Bibliography


[CHECKER GAME, 2013]


[GURGEL et al., 2010] GURGEL, A.; DANTAS, F.; GARCIA, A.. Um estudo de composições de padrões de projeto em caesarj. In: IV LATIN
AMERICAN WORKSHOP ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, Salvador, July 2010. SBC.


[NARAYANAN et al., 2006] SRINIVAS NARAYANAN, KAMAL GOVINDRAJ, B. T.-P. N.. Use of aop in j2ee application performance monitor-


[SANTANNA et al., 2003] SANT’ANNA, C.; GARCIA, A.; CHAVEZ, C.; LUCENA, C.; VON STAA, A. On the reuse and maintenance of


[WHITTLE et al., 2009]


A

Appendix A - Target Applications

This Appendix presents the three evolving software applications used in this thesis, which are MobileMedia (Section A.1), iBatis (Section A.2) and GameUp (Section A.3).

A.1

MobileMedia

It is a program family that provides support to manage (create, delete, visualize, play, send) different types of media (photo, music and video) on mobile devices. During the SPL development and evolution, the initial core architecture was systematically enriched with mandatory, optional and alternative features. Seven releases of the MobileMedia were analyzed. The core features are: create/delete media (photo, music or video), label media, and view/play media. Some varying features, amongst others, are: transfer photo via SMS, count and sort media, copy media and set favorites. We decided to re-implement with advanced mechanisms all the variabilities, whenever it made sense, those ones that are implemented with aspects in the original AspectJ implementation. Table A.1 summarizes the

<table>
<thead>
<tr>
<th>Table A.1: Mobile Media Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System Type - Size</strong></td>
</tr>
<tr>
<td>Programming Language</td>
</tr>
<tr>
<td>Number of versions</td>
</tr>
<tr>
<td>Number of selected versions</td>
</tr>
<tr>
<td>Avg. # of Modules</td>
</tr>
<tr>
<td>Avg. # of Program Elements</td>
</tr>
<tr>
<td>Avg. # of Compositions</td>
</tr>
</tbody>
</table>
A.2

iBatis

iBatis is a Java-based open source framework for data mapping and it uses two main APIs: (i) SQL Maps for reducing JDBC code; and (ii) Data Access Objects (DAO) for abstracting the persistence implementation details. It is composed of more than 60 releases developed incrementally and its development is characterized as a reactive approach. Initially, four releases were chosen and implemented using the AspectJ and CaesarJ language in its essential. The functionalities that were refactored were: concurrency, type mapping, design patterns, error context, exception handling, connection, session, and transaction.

Table A.2: iBatis Details

<table>
<thead>
<tr>
<th>System Type - Size</th>
<th>Framework - 110 KLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Language</td>
<td>Java (OO), AspectJ (AOP) and CaesarJ (FOP)</td>
</tr>
<tr>
<td>Number of versions</td>
<td>60</td>
</tr>
<tr>
<td>Number of selected versions</td>
<td>04</td>
</tr>
<tr>
<td>Avg. # of Modules</td>
<td>6010</td>
</tr>
<tr>
<td>Avg. # of Program Elements</td>
<td>18920</td>
</tr>
<tr>
<td>Avg. # of Compositions</td>
<td>22301</td>
</tr>
</tbody>
</table>

A.3

GameUP

GameUp is a SPL developed following the reactive approach. It encompasses three open-source board games where each of them is an SPL: Shogi, JHess and Checkers. Checkers is an American checker whereas Shogi and JHess are chess games. All of them provide features to manage various functionalities for customizing the board (e.g. indicating moveable pieces) and the matches between players (e.g. indicating player turns). New feature combinations make possible to generate several additional products, which could not be derived from the individual SPLs. In fact, the SPLs involve more than thirty five features that embrace several categories of features. The SPL variabilities provide an extensive list of integration scenarios.
### Table A.3: GameUP Details

<table>
<thead>
<tr>
<th>System Type - Size</th>
<th>Software Product Line - 3 KLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Language</td>
<td>Java (OO), AspectJ (AOP) and CaesarJ (FOP)</td>
</tr>
<tr>
<td>Number of versions</td>
<td>06</td>
</tr>
<tr>
<td>Number of selected versions</td>
<td>06</td>
</tr>
<tr>
<td>Avg. # of Modules</td>
<td>308</td>
</tr>
<tr>
<td>Avg. # of Program Elements</td>
<td>1963</td>
</tr>
<tr>
<td>Avg. # of Compositions</td>
<td>7250</td>
</tr>
</tbody>
</table>
Appendix B - Questionnaire of the Experiment

This appendix presents the questionnaire given to the participants involved in the experiment. Basically, the participants had to determine their level of knowledge and experience in developing and evolving software systems.

