Feature Extraction

Marco Túlio Valente
DCC - UFMG

Software Product Lines

- Surgimento do termo: Workshop do SEI, 1996
  - Ou então: On the Design and Development of Program Families, David Parnas, 1976

- SEI: “A SPL is a set of software intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.”

- Inspiração: linhas produtos industriais (customização em massa)
  - Exemplo: indústria automobilística
  - Plataforma de carro comum; a partir dessa plataforma são fabricados diversos carros; que possuem diversos itens opcionais

Motivação: HTC Android

http://developer.htc.com/

htc
quietly brilliant

Developer center

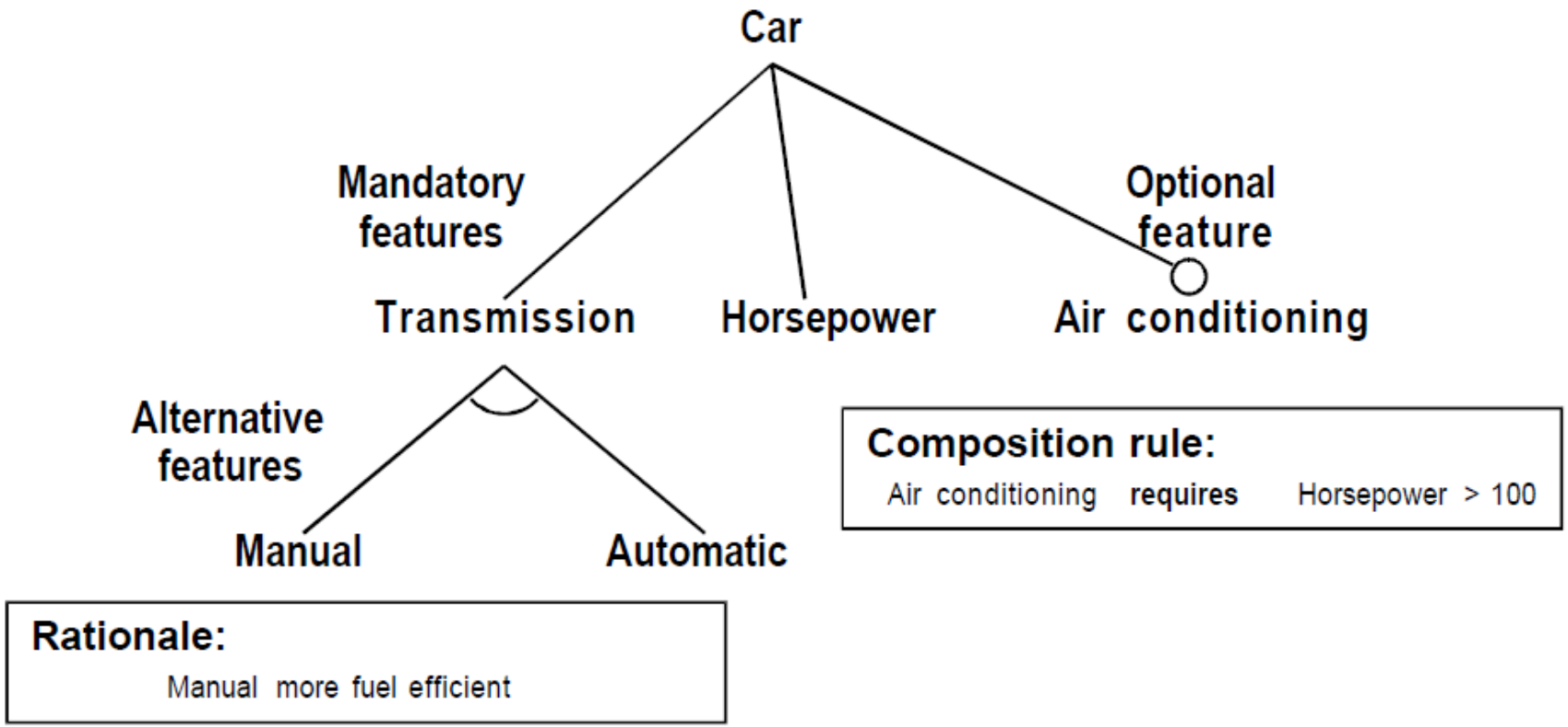
placeholder

The HTC Developer Center is a place for developers to obtain essential resources to help you in developing great applications on HTC's Android and Windows Mobile phones.

