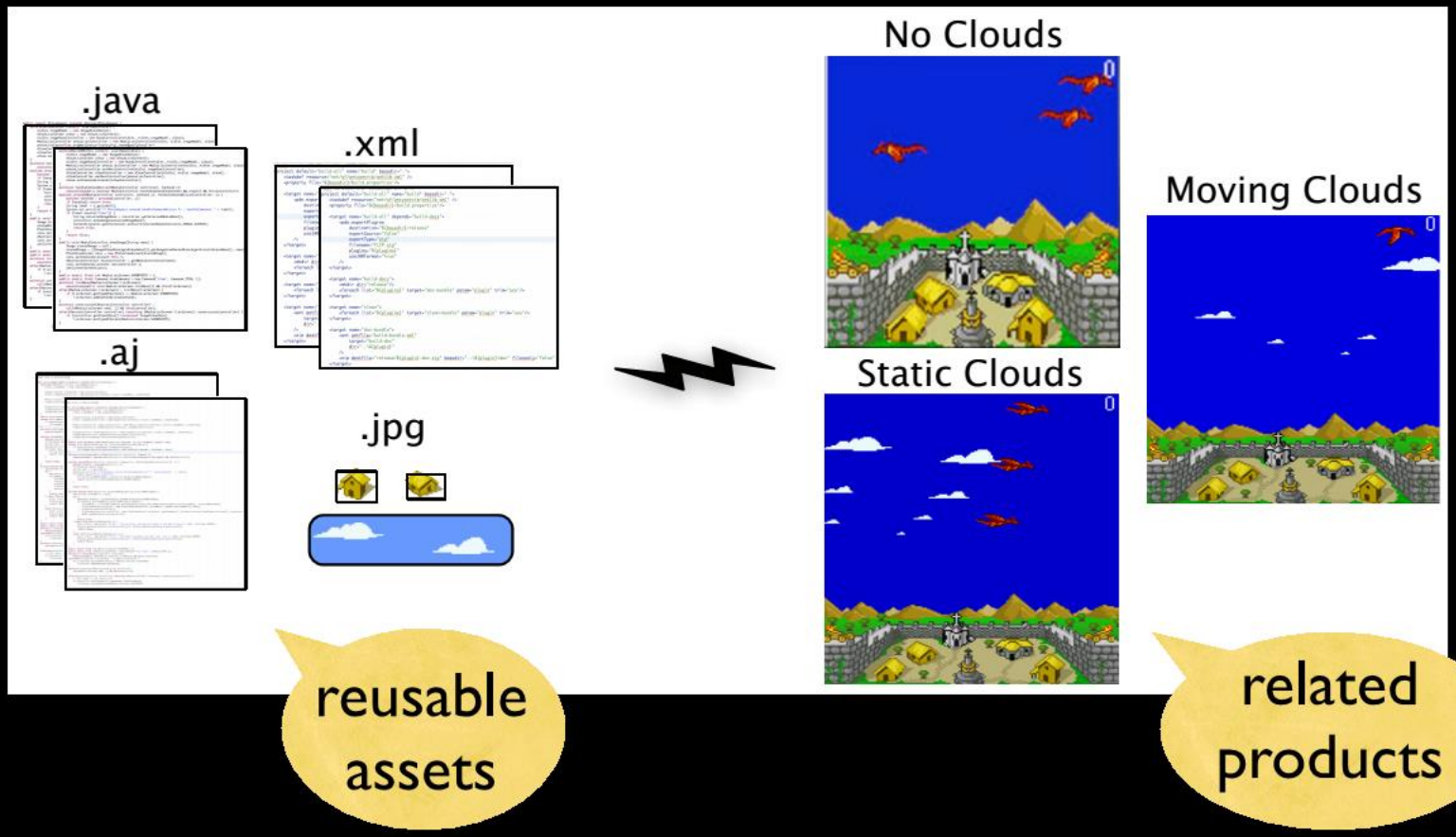
Modular Reasoning for Software Product Lines with Emergent Feature Interfaces

Jean Melo

Advisor: Paulo Borba

Software Product Line



cost time to market quality

Preprocessors
(a.k.a. *conditional compilation*)

- Often used to implement the variability (features) of SPLs

```
…
char_u *fname_res = …;
…
#ifdef UNIX || WIN3264
…
tail = make_percent_swname(fname_res);
…
#endif
…
```

Preprocessors

- Often used to implement the variability (features) of SPLs
- But,

```
...
char_u *fname_res = ...;
...
#ifdef UNIX || WIN3264
...
tail = make_pertent_swname(fname_res);
...
#endif
...
```

Code Pollution

No Separation of Concerns

Error-prone

Problem

- Code maintenance can **break** a variant of a SPL. For example, changes in one feature might **affect** another one

```
…
char_u *fname_res = *fname;
…
#ifdef UNIX || WIN3264
…
tail =
make_percent_swname(fname_res);
…
#endif
…
```

```
…
char_u *fname_res = null;
…
#ifdef UNIX || WIN3264
…
tail =
make_percent_swname(fname_res);
…
#endif
…
```

Example #1: Bug in Vim editor

Problem

- Code maintenance can **break** a variant of a SPL. For example, changes in one feature might **affect** another one

```
…
char_u *fname_res = *fname;
…
#ifdef UNIX || WIN3264
…
tail =
make_percent_swname(fname_res);
…
#endif
…
```

```
…
char_u *fname_res = null;
…
#ifdef UNIX || WIN3264
…
tail =
make_percent_swname(fname_res);
…
#endif
…
```

Compilation error

Example #1: Bug in Vim editor

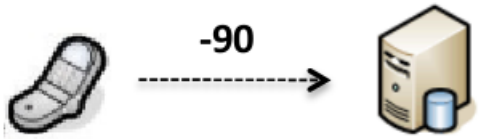
Problem

```
public void computeLevel() {
    ...
    totalScore = perfectCurvesCounter * PERFECT_CURVE_BONUS
               + perfectStraightCounter * PERFECT_STRAIGHT_BONUS
               + gc_levelManager.getCurrentCountryId()
               - totalLapTime * SRC_TIME_MULTIPLIER;
    ...
    Arena feature
}
```

```
NetworkFacade.setScore(totalScore);
NetworkFacade.setLevel(this.gc_getCurrentLevel
());
```

```
public void setScore(int s) {
   score = (s < 0) ? : 0 : s;
}
```

Negative score appears correctly in the game! Let's commit!



Zero instead of -90!

Example #2: Maintenance in the *best lap* product line

Problem

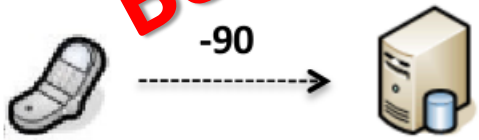
```
public void computeLevel() {
    ...
    totalScore = perfectCurvesCounter * PERFECT_CURVE_BONUS
               + perfectStraightCounter * PERFECT_STRAIGHT_BONUS
               + gc_levelManager.getCurrentCountryId()
               - totalLapTime * SRC_TIME_MULTIPLIER;
    ...
    Arena feature
}
```

```
NetworkFacade.setScore(totalScore);
NetworkFacade.setLevel(this.gc_getCurrentLevel
());
```

```
public void setScore(int s) {
   score = (s < 0) ? : 0 : s;
}
```

Behavioral error

Negative score appears correctly in the game! Let's commit!



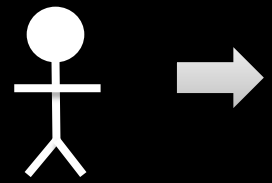
Zero instead of -90!

