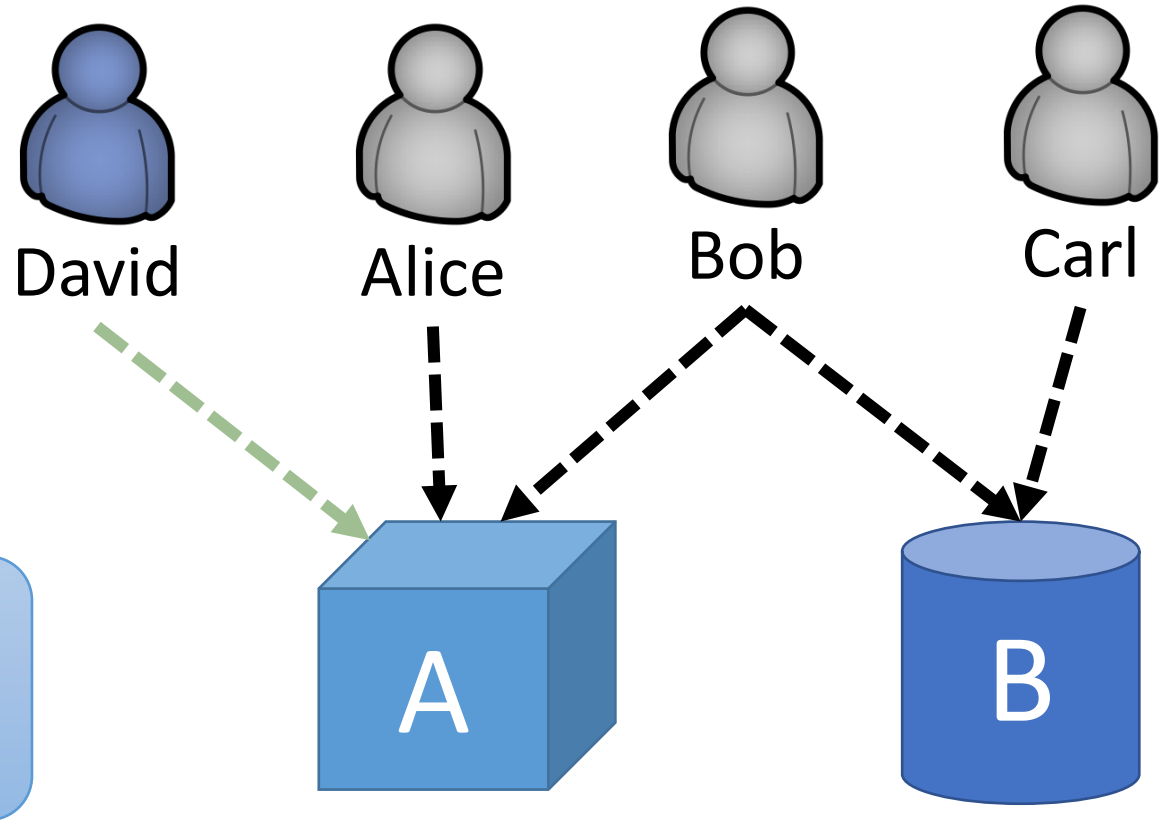
F. Rahman and P. Devanbu, "*How, and why, process metrics are better*," ICSE 2013
Source code metrics are probably too static !

External Turnover

External Newcomer

No or poor knowledge

More bugs [Rahman & Devanbu, ICSE'11]



External Leaver

Loss of knowledge

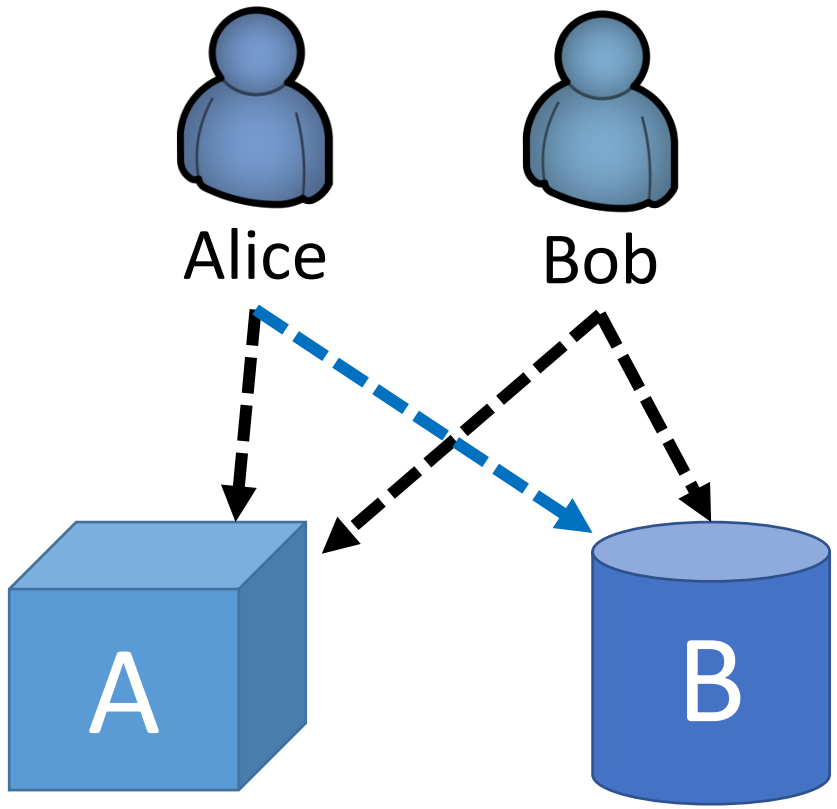
More bugs [Mockus, FSE'10]

Iternal turnover

Internal Newcomer

Alice is increasing her knowledge of the project

Less bug [Mockus & Weiss, Bell Labs Tech J., 2000]



Internal Leaver

Changing responsibility is good for the motivation

Less bug [Kanter, 1976]

