A Quantitative Study of the Adoption of Design Patterns by Open Source Software Developers

Michael Hahsler
Department of Information Business
Vienna University of Economics and Business Administration

Phone: +43 1 31336-6081
Fax: +43 1 31336-739
Email: hahsler@ai.wu-wien.ac.at

Preprint:
Chapter in S. Koch (Ed.): Free/Open Source Software Development, Idea Group, Inc.
To appear: 2004

A Quantitative Study of the Adoption of Design Patterns by Open Source Software Developers

Abstract – Several successful projects (Linux, Free-BSD, BIND, Apache, etc.) showed that the collaborative and self-organizing process of developing open source software produces reliable, high quality software. Without doubt, the open source software development process differs in many ways from the traditional development process in a commercial environment. An interesting research question is how these differences influence the adoption of traditional software engineering practices.

In this chapter we investigate how design patterns, a widely accepted software engineering practice, are adopted by open source developers for documenting changes. We analyze the development process of almost 1,000 open source software projects using version control information and explore differences in pattern adoption using characteristics of projects and developers. By analyzing these differences we provide evidence that design patterns are an important practice in open source projects and that there exist significant differences between developers who use design patterns and who do not.

Keywords - Software Development, Design Patterns, Open Source

INTRODUCTION

The growing need for reliable software has made software engineering an important industry in the last few decades. The steady progress produced an enormous number of different approaches, concepts, and techniques: structured analysis and design, the object oriented paradigm, agile software development, component based systems, frameworks, and design patterns, just to mention a few. These techniques where developed with traditional software development in a commercial environment in mind. Recently, a new organizational form of collaborative software development, the open source movement (Raymond, 1999), gained popularity. Open source software (OSS) development differs from traditional forms in many respects. For example, the source code is publicly shared and therefore rigorously peer reviewed, the development teams are often geographically dispersed and instead of extensive unit tests massive system-level testing by large user communities is conducted (Perpich et al., 1997; Vixie, 1999; Jorgensen, 2001; Dempsey et al., 2002). Quantitative research is the key to understand how these differences influence the adoption of existing software engineering practices or how software engineering practices are adapted to the needs of open source development.

In this chapter we study how design patterns are used in open source software development teams. We are interested in the question if design patterns are useful for open source development and if so, are there factors that influence their adoption. To gain an insight into the application of design patterns we analyze historic data of the development process of OSS projects.

This chapter is organized as follows: As the starting point for the chapter we review literature about design patterns to identify how patters are used for traditional software development. Next, we describe the research method employed by the study. We present the used data set and its main characteristics followed by the analysis of the data set and the discussion of the results. We conclude the chapter with the main findings and point out directions for further research.

BACKGROUND AND RELATED LITERATURE

Design patterns describe non-obvious solutions in a standard written form for recurring software design problems in a certain context. They are normally developed by experts from their experiences with many existing systems and represent good and flexible solutions.

Since the introduction of the first software pattern catalog containing 23 design patterns by Gamma, Helm, Johnson, and Vlissides (1995), design patterns were rapidly accepted by the software engineering community and their use is now strongly facilitated by the Unified Modeling Language (UML), the standardized notation for object oriented analysis and design. The number of publications about design patterns has soared, and even several conference series on the topic were initiated. In the US the conference series has the name Pattern Languages of Programs (PLoP) and in other parts of the world conference series like EuroPLoP, KoalaPLoP, ChiliPLoP were started. These conferences, as well as most publications, focus on the development of new and improved design patterns, but the research on the actual adoption of design patterns by commercial and especially by open source software developers is still underdeveloped.

In their book about design patterns Gamma et al. (1995) state that they expect design patterns will provide (a) a common vocabulary, (b) a documentation and learning aid, (c) an adjunct to existing methods, and (d) a target for refactoring. To underpin these expectations there were some early publications by practitioners that describe experiences with design patterns in an industrial setting as well as for training (Helm, 1995; Beck et al., 1996; Goldfedder & Rising, 1996). However, these publications represent personal experience reports with no empirical data to support their claims. In the joint paper "Industrial Experience with Design Patterns" (Beck et al., 1996) co-authored by Kent Beck (First Class Software), James O. Coplien (AT&T), Ron Crocker (Motorola Inc.), Lutz Dominick and Frances Paulitsch (Siemens AG), Gerard Meszaros (Bell Northern Research) and John Vlissides (IBM Research) these 7 experts describe the efforts they and their companies put into design patterns and the resulting experiences. The paper contains a table of the most important observations sorted by the number of experts who mentioned them. This can be interpreted as the results of interviewing experts. The following observations were mentioned by all experts (Beck et al., 1996):

1. Patterns are a good communications medium.
2. Patterns are extracted from working designs.
3. Patterns capture design essentials.