Example #2: Maintenance in the *best lap* product line

To minimize the feature modularization problem…

Emergent Interfaces

- The developer selects the **maintenance point**

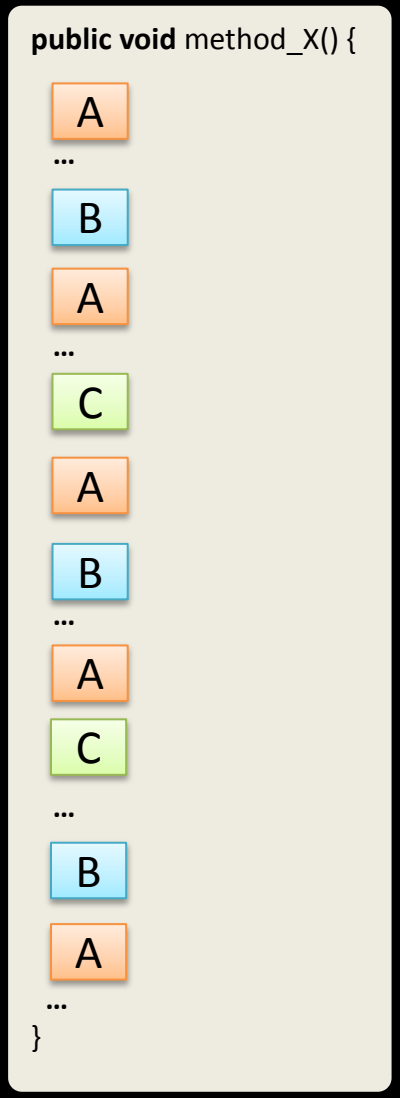


```
…
char_u *fname_res = …;
…
#ifdef UNIX || WIN3264
…
tail = make_percent_swname(fname_res);
…
#endif
…
```

Code analysis is performed…

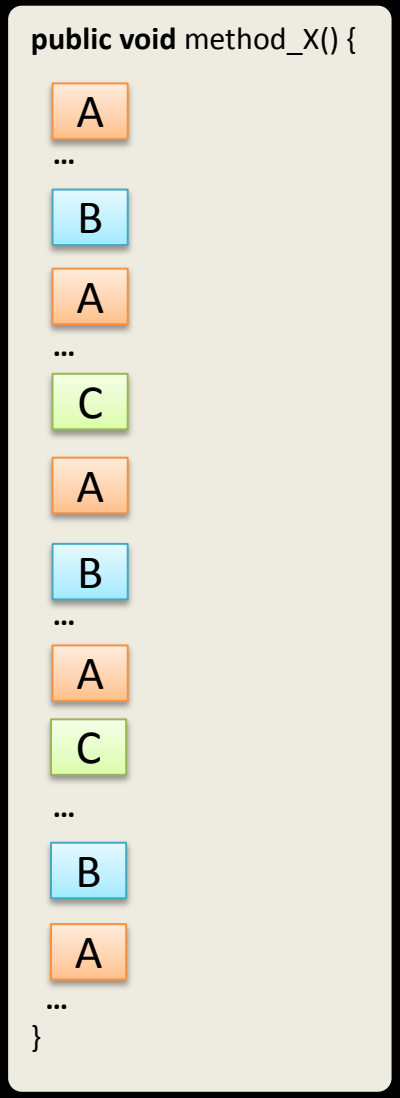
Be careful! You provide *fname_res for the UNIX and WIN3264 features.

Let’s maintain the feature A!



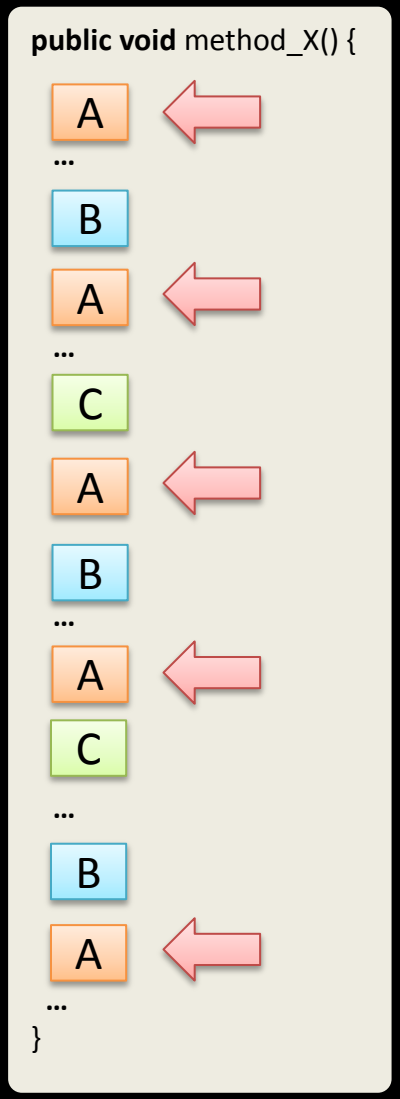
Let’s maintain the feature A!

What we need to do using EI?



Let’s maintain the feature A!

What we need to do using EI?



Select all fragments of the feature A one-by-one! ☹

Emergent Interfaces…

… capture dependencies between a feature maintenance point and **parts of other feature** implementation, but they do not provide an **overall feature interface** considering all parts in an integrated way

Maintenance in the GUI feature!

```
public static void main(String args[]) {
      String title;

      //#ifdef PT_BR
      title = "JCalc - Calculadora Padrão e Científica";
      ...
      //#endif

     //#ifdef GUI
     JFrame frame = new JFrame(title);
     ...
     //#endif

     //#ifdef LOOKANDFEEL
     initLookAndFeel(frame);
     //#endif

     //#ifdef GUI
     ...
     JCalcStandardFrame myFrame = new JCalcStandardFrame();
     JPanel myPane = myFrame.getPane();

     frame.getContentPane().add(myPane, ...);
     ...
     //#endif
}
```

Maintenance in the GUI feature!

```
public static void main(String args[]) {
      String title;

      //#ifdef PT_BR
      title = "JCalc - Calculadora Padrão e Científica";
      ...
      //#endif

      //#ifdef GUI
      JFrame frame = new JFrame(title);
      ...
      //#endif

      //#ifdef LOOKANDFEEL
      initLookAndFeel(frame);
      //#endif

      //#ifdef GUI
      ...
      JCalcStandardFrame myFrame = new JCalcStandardFrame();
      JPanel myPane = myFrame.getPane();

      frame.getContentPane().add(myPane, ...);
      ...
      //#endif
}
```

Provides *frame* **to** LOOKANDFEEL

Maintenance in the GUI feature!

```
public static void main(String args[]) {
      String title;

      //#ifdef PT_BR
      title = "JCalc - Calculadora Padrão e Científica";
      ...
      //#endif

      //#ifdef GUI
      JFrame frame = new JFrame(title);
      ...
      //#endif

      //#ifdef LOOKANDFEEL
      initLookAndFeel(frame);
      //#endif

      //#ifdef GUI
      ...
      JCalcStandardFrame myFrame = new JCalcStandardFrame();
      JPanel myPane = myFrame.getPane();

      frame.getContentPane().add(myPane, ...);
      ...
      //#endif
}
```

Provides *frame* **to** LOOKANDFEEL

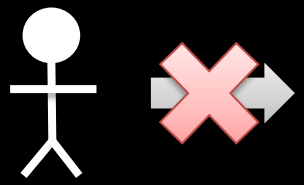
No dependencies found!

⋮

nth **fragment**

Another problem is…

Dependencies among heterogeneous artifacts!



```
<!-- #if($Bibtex) -->
...
<g:link controller="Publication" action="generateBib">Bibtex</g:link>

<!-- #end -->
```

show.gsp

```
//#if($ImportBibtex)
def generateBib()
        def publication = Publication.get(params.id)
        ...
}
//#end
```

PublicationController.groovy

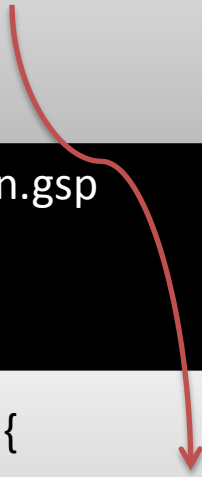
Emergent Interfaces…

…do not capture dependencies from different kinds of artifacts (.java, .groovy, .jsp, .gsp, etc.)