Kernel Source Code and Binaries for HTC Android Phones

Name	File Size	Download
HTC Desire S – 2.6.35 kernel source code	89.5 MB	
HTC Thunderbolt - 2.6.32 kernel source code	87 MB	
HTC Incredible S - 2.6.35 kernel source code	89.5 MB	
T-Mobile myTouch 3G Slide - Froyo MR -2.6.32 kernel	80.2 MB	
HTC Wildfire - Froyo MR - 2.6.32 kernel source code	82.3 MB	
Droid Eris by HTC - Eclair MR - 2.6.29 kernel source code	75.2 MB	
HTC Gratia - 2.6.32 kernel source code	86.3 MB	
HTC Inspire 4G - 2.6.32 kernel source code	87 MB	
T-Mobile myTouch 4G – Froyo QMR – 2.6.32 kernel source code	81.9 MB	
HTC Desire HD – Froyo QMR – 2.6.32 kernel source code	81.8 MB	
HTC Aria – Froyo MR – 2.6.32 kernel source code	86.3 MB	

Feature Model



SPL Extraction

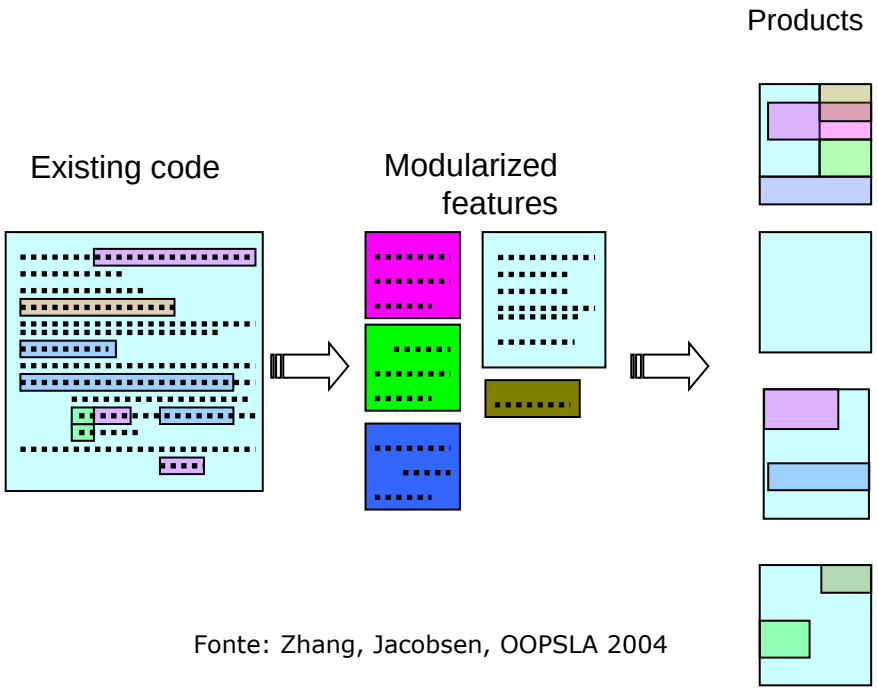
Motivation

- SPL extraction is a time-consuming task

- Two approaches for extracting SPL:
  - Compositional-based
  - Annotation-based

Compositional-based Approaches

- Aspects (AspectJ)