- **Date of birth:** dd — mm — aaaa
- **Gender:** () Male () Female

1. Which design(s) technologies are you familiar with?
   a. Aspect-oriented (AO) design
   b. Feature-oriented (FO) design
   c. None of the above
      If you selected option "c", which modelling languages have you already used?
      ________________________________________________________________
      ________________________________________________________________

2. Which programming language(s) are you familiar with?
   a. AspectJ
   b. AspectJ dialects such as Spring and Jboss AOP
   c. CaesarJ
   d. None of the above
      If you selected option "d", which programming languages have you already worked with?
      ________________________________________________________________
      ________________________________________________________________

3. If you are familiar with AO and FO designs, please tick which of the following activities you have performed.
Appendix B. Questionnaire of the Experiment

1. I took part in discussions about software design
2. I took part in discussions about software design and their evolution
3. I have designed software systems
4. I have designed and evolved software systems
5. I have analysed quality attributes (e.g. reusability and reusability) of software design

4. If you are familiar with design technologies (AOP and FOP), how long have you worked with them?
   a. 0 to 6 months
   b. 6 months to 1 year
   c. 1 to 3 years
   d. More than 3 years

5. If you are familiar with programming languages (Question 2), how long have you worked with them?
   a. 0 to 6 months
   b. 6 months to 1 year
   c. 1 to 3 years
   d. More than 3 years

6. How do you classify your level of experience in the following topics? It can be ”Advanced” (more than 3 years of experience), ”Intermediate” (1 to 3 years of experience), ”Basic” (6 months to 1 year) or ”No experience” (0 to 6 months)

6.1 Feature-Oriented Software Development
   a. Advanced
   b. Intermediate
   c. Basic
   d. No experience

6.2 Aspect-Oriented Software Development
   a. Advanced
   b. Intermediate
   c. Basic
   d. No experience

6.3 Software Evolution
   a. Advanced
   b. Intermediate
c. Basic
d. No experience

6.4 UML-based Design
a. Advanced
b. Intermediate
c. Basic
d. No experience

ADDITIONAL COMMENTS:

____________________________________________________________________________

____________________________________________________________________________
Appendix C - Experiment Details

This appendix presents the description of the program modification tasks of the experiment (Section C.1). All the tasks must be implemented with the AspectJ programming language. The participants had to answer all the tasks within 60 minutes. This experiment is based on elements of a family of board games, called GameUP (Section A). All these games provide functionalities for customizing the board (e.g. indicating moveable pieces) and the matches between players (e.g. indicating player turns). In addition, Section C.2 presents the feedback questionnaire provided by the experiment participants.

C.1 Description of Tasks

1. The GameUP developers must evolve the games with a new functionality, which consists of displaying a new screen before the match starts, where the player can specify whether the game will be played on the network or not. This new functionality must be implemented through a method named `setupGame()` using AspectJ mechanisms. The existing method `startGame()` is in charge of initializing the game. The level of detail expected in your answer is the same as illustrated in the aspect `BoardStartup`, `GameStatus`, `ControllerBase` and `NickNameDefinition`. 
Code C.1: Code of Task 1

```java
class MainFrame {
    int status;
    GameController controller;
    void startGame() {
        ...  
        Controller.setStatus(status);
    }
    void stopGame() { //stop the game }
}

aspect BoardStartup {
    pointcut setGame(): call(void *.startGame());
    void setBoardGame() {
        //make the board available to the match
    }
    before(): setGame(){ setBoardGame( );}
}

class GameController {
    int status;
    void setStatus(int x){ this.status = x; }
}

abstract aspect ControllerBase {
    pointcut setData(): call(* *.set());
}

aspect GameStatus extends ControllerBase{
    after(): setData(){ //Print a message }
}

aspect NickNameDefinition {
    pointcut setId(): call(void *.startGame());
    before(): setId(){ //display screen to specify nickname
    }
    ...
}

aspect ExecutionOrder {
    declare precedence: BoardStartup, NickNameDefinition
}
```
Figure C.1: UML Design for Task 1 - Class Diagram
Figure C.2: UML Design for Task 1 - Sequence Diagram
Figure C.3: UML+ Design for Task 1 - Class Diagram
Figure C.4: UML+ Design for Task 1 - Sequence Diagram
2. The player’s status is indicated by a colored button. When the button is green (status=0) it indicates that the player can play. If the button is red (status=1) the player cannot play due to some restrictions and another match must be started. Having said this, you are required to add a new functionality which aims to change the button color to yellow (status=2) when a player suffers a penalty and passes your next turn on to your opponent. This new functionality must be implemented in a separate aspect using an after returning advice.