Dependencies between GSP-Groovy

```
<g:form id="log" controller="auth" action="signIn" >
    <!-- ... -->
    <g:textField name="username" required="true"/>
    <!-- ... -->
</g:form>
```

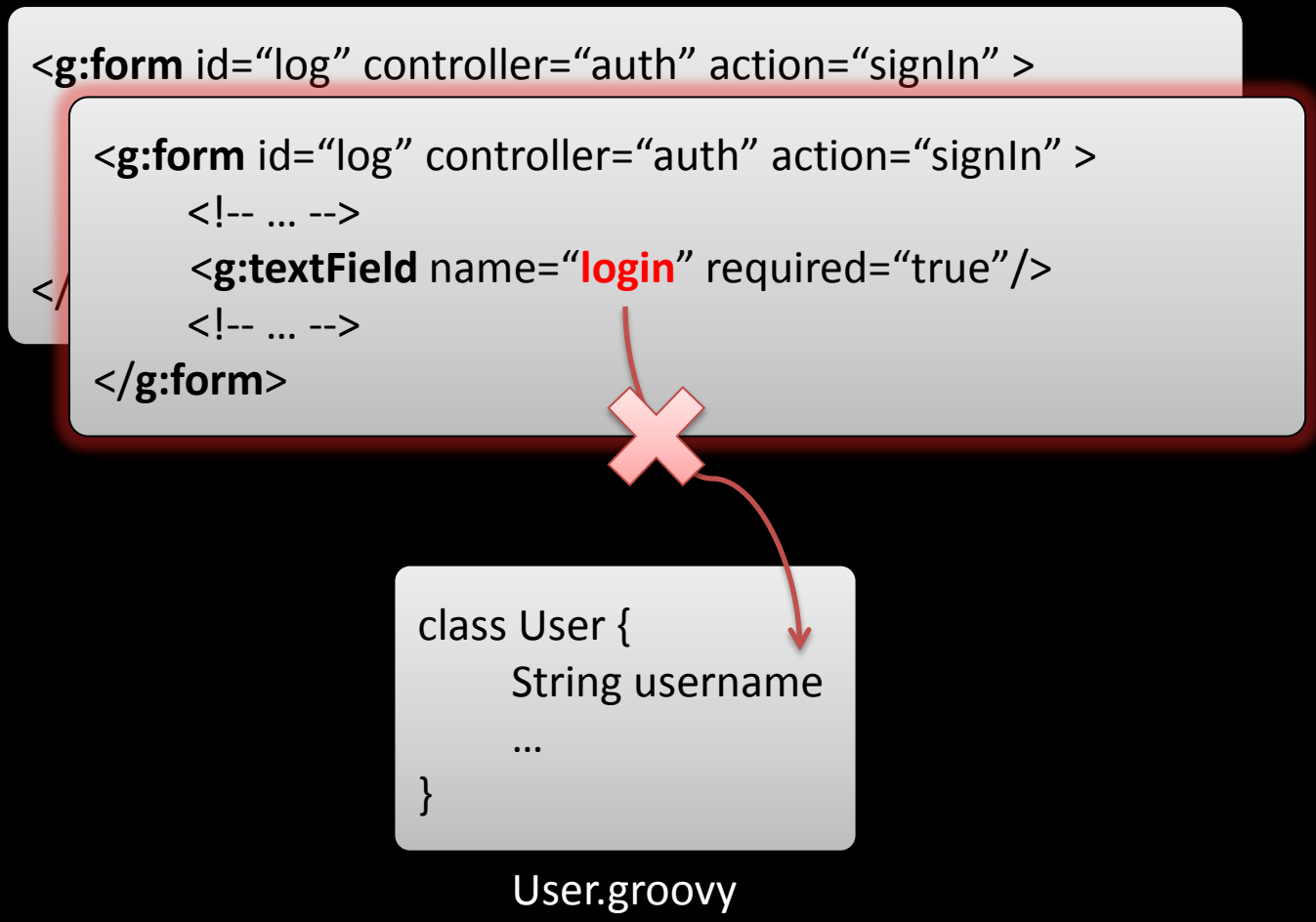
login.gsp



```
class User {
    String username
    ...
}
```

User.groovy

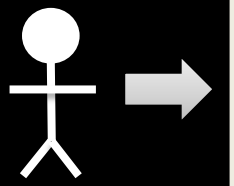
Dependencies between GSP-Groovy



To address these problems…

Emergent Feature Interfaces!

EFI for a single language settings

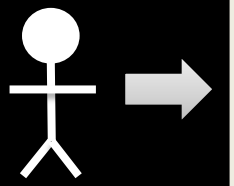


```
public static void main(String args[]) {
    ...
    //#ifdef GUI
    JFrame frame = new JFrame(title);
    initResolution(frame);
    //#endif

    //#ifdef LOOKANDFEEL
    initLookAndFeel(frame);
    //#endif
    ...
}
```

Requires *title* **from** PT_BR
Provides *frame* **to** LOOKANDFEEL

EI vs. EFI



```
public static void main(String args[]) {
    ...
    //#ifdef GUI
    JFrame frame = new JFrame(title);
    initResolution(frame);
    //#endif

    //#ifdef LOOKANDFEEL
    initLookAndFeel(frame);
    //#endif
    ...
```

Provides *frame* **to** LOOKANDFEEL

No dependencies found!

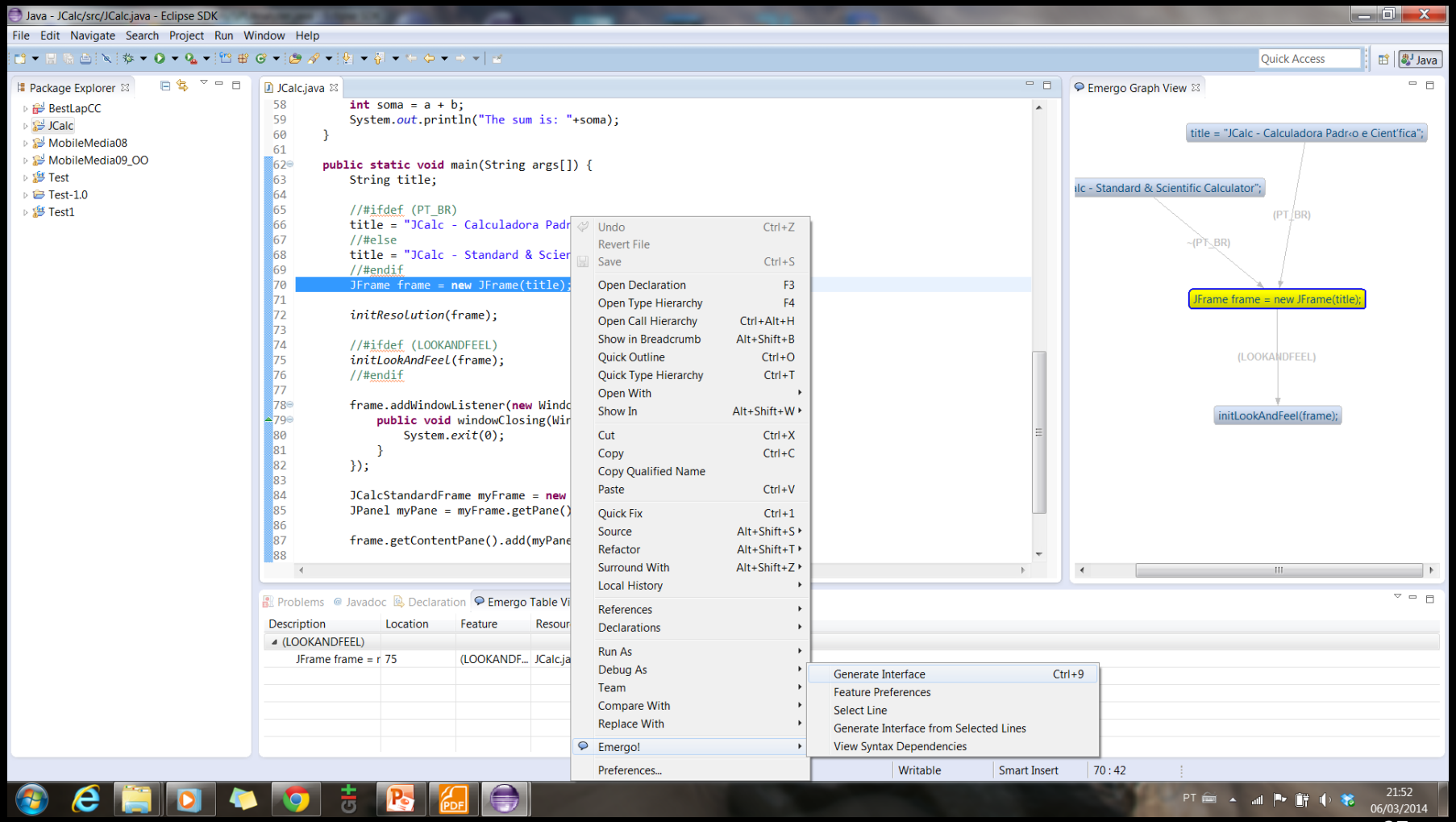
⋮

nth **fragment**

Requires *title* **from** PT_BR
Provides *frame* **to** LOOKANDFEEL

Emergo

A tool for improving the maintainability of SPLs



Single language evaluation (EI versus EFI)

Evaluation main goals

Is there any difference between EI and EFI in terms of size and precision?

How do EFI dependency detection compare to EI?