Fonte: Zhang, Jacobsen, OOPSLA 2004

- Physical “modularization”
- Slow adoption

Annotation-based Approaches

- Example: preprocessors

```
boolean push(Object o) {
  Lock lk = new Lock();
  if (lk.lock() == null) {
    Log.log("lock failed");
    return false;
  }
  elements[top++]= o;
  size++;
  lk.unlock();
  if ((size % 10) == 0)
    snapshot("db");
  if ((size % 100) == 0)
    replicate("db","srv2");
  return true;
}
```

```
boolean push(Object o) {
  #ifdef MULTITHREADING
  Lock lk = new Lock();
  if (lk.lock() == null) {
    #ifdef LOGGING
    Log.log("lock failed");
    #endif
    return false;
  }
  #endif
  elements[top++]= o;
  size++;
  #ifdef MULTITHREADING
  lk.unlock();
  #endif
  #ifdef SNAPSHOT
  if ((size % 10) == 0)
    snapshot("db");
  #endif
  #ifdef REPLICATION
  if ((size % 100) == 0)
    replicate("db","srv2");
  #endif
  return true;
}
```

- Widely adopted
- Problems: annotation hell; code pollution

Visual Annotations

- CIDE: Colored IDE (Eclipse + background colors)

```
boolean push(Object o) {
  Lock lk = new Lock();
  if (lk.lock() == null) {
    Log.log("lock failed");
    return false;
  }
  elements[top++]= o;
  size++;
  lk.unlock();
  if ((size % 10) == 0)
    snapshot("db");
  if ((size % 100) == 0)
    replicate("db","srv2");
  return true;
}
```

```
boolean push(Object o) {
  Lock lk = new Lock();
  if (lk.lock() == null) {
    Log.log("lock failed");
    return false;
  }
  elements[top++]= o;
  size++;
  lk.unlock();
  if ((size % 10) == 0)
    snapshot("db");
  if ((size % 100) == 0)
    replicate("db","srv2");
  return true;
}
```

- Less code polution than #ifdefs
- Problem: colors assigned manually (repetitive, error-prone etc)

Extracting Software Product Lines: A Case Study Using Conditional Compilation

Marcus Vinícius Couto, Marco Tulio Valente
Eduardo Figueiredo

15th CSMR - March, 2011 – Oldenburg, Germany

Software Product Lines

- Goal: variable software systems
- Systems: core components + features components
- Product: core + specific set of features

Our Solution: ArgoUML-SPL

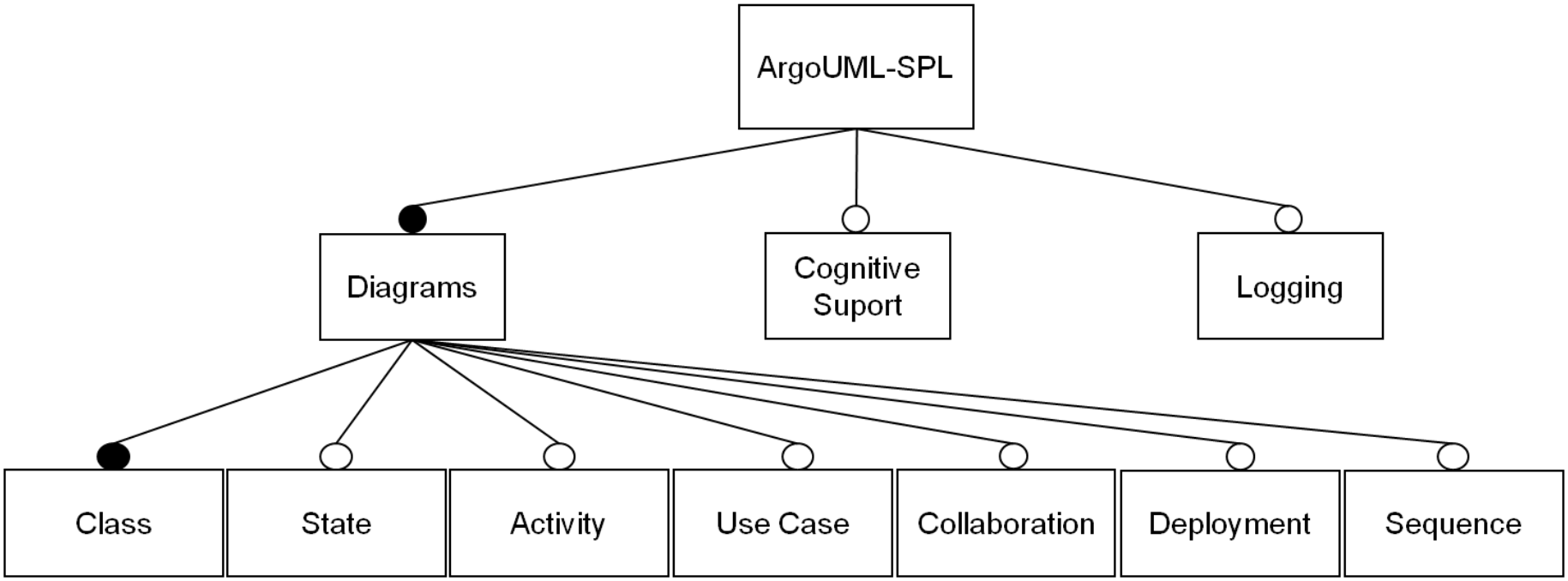
- We decided to extract our own -- complex and real -- SPL
- Target system: ArgoUML modelling tool (120 KLOC)
- Eight features (37 KLOC ~ 31%)
- Technology: conditional compilation
- Baseline for comparison with tools (e.g. CIDE+) and languages (e.g. aspects) for SPL implementation

