Supporting Framework Use via Automatically Extracted Concept-Implementation Templates

Abbas Heydarnoori
Krzysztof Czarnecki
Thiago Tonelli Bartolomei

Generative Software Development Lab
University of Waterloo, Canada
Outline

- Introduction and Motivation
- The FUDA Framework Comprehension Technique
- Evaluations
- Concluding Remarks
Introduction

- **Object-oriented application frameworks** are widely used to develop new applications

- Frameworks provide **domain-specific concepts**
  - *Example*: JFace offers implementation for context menu and tree viewer

- Framework-based applications are constructed by writing **completion code** that instantiates these concepts
Main Difficulties of Frameworks

- Complex and difficult to learn APIs
- Lack of manuals and documentation
Proposed Solution

- Apply the *Monkey See/Monkey Do* Rule \cite{Gamma et al., 2004}

  - “Use existing framework applications as a guide to develop new applications”

- Code difficult to find due to scattering and tangling
Motivating Example
Motivating Example

- Instantiating Framework Classes
- Implementing Interfaces
- Sub-classing Framework Classes
- Calling Framework Methods
Related Work

- **Framework usage comprehension tools**, such as Strathcona [ICSE’05] and FrUiT [ETX’06]
  - Apply static analyses
  - Aim fine-grained API elements

- **Concept location tools**, such as SNIAFL [ICSE’04] and SITIR [ASE’07]
  - Unaware of a framework API
  - Results contain application-specific instructions
FUDA Framework
Comprehension Technique
FUDA Framework Comprehension Technique

- Automatically extracts *implementation templates* for framework-provided concepts

- *Concept Implementation Template*: A Java-like representation of the implementation steps that are necessary to instantiate a given concept
A Sample Template

Basic Steps

- Packages to Import
- Interfaces to Implement
- Methods to Implement
- Classes to Subclass
- Objects to Create
- Methods to Call

Additional Information

- Call Nesting
- Order of Calls
- Object Passing Patterns
- Statement Repetition Info.

```java
import org.eclipse.jface.action.Separator;
import org.eclipse.jface.viewers.Viewer;
import org.eclipse.jface.action.Action;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.action.IMenuListener;
import org.eclipse.swt.widgets.Control;

public class AppMenuListener implements IMenuListener {
    public void menuAboutToShow(menuManager) {
        Separator separator = new Separator(String)||(); //REPEAT
        menuManager.add(separator)||(appAction); //REPEAT
    }

    public class AppAction extends Action {
    }

    public void someMethod() {
        Viewer viewer = ...;
        Control control = viewer.getControl(); //MAY REPEAT
        AppAction appAction = new AppAction(); //MAY REPEAT
        appAction.setText(String); //MAY REPEAT
        appAction.setToolTipText(String); //MAY REPEAT
        MenuManager menuManager = new MenuManager(String)||(String,String)||(String||());
        menuManager.setRemoveAllWhenShown(boolean);
        AppMenuListener appMenuListener = new AppMenuListener();
        menuManager.addListener(appMenuListener);
        Menu menu = menuManager.createContextMenu(control);
    }
```
The FUDA Approach Overview

Manual Steps

Determining the Concept Definition Question

Selection of Sample Applications and Execution Scenarios

Trace Collection

Concept Implementation Template

Automated Template Generation
Trace Collection

- Traces only the calls at the framework boundary
  - The API trace consists of *API interaction events*
Direction of Events

Application

Framework

Incoming

Framework API

Outgoing

Framework-Stipulated Code

Outgoing

Somewhere in the Application

Somewhere in the Framework
A Sample API Trace for the Concept Context Menu

Events involved in the implementation of the context menu

The marked events when the context menu is invoked
API Trace Slicing

- Identifies relevant events before and after the marked region

  - Related events use common objects as targets, parameters, or return values
Sample Sliced API Trace

- An approximation of the actual dependencies among API calls
- Could have both false positives and false negatives
Event Generalization

- Allows comparing traces in terms of framework API types
- Replaces application-specific names with appropriate framework names
- A static analysis on the type hierarchy of the event’s target
Fact Extraction and Template Generation

- Extracting facts about the call occurrences, nesting, and dependency
- Determining common facts across traces
- Template generation from common facts
Evaluations

Template Extraction Evaluation
Template Usage Evaluation
Template Extraction Evaluation

- **Evaluation Hypothesis:** FUDA can extract templates with high precision and recall from only two traces and two sample applications
  - Aimed to keep the number of traces minimal
Selection of Frameworks and Concepts

- Four complex, widely-used frameworks:
  - Eclipse, JFace, GEF, Java2D

- Fourteen concepts:
  - Six based on prior knowledge
  - Eight from developer forums
  - Covered different characteristics: **scope**, **slicing**, **frequency**, **complexity**, and **atomicity**
Selection of Sample Applications

- Two applications per concept from different sources
  - Available at hand
  - Packaged with the desired framework
  - Online repositories
  - Suggested by others
Experiment Performance

- Prototyped FUDA

  - **FUDA Profiler**
    - Provides a GUI for collecting marked traces
    - Uses *AspectJ* to instrument applications

  - **FUDA Analyzer**
    - An Eclipse plug-in for generating templates out of traces
Calculation of Precision and Recall

- Requires a reference
  - Created based on prior experience, framework documentation, and/or actual implementation
Results

- **Precision**: 59% - 100%
  - Presented instructions being correct

- **Recall**: 79% - 100%
  - Required instructions being presented

- Application similarities caused false positives

- Slicing eliminated 18% - 80% of false positives
  - Mainly useful for similar applications
  - No effect for different applications
Template Usage Evaluation

- Evaluated the usage of templates by developers in the implementation of concepts

  - Asked developers to use either documentation or templates
Research Questions

- $Q_1$: Are templates as effective as documentation in aiding the developers?
  - If yes, they can serve as a substitute when no documentation is available

- $Q_2$: What is the influence of template quality and its usage strategies on the quality of resulting implementations?
Subjects

- Recruited twelve subjects
  - A mixture of students and professionals
  - Highly skilled Java programmers
  - Except one, all had industry experience
Task Assignment

- Assigned two tasks to each subject:
  - One simple, one complex
  - One using template, one using documentation

- Random and balanced over the concept complexity and documentation aid

- Constrained by prior knowledge of the concepts
Data Analysis

- Quantitatively analyzed via:
  - Statistical analyses of development times

- Qualitatively analyzed via:
  - Inspection and execution of resulting implementations
  - Careful examination of questionnaires and interviews
Quantitative Analysis Results

- The choice of documentation aid had little influence on the development time

  - Statistical analysis failed in providing evidence that templates and documentation are different (or equivalent) in providing aid

- However:
  - The observed differences due to documentation aid were small
  - The concept complexity had much greater impact on productivity
Qualitative Analysis Results

- Only two buggy implementations
  - One for each documentation aid

- All except one used templates together with sample applications
  - That subject had a buggy implementation
  - Use templates with sample applications
Concluding Remarks

Strengths and Weaknesses of the Approach
Conclusions
Strengths of the Approach

- Templates and documentation were similarly effective in the experiment
  - Not statistically significant; larger experiment needed
- Highly automated
- Needs only a few sample applications
- Traces only the API interactions
- Dynamically detects the actual API elements involved
Weaknesses of the Approach

- Results depend on sample applications
- Designing concept invoking scenarios not always obvious
- Setting up the runtime environment could be challenging
Thank You!

Questions?
Contact Information:

Generative Software Development Lab
University of Waterloo, Canada

gsd@gsd.uwaterloo.ca