Study settings: experimental objects

System	Version	# methods	MDi	MDe
Best lap	1.0	343	20.7%	11.95%
Juggling	1.0	413	16.71%	11.14%
Lampiro	10.4.1	1538	2.6%	0.33%
MobileMedia	0.9	276	7.97%	5.8%
Mobile-rss	1.11.1	902	27.05%	23.84%

methods: Number of Methods; MDi: Methods with Directives;
MDe: Methods with Dependencies

Study settings: method and maintenance point selection

- Random selection of methods
  - Only methods with dependencies
  - No dependency: **empty interfaces**
  - We use MDe to select ten methods
- Random selection of maintenance points
  - Only suitable maintenance points
  - No comment or whitespace as well as method/class declaration
  - Feature-sensitive dataflow analysis is **intra-procedural**

Evaluation results

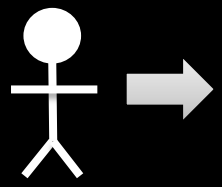
- EI return ‘No dependencies found!’ in all cases that the maintenance point is not an assignment (i.e. do not have required interfaces)
- Methods with many dependencies favor EFI
- EI do not provide support for feature selection as a maintenance point

Threats to validity

- Only 10 methods were chosen (five SPLs)
- Unavailable feature models
- Manually computing EI and EFI
- More studies are required to draw more general conclusions

Regarding the cross-language dependencies…

Cross-Language Analysis!

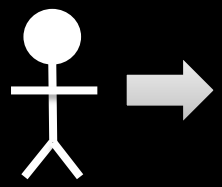


```
<!-- #if($Bibtex) -->
...
<g:link controller="Publication" action="generateBib"/>
...
<!-- #end -->
```

show.gsp

Requires *generateBib* **from** PublicationController

Cross-Language Analysis!



```
<!-- #if($Bibtex) -->
...
<g:link controller="Publication" action="generateBib"/>
...
<!-- #end -->
```

show.gsp

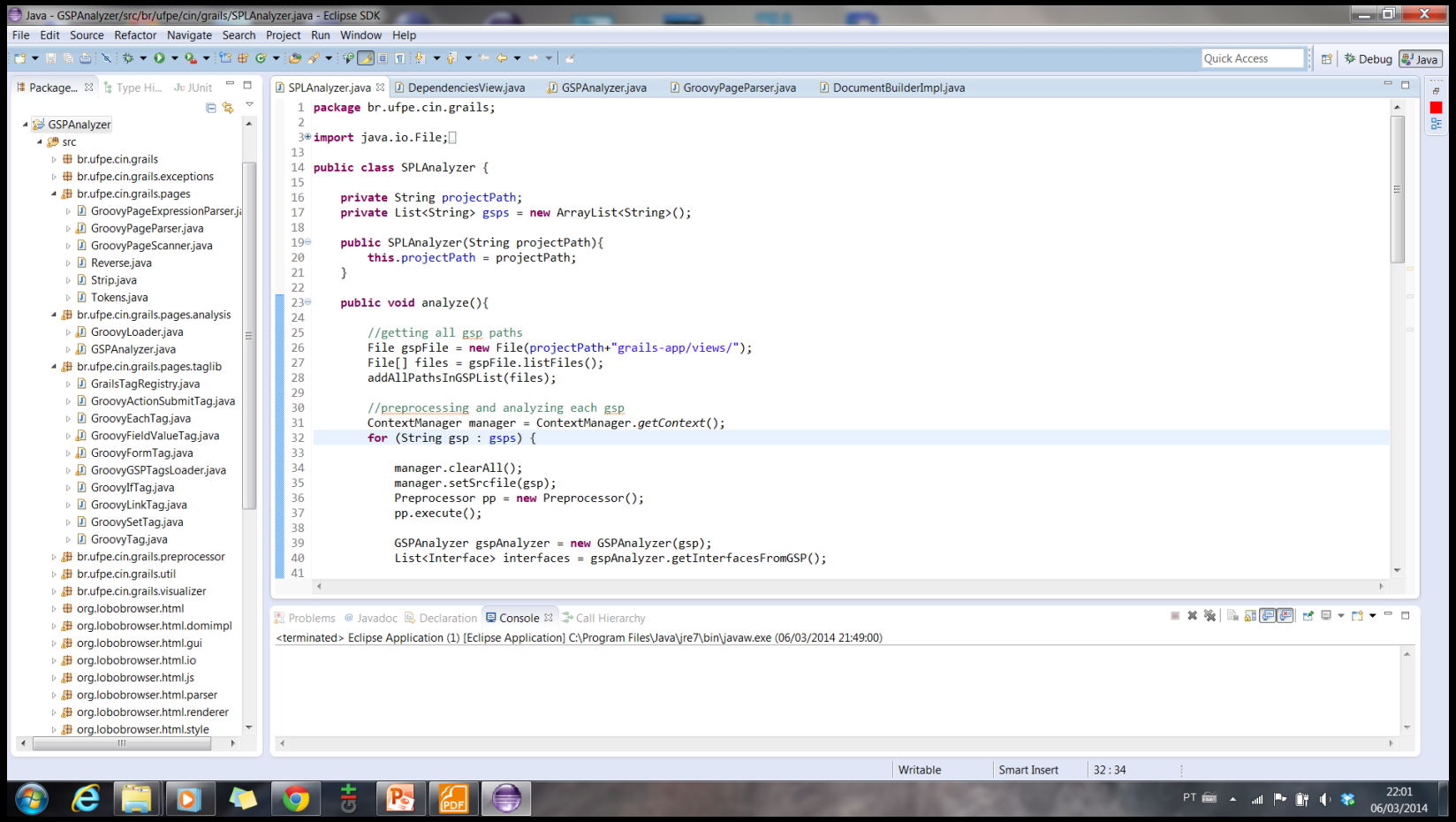
Requires *generateBib* **from** PublicationController

```
//#if($ImportBibtex)
def generateBib() {
        def publication = Publication.get(params.id)
        render(text: publication.generateBib(),
contentType: "text/txt", encoding: "UTF-8")
}
//#end
```

PublicationController.groovy

GSPAnalyzer

A tool for capturing cross-language feature dependencies in Grails SPLs



Multi-language evaluation (A case study with RGMS)

Evaluation main goals

Measure the number of occurences of cross-language dependencies in the RGMS

Identify different types of dependencies among heterogeneous artifacts

RGMS SPL

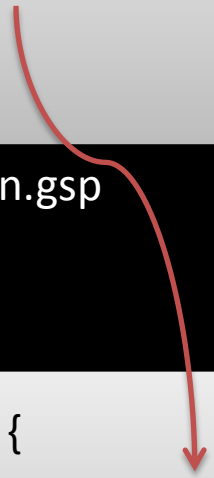
Research Group Management System
Developed during an SPL course
More than 50 students implemented it
Using the Grails framework

RGMS Stats	
# artifacts	371
# languages involved	7
# features	17
# LOC	40026

Satisfied dependency

```
<g:form id="log" controller="auth" action="signIn" >
    <!-- ... -->
    <g:textField name="username" required="true"/>
    <!-- ... -->
</g:form>
```

login.gsp



```
class User {
    String username
    ...
}
```

User.groovy

Unsatisfied dependency

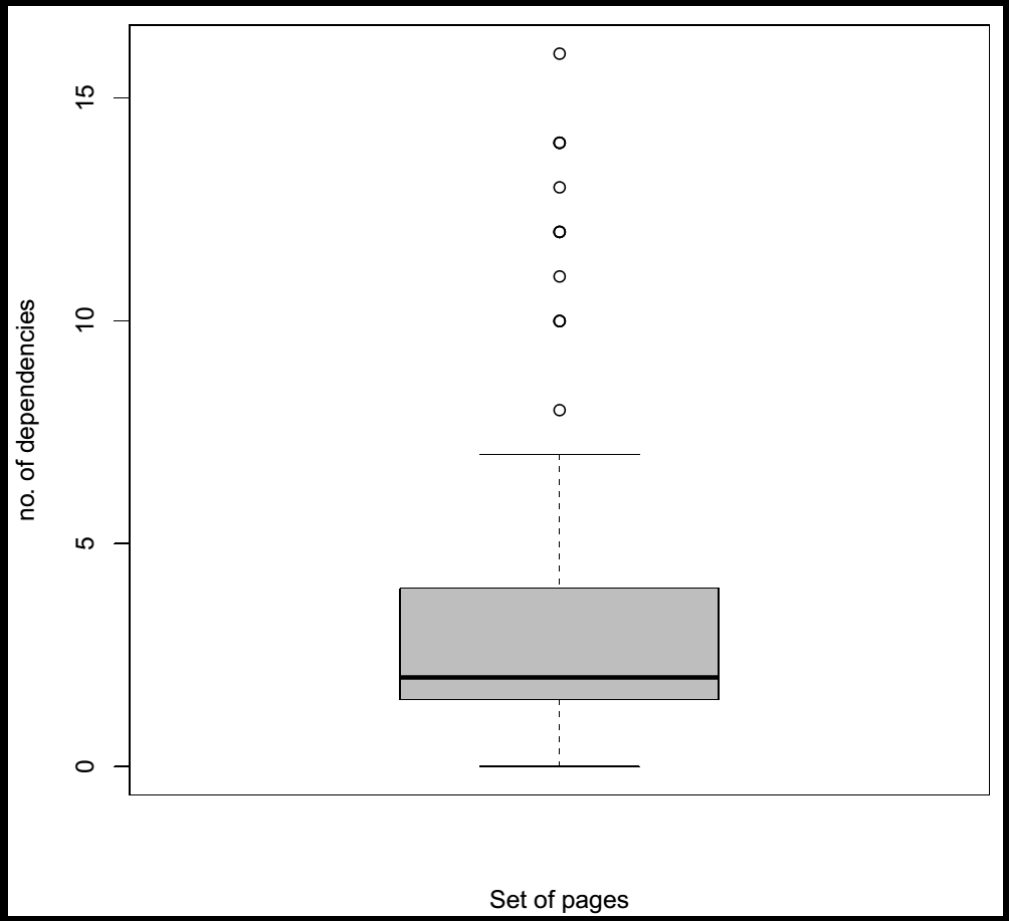
```
<fieldset class="buttons" >
        <!-- ... -->
        <g:actionSubmit action="share" value="Share on Facebook"/>
        <!-- ... -->
</g:form>
```