The first observation is the most prominent benefit of design patterns. In Gamma et al. (1995) two of the expected benefits are that design patterns provide "a common design vocabulary" and "a documentation and learning aid" which also focus on the communication process. Prechelt et al. (2002) studied this aspect with two controlled experiments using undergraduate and graduate students to perform maintenance tasks for small programs. The experiments showed that explicit documentation of a used pattern has a positive influence on the speed and the error rate of the maintenance task.

Efficient communication is certainly also a key issue for OSS development. Often, there is no explicit design document for new OSS projects and the design emerges later on from the implementation (Vixie, 1999). This means that collaborating developers need to communicate the design in part directly by code. But since to infer design from code artifacts is known to be hard, this would make collaboration almost impossible unless there are some widely accepted design practices that are known within the software engineering community. Design patterns try to capture such design practices and give them a unique name to communicate them efficiently even as short comments within the code.

Seen et al. (2000) look at the adoption of design patterns in a commercial environment from a perspective of the diffusion of innovation theory. Although they rate the overall adoption rate for design patterns as moderate they identify an area where design patterns score very high. The area is concerned with properties which influence the individual motivation for adoption which are especially interesting for open source developers. Specifically, patterns require no infrastructure investment, they can be adopted bottom-up and visible pattern adoption advertises competence. All three properties are certainly more important in an open source environment than in a traditional company where the necessary infrastructure is provided and the management controls the development process.

RESEARCH METHOD

To investigate the adoption of design patterns by open source software developers we analyze the development process of OSS projects by using publicly available version control data accessible via the Source Forge Web site (http://www.sourceforge.net). This approach is inexpensive and non-intrusive (Cook & Votta, 1998; Atkins et al., 1999) and was already successfully used to analyze the development of the Apache Web Server project (Mockus et al., 2000) and of the GNOME project (Koch & Schneider, 2002), both large scale open source projects.

Source Forge currently hosts over 59,000 open source projects and has over 597,000 registered users (April 2003). It provides the projects with a version control facility as well as a presentation platform and communication channels for developers and users. For Source Forge each developer has a unique pseudonym, the user name, which can be used throughout all projects he or she participates in. Each project has a home page with general information about the project like the project name, a short description of the project, the developers in charge of the project (administrators), the development status of the project (alpha, beta, production, etc.), the intended audience (e.g. developers or end users), the programming languages used, and more general information.

For this study we analyze projects which use the object-oriented programming language Java and employ the version control tool Concurrent Versions Control (CVS) (Fogel, 1999). The CVS tool supports parallel development by several developers, comments for modifications of the code, control of releases, reversing modifications, generating history logs for the projects and for each individual file, and much more. A project is defined as a collection of individual files which are stored together with version control information in a CVS repository. New files can be added to the project and existing files can be modified by the developers of the project. To modify files using version control the developer has to obtain a version of the files from the repository, change the files locally and commit the modifications to the repository (execute a *check-in* for the files). During the check-in (often called modification request) the developer is encouraged to add a short log message which explains the purpose of the modifications and therefore makes it easier to understand the changes in the code later on.

CVS records the modifications in each file in lines of code (LOCs) added and LOCs deleted by the developer. The definition of LOCs used by CVS is the number of physical lines. There is no distinction between program statements, comments or other arbitrary text. We adopt this definition for our study. Furthermore, CVS does not explicitly record changes in a line; instead it records a changed line as a line deleted and a new line added. Therefore, the growth of LOCs for a check-in (the delta) is the difference between the LOCs added and the LOCs deleted.