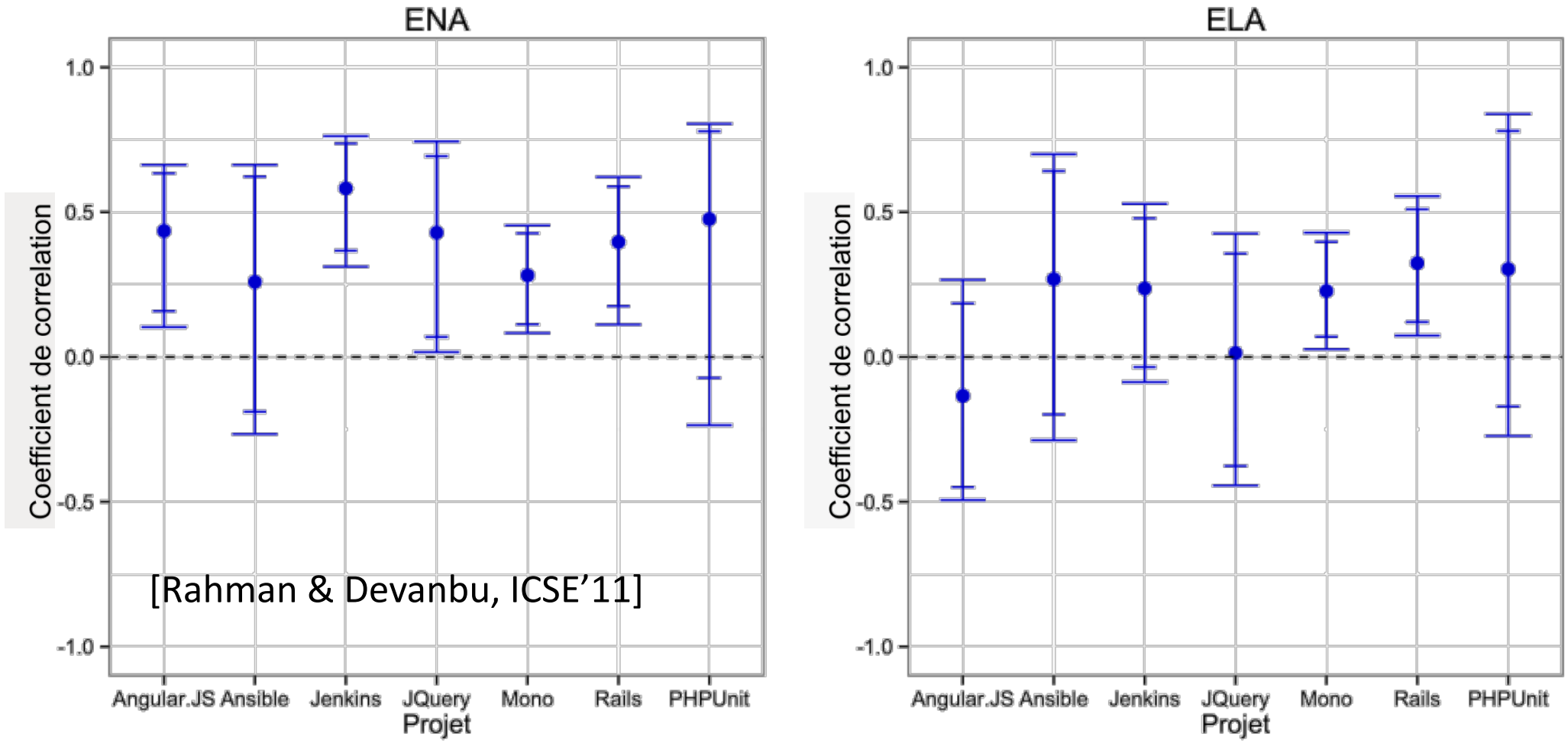
Research Questions

1. *Is turnover an important phenomena?*

2. *Is there any schema for turnover?*

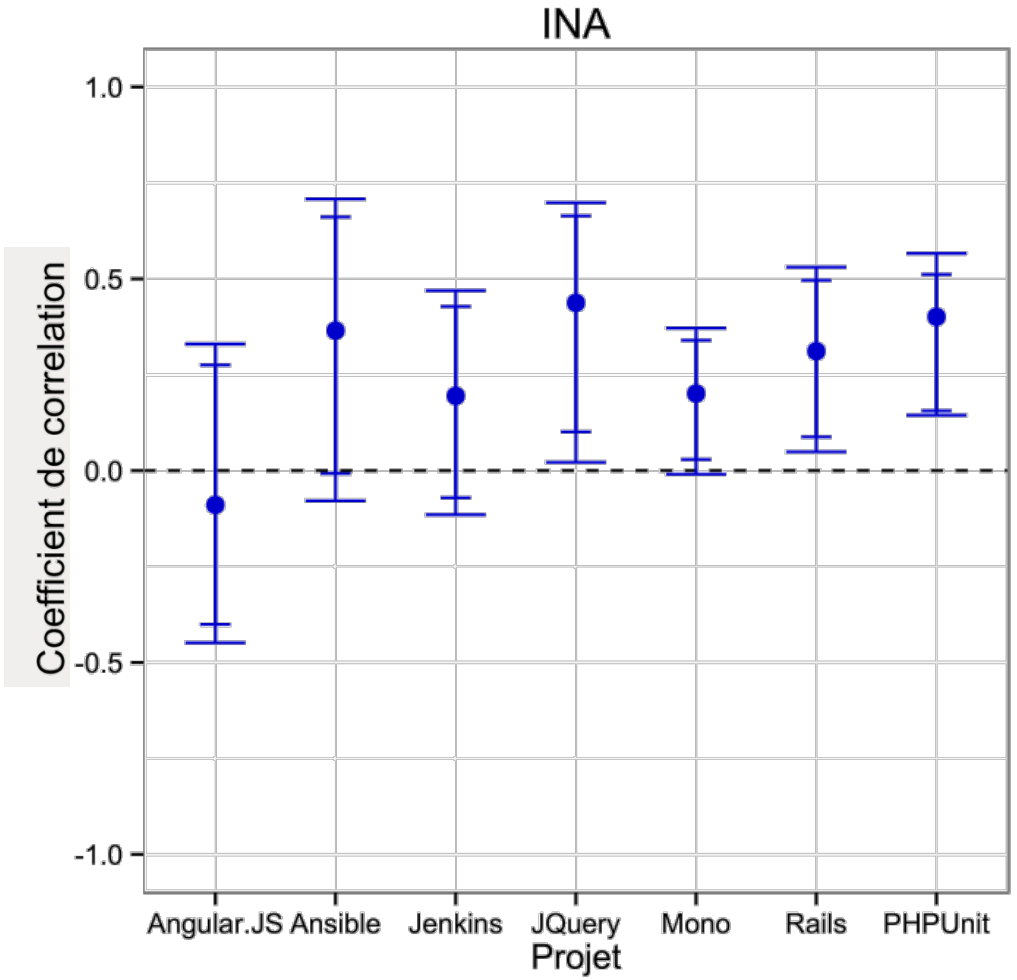
3. Is there any relationship between turnover and quality?

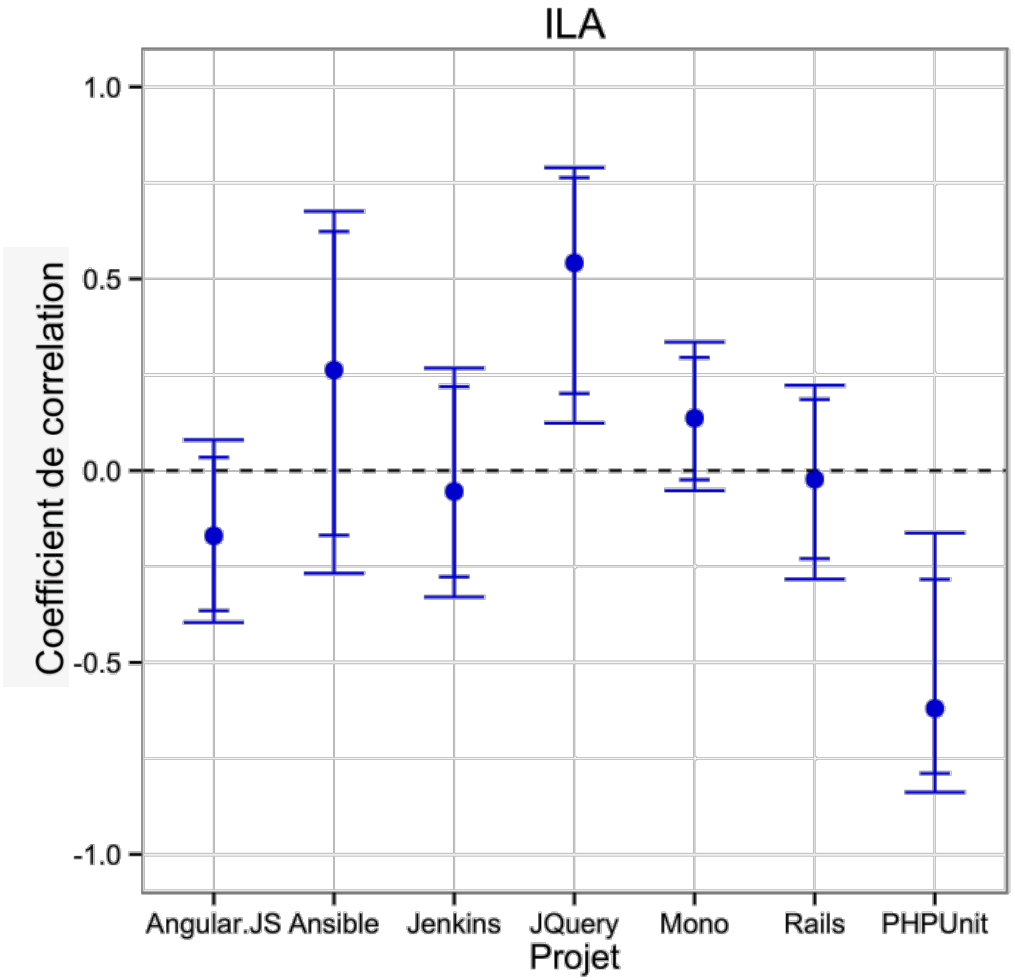
External Turnover



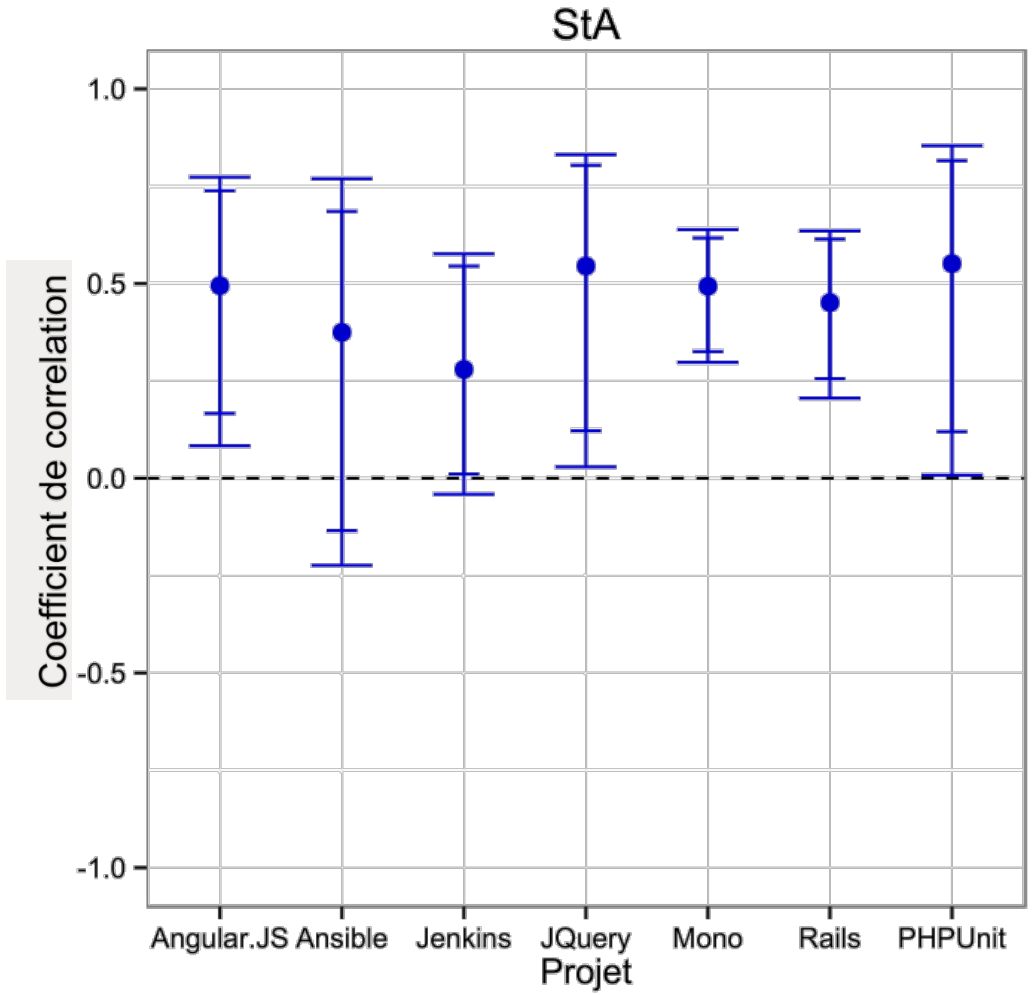
[Rahman & Devanbu, ICSE'11]

