Session 3

High level dynamic analysis views

Bas Cornelissen, *Delft Univ. of Technology*
Combining Reverse Engineering Techniques for Product Lines

Dharmalingam Ganesan, Isabel John and Jens Knodel
Motivation

• Migration towards product lines
  – “Recover and reconstruct” strategy

• Approach
  – Identify and prioritize relevant assets
  – Combine reverse engineering techniques to recover these assets
  – Visualization
    • Hierarchical graphs
    • Message sequence charts
Combining techniques

- Information sources: code, documentation and running system
Document analysis

• CaVE method
  – FAQs, tutorials, user guides, designer documentation

• Extraction
  Concepts, features, use cases, relationships

• Conceptual architectural view
Static analysis

• SAVE method
  – Compare as-intended architecture with as-is architecture
  – Use conceptual model as starting point
  – Obtain source model by parsing the code
  – Reduce no. of lowlevel components
  – Iterate until initial model and source model are aligned
Dynamic analysis

- Extraction of behavioral views
Final words

• Systematical reconstruction
  – Recovery of assets for use in product lines
  – Combination of techniques covers multiple grounds

• Effort?
• Other visualizations?
• Scalability?
Higher Abstractions for Dynamic Analysis

Marcus Denker, Orla Greevy and Michele Lanza
Motivation

• Dynamic analysis
  – Code instrumentation & registration of runtime behavior
  – Requires detailed knowledge of target language
Reinventing the wheel

- Multiple implementations for instrumentation
  - Too much effort
Non-flexible solutions

- Tight coupling between tool and environment
  - Alternate VM requires reimplementation
Proposition

• New abstraction layer
  – Based on behavioral reflection
Behavioral reflection

• Allows a program to modify (at runtime):
  – its own code
  – the semantics and implementation of its own programming language

• Complete dynamic analysis
  – Comprises more than just method executions
  – Need for a reflective meta representation that describes all behavioral aspects
Behavioral framework

- Additional abstraction layer

![Diagram showing Behavioral Reflection Framework, Tracer1, Tracer2, Bytecode modification, Virtual Machine, and Specialized Virtual Machine]
Requirements

• Runtime installation
• Unanticipated use
• Fine-grained selection
• Implementation hiding
• Performance
Final words

• Generic abstraction layer
  – Allows for portable tools
  – Relieves developers of lowlevel detail concerns

• Several requirements
  – Can these be realized?
Discussion

• Feasible?

• How does this abstraction layer compare to Aspect Oriented Programming?
Capturing How Objects Flow at Runtime

Adrian Lienhard, Stephane Ducasse, Tudor Girba and Oscar Nierstrasz
Motivation

• Dynamic techniques are generally based on trace views
  – Too low level of abstraction for OO systems

• Idea: capture object lifecycles
  – Take aliasing into account
  – Follow propagation of objects at runtime
  – Meta model
Aliases

• Created when an object is:
  – instantiated
  – stored in a field
  – stored in a local variable
  – passed as an argument
  – returned from a method execution
Relating static to dynamic information

• Serves two purposes
  – Check whether objects paths are as expected
  – Identification of important classes in the object flow
Characterizing object flows

• Purpose
  – Shows an object’s interaction with other objects during its lifecycle

Legend
- instance creation
- field alias
- argument alias
- return alias
Object-centric debugging

• Purpose
  – Support in finding causes and effects of errors
Conclusion

• Need for views on object referencing
  – Alias analysis yields promising results
  – Serves various purposes
Discussion

• Scalability
  – Performance overhead: factor 10
  – Implement aliases at a lower level in the VM
  – Room for improvement?
  – Scalable object visualizations?