To analyze the application of design patterns we have to identify the patterns in the projects. Design patterns are design artifacts that result in special constructions in the final code, e.g., several objects that interact in a certain way. It is very difficult to infer the application of design patterns automatically from code (see e.g. Antoniol et al. (1998) for an automatic approach). However, CVS provides us with additional information, the log messages for changes, which Mockus and Votta (2000) already successfully used to classify maintenance activities. For this study we analyze the log messages to identify the application of design patterns. Of course, design patterns can be applied without mentioning them in the log message. Therefore, we can only identify the application of design patterns when they are used for documentation of changes and to support communication between developers. This is one of the major contributions of patterns stressed throughout the literature, namely that the names of design patterns become part of a common language which developers use to communicate design more efficiently (Gamma et al., 1995; Buschmann et al., 1996; Vlissides, 1998). However, it is important to note that with this approach we only analyze the communication aspect of patterns. To analyze the influence of patterns on software quality or the usage of patterns in general other approaches are necessary which might include analyzing bug tracking databases, interviews with developers, extensive manual code reviews, and controlled experiments.

For this study we only use the original set of the 23 design patterns introduced by Gamma et al. (1995). Although many other design patterns were introduced in the literature (e.g. in Coplien and Schmidt (1995), Vlissides et al. (1996) and Martin et al. (1998)), these 23 patterns are still the most popular and best known patterns. For the analysis we first extracted the CVS log messages for each project using Java. We parsed the messages using regular expressions, identified changes to files that constitute together a check-in, and stored the information in a relational database. All analyses in the subsequent sections are performed using standard SQL select statements on the data base and a standard statistical package.

THE DATA SET

The used data set includes 988 open source projects from Source Forge using Java as the main programming language. The projects were downloaded between August and September 2001 and were selected by the following criteria: Only projects that enabled CVS and that have had already more than 1,000 LOCs Java code. In total the selected projects contain almost 120,000 files (with the extension *.java*) with more than 19.5 million LOCs and 1,487 different developers worked on them. Figure 1 depicts the distribution of the size of the projects in LOCs. The project sizes in number of files follows a similar distribution. Most of the projects are rather small with a mean around 20,000 LOCs or 120 files, but there is a significant amount of much bigger projects. The biggest project has almost 1 million LOCs and almost 6,000 files. The sizes of the developer teams for the projects show a similar distribution with many projects with only a single developer (see Figure 2). The mean of the team size is 1.84 and the biggest team consists of 73 developers.

Figure 3 shows the number of projects by the development status used by Source Forge. The status ranges from 1 to 6 (1 planning, 2 pre-alpha, 3 alpha, 4 beta, 5 production/stable and 6 mature) and gives an idea about the project in its development live-cycle. Most of the selected projects with more than 1,000 LOCs have a status of 3 and 4. Fewer projects with status 1 and 2 already reached the minimum LOCs. For the status 5 and 6 there are fewer Java projects in Source Forge indicating that most of the analyzed projects can be considered still in the design and implementation phases of the software life-cycle. In Table 1 we summarize the statistics of the analyzed projects.