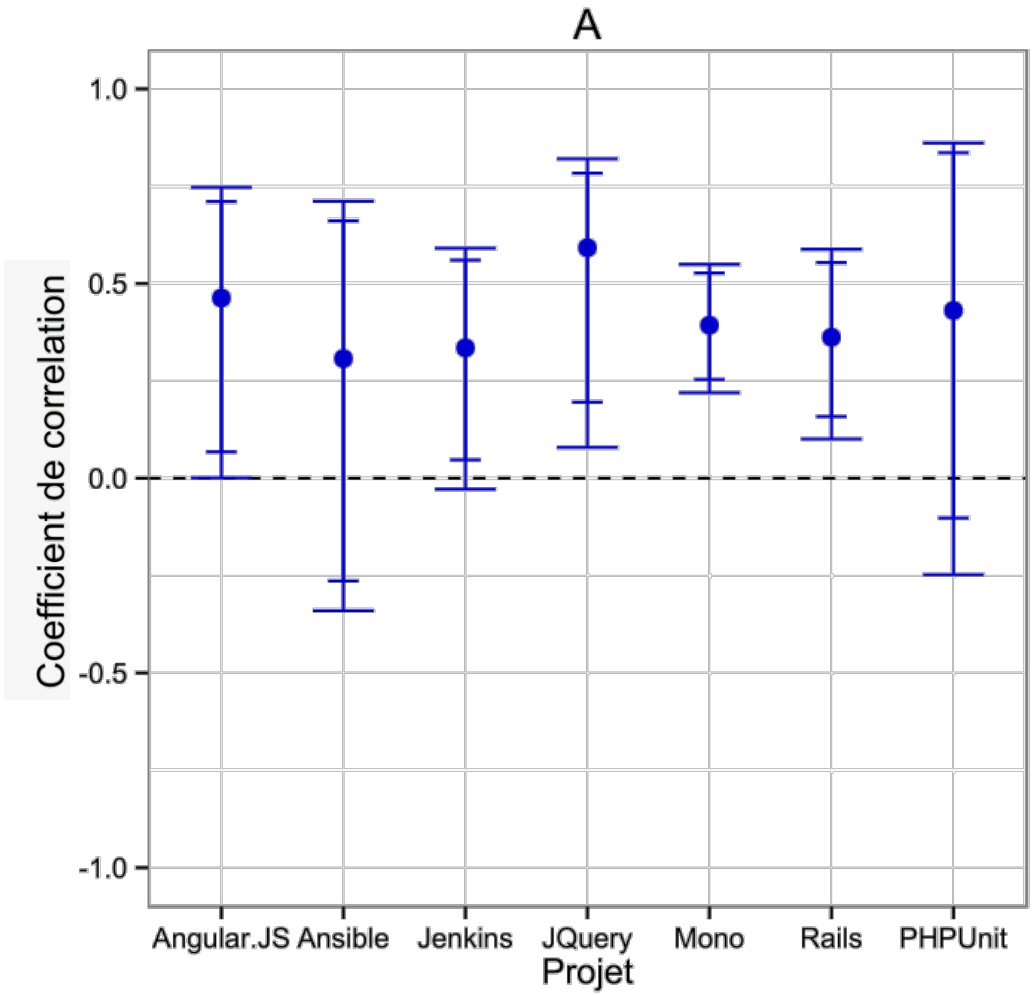
Internal Turnover



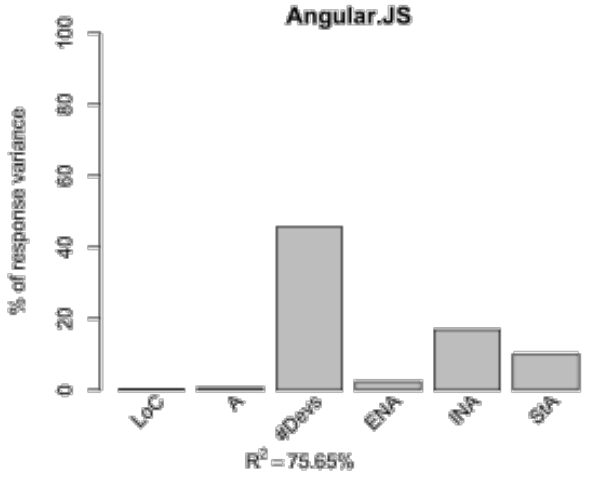


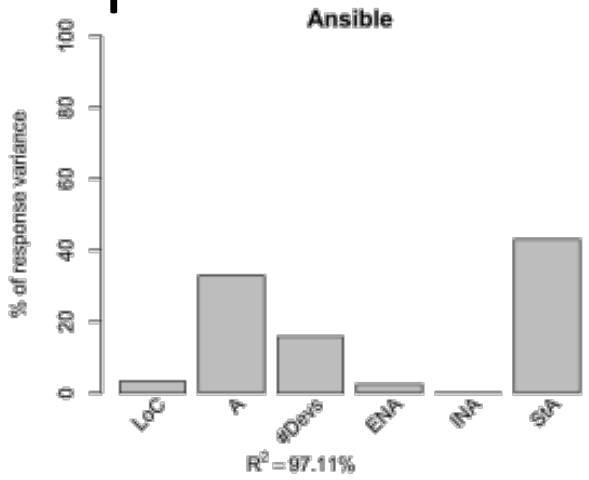
Stayers

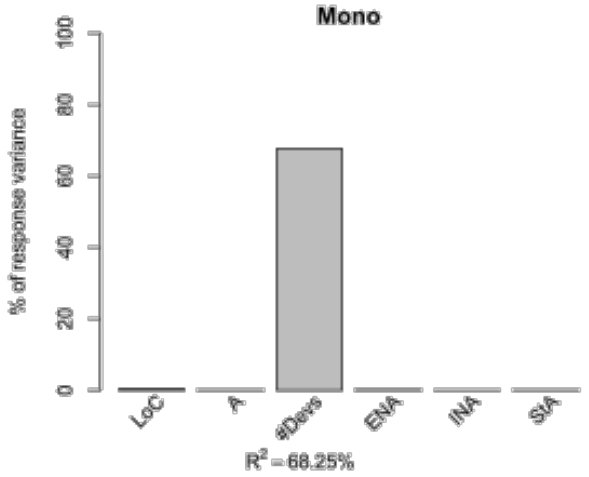

StA
1.0
0.5
0.0
-0.5
-1.0
Coefficient de correlation
Angular.JS
Ansible
Jenkins
JQuery
Mono
Rails
PHPUnit
Projet

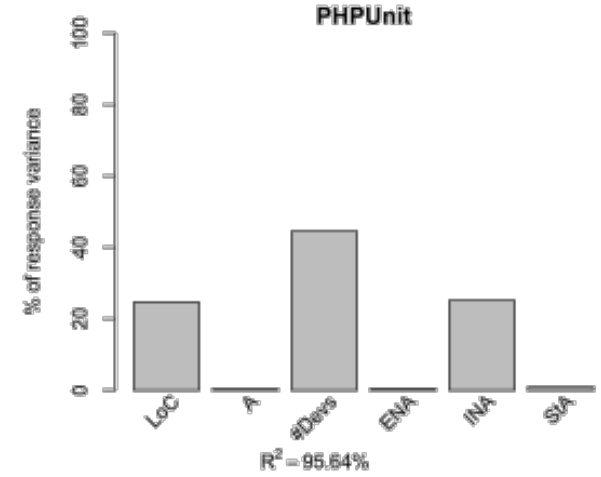

A
1.0
0.5
0.0
-0.5
-1.0
Coefficient de correlation
Angular.JS
Ansible
Jenkins
JQuery
Mono
Rails
PHPUnit
Projet

