Enhancing Feature Interfaces for Supporting Software Product Line Maintenance

Bruno B. P. Cafeo
bcafeo@inf.puc-rio.br
Motivation

Software Product Line (SPL)

Feature Model
Motivation

Boundaries of features and modules of a programming language

Legend: PE – Program Element

Feature Model

Source code

PE1
PE2
PE3
PE4
PE5
PE6
PE7
PE8
Feature Dependencies in the source code of a SPL

Legend: PE – Program Element
Motivation

Feature Dependencies in the source code of a SPL

Legend: PE – Program Element

Feature Model

Source code
Feature Dependencies in the source code of a SPL

Legend: PE – Program Element  **XX** Maintenance problems related to the presence of a feature dependency

Feature Model

Source code

Legend: PE – Program Element  **XX** Maintenance problems related to the presence of a feature dependency
Two points to explore

- Identification and understanding of feature dependency properties and their impact on SPL maintenance
- Feature modularity improvement
Two points to explore

- Identification and understanding of feature dependency properties and their impact on SPL maintenance
- Feature modularity improvement
There is no conceptual framework that characterizes feature dependency properties in the source code.

Revealing properties that may exert an impact on SPL maintenance would be interesting.
Results

- Identification of properties and definition of metrics based on these properties

- Correlation of these metrics with SPL maintenance problems

- Comparison of our metrics with conventional metrics
Two points to explore

- Identification and understanding of feature dependency properties and their impact on SPL maintenance

- Feature modularity improvement
As in a conventional stand-alone software (i.e. non-SPLs), interfaces should help the understanding of the communication between features in SPLs.

Program elements configuring dependencies are part of an implicit feature interface.
Challenge

Complex feature interfaces

- Feature interfaces may become large
- Several elements are member of an implicit interface
- Implicit feature interfaces are not cohesive
- Groups of elements act together for the purpose of a dependency

REGULAR_ACCOUNT

- account_number
- type
- sort_code
- balance
- paid_in[]
- paid_out[]
- getEndofMonthBalance()
- getBalance()
- makeAPayment()
- getFullStatement()
- viewAccountDetails()
- getTaxes()
Towards a Solution

- The proposed solution relies on the idea of *Interface Segregation Principle (ISP)* that states that

  
  “clients should not be forced to depend upon interfaces that they do not use”

- We observe the idea of ISP from the point of view of SPL maintenance and we argue that

  Developers should not be forced to understand parts of an interface that are not useful to their tasks
Towards a Solution (simple example)

Identifying feature interface elements

Feature Interface

att1
att2
att3
att4
att5
att6
att7
method1
method2
method3
method4
method5

Feature blue

Legend: PE – Program Element

Source code

{att1, att2, method1, method2}

PE1

PE6

PE4

PE2

PE3

PE5

PE7

PE8

{att3, att4, att5, method3}

{method4, method5, att6, att7}
Towards a Solution (simple example)

Understanding the feature interface

Feature Interface

- att1, att2, method1, method2
- att6, att7, method4, method5

Relevant elements of the collaboration

Feature blue

Legend: PE – Program Element

Source code

Important part of the code

{att1, att2, method1, method2}

{att3, att4, att5, method3}

{method4, method5, att6, att7}

Legend:

PE1
PE2
PE3
PE4
PE5
PE6
PE7
PE8

6/2/14
- Conduct empirical studies to compare
  - The reduction of maintenance side effects
  - The SPL maintenance effort
Thanks!
Enhancing Feature Interfaces for Supporting Software Product Line Maintenance

Bruno B. P. Cafeo
bcafeo@inf.puc-rio.br