2015 12th Working Conference on Mining Software Repositories

Modifications, Tweaks, and Bug Fixes in Architectural Tactics

Mehdi Mirakhorli
Software Engineering Department
Rochester Institute of Technology
Rochester, NY, USA
mehdi@se.rit.edu

Jane Cleland-Huang
School of Computing
DePaul University
Chicago, IL. USA
jhuang@cs.depaul.edu

***Abstract*—Architectural qualities such as reliability, performance, and security, are often realized in a software system through the adoption of tactical design decisions such as the decision to use redundant processes, a heartbeat monitor, or a specific authentication mechanism. Such decisions are critical for delivering a system that meets its quality requirements. Despite the stability of high-level decisions, our analysis has shown that tactic-related classes tend to be modified more frequently than other classes and are therefore stronger predictors of change than traditional Object-Oriented coupling and cohesion metrics. In this paper we present the results from this initial study, including an analysis of why tactic-related classes are changed, and a discussion of the implications of these findings for maintaining architectural quality over the lifetime of a software system.**

***Keywords*-Architectural decisions, tactics, modifications, bugs, metrics**

I. Introduction

The architectural design of high-performance and safety-critical systems is driven by quality concerns such as reliability, dependability, security, and performance. Such concerns are often addressed through a series of design decisions, many of which involve implementing and deploying standard *architectural tactics* [2] such as the heartbeat tactic to monitor the availability of a critical component, or redundant processes with a voting coordinator to help prevent individual errors from causing the system to fail. Architectural tactics are therefore a critical element of the design and must be carefully implemented and maintained in the deployed system.

Unfortunately, numerous studies have shown that architectural quality tends to erode as a result of ongoing modifications and maintenance efforts [7]. Several researchers such as Kruchten et.al [5], have therefore highlighted the importance of documenting design decisions and their rationales so that architectural knowledge can be preserved and quality can be maintained over the lifetime of the system.

In our prior work [7], we conducted a study of architectural tactics in 20 performance-centric projects including Google Chrome and Hadoop. Our study showed that tactics such as heartbeat, resource pooling, and redundancy were found across many of the systems. These results are reported in Figure 1. We also informally observed that classes which implemented architectural tactics tended to be modified more frequently than other classes. In this paper we report on an empirical study we conducted to evaluate this phenomenon. The results of our study are quite surprising, and show, not only that tactic-related classes change more frequently than their counterparts, but also that the presence of a tactic in a class serves as a stronger predictor of change than more traditional Object-Oriented (OO) coupling and cohesion metrics [3].

II. Tactic-Related Change

Two projects were selected for an in-depth analysis of tactic-related change according to the following criteria: (i) they represented two different application domains of performance-centric and business-centric applications, (ii) they had readily available change logs which differentiated between bugs and other types of modifications, and (iii) their architectural design documents were accessible. The first project was *Hadoop*, which is a framework for reliable, distributed computation. It was developed using tactics such as Heartbeat, Scheduling, Thread Pooling, Authentication, Authorization(Kerbrose), and Audit Trail. The second project was *ofBiz* which is an Enterprise resource planning application implemented using architectural tactics such as scheduling, Queueing, wrapper, thread pool, and session pool. For each of these projects we extracted change logs: i.e. a list of Java classes modified as part of a release (excluding bug fixes), and bug fixes: a list of Jave classes modified in order to fix a bug. For ofBiz, these data represented changes from January 2009 to November 2011, while for Hadoop they represented changes from November 2008 to November 2011.

A. Stability of Tactical Decisions

It is widely understood that architectural decisions are relatively stable in a design, meaning that Software Architects infrequently eliminate or replace architectural decisions between releases. This observation was supported by our study, as we found no evidence of architectural tactics being replaced by other tactics. Nevertheless our study showed that despite this stability, tactic-related classes were modified quite frequently.