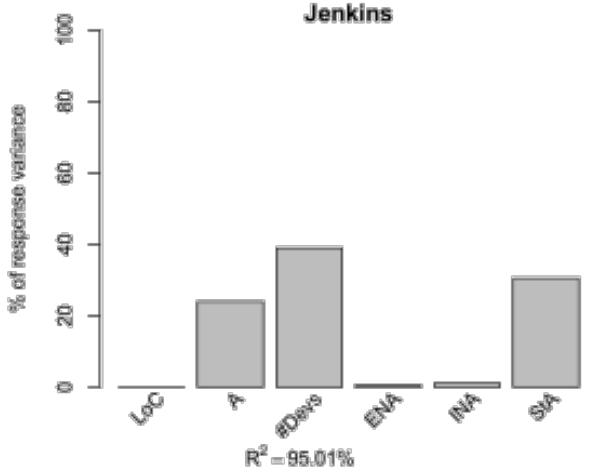
Relative Importance

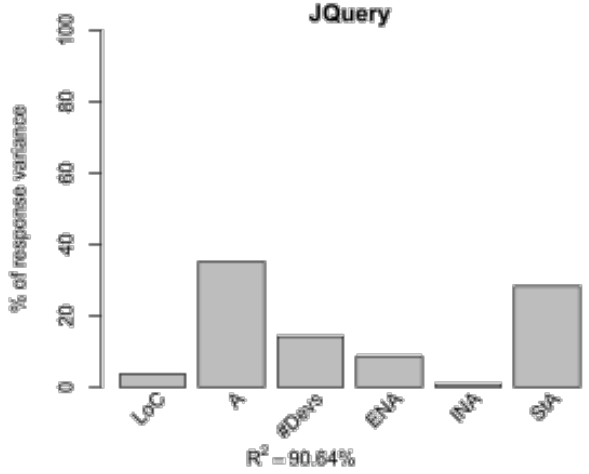


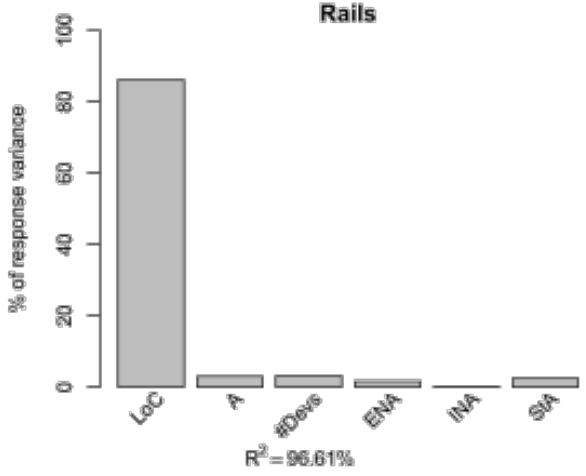












Analyse de dépôt, quelle méthode et quels outils

Le chercheur et le bon chercheur !

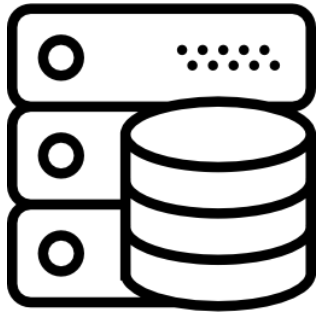
Mining Software Repository

- Use open source software repository to uncover knowledge on **software evolution**
- Leverage on **source code** and **repository metadata**
- Necessitate to use **advanced statistics** in order to validate assumptions and hypotheses

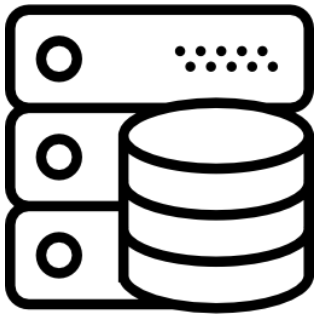
Challenge

- Do developers write tests before code ?

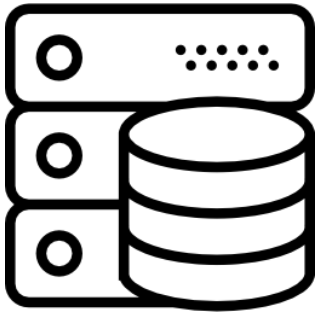
jQuery



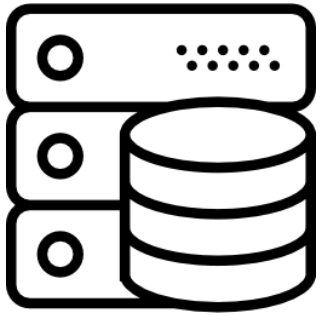
Apache



...



Eclipse

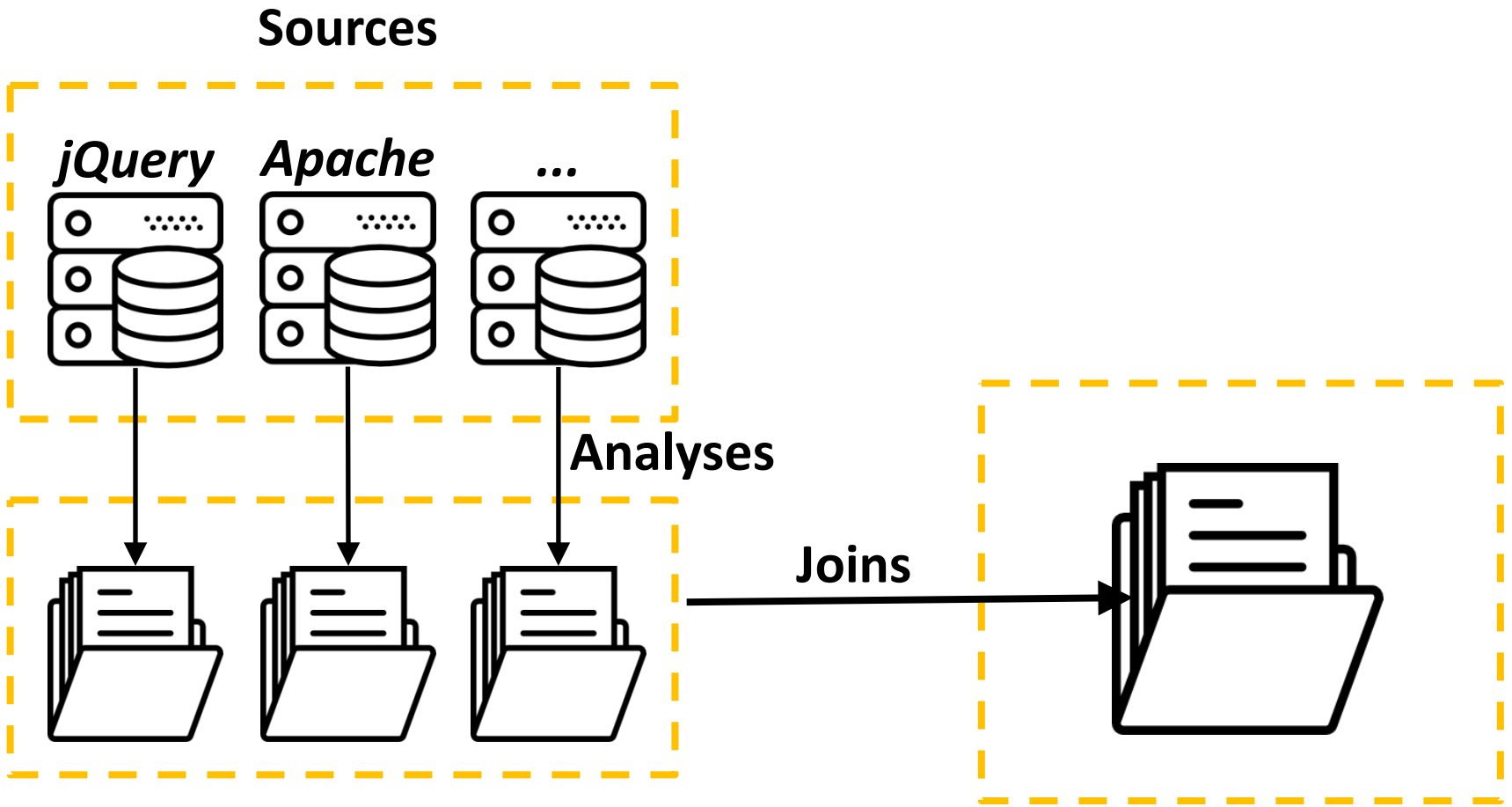


Problems

- **Heterogeneous** sources of data
- Need **various scripts** in **various languages** in order to run
- Need to use **scaling** storage technologies
- Must be easy to reuse for **reproducibility**

A handmade solution is not going to cut it...