In this CSMR Paper/Talk

- We report our experience extracting a SPL for ArgoUML
  - ArgoUML-SPL
  - Extraction Process
  - Characterization of the Extracted SPL

ArgoUML-SPL

Feature Model



Feature Selection Criteria

- Relevance:
  - Typical functional requirements (diagrams)
  - Typical non-functional concern (logging)
  - Typical optional feature (cognitive support)

- Complexity:
  - Size
  - Crosscutting behavior (e.g. logging)
  - Feature tangling
  - Feature nesting

Extraction Process

Extraction Process

- Pre-processor: `javapp`
  - http://www.slashdev.ca/javapp

- Extraction Process:
  - ArgoUML's documentation:
    - Search for components that implement a given feature
    - E.g.: package org.argouml.cognitive
  - Eclipse Search:
    - Search for lines of code that reference such components
  - Delimit such lines with #ifdefs and #endifs

- Effort:
  - 180 hours for annotating the code
  - 40 hours for testing the various products

Example

```
1  public List getInEdges(Object port) {
2    if (Model.getFacade().isAStateVertex(port)) {
3      return new ArrayList(Model.getFacade().getIncomings(port));
4    }
5    //#if defined(LOGGING)
6    //@#$LPS-LOGGING:GranularityType:Statement
7    //@#$LPS-LOGGING:Localization:BeforeReturn
8    LOG.debug("TODO: getInEdges of MState");
9    //#endif
10   return Collections.EMPTY_LIST;
11 }
```

Characterization

Metrics

- Metric-suite proposed by Liebig et al. [ICSE 2010]

- Four types of metrics:
  A. Size
  B. Crosscutting
  C. Granularity
  D. Location

(A) Size Metrics

- How many LOC have you annotated for each feature?
- How many packages?
- How many classes?

Size Metrics

Product	LOC	NOP	NOC
Original, non-SPL based	120,348	81	1,666
Only COGNITIVE SUPPORT disabled	104,029	73	1,451
Only ACTIVITY DIAGRAM disabled	118,066	79	1,648
Only STATE DIAGRAM disabled	116,431	81	1,631
Only COLLABORATION DIAGRAM disabled	118,769	79	1,647
Only SEQUENCE DIAGRAM disabled	114,969	77	1,608
Only USE CASE DIAGRAM disabled	117,636	78	1,625
Only DEPLOYMENT DIAGRAM disabled	117,201	79	1,633
Only LOGGING disabled	118,189	81	1,666
All the features disabled	82,924	55	1,243

LOC: Lines of code; NOP: Number of packages; NOC: Number of classes

Size Metrics

Feature	LOF	
Cognitive Support	16,319	13.59%
Activity Diagram	2,282	1.90%
State Diagram	3,917	3.25%
Collaboration Diagram	1,579	1.31%
Sequence Diagram	5,379	4.47%
Use Case Diagram	2,712	2.25%
Deployment Diagram	3,147	2.61%
Logging	2,159	1.79%
Total	37,424	31.10%

LOF: Lines of Feature code

(B) Crosscutting Metrics

- How are the #ifdefs distributed over the code?
- How many #ifdefs are allocated for each feature?
- Are “boolean expressions” common (e.g. #ifdef A && B)?