978-0-7695-5594-2/15 $31.00 © 2015 IEEE
DOI 10.1109/MSR.2015.44

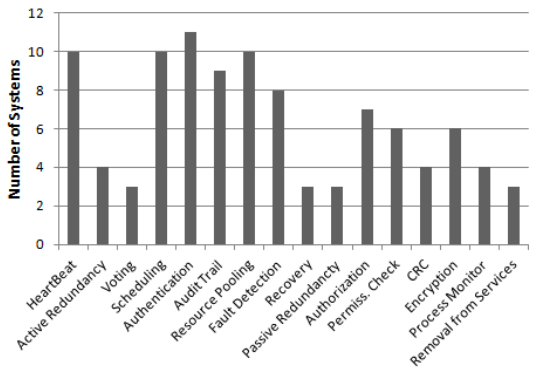


Figure 1. Analysis of Tactics Across Open-Source Projects

To analyze whether the modification rate of tactic-related classes was higher than non-tactic ones we computed the number of changes per class and then compared the medians of both groups using the non-parametric Wilcoxon rank test, since change counts have right-skewed distribution and standard two-sample t-test procedure can only be used for symmetric distributions. The test showed that the difference between median change counts is statistically significant (p-value< 0.001), providing evidence that tactic related classes changed more frequently than non-tactic related ones. For instance, average changes in ofBiz system for tactic-related classes was 5.81, compared to an average change of 2.8 for non tactic classes. Similarly for Hadoop, average change count in tactic-related classes was 4.57, while for other classes it was 1.65.

Figures 2(a) and 2(b) plot the change history against size and complexity for ofBiz and Hadoop. The X-axis shows the size of a class and Y-axis shows the complexity measured by Weighted Methods per Class (WMC) [3]. Unsurprisingly, the results show a strong positive correlation between the size of the class (measured by Lines of Code) and the complexity measured in terms of WMC values. The bubble chart depicts tactic-related classes in pale gray, and unrelated classes in dark gray. The number of changes (modifications or bug fixes) made to each individual class are directly proportional to the size of each circle (class), such that classes which change more frequently are larger than those that change less frequently.

For both ofBIZ and Hadoop, the graph visually confirms our earlier results that tactic-related classes change more frequently than unrelated ones. It is interesting to note that all of the larger classes, and many of the classes with higher WMC values, are tactic-related. A more surprising observation is that the majority of small and less complex classes which change more frequently are tactic-related classes rather than non-tactical ones.

B. Software Defects in Tactical Classes

We performed a similar analysis of bug density for tactical vs. non-tactical classes by computing the number of bugs per class. The non-parametric Wilcoxon rank test indicated that tactic related classes had significantly more bugs than non tactic-related ones (p-value< 0.001). For example, the average number of bugs in tactic-related classes for ofBiz was 2.71 compared to an average of 0.94 bugs for non tactic-related classes. For Hadoop, the average number of bugs was 3.24 for tactic classes and 0.72 for non-tactic ones.

The bubble charts shown in Figures 2(c) and 2(d) confirm well-established findings that as the complexity of a class increases, the number of bugs also tends to increase [1] [4]. In these charts, the size of the bubbles represent the number of bugs. The graphs visually confirm our earlier findings that classes with the most bugs (i.e. the larger circles) tend to be tactic-related, regardless of size and complexity of the class.

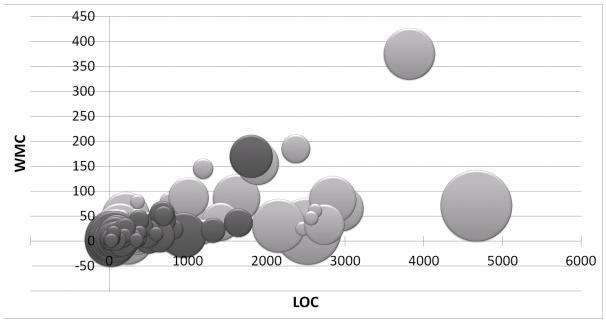
These results show quite clearly that in the case of ofBIZ and Hadoop, the tactic-related factor might serve as a better bug predictor than more traditional complexity metrics such as WMC. We explore this conjecture in the next section.