Diggit



Diggit

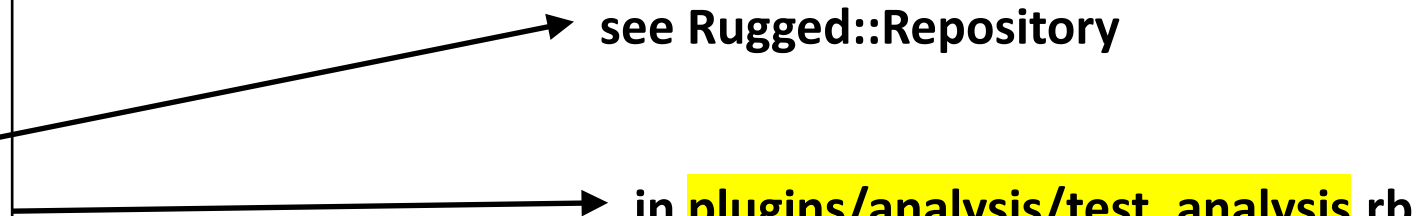
- Manages a set of **sources**
- Manages a set of **analyses** that produce data out of a source
- Manages a set of **joins** that produce data out of the analyses' data
- Manages **addons** that are services to help coding analyses or joins (i.e. db addon)
- Is written in **ruby**

Managing sources

```
$ dgit init
$ dgit sources add https://github.com/jrfaller/test-git
$ dgit sources add https://github.com/jrfaller/diggit
$ dgit sources
$ dgit clones perform
$ dgit status
```

An analysis

```
class TestAnalysis < Diggit::Analysis

   def run

            puts “Run on #{source.url}”

            puts “I have bindings to the

                        repository: #{repo}”

   end


   def clean

            puts “Clean on #{source.url}”

   end

end
```



Managing analyses

```
$ dgit analyses add test_analysis
$ dgit analyses perform
> “Run on https://github.com/jrfaller/test-git”
> “Run on https://github.com/jrfaller/diggit”
$ dgit analyses perform -m clean
> “Clean on https://github.com/jrfaller/test-git”
> “Clean on https://github.com/jrfaller/diggit”
```

A join

```
class TestJoin < Diggit::Join

  require_analyses “test_analysis”

  def run

        puts “Run on #{sources}”

  end


  def clean

          puts “Clean on #{sources}”

  end

end
```

in plugins/join/test_join.rb

Managing joins

```
$ dgit joins add test_join
$ dgit joins perform
> “Run on (https://github.com/jrfaller/test-git, https://github.com/jrfaller/diggit)”
$ dgit joins perform -m clean
> “Clean on (https://github.com/jrfaller/test-git,https://github.com/jrfaller/diggit)”
```

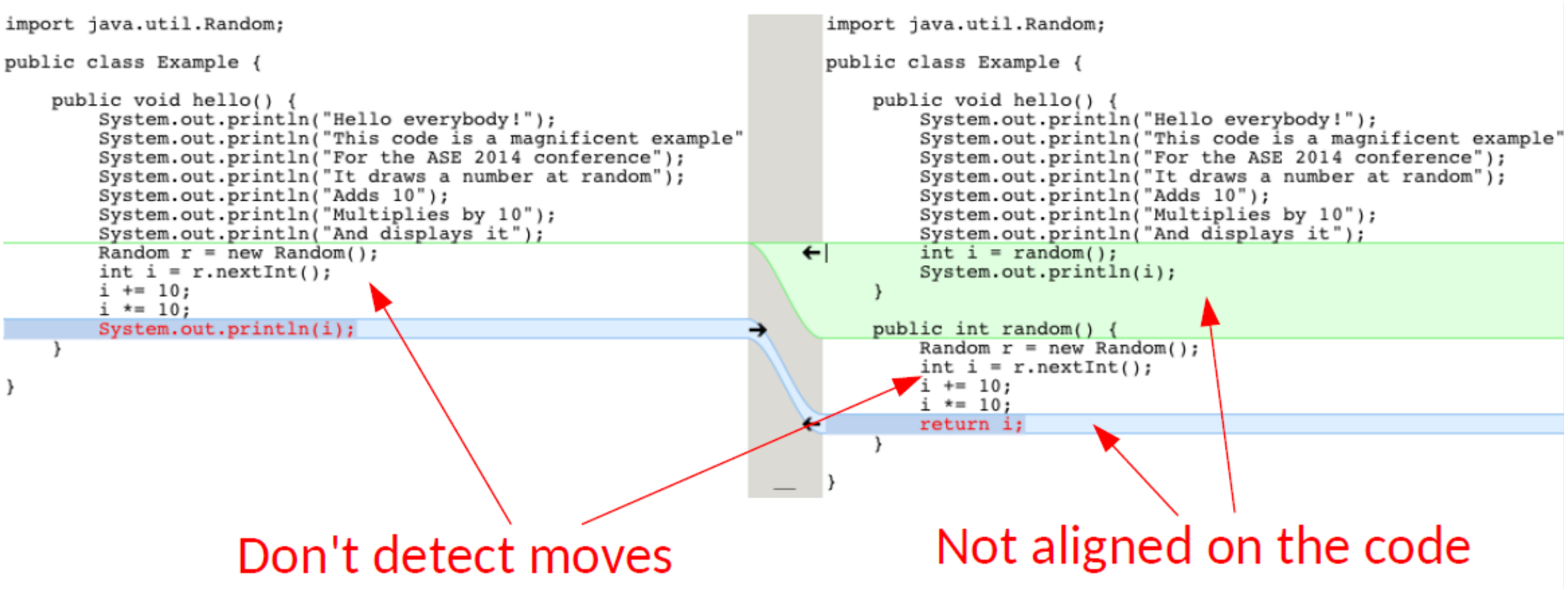
Exercice ?

- Essayez de savoir si quelqu'un commit rarement et là où il ne devrait pas …
- Essayez de savoir quels commits ont introduit des défauts …

Textual differencing

- Source code model is a sequence of text lines
- Possible actions are
  - Delete a text line
  - Insert a text line
- Problem solved: find one shortest sequence of actions
  - Transforming the previous source code into the current source codes

Current approaches



A NP-Hard problem

- Three choices
  - Source code model
  - Set of possible actions
  - Problem solved
- Leads to NP-hard problems
  - Textual diff with move is NP-hard
  - Labelled graph diff is NP-hard, even with basic actions
  - Unordered tree diff is NP-hard, even with basic actions

GumTree

- Source code model
  - A labelled ordered rooted tree (AST)
- Possible actions
  - Node insertion
  - Node deletion
  - Node relabel
  - Subtree move
- Problem solved
  - Short sequence that corresponds to a developer intent

GumTree process

1. Parse code files to our code agnostic tree structure
2. **Find mappings between nodes**
   - Like developers proceed
3. Deduce code edition actions
4. Output code difference

