Analisis Co-Changes Pada Implementasi Bridge Design Pattern Pada Aplikasi Mobile

Ganesha Danu Enastika[1], Gede Agung Ary Wisudiawan[2], Shinta Yulia Puspitasari[3]

[1,2,3]Fakultas Informatika, Universitas Telkom, Bandung
[1]ganeshadanu@student.telkomuniversity.ac.id, [2] degunk@telkomuniversity.ac.id,
[3]shintayulia@telkomuniversity.ac.id

Abstrak

Pada pengembangan perangkat lunak secara iterative dan incremental, terjadi perubahan pada komponen-komponen berbeda dalam codebase pada setiap iterasinya. Sehingga dibutuhkan sistem yang modular untuk mengakomodir perubahan-perubahan tersebut dengan mudah. Dalam usaha untuk meningkatkan modularitas, pengembang perangkat lunak umumnya mengunakan design pattern dalam *refactoring* sebagai salah satu solusi. Namun dalam penerapan design pattern, seringkali *trial and error* digunakan untuk menentukan komponen yang akan di-*refactor*. Salah satu cara untuk menentukan komponen yang dapat di-*refactor* adalah dengan memanfaatkan *co-changes* untuk mengetahui komponen-komponen yang dikategorikan sebagai *evolutionary smell*. Namun saat ini, metode tersebut hanya diterapkan dengan menggunakan refactoring yang sederhana. Penelitian ini berfokus untuk mencari tahu dampak implementasi bridge design pattern dengan memanfaatkan co-changes. Penelitian dilakukan dengan membangun *fine-grained co-changes clusters* dan menentukan komponen-komponen yang dikategorikan sebagai *evolutionary smell* dari *clusters* yang dibangun. Komponen-komponen tersebut kemudian di-*refactor* menggunakan *bridge design pattern*. Dengan membandingkan *object oriented metrics* sebelum dan sesudah *refactoring* untuk mengetahui pengaruh dari *refactor* yang dilakukan. Analisis *object oriented metrics* menunjukkan bahwa terdapat perbaikan pada *cohesion* (LCOM) sebesar 1 hingga 246, dan penurunan kompleksitas (WCM) sebesar 1 hingga 34. Namun, implementasi bridge design pattern dapat menurunkan atau justru meningkatkan tingkat coupling (CBO) sebanyak 1 hingga 2 bergantung dari apakah masih terdapat *dependency* lain diluar dari *class* pada *struktur bridge design pattern* yang diterapkan. Selain itu, Penerapan *bridge design pattern* dapat menyebabkan potensi *co-changes dependency*. Penelitian ini juga menemukan bahwa mayoritas dari komponen yang dikategorikan sebagai *evolutionary smell* memiliki fungsionalitas sebagai sumber data.

Kata Kunci: co-changes, design pattern, refactoring

Abstract

On iterative and incremental software development, changes are happening on various components within the codebase on every iteration. Thus, a modular system is needed to accommodate those changes easily. To improve modularity, software developers often use design patterns when refactoring as a solution. But, when implementing design patterns, more often than not trial and error were used as a means to decide which component to be refactored. The other way that could be used to decide which component to be refactored is by leveraging co-changes to find out which component could be categorized as evolutionary smell. But, as of now, this method is to be used to implement simple refactoring. This experiment focuses to find out the impact of implementing a design pattern by leveraging co-changes. This experiment was done by building fine-grained co-changes clusters and determining which component within the clusters to be categorized as evolutionary smell, then to be refactored by implementing a design pattern. By comparing object-oriented metric before and after the refactoring to find out how the refactor affect the system. The analysis done on the object-oriented metrics found out that implementation of bridge design pattern improves cohesion (LCOM) by 1 up to 246, and reduces complexity (WCM) by 1 up to 34. However, implementation of bridge design pattern may reduce or even increase coupling (CBO) by 1 up to 2 of the system depending on whether or not there are dependencies of classes out of the bridge design pattern structure. This experiment also found out that most of the components categorized as the evolutionary smell are the component responsible to retrieve data from a data source.

Keywords: co-changes, design pattern, refactoring