Code C.2: Code of Task 2

```java
class Player {
    int status = 0;
    int id = 0;
    void setStatus(int x) {
        status = x;
    }
}

class GameController {
    Player p;
    void addPenalty(int code) {
        p.setStatus(code);
    }
    int checkMove() {
        // check if a move is valid
    }
}

class Score {
    ...
    void setScore(Player player, int score) {
        if (player.getStatus() == 0) {
            // update current score
        } else { // bring the score to zero
        }
    }
}

classMainFrame{
    GameController g;
    void initPlay() {
        g.checkMove();
    }
    ...
}
```
abstract aspect ControllerBase {
    pointcut setData(): call(* *.set());
}

aspect GameStatus extends ControllerBase{
    after(): setData(){ //Print a message }
}
Figure C.5: UML Design for Task 2 - Class Diagram
Figure C.6: UML Design for Task 2 - Sequence Diagram
Figure C.7: UML+ Design for Task 2 - Class Diagram
Figure C.8: UML+ Design for Task 2 - Sequence Diagram
3. The current version of Game UP allows both saving the board configuration in JPG format and informing the player about the result of such operation by message on the screen. The GameUP developers need your help to evolve the source code below in order to allow saving the game in XML format in addition to the JPG format. The level of detail expected in your answer is the same as illustrated in the code bellow.

Code C.3: Code of Task 3

```java
class GameController {
    void saveBoard () {
        // save board in JPG format
        ...
    }
    ...
}

aspect BoardTracing {
    pointcut traceSaveBoard() : call(void GameController.
        saveBoard( ))
    after () : traceSaveBoard () {
        System.out.println("An JPG board image was successfully saved");
    }
    ...
}

abstract aspect ControllerBase {
    pointcut setData(): call(* .set(*));
}

aspect GameStatus extends ControllerBase {
    after () : setData( ) { // Print a message }
}
```
Figure C.9: UML Design for Task 3 - Class Diagram
Figure C.10: UML Design for Task 3 - Sequence Diagram
Figure C.11: UML+ Design for Task 3 - Class Diagram
Figure C.12: UML+ Design for Task 3 - Sequence Diagram
4. Help GameUP developers to add the method `int saveScore()` to the class `PersistentData` using AspectJ’s mechanisms. This method aims at providing a new functionality, which is to save the player’s score at the end of each game match.

Code C.4: Code of Task 4

```java
class PersistentData {
    ...
    int saveJPG() { ... }
    int saveXML() { ... }
    ...
}

aspect BoardTracing{
    pointcut traceSaveBoard(): call(int PersistentData.save*( ));
    ...
    after(): traceSaveBoard() {
        System.out.println("Board Saved");
    }
    ...
}
```
Figure C.13: UML Design for Task 4 - Class Diagram
Figure C.14: UML Design for Task 4 - Sequence Diagram
Figure C.15: UML Design for Task 4 - Class Diagram
Figure C.16: UML+ Design for Task 4 - Sequence Diagram
C.2

Feedback Questionnaire

1. In your opinion, which were the characteristics present in the source code and also in the UML-based diagrams that make the changes more difficult to be performed? Why were these specific characteristics hindering the implementation of the required changes?

________________________________________________________________________

2. Please qualify to what extent the factors below exerted some influence on your execution of the changes required in the experiment. For each factor you must choose only one option considering the range from ”No Influence” to ”Extremely High”.

Factor 1: The scope of the pointcuts involved in the changes. Scope refers to the set of program elements (joinpoints) picked out by a given pointcut.

a. No Influence  
b. Extremely Low  
c. Low  
d. High  
e. Extremely High  

Factor 2: The occurrence of multiple aspects sharing the same joinpoint.

a. No Influence  
b. Extremely Low  
c. Low  
d. High  
e. Extremely High  

Factor 3: The existence of different types of modules, i.e. classes and aspects, involved in a pointcut.

a. No Influence  
b. Extremely Low  
c. Low  
d. High  
e. Extremely High  

Factor 4: The dependency among aspects and classes are highly based on the program language syntax.
Appendix C. AppendixC - Experiment Details

Factor 5: The occurrence of methods with similar signature pattern (e.g. names starting with the prefixes "set" and "get").

a. No Influence
b. Extremely Low
c. Low
d. High
e. Extremely High