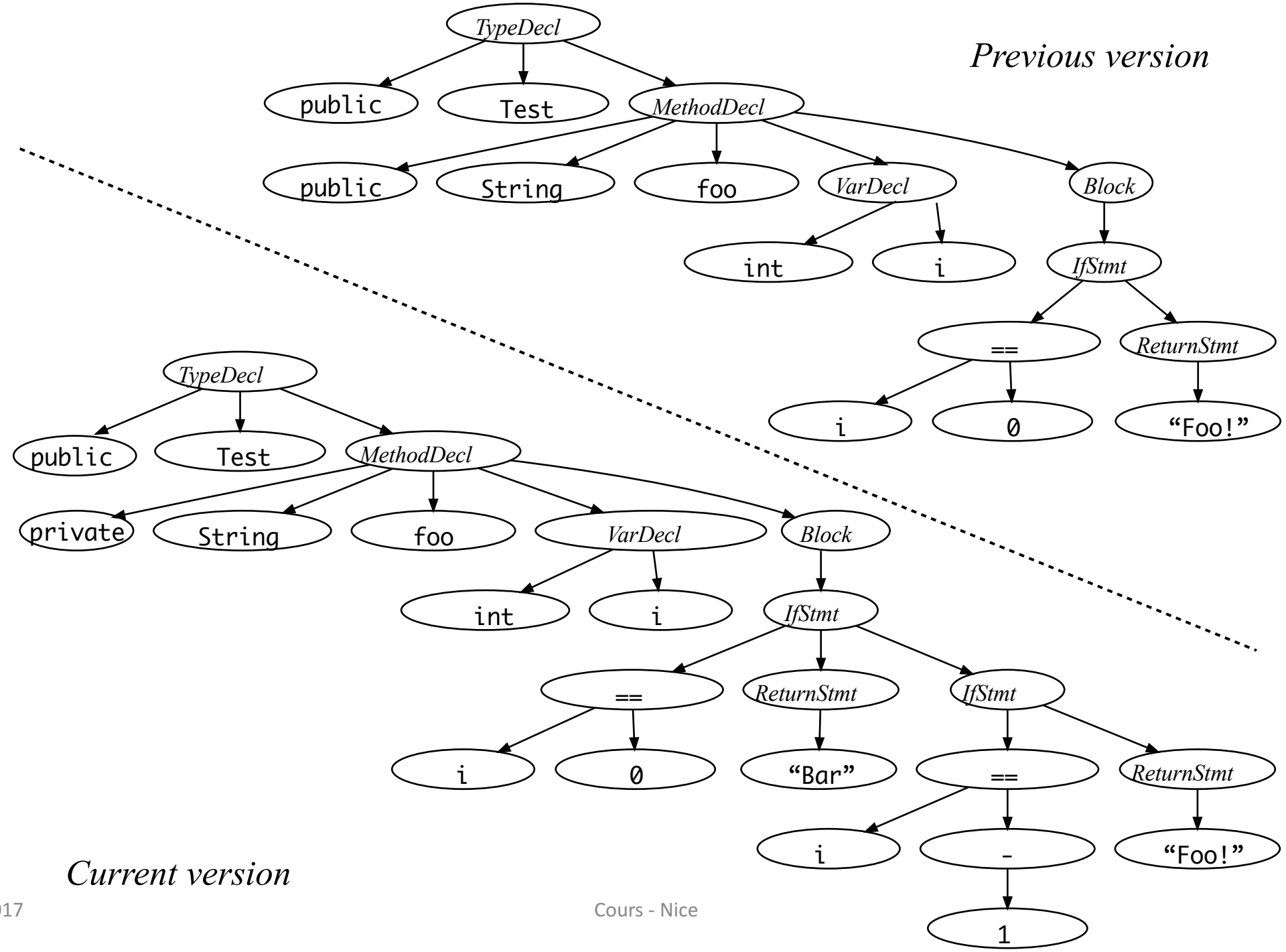
Finding mappings

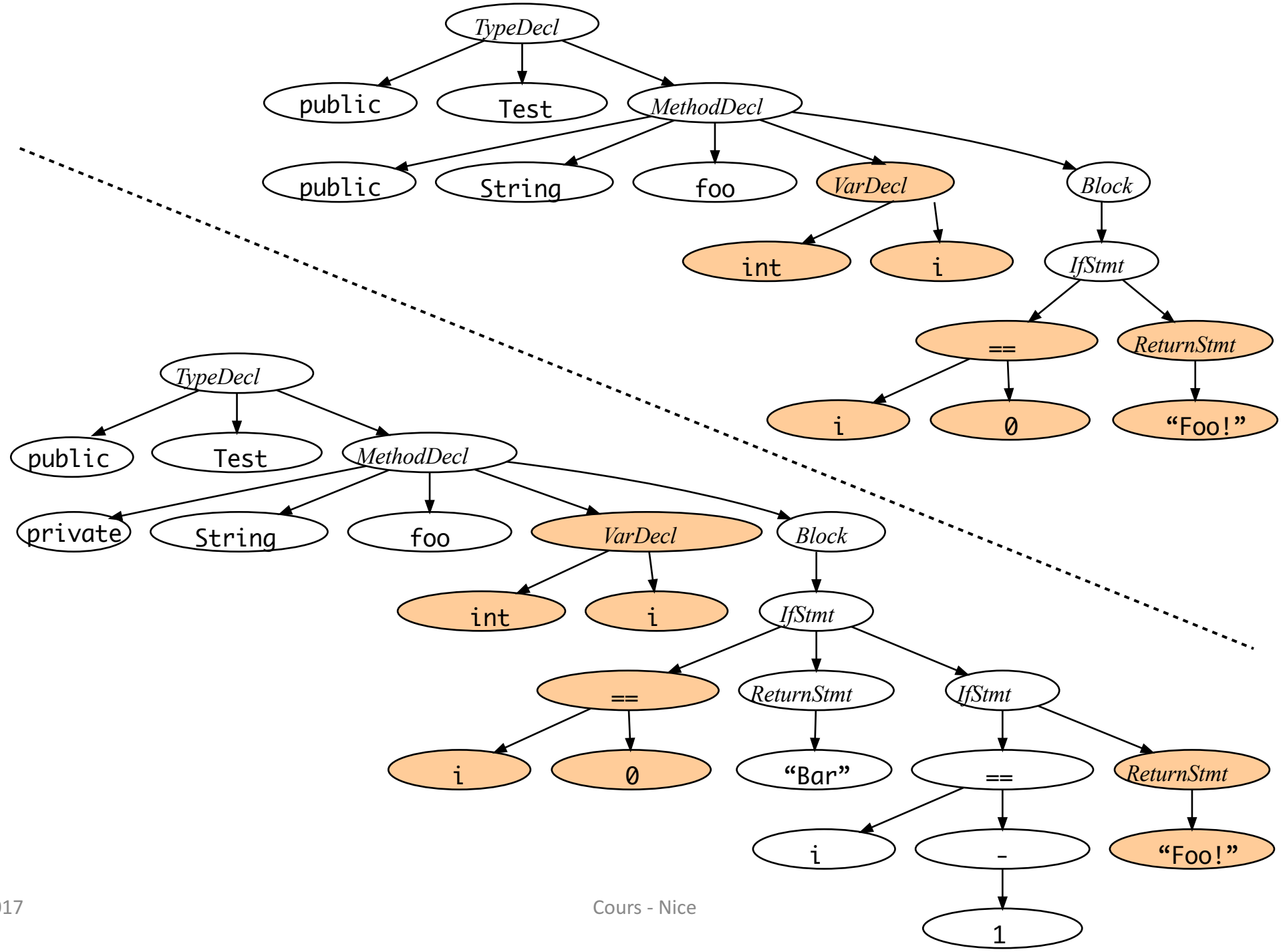
1. Top-down phase
   - Find biggest chunks of unmodified code
2. Bottom-up phase
   - Propagate mappings to the containers
   - Extend mappings in the left-over code

