Faults and Their Relationship to Implemented Patterns, Coupling and Cohesion in Commercial C# Software

Matt Gatrell, Brunel University, UK
Steve Counsell, Brunel University, UK

ABSTRACT

This paper documents a study of fault proneness in commercial, proprietary software and attempts to determine whether a relationship exists between class faults and the design context of a class, namely the coupling and cohesion of a class, and whether the class is a participant of common design patterns. The authors studied a commercial software system for a 24 month period and identified design pattern participants by inspecting the design documentation and source code; coupling and cohesion metrics were measured by inspecting the source code with a tool; we also extracted fault data for the same period to determine whether a relationship existed between the design context and the fault propensity of a class. Results showed that design pattern participant classes were marginally more fault-prone than non-participant classes. The Adaptor, Method and Singleton patterns were found to be the most fault-prone of thirteen patterns explored. Coupling was found to have a significant relationship with the fault proneness of classes in the system; efferent coupling was a stronger indicator of fault propensity than afferent coupling. Cohesion, when measured using the LCOM(HS) metric, was not found to have a strong relationship with fault proneness.

Keywords: C#, Cohesion, Coupling, Fault, OO, Pattern

1. INTRODUCTION

Identifying fault prone areas within a software system over time can help identify problem areas and can be instrumental in predicting future maintenance requirements and prioritisation of resources. Being able to identify potential problem areas from the design context of the software allows avoidance at design time, and detection through automated tools during the whole lifecycle. In this paper we attempt to determine whether a relationship exists between class fault proneness and the design context of a class, specifically, the coupling and cohesion of a class, and whether the class is a participant of one or more common design patterns. We examined a subset of a large commercial system, consisting of 266 thousand lines of C# code,
7439 classes and 79964 methods. We collected fault data, coupling and cohesion metrics and identified design pattern participants over a 24 month period.

Coupling measures the number of dependants and dependencies of a class. Classes with lower coupling are said to be preferred as they tend to be easier to reuse and are less prone to change resulting from changes rippling through a dependency chain. As such, a class with low coupling may be expected to be less fault prone than a class with high coupling. Coupling measures how strongly related the functionality within a class is (Parnas, 1972); for example, a cohesive class may have a number of methods that perform a set of related operations on a common data entity; an uncohesive class may have a set of unrelated methods each offering an operation on a data entity distinct from the data entities used by the other methods in the class. Classes that are more cohesive tend to be more reusable and comprehensible. As such, a cohesive class may be expected to be less fault prone than a less cohesive class. The coupling and cohesion of a class are often considered together as a pair of measurements.

Object-oriented design patterns are reusable descriptions or templates showing the relationships and interactions between classes and objects (Bishop, 2008; Buschmann, Meunier, Rohnert, Sommerlad, & Stal, 1996 Grand, 2002). Many design patterns including some of those described by Gamma, Helm, Johnson, and Vlissides (1995) promote adaptability by supporting specialization of the pattern-based classes. A system built using design patterns can therefore be adapted by creating concrete classes with desired new functionality rather than by direct modification of the existing set of core pattern-based classes. It is therefore reasonable to expect that design pattern ‘participant’ classes (i.e., those core classes) would have a relatively lower propensity for change relative to all other classes in a system since, in theory, they should remain untouched by developers. A previous study by Bieman, Jain, and Yang (2001) and then subsequently replicated by Gatrell, Counsell, and Hall (2009) have both shown (for the respective systems studied) that the opposite was true; design pattern participants were actually more change-prone than non-pattern classes. Discussion with developers in the latter revealed that over-familiarity with a core set of design patterns was the reason why developers changed classes rather than used specialisation or other adaptations. A key strength of design patterns seems to be the cause of their subversion. In this paper, we extend those two studies to explore whether design pattern participants were more fault-prone than non-pattern classes. We manually inspected the system (design and code) to identify intentional design patterns, using the same methodology as Bieman’s previous study (Bieman, Jain, & Yang, 2001) and extracted fault data from the system’s source control system.

The remainder of the paper is organised as follows. In the next section, we describe the motivation for the study and related work. In Section 3, we describe preliminaries such as the system studied and the metrics extracted. We then analyse the data (Section 4) exploring three facets of the data: patterns, coupling and cohesion, before discussing the threats to the validity of the study (Section 5). Finally, we conclude and point to further work in Section 6.

2. MOTIVATION AND RELATED WORK

The motivation for this research arises from several sources. First, design patterns are commonly used in both open-source and proprietary software and are seen as an exemplar of sound software engineering practice; knowledge of their change trends and fault-proneness is valuable information for the developer and could also inform future maintenance activity, resource prioritization, and QA planning. While we agree that in theory use of design patterns should be condoned, we see the study presented as an exploration of actual design pattern usage and evolution in an industrial setting (Kemerer & Slaughter, 1999); the purpose of any empirical software study is to support (or refute) hypoth-
Related Content

Using Description Logics for the Provision of Context-Driven Content Adaptation Services
www.igi-global.com/article/using-description-logics-provision-context/39101?camid=4v1a

Cyber Physical Control Systems
www.igi-global.com/chapter/cyber-physical-control-systems/121260?camid=4v1a
Advancing Cyber Resilience Analysis with Performance-Based Metrics from Infrastructure Assessments
www.igi-global.com/article/advancing-cyber-resilience-analysis-performance/76356?camid=4v1a

BROOD: Business Rules-Driven Object Oriented Design
www.igi-global.com/chapter/brood-business-rules-driven-object/29434?camid=4v1a