Periodico/show.gsp

```
class PeriodicoController {
        //...
        //#if($Facebook)
        def share() { ... }
        //#end
        //...
}
```

PeriodicoController.groovy

Evaluation results



Evaluation summary	
# pages analyzed	76
# satisfied dependencies	304
# unsatisfied dependencies	4

Threats to validity

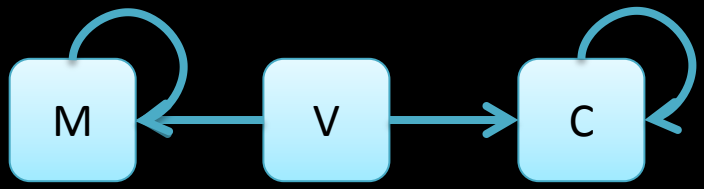
- RGMS is an academic software product line
- Our tool has not recognize the Groovy constructs *hasMany* and *belongsTo* yet
- We need of other case studies with different sizes, purposes, architectures, granularity, and complexity

Concluding remarks

- Our work helps to achieve independent feature comprehensibility and, consequently, to maintain features in SPLs
  - Looking for feature dependencies
  - Providing a global interface considering all fragments in an integrated way, and,
  - Supporting different types of artifacts

Limitations

- We capture only data dependencies
- So far, we only make syntactic analysis of Grails programs and
- Capture dependencies among...



Legend:

M Model

V View

C Controller

Future work

- Provide inter-procedural analysis to capture feature dependencies among methods, classes, and components
- More studies are required to draw more general conclusions
  - Benefits of supporting different types of artifacts

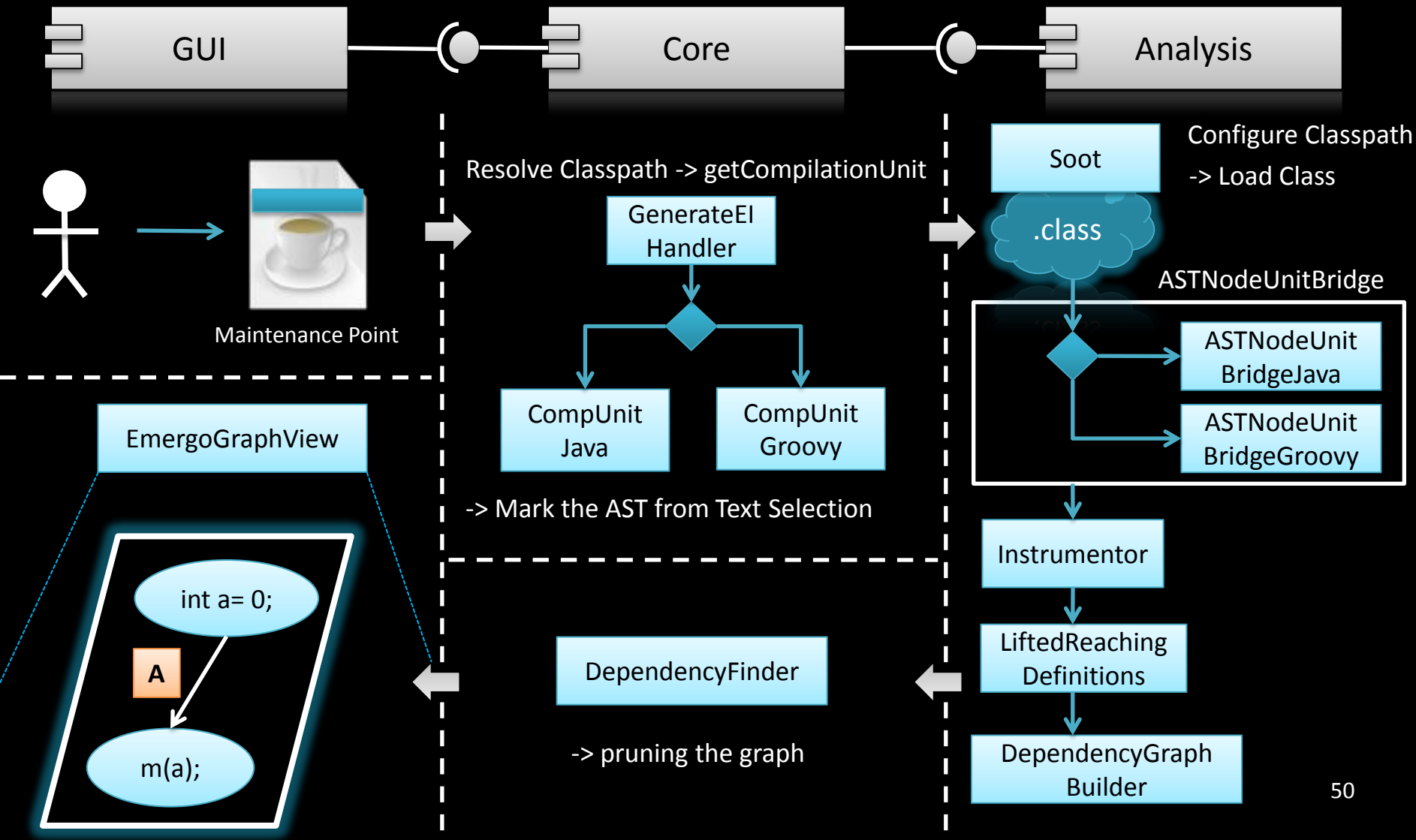
Modular Reasoning for Software Product Lines with Emergent Feature Interfaces

Jean Melo (jccm@cin.ufpe.br)