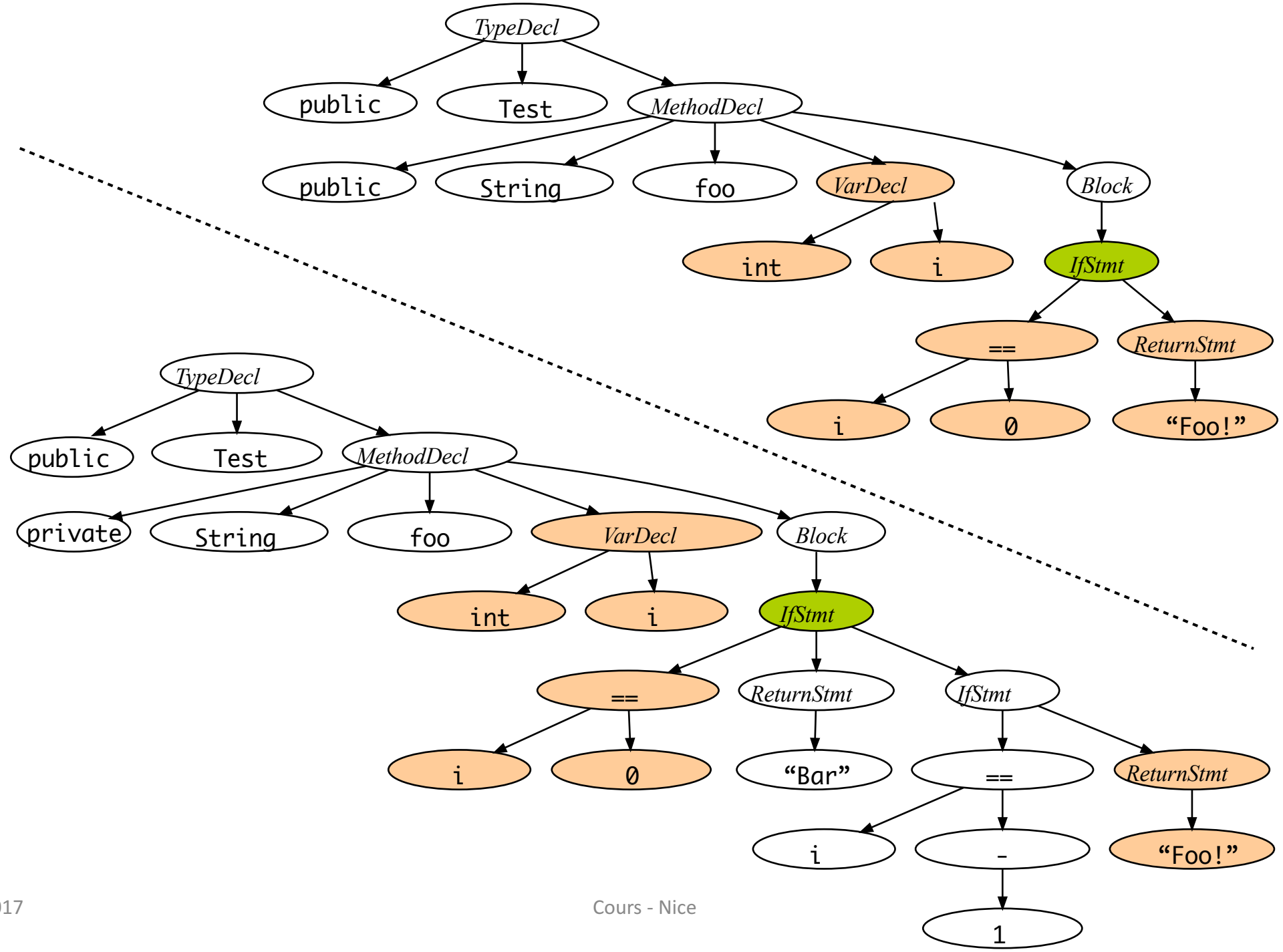
Example

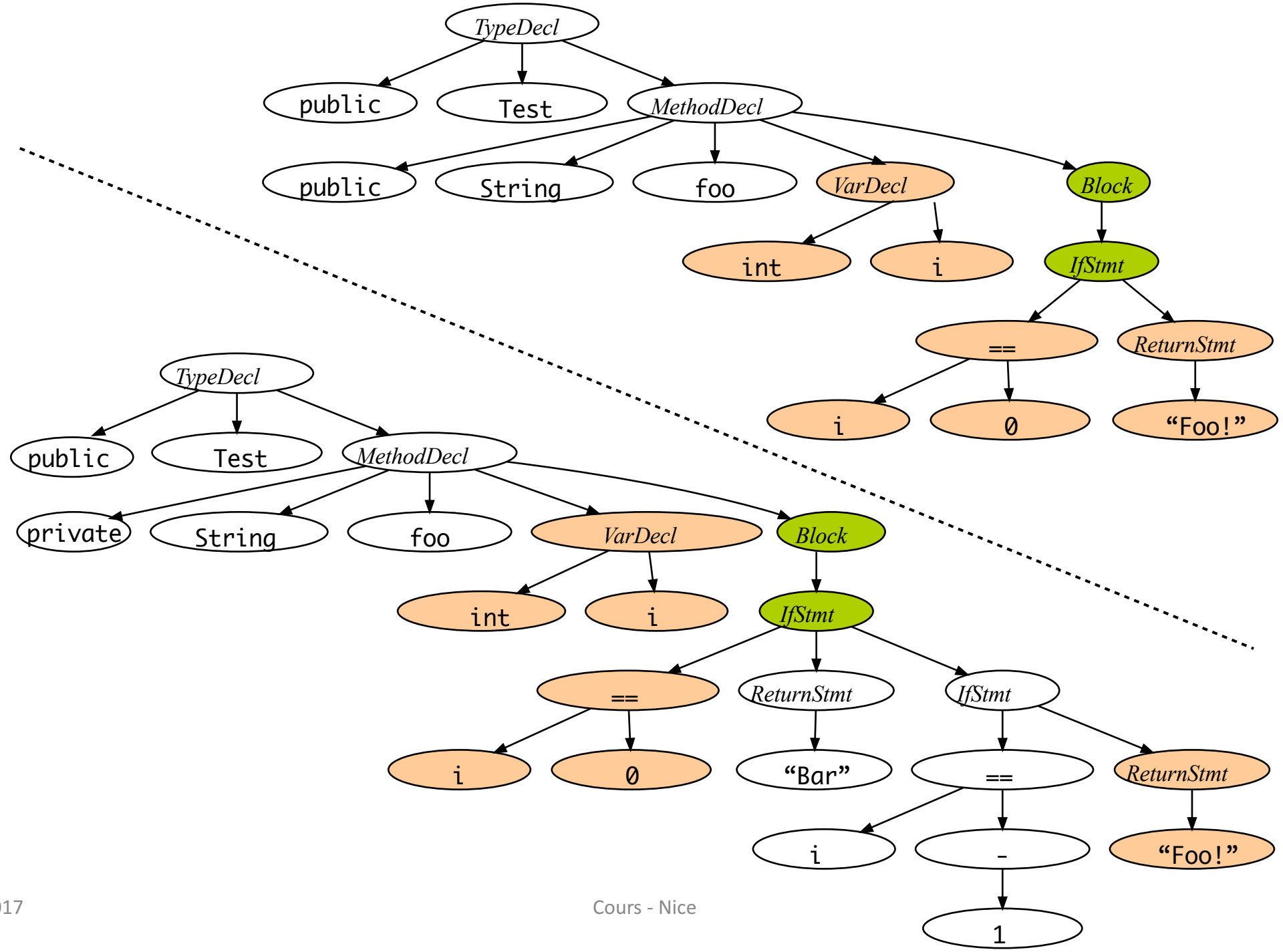
```
public class Test {
    public String foo(int i) {
        if (i == 0)
            return "Foo!";
        }
}
```

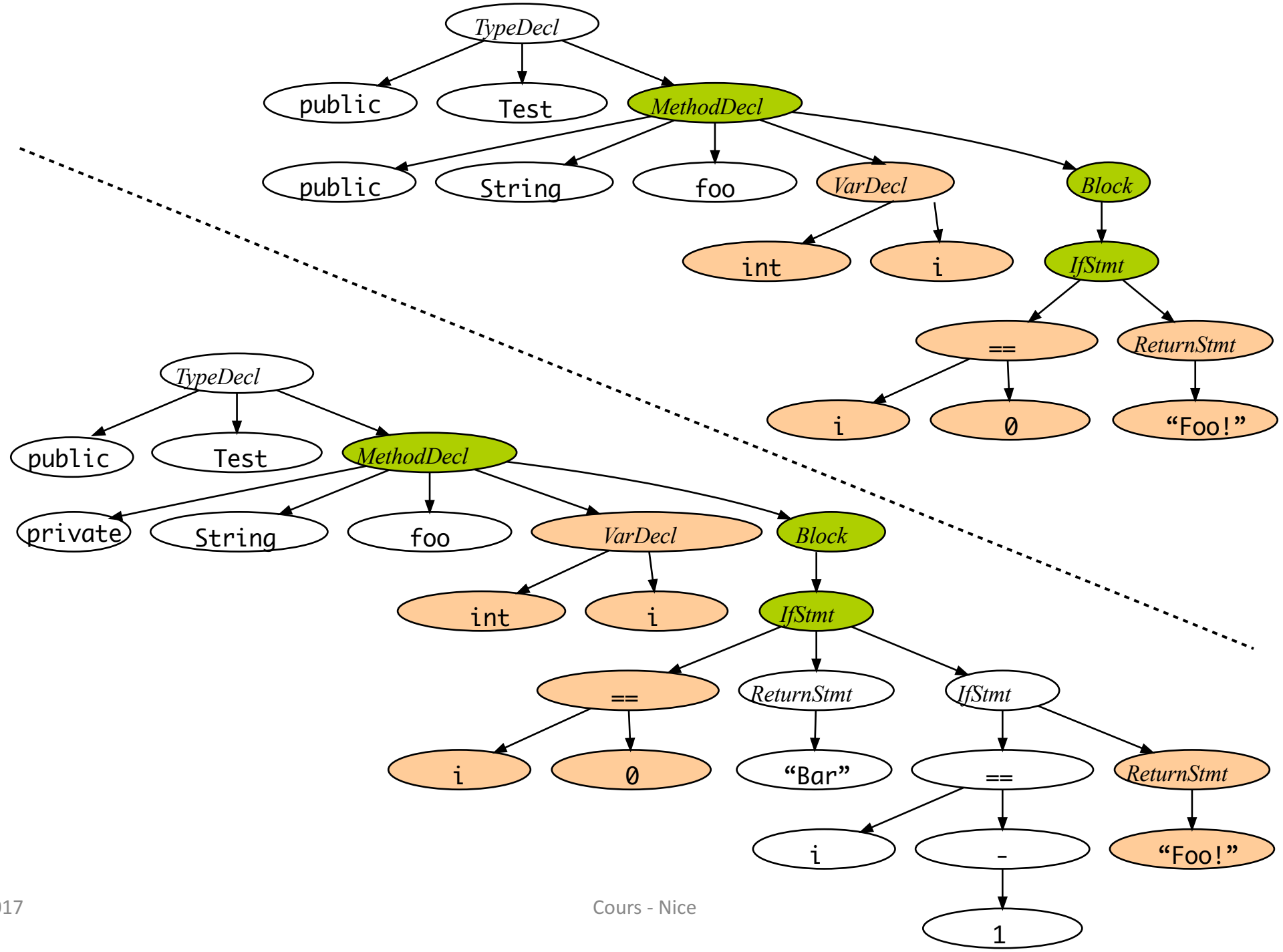
```
public class Test {
    private String foo(int i) {
        if (i == 0)
            return "Bar";
        else if (i == -1)
            return "Foo!";
    }
}
```