C. Tactics vs. OO Metrics

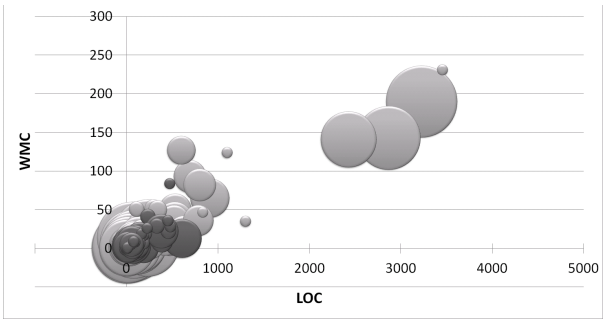
We also investigated the association between several Chidamber and Kemerer's OO metrics [3] and bugs, versus the association between the tactic-factor and bugs. *WMC (Weighted Methods per Class)* measures the complexity of an individual class by summing the weighted methods. *DIT (Depth of Inheritance Tree)* computes the number of ancestors of a class. *RFC (Response For a Class)* computes the number of methods which can be directly or indirectly executed in response to a message to an object of that class. *NOC (Number Of Children)* is the number of direct descendants for each class. *CBO (Coupling Between Object classes)* shows the number of classes to which a given class is coupled. *LCOM (Lack of Cohesion on Methods)* measures dissimilarity of methods in a class.

For purposes of our study, we computed all of the metrics using *"Understand"* source code analyzer. A statistical analysis using Poisson log-linear models was conducted to evaluate the impact of C & K metrics and tactic class type on change and bug counts. Tables 3(a), 3(b), 3(c) and 3(d) show the estimated parameters of the four Poisson log-linear models describing Hadoop and ofBiz bug counts and change counts using C&K metrics and tactic-related classes.

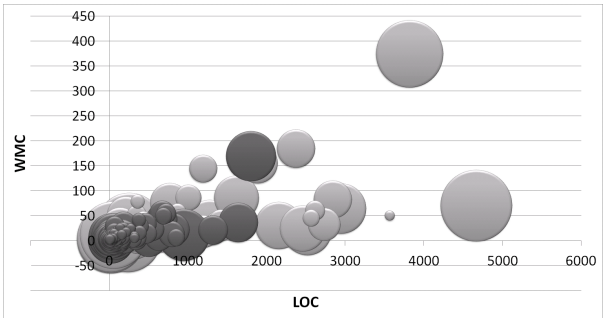
Results showed that for the Hadoop System, all the C&K metrics had a significant association (p-values < 0.05) with the number of bugs in the system, while for ofBiz only CBO and WMC were considered significant indicators (p-values < 0.001). For both systems, the tactic-factor had a significant effect (p-values < 0.001) on the number of bugs in classes. The model parameters provided a measure (on the log scale) of the effect of each factor on the variable of interest. For the Hadoop system, for example, the parameter value of 1.299 for the tactic factor indicated that on average tactic related classes were likely to have exp(1.299) = 3.667 times more bugs than non-tactic related classes. While an increase of one unit in either CBO or RFC



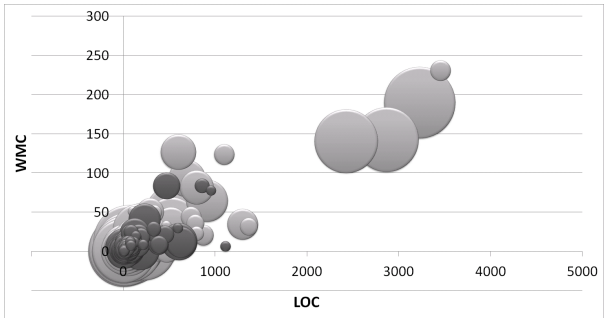
(a) Changes in ofBiz



(b) Changes in Hadoop



(c) Bugs in ofBiz



(d) Bugs in Hadoop

Figure 2. Changes and Bugs in Tactic-Related Classes vs. Non-Tactical classes

metrics corresponded on average to a 1% increase in the number of bugs. All other metrics (i.e. NOC, LCMO, DIT, and WMC) showed a slightly negative association. Similar results were found for the ofBiz system where tactic related classes were likely to have 2.54 times more bugs than non-tactic related classes. One important observation from all four models displayed in Figure 3 is that the tactic factor had a far stronger impact on both bug and change counts than any of the C &K metrics as shown by the extremely large Wald Chi-square statistic value for the tactic factor. In fact, some C & K metrics showed weak to no association with either bug or change counts.