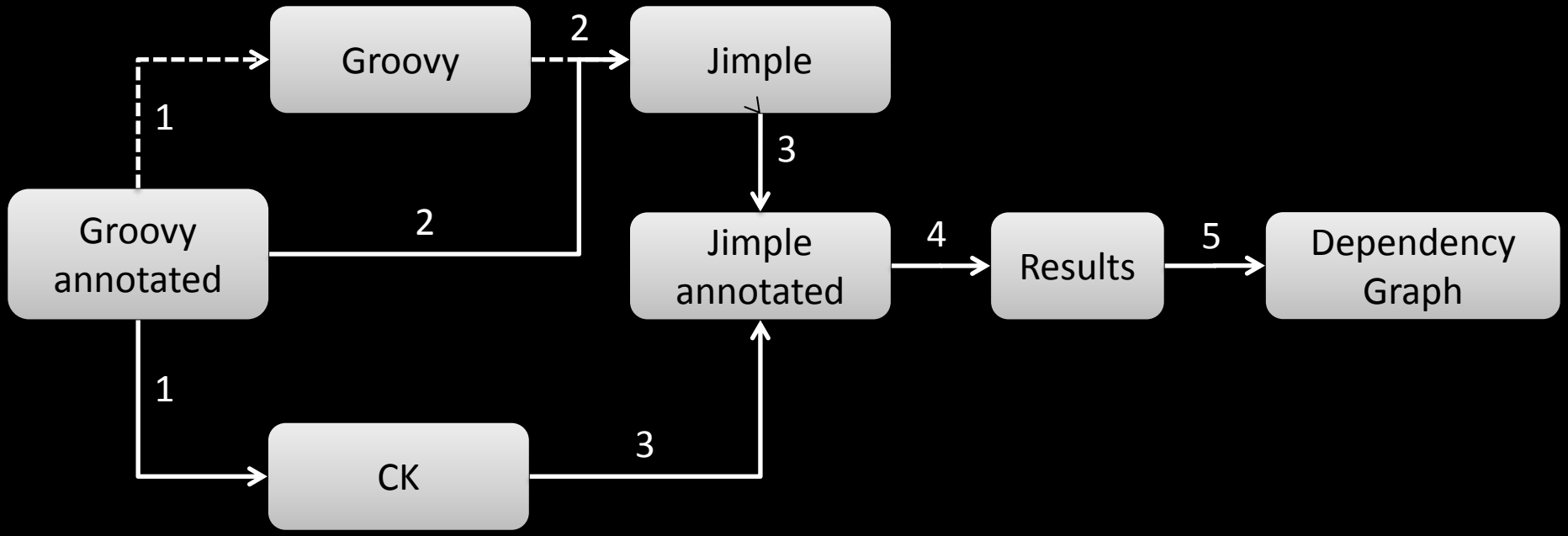
Advisor: Paulo Borba

BACK-UP SLIDES

Emergo's Architecture

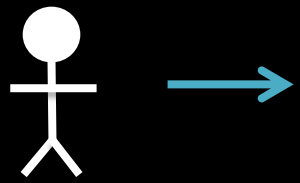


Our proposal in a nutshell



(1) Preprocessing;
(2) Compiling;
(3) Instrumenting;
(4) Dataflow analysis;
(5) Transformation of the analysis' results.

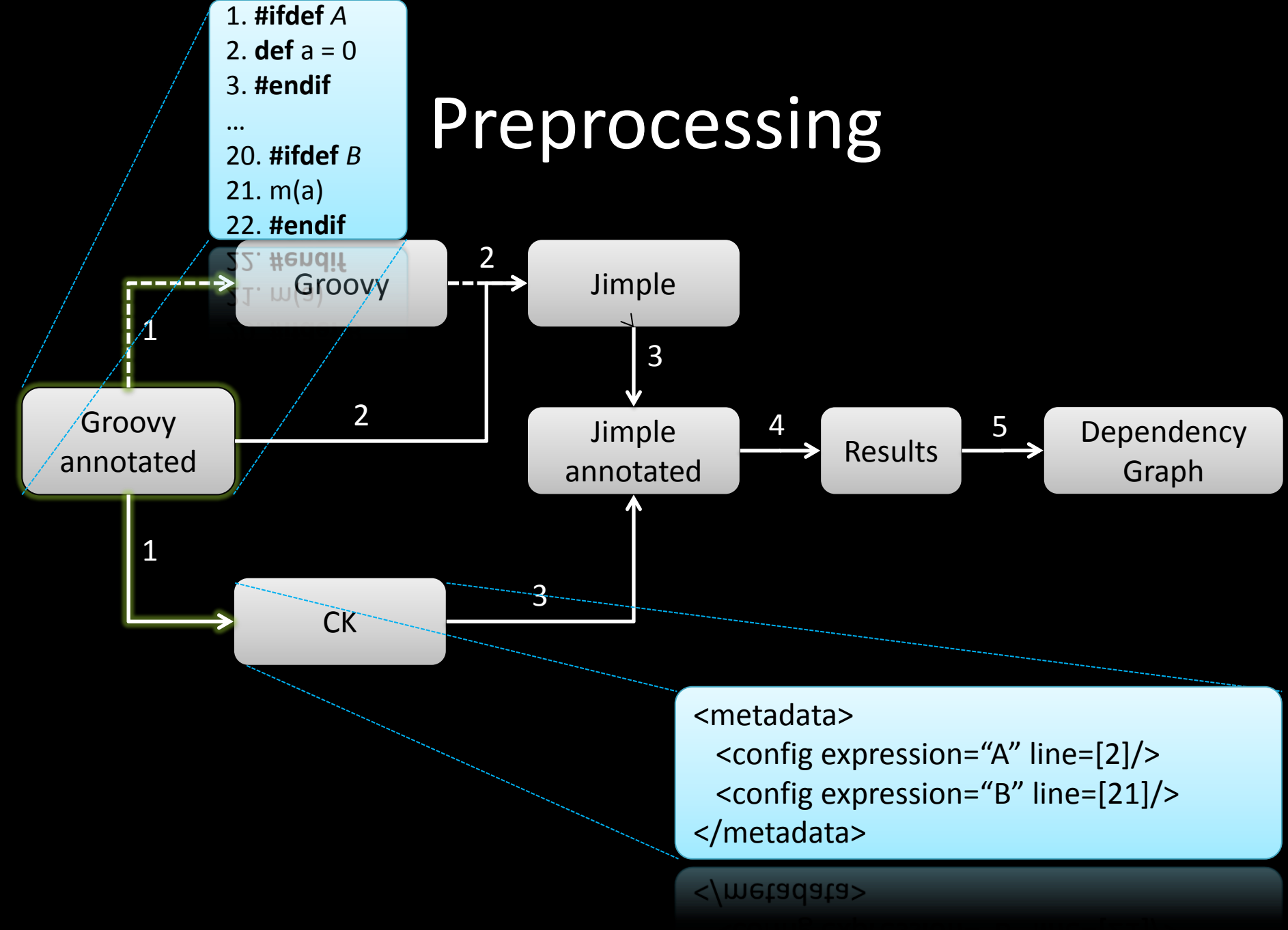
Step-by-step



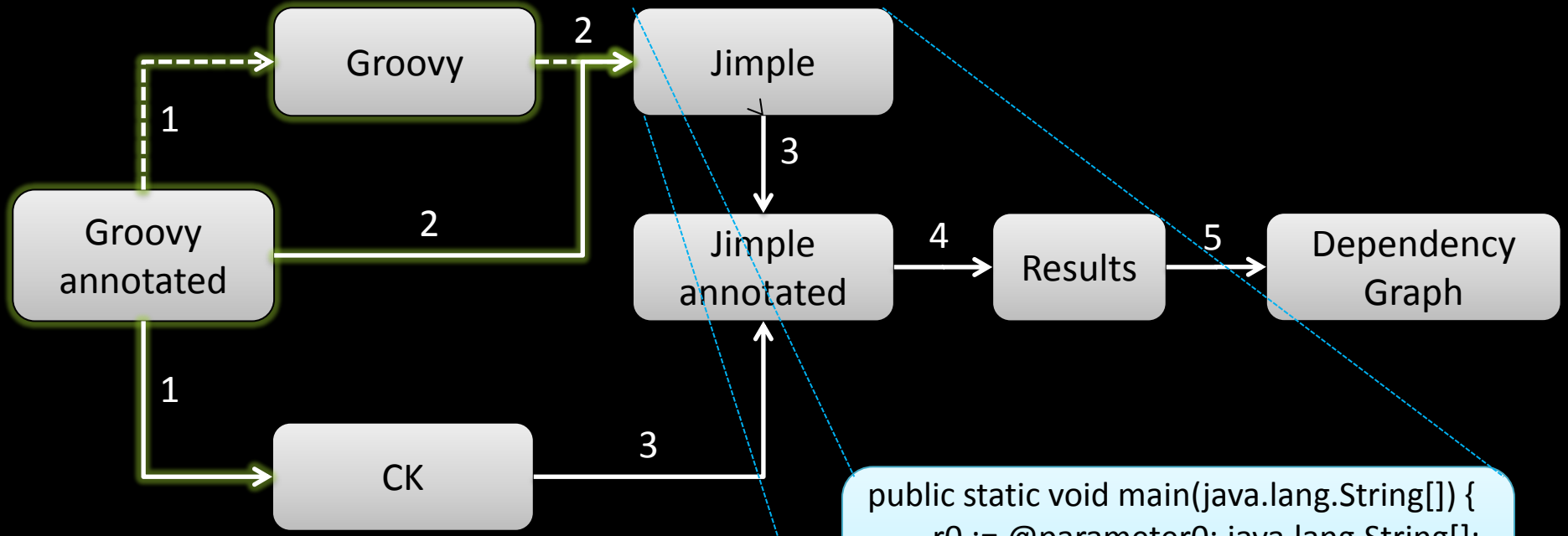
```
static void main(String[] args) {
        //#ifdef A
        def a = 0
        //#endif
        ...
        //#ifdef B
        m(a)
        //#endif
}
```