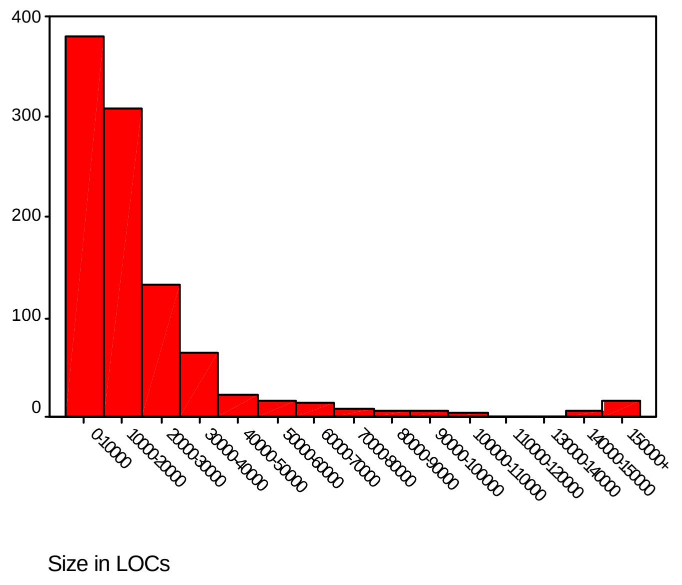

Figure 1: Number of projects by size in LOC

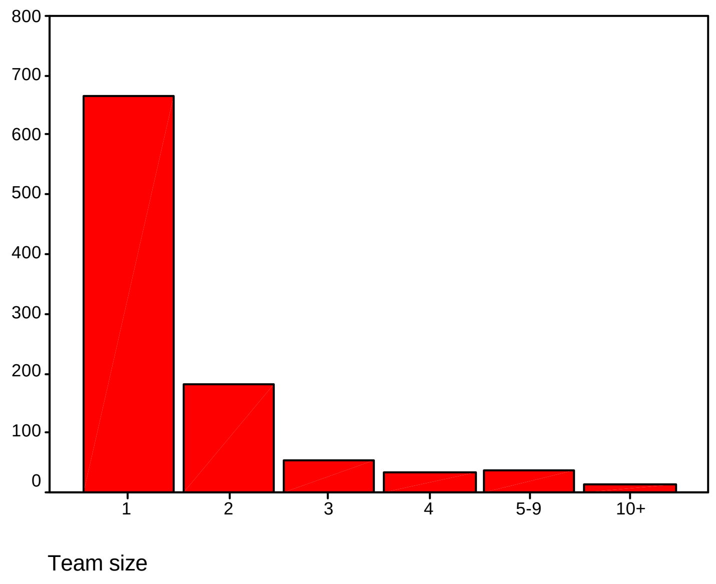

Figure 2: Number of projects by team size

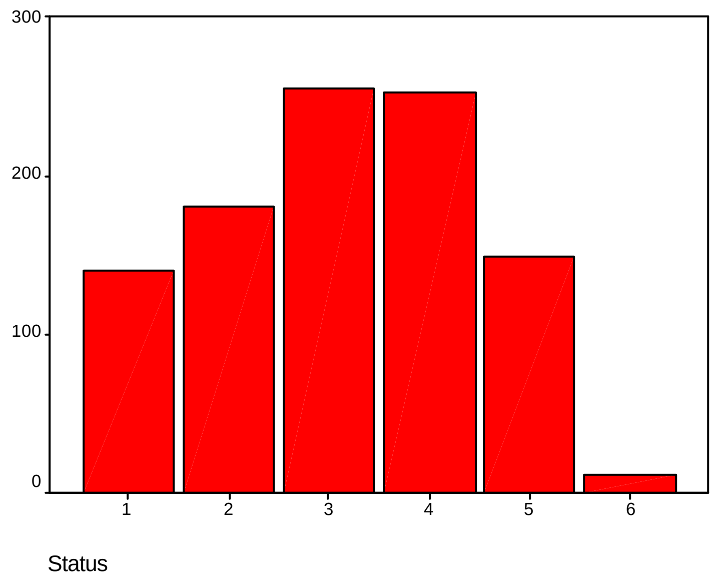

Figure 3: Number of projects by status

Table 1: Descriptive statistics of the main project variables

Case Summaries

	Status	Size in LOCs	Size in files	Team size
N	988	988	988	988
Minimum	1	1001	1	1
Maximum	6	961961	5951	73
Median	3.00	7279.50	50.00	1.00
Mean	3.13	19908.79	121.14	1.84
Variance	1.703	2952336139	104534.022	9.468

For the analyzed changes made to the projects we excluded the initial check-in of new files since new files added to the Source Forge CVS repository often have already a considerable size and it is impossible to say how many developers, and who, worked on this file already. We only know the developer who checked-in the file. This would distort the results in an unpredictable way. We observed about 97,300 check-ins where 6.65 million lines of code were changed or added in almost 329,000 files. A check-in affected on average 3.38 files and increased the code base of the project by almost 20 lines. Table 2 summarizes the descriptive statistics of the analyzed changes.

Table 2: Descriptive statistics of the analyzed changes (check-ins)

	Number of files	LOCs added/changed	LOCs deleted	Increase in LOCs
N	97292	97292	97292	97292
Minimum	1	0	0	-5375
Maximum	644	27189	26998	18018
Median	1.00	7.00	3.00	.00
Mean	3.38	68.40	48.92	19.48
Variance	77.328	172224.284	143837.124	25235.507

RESULTS

In this section we present the results of our explorative analysis of the adoption of design patterns by open source developers. The section is divided into three parts. First, the results of the pattern identification are presented. Second, we try to find out if characteristics of the project (e.g., size) make the usage of design patterns more likely. And finally we analyze if there are specific characteristics of developers who use design patterns (e.g., extent of participation).

The Identified Design Patterns

To identify patterns in the log messages we first searched all messages for the names of the 23 design patterns introduced by Gamma et al. (1995). In the log messages of 1,475 of the 97,292 observed check-ins we found the name of at least one pattern. The names found most often were "command" (142 counts) and "state" (80).