Crosscutting Metrics (Example)

```
1  ...
2  //#if defined(STATEDIAGRAM) or defined(ACTIVITYDIAGRAM)
3  if ((
4    //#if defined(STATEDIAGRAM)
5    type == DiagramType.State
6    //#endif
7      //#if defined(STATEDIAGRAM) and defined(ACTIVITYDIAGRAM)
8      ||
9      //#endif
10     //#if defined(ACTIVITYDIAGRAM)
11     type == DiagramType.Activity
12     //#endif
13     )
14    && machine == null) {
15    diagram = createDiagram(diagramClasses.get(type), null, namespace);
16 } else {
17 //#endif
18   diagram = createDiagram(diagramClasses.get(type), namespace, machine);
19 //#if defined(STATEDIAGRAM) or defined(ACTIVITYDIAGRAM)
20 }
21 //#endif
22 ...
```

SD(STATEDIAGRAM) = 3

SD(ACTIVITYDIAGRAM) = 4

TD(STATEDIAGRAM,ACTIVITYDIAGRAM) = 3

SD: Scattering Degree; TD: Tangling Degree

Scattering Degree (SD)

Feature	SD	LOF/SD
Cognitive Support	319	51.16
Activity Diagram	136	16.78
State Diagram	167	23.46
Collaboration Diagram	89	17.74
Sequence Diagram	109	49.35
Use Case Diagram	74	36.65
Deployment Diagram	64	49.17
Logging	1287	1.68

Tangling Degree (TD)

Pairs of Features	TD
(State Diagram, Activity Diagram)	66
(Sequence Diagram, Collaboration Diagram)	25
(Cognitive Support , Sequence Diagram)	1
(Cognitive Support , Deployment Diagram)	13

(C) Granularity Metrics

- What is the granularity of the annotated lines of code?
  - How many full packages have been annotated?
  - And classes?
  - And methods?
  - And just method bodies?
  - And just single statements?
  - And just single expressions?

Granularity Metrics

Feature	Package	Class	Interface Method	Method	Method Body
Cognitive Support	11	8	1	10	5
Activity Diagram	2	31	0	6	6
State Diagram	0	48	0	15	2
Collaboration Diagram	2	8	0	5	3
Sequence Diagram	4	5	0	1	3
Use Case Diagram	3	1	0	1	0
Deployment Diagram	2	14	0	0	0
Logging	0	0	0	3	15

Granularity Metrics

Feature	ClassSignature	Statement	Attribute	Expression
Cognitive Support	2	49	3	2
Activity Diagram	0	59	2	6
State Diagram	0	22	2	5
Collaboration Diagram	0	40	1	1
Sequence Diagram	0	31	2	3
Use Case Diagram	0	22	1	0
Deployment Diagram	0	13	1	3
Logging	0	789	241	1

(D) Localization Metrics

- Where are the #ifdefs located?
  - In the beginning of a method
  - In the end of a method
  - Before a return statement

- Important for example to evaluate a migration to composition-based approaches (e.g. aspects)

Localization Metrics

Feature	StartMethod	EndMethod	BeforeReturn	NestedStatement
Cognitive Support	3	5	0	10
Activity Diagram	2	20	2	19
State Diagram	2	19	3	12
Collaboration Diagram	1	10	3	3
Sequence Diagram	0	9	3	7
Use Case Diagram	0	2	0	1
Deployment Diagram	0	0	0	3
Logging	127	21	89	336

Demo CIDE+