Sample code

Preprocessing

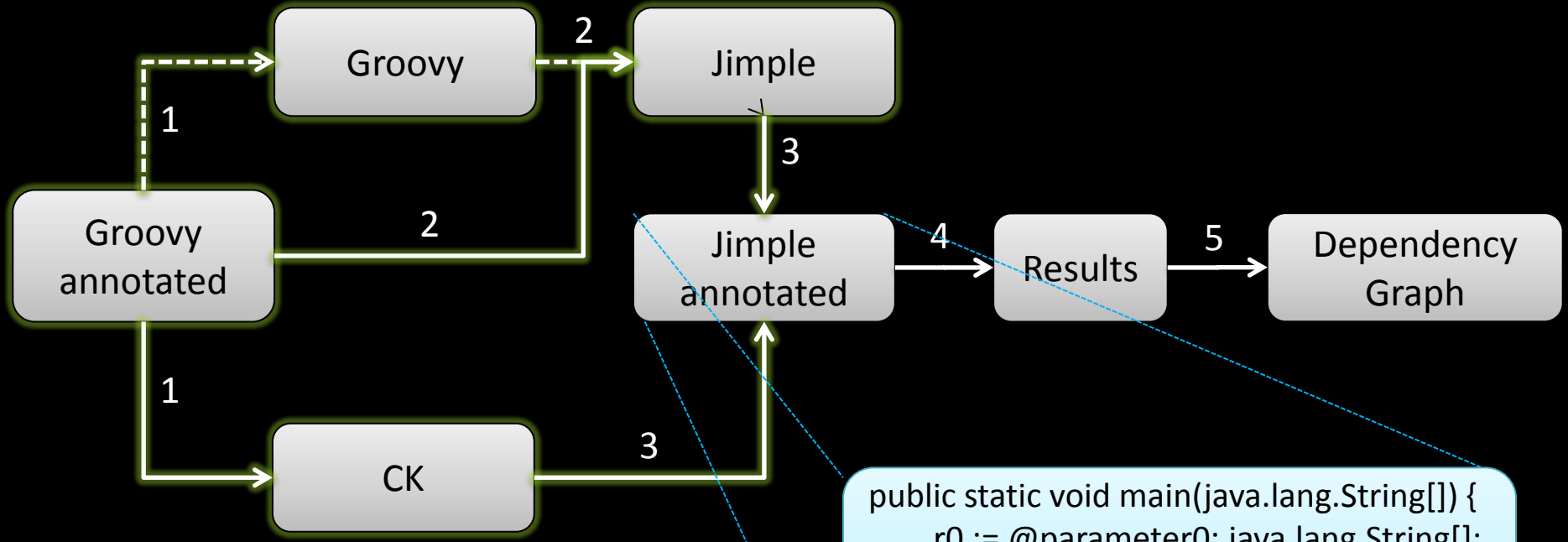


Compiling



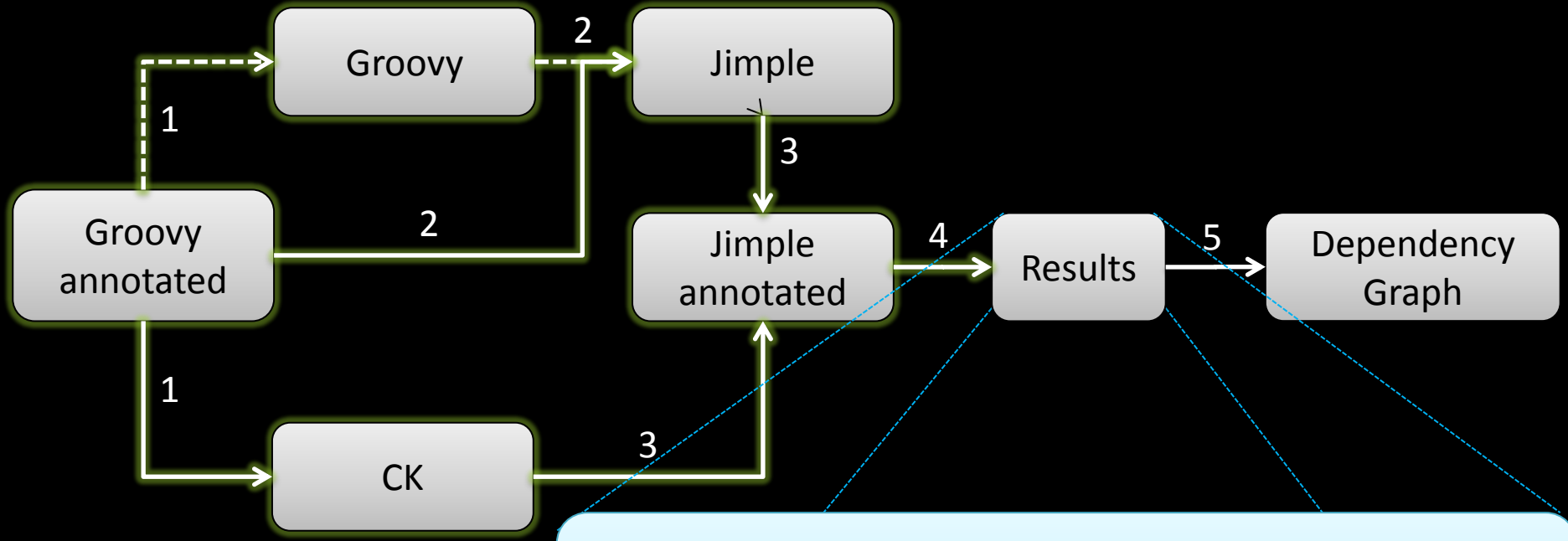

```
public static void main(java.lang.String[]) {
    r0 := @parameter0: java.lang.String[];
    b0 = 0;
    …
    virtualinvoke<void m(int)>(b0);
    return;
}
```

Instrumenting




```
public static void main(java.lang.String[]) {
    r0 := @parameter0: java.lang.String[];
    b0 = 0;  A
    …
    virtualinvoke<void m(int)>(b0);  B
    return;
}
```

Dataflow analysis



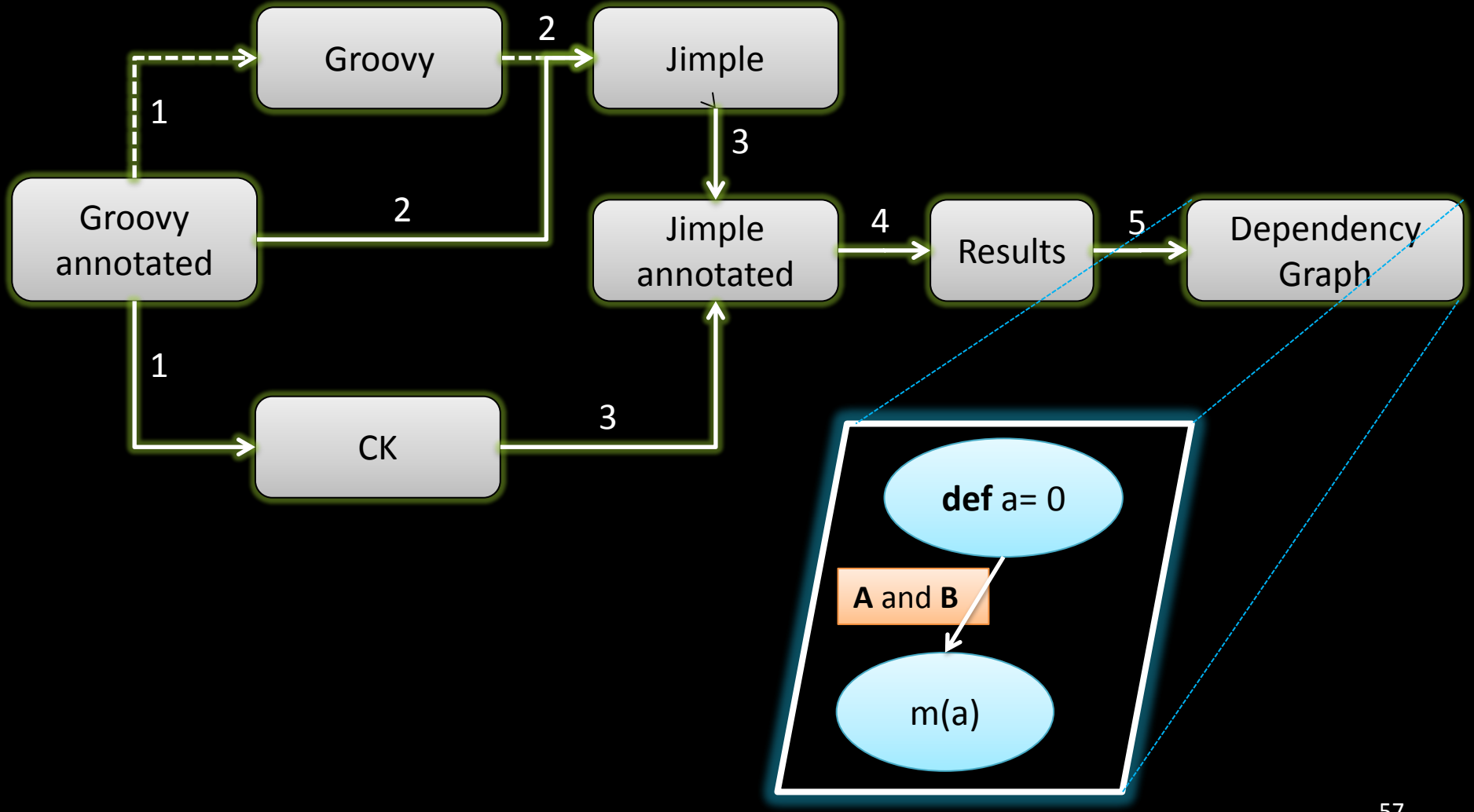
Legend:

0 -> No feature
1 -> A
2 -> B
3 -> A and B

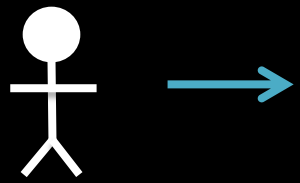
r0 := @parameter0: java.lang.String[] => **{3={}, 2={}, 1={}, 0={}}**
b0 = 0 => **{3={}, 2={}, 1={}, 0={}}**
...
virtualinvoke r1.<void m(int)>(b0) => **{3={b0 = 0}, 2={}, 1={}, 0={}}**
...

...

Transformation of the analysis' results



Emergent Feature Interface



```
static void main(String[] args) {
        //#ifdef A
        def a = 0
        //#endif
        ...
        //#ifdef B
        m(a)
        //#endif
}
```

Provides a = 0 **to** B
[**Configuration**: A and B]

Dependency between GSP-Groovy

```
…
<form id="log" action="auth/signIn" method="post" >
        <td>Username: </td>
        <input name="username" required="true”/>
</form>
…
```

.gsp

```
class Member {
        String username
        …
}
```

.groovy

Dependency between GSP-Groovy

```
…
<form id="log" action="auth/signIn" method="post" >
        <td>Username: </td>
        <input name="username" required="true”/>
</form>
…
```

.gsp

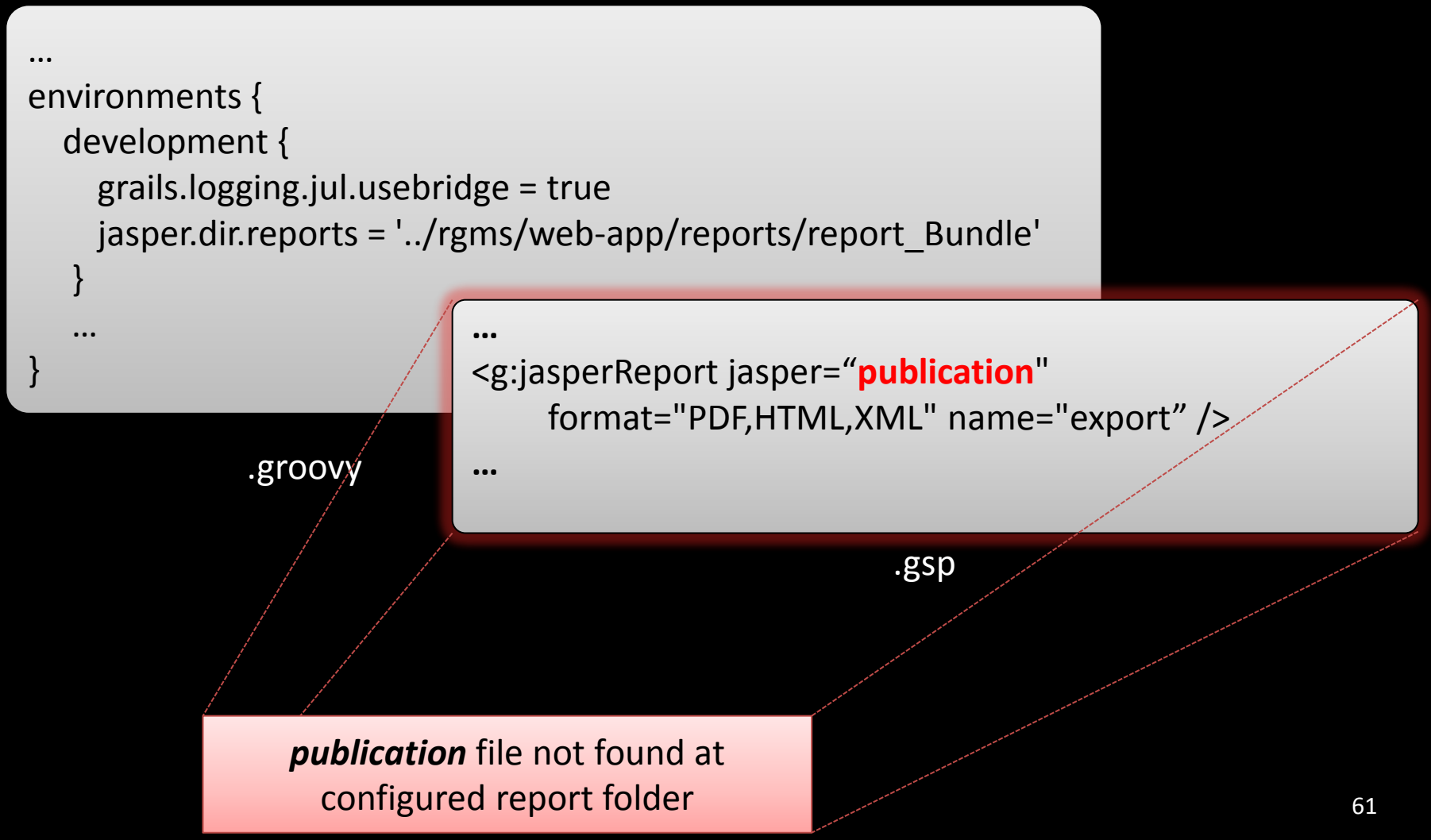
```
class Member {
        String username
```

```
…
<form id="log" action="auth/signIn" method="post" >
        <td>Username: </td>
        <input name=“login” required=“true”/>
</form>
…
```

.gsp

Unmatched ***login*** element (params) between page and code.

Dependency among GSP-Groovy-jrxml



Dependency between Groovy code

```
def user = Member.findByUsername(params.username)
…
log.info "Redirecting to '${targetUri}'.”
redirect(uri: "/member/list")
…
```

.groovy

Dependency between Groovy code

```
def user = Member.findByUsername(params.username)
…
log.info "Redirecting to '${target
redirect(uri: "/member/list")
…
```

.groovy

```
def user = Member.findByUsername(params.username)

if (user?.passwordChangeRequiredOnNextLogon) {
        log.info "Redirecting to '${targetUri}'."
        redirect(action: newPassword)
} else {
        log.info "Redirecting to '${targetUri}'."
        redirect(uri: "/member/list")
}
```

.groovy

newPassword action not found on the current controller.

Dependency between GSP-Groovy

```
…
<form id="log" action="auth/signIn" method="post”>
…
</form>
```

.gsp

```
…
def signIn = {
// Authentication code
}
…
```

.groovy

Dependency between GSP-Groovy

```
…
<form id="log" action="auth/signIn" method="post”>
…
</form>
```

.gsp

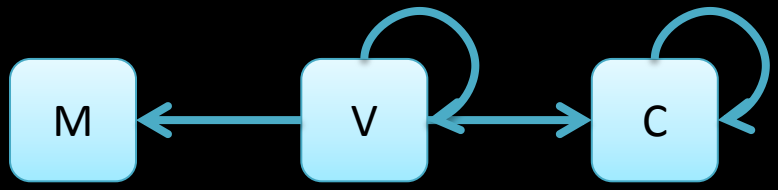
```
…
def signIn = {
// Authentication code
}
…
```

.groovy

```
…
<form id="log" action="auth/login" method="post”>
…
</form>
```

login action not found on the auth controller.

We have dependencies among…



Legend:

M Model

V View

C Controller