Of course we expect a high number of false positives since several pattern names (e.g. Prototype) are also terms frequently used in software engineering with a meaning other than the design pattern. To quantify the rate of false positives we manually inspected the log message and the code of a random sample of 100 out of the 1,475 check-ins where pattern names were found. For 69 changes it was clear from the log messages whether a design pattern was meant or not and for the remaining 31 changes we had to inspect the source code. In the inspected sample we found a false positive rate of 69%. For example, "command" was almost always used in connection with changing the command-line interface of the software which is not related to the application of a design pattern. To reduce this problem we required the pattern names, which are also common terms for software development or which have a meaning in the projects' problem domains (command, interpreter, prototype, proxy, state, and strategy), to be accompanied by the term "pattern" which reduced the number of check-ins with patterns to 343.

Table 3: Accuracy of pattern identification from log messages

			Contains a pattern		
			no	yes	Total
Classified as pattern	no	Count	65	7	72
		%	90.3%	9.7%	100.0%
	yes	Count	4	24	28
		%	14.3%	85.7%	100.0%
Total		Count	69	31	100
		%	69.0%	31.0%	100.0%

Table 3 depicts the accuracy of this approach. 85.7% of the check-ins identified as using a pattern really use the pattern reducing the rate of false positives to 14.3%. Overall 89 out of the 100 inspected check-ins were classified correctly. Note that this approach only identifies the application of design patterns where their full original names are used in the log messages. Therefore, the actual usage of design patterns is considerably higher and includes the following cases:

1. Alternative names are used (e.g., Virtual Constructor instead of Factory Method; some alternative names can be found in Gamma et al. (1995)).
2. Design patterns are not mentioned in the log message but are obvious from comments in the code or the names of classes, methods and files.
3. Design patterns are used without giving any clue.

Although it is possible to incorporate some of these cases into an identification heuristic we use the simpler and therefore more robust approach described above.

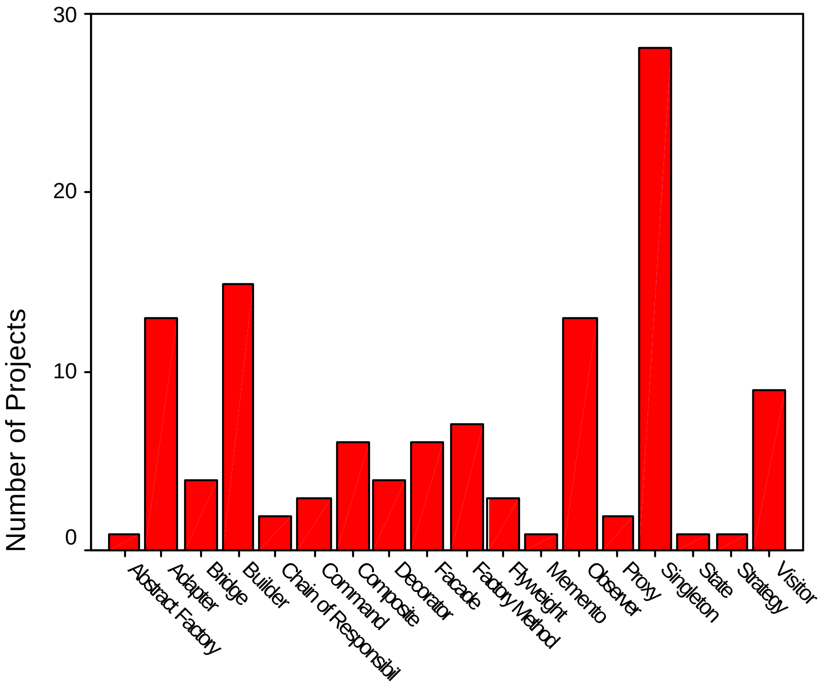

Figure 4: Number of projects using individual patterns

In Figure 4 we show the number of projects in which we identified each individual pattern. The pattern mentioned most by far in the analyzed projects is the Singleton pattern. The reason for this is that the Singleton pattern is very simple and is easy to implement in Java. This leads us to the conclusion that for Java the application of the pattern Singleton seems to provide important design advantages at little implementation effort. During manual inspection of the log messages we observed that out of 8 inspected changes concerning the Singleton pattern 4 changes consisted of removing existing Singletons. This perhaps indicates overuse of the pattern caused by its simplicity. Further analysis is needed to investigate the usage of different patterns and pattern types, but this is outside the scope of this chapter.