III. Analysis of Change

In order to interpret these findings we analyzed a selection of bugs and modifications in the Hadoop System. Out of all the modifications made to Hadoop classes, 33% were bug related and 67% were for non bug-related changes; however we analyzed 50 randomly selected modifications and 50 randomly selected bug fixes.

A. Change Categories in Tactic Related Classes

As a result of our analysis, we identified five primary categories of non bug-related changes. Each of these is explained below along with the percentage of modifications falling under each category.

Tweaking (29%): Architectural tactics are hierarchical in nature meaning that higher level decisions are often implemented through a combination of lower level ones [6]. These lower-level decisions are used to tune the effectiveness of a tactic in satisfying the qualities. Changes in this category tweak the tactic to enhance the software quality. For example, the heartbeat tactic was fine-tuned by changing the health-checking interval. This had a direct effect on performance and reliability of the Hadoop system.

Extensions (20%): Certain changes involve adding new features to the tactic, or extending the tactic into other parts of the system. Although architects start by making high-level tactical decisions, details are often identified during implementation. For example, the authentication tactic was extended to work over RPC communication and also to log authentication results.

Content Refactoring (25%): The tactic is not extended nor tweaked to increase the quality level, but is refactored to enhance the code structure and comprehension. In some cases, the tactic related class is refactored to reflect changes related to a non-tactical part of the class.

Structure Refactoring (21%): Many times different changes to tactic related classes resulted in refactoring the structure of the system. This includes, adding or removing a class, or moving a Java file from one sub-system to another sub-system. Such refactorings suggest that some tactical classes use inflexible structures.

Clarification (5%): The program is modified simply to clarify the implemented code. For example, comments are added, or the meta data of the program such as the authors' names are modified.

Figure 4(a) reports the break down of each of these change categories, and shows that 53% of the changes were directly

Parameter	B	Std. Error	Hypothesis Test		Exp(B)
			Wald Chi-Square	p-value	
(Intercept)	.169	.0836	4.100	.043	1.184
Tactic	1.299	.0616	444.520	.000	3.667
CBO	.010	.0019	27.514	.000	1.010
NOC	-.100	.0296	11.365	.001	.905
RFC	.007	.0010	40.670	.000	1.007
LCMO	-.002	.0008	4.010	.045	.998
DIT	-.395	.0403	96.131	.000	.674
WMC	-.004	.0017	5.627	.018	.996

(a) Bugs-Hadoop

Parameter	B	Std. Error	Hypothesis Test		Exp(B)
			Wald Chi-Square	p-value	
(Intercept)	.721	.0568	161.363	.043	1.184
Tactic	.861	.0430	400.285	.000	3.667
CBO	.007	.0011	43.780	.000	1.010
RFC	.004	.0008	26.051	.001	.905
DIT	-.235	.0277	72.104	.000	1.007

(b) Changes-Hadoop

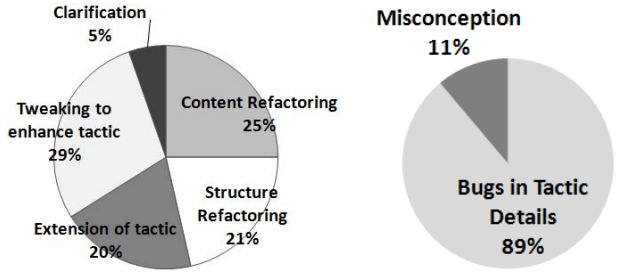
Parameter	B	Std. Error	Hypothesis Test		Exp(B)
			Wald Chi-Square	p-value	
(Intercept)	-.230	.0607	14.367	.000	.794
Tactic	.932	.0659	199.913	.000	2.541
CBO	.010	.0019	28.655	.000	1.010
WMC	.004	.0007	36.719	.000	1.004