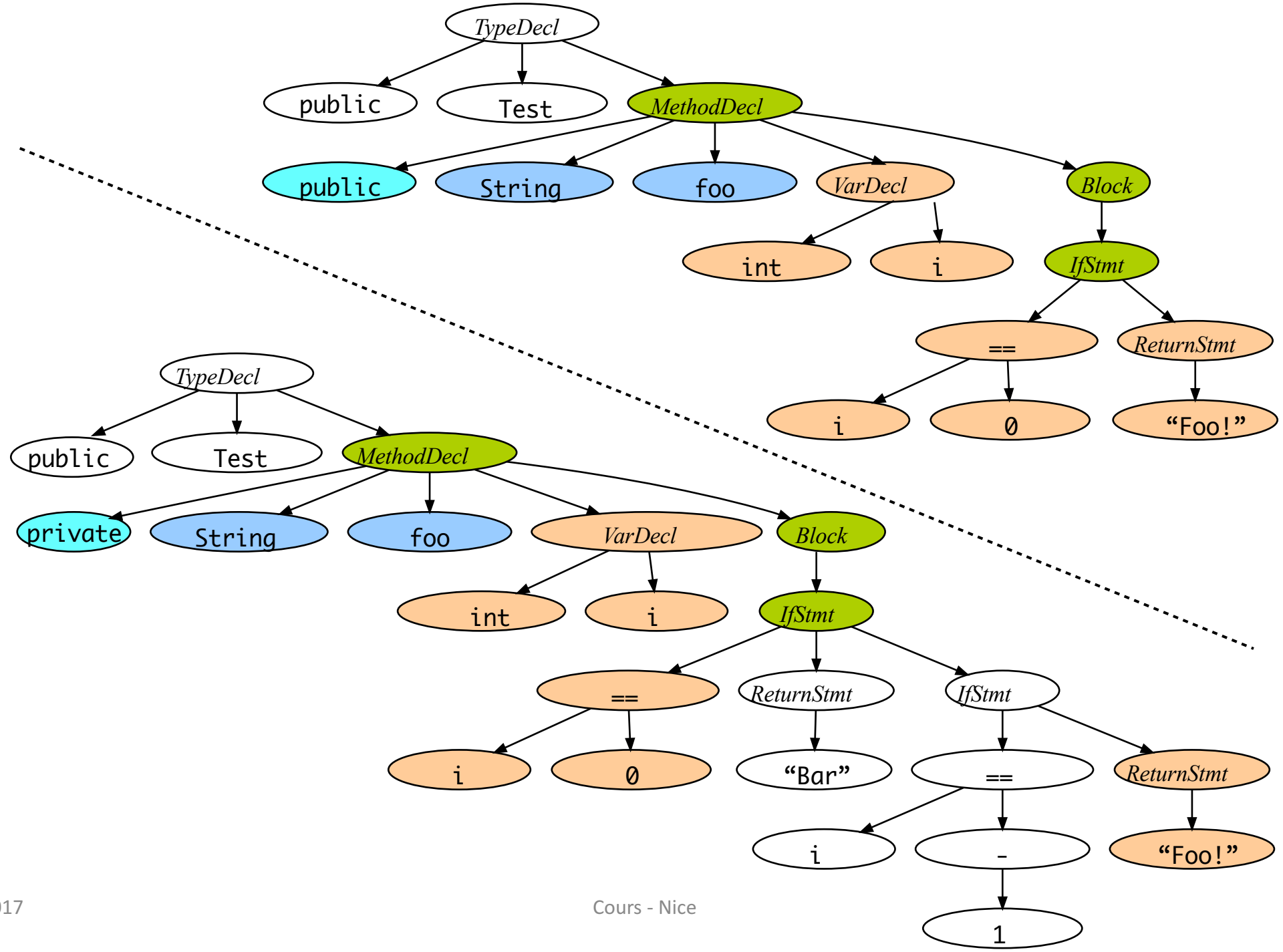

Previous version
TypeDecl
public
Test
MethodDecl
public
String
foo
VarDecl
Block
int
i
IfStmt
==
ReturnStmt
i
0
"Foo!"
TypeDecl
public
Test
MethodDecl
private
String
foo
VarDecl
Block
int
i
IfStmt
==
ReturnStmt
IfStmt
i
0
"Bar"
==
ReturnStmt
i
-
"Foo!"
1
Current version

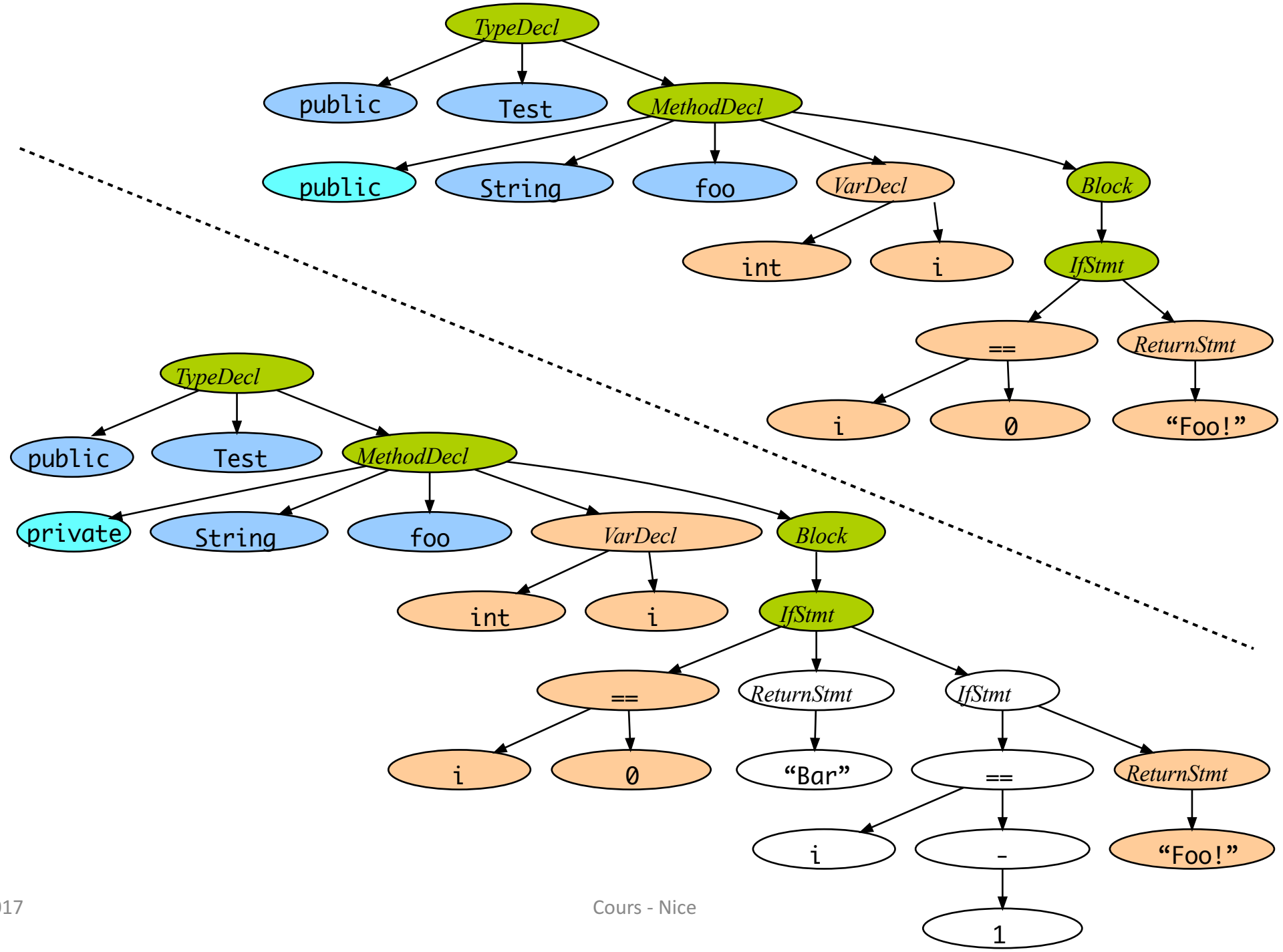

TypeDecl
public
Test
MethodDecl
public
String
foo
VarDecl
Block
int
i
IfStmt
==
ReturnStmt
i
0
"Foo!"
TypeDecl
public
Test
MethodDecl
private
String
foo
VarDecl
Block
int
i
IfStmt
==
ReturnStmt
IfStmt
i
0
"Bar"
==
ReturnStmt
i
-
"Foo!"
1


TypeDecl
public
Test
MethodDecl
public
String
foo
VarDecl
Block
int
i
IfStmt
==
ReturnStmt
i
0
"Foo!"
TypeDecl
public
Test
MethodDecl
private
String
foo
VarDecl
Block
int
i
IfStmt
==
ReturnStmt
IfStmt
i
0
"Bar"
==
ReturnStmt
i
-
"Foo!"
1


TypeDecl
public
Test
MethodDecl
public
String
foo
VarDecl
Block
int
i
IfStmt
==
ReturnStmt
i
0
"Foo!"
TypeDecl
public
Test
MethodDecl
private
String
foo
VarDecl
Block
int
i
IfStmt
==
ReturnStmt
IfStmt
i
0
"Bar"
==
ReturnStmt
i
-
"Foo!"
1


TypeDecl
public
Test
MethodDecl
public
String
foo
VarDecl
Block
int
i
IfStmt
==
ReturnStmt
i
0
"Foo!"
TypeDecl
public
Test
MethodDecl
private
String
foo
VarDecl
Block
int
i
IfStmt
==
ReturnStmt
IfStmt
i
0
"Bar"
==
ReturnStmt
i
-
"Foo!"
1


TypeDecl
public
Test
MethodDecl
public
String
foo
VarDecl
Block
int
i
IfStmt
==
ReturnStmt
i
0
"Foo!"
TypeDecl
public
Test
MethodDecl
private
String
foo
VarDecl
Block
int
i
IfStmt
==
ReturnStmt
IfStmt
i
0
"Bar"
==
ReturnStmt
i
-
"Foo!"
1


TypeDecl
public
Test
MethodDecl
public
String
foo
VarDecl
Block
int
i
IfStmt
==
ReturnStmt
i
0
"Foo!"
TypeDecl
public
Test
MethodDecl
private
String
foo
VarDecl
Block
int
i
IfStmt
==
ReturnStmt
IfStmt
i
0
"Bar"
==
ReturnStmt
i
-
"Foo!"
1

Example

Previous version

```
public class Test {
   public String foo(int i) {
      if (i == 0)
         return "Foo!";
      }
}
```

Current version

```
public class Test {
   private String foo(int i) {
      if (i == 0)
         return "Bar";
      else if (i == -1)
         return "Foo!";
   }
}
```

Gumtree

- From textual diff to tree diff
- Understanding source code changes
- Falleri et al. ASE 2014
- https://github.com/GumTreeDiff/gumtree

Conclusion

Fouiller les dépots

- Le MSR (Mining Software Repository) pour comprendre les difficultés des développeurs et valider les solutions mises en place
- Quels sont les facteurs d'impacts sur le développement ?
- Comment prendre en compte les spécificités des projets ?

L’aventure ProMyze

- Outil de suivi de développement
- Observer les actions de développement
- Analyser l’impact des actions
- Faire levier sur l’implication des développeurs : gamification