Reflecting on Changeability through Kalah

Matthew Eden
Department of Electrical, Computer and Software Engineering
University of Auckland
Auckland, New Zealand
mede607@aucklanduni.ac.nz

***Abstract*—As part of an investigation into changeability, I was tasked with creating an implementation of Kalah that followed an object-oriented design and aimed to have high changeability. This design was then subjected to 2 change cases, one concerned with the game output and another concerned with replacing one player with a bot. Through these change cases I gained some understanding of changeability and what it means in practice.**

***Index Terms*—kalah, mancala, changeability, reusability, java, solid, object-oriented design**

I. Introduction

To investigate changeability and better understand the abstract ideas presented in the course *SOFTENG 701 - Advanced Software Engineering Development Methods*, I was tasked with creating an implemention of the traditional game *Kalah* in Java 7. The aims of this design was to have high changeability, where changeability is defined as “How much it costs to make the necessary changes to existing code (not new code) once those changes have been identified” [5]. The changeability of this design was then tested through two change cases, the first of which was concerned with the display of the game and the second with the actions of the player.

II. Designing for Changeability

Looking back on my original design, there are several aspects of it that do not hold up to scrutiny when considering changeability. However, this is to be expected as at that point in time I did not have a good formal understanding of changeability, as it has been presented in this course. Previously, if I had been asked about how one might design code such that it is changeable, the advice I would have given would be in regards to ensuring atomicity in methods and classes. I would give this advice with the mindset that a changeable design is one that is ‘pluggable’; new functionality can be added in without much change to the current design and elements of the design can be taken out and put into other designs. Although this advice does sound good at face value, and has done well for me thus far, it lacks formality. In contrast to the view of changeability presented in this course, I wouldn’t have made mention of a *changeability index*, or considered that a design is still changeable regardless of how much new code is written for a change case. I also would not have considered much of the advice presented in this course, such as design patterns or the SOLID principles.

Having now gotten to a point where I am aware of this advice, I can start to see some flaws in my original design. Particularly in regards to SOLID principles, such as the *Single Responsibility Principle* [8]. I have a class that violates this principle, `GameIO`. By the very nature of the class, it has two responsibilities: player input and game output. This was intentional at the time as I had been following a game design pattern I used in a previous project that used a similar context (the context being a local multiplayer command-line Java game). This decision did not support changeability from a design perspective, if we consider that violating one of the SOLID principles reduces changeability. On the other hand, if we consider the changeability index (CI) [5] for

III. Understanding Changeability through Change Cases

Although the changeability of a design can be evaluated through looking the design principles being followed, ultimately, whether a design is changeable can only be seen through the experience of attempts to change it. That is to say, change cases. The reason why this is the case is that changeability is concerned with how changeable a design is. Yet, if one never changes the design, to talk about the changeability of it appears futile. For my design, there were two change cases that I successfully performed and both of them provided insights into changeability.

The first change case was concerned with modifying game output [2] illustrated the changeability of my design. In that reflection, I concluded that my design had good changeability due to the ease with which I implemented that change case. I provided such arguments as “[it] resulted in minimal impact on the functionality of the implementation”, and it can been seen in the change plan that the majority of the changed code occured in a single method within a single class. If we then look at the changeability index from that change case, this also supports the idea that there was good changeability.

$$CI = (3/7) * 0.1 = 0.04286 \tag{1}$$

Values for CI that fall below `1.0` are considered indicators of ’good’ changeability, and values that exceed `1.0` are considered indicators of ’bad’ changeability. So the CI here tells us that for this change case there was ’good’ changeability.

The second change case was concerned with adding a bot in place of the second player, changing the game from multiplayer (with two human players) into singleplayer (a human player against a computer player) [3]. In implementing that change case, I ended up writing a large amount of new code due to being unable to easily modify the pre-existing code. I then used this to conclude that the changeability of my design could be improved, in spite of the fact that very little of the pre-existing code was modified. This conclusion *did not* follow the definition of changeability provided in this course. Again, if we consider the CI, we can see that the design has good changeability.

$$CI = (1/7) * 0.3 = 0.04286 \tag{2}$$

IV. Moving Forward with Changeability in Mind

References

[1] M. Eden, "Implementing Kalah with a Changeable Object-Oriented Design", in Canvas, SOFTENG 701, Assignment 3 Submission

[2] M. Eden, "Implementing the Change Board Orientation change case", in Canvas, SOFTENG 701, Assignment 4 Submission

[3] M. Eden, "Implementing the Best move or First robot change case", in Canvas, SOFTENG 701, Assignment 5 Submission

[4] B. Meyer, "Reusability: The Case for Object-Oriented Design", in IEEE Software, vol. 4, no. 2, pp. 50-64, March 1987, doi: 10.1109/MS.1987.230097

[5] E. Tempero "SOFTENG 701 - Lecture 01b - Changeability", in Canvas, SOFTENG 701, May 2020

[6] E. Tempero "SOFTENG 701 - Lecture 03 - How to improve design", in Canvas, SOFTENG 701, May 2020

[7] E. Tempero "SOFTENG 701 - Lecture 04 - Learning to Measure", in Canvas, SOFTENG 701, May 2020

[8] E. Tempero "SOFTENG 701 - Lecture 07 - SOLID Design Prinicples", in Canvas, SOFTENG 701, May 2020

[9] E. Tempero "SOFTENG 701 - Lecture 09 - Counting Objects to evaluate object-oriented design", in Canvas, SOFTENG 701, June 2020