(c) Bugs-ofBiz

Parameter	B	Std. Error	Hypothesis Test		Exp(B)
			Wald Chi-Square	p-value	
(Intercept)	.923	.0655	198.516	.000	2.516
Tactic	.545	.0404	181.725	.000	1.725
CBO	.012	.0013	73.680	.000	1.012
WMC	.004	.0004	107.407	.000	1.004
NOC	-.023	.0101	4.936	.026	.978
LCMO	.002	.0006	11.099	.001	1.002
DIT	-.102	.0254	16.121	.000	.903

(d) Changes-ofBiz

Figure 3. Changes and Bugs in Tactic-Related Classes vs. Non-Tactical classes



(a) Type of Changes (b) Cause of Bugs

Figure 4. Changes & Bugs in Tactic-Related vs. Non-Tactical classes

related to modifying the tactic (i.e. tweaks and extensions), while the remaining 47% were more general changes.

B. Bug Categories in Tactics

Our analysis of bug fixes resulted in the identification of two primary categories. The percentage of bug fixes for each category are shown below.

Bugs in Tactic Details (89%): These bugs occurred due to problems in the implementation of a tactic, for example, an unhandled exception, type mismatch, or missing values in a configuration file.

Wrong Implementation (11%): These bugs involved misconceptions in the use of the tactic, so that the tactic failed to adequately accomplish its architectural task. These kinds of bugs caused the system to crash under certain circumstances. For example, in one case a replication decision with a complex synchronization mechanism was misunderstood for different types of replica failure.

IV. Implications and Conclusions

Our study of Hadoop and ofBIZ has suggested that tactic-related classes have a strong tendency to exhibit faults and/or to undergo refactorings, tweakings, and other kinds of modifications; and that in many cases, they result in a larger and more complex implementation than non-tactic classes. As the quality of the entire architecture is significantly dependent upon architectural tactics, it makes sense to invest more effort to design, implement, and test, tactic-related classes. There are numerous ways to implement tactics, some of which are more flexible to change, less complex and easy to maintain. For example we observed the *Heartbeat* tactic implemented across various high-performance systems using the observer pattern, the decorator pattern, and also numerous ad-hoc solutions. If developers understood the high volatility of tactic-related classes they might invest more effort to build them more robustly.

The study presented in this paper represents only an initial investigation into the relationship between architectural significant classes, such as tactics, and long-term change. It does, however, highlight an important relationship between architecturally significant classes and change, and therefore sheds some light on why architectural solutions tend to erode over time. Future work we will investigate this further.

V. Acknowledgments

The work described in this paper was partially funded by NSF grant CCF-0810924.

References

[1] V. Basili, L. Briand, and W. Melo. A validation of object-oriented design metrics as quality indicators. *Software Engineering, IEEE Transactions on*, 22(10):751 –761, oct 1996.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Adison Wesley, 2003.

[3] S. Chidamber and C. Kemerer. A metrics suite for object oriented design. *IEEE Trans. on Software Eng.*, 20:476–493, 1994.

[4] T. Gyimóthy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Trans. Software Eng*, 31(10):897–910, 2005.

[5] P. Kruchten, R. Capilla, and J. C. Dueas. The decision view's role in software architecture practice. *IEEE Software*, 26(2):36–42, 2009.

[6] M. Mirakhorli and J. Cleland-Huang. Tracing architectural concerns in high assurance systems (NIER track). In *International Conf. on Software Engineering, New Ideas and Emerging Results Track, ICSE*, 2011.

[7] M. Mirakhorli, Y. Shin, J. Cleland-Huang, and M. Cinar. A tactic-centric approach for automating traceability of quality concerns. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 639–649, Piscataway, NJ, USA, 2012. IEEE Press.