Analysis of Differences Between Projects

For Source Forge all development efforts are organized in projects as the basic unit of coordination. A project has one or several administrators who coordinate the development of the project and organize the cooperation between the developers. In this section we investigate if characteristics of projects (e.g., development status, project size, team size, number of check-ins) are related to the application of design patterns for documenting changes.

Most of the projects in the data set do not apply patterns for documenting changes in the code. Only for a small fraction of projects, 86 out of 988 projects, we found one or more patterns. To exclude very small projects which are still in their planning phase and therefore do not have enough code which can contain design patterns, we only select the projects for which we observed at least 1,000 LOCs being added or changed. This leaves 519 projects for analysis. The general trend of the distributions of project sizes and team sizes of this sample is similar to the distributions of the whole data set. Only for the development status more projects with a status smaller than 3 do not meet this criterion resulting in a shift towards projects with status 3 and up. As the indicators of pattern usage in a project we used the number of developers using patterns and the number of distinct patterns used in a project. Table 4 contains the descriptive statistics for the selected projects.

Table 4: Descriptive statistics for the selected projects

Case Summaries

	Status	Size in LOCs	Size in files	Number of check-ins	Team size	Developer using patterns	Number of distinct patterns
N	519	519	519	519	519	519	519
Minimum	1	1038	1	4	1	0	0
Maximum	6	961961	4265	4146	73	10	5
Median	3.00	12126.00	79.00	68.00	2.00	.00	.00
Mean	3.30	27172.79	162.34	164.47	2.47	.20	.22
Variance	1.604	3.9E+09	113925	119751.852	17.029	.401	.359

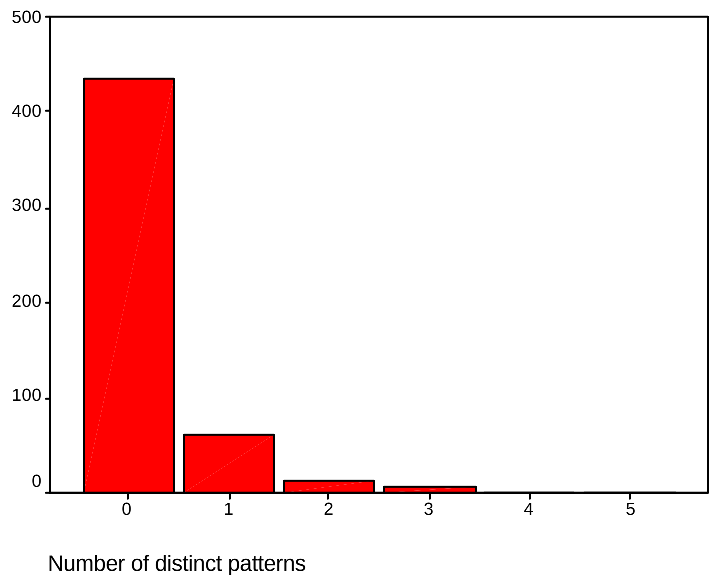

Figure 5: Projects by number of different design patterns used

In Figure 5 we show the number of projects using 0,1,...,5 different design patterns in the project. For 83 projects (16% of the 519 projects) at least one design pattern was used.

To explore the relationship between the main project variables (development status, size of the project, the number of check-ins, and the team size) and the use of design patterns for documenting changes we calculated the correlation coefficients. Since the distributions are highly skewed we use non-parametric Spearman's rank order correlation. All reported correlation coefficients are significant with $p < .01$.

The two measures of project size, LOCs and the number of files, are highly correlated with a coefficient of .90. Both measures are correlated with the number of check-ins, with a coefficient around .60. Team size is correlated with the number of check-ins (.48) and with the project size (around .30). These correlations reflect the intuitive relationship with larger projects having more check-ins and also tend to be developed by bigger teams.

Both indicators of pattern usage (number of different patterns and developers using patterns) seem highly correlated with each other in the data set (a significant coefficient of

.99) and are significantly correlated with the number of check-ins (.33), the size of the project (around .19) and the team size (around .16). However, all these correlations are artifacts occurring only because we have a very small proportion of projects with patterns (creating the high correlation coefficient between the two indicators) and because the probability of detecting a pattern increases with the number of check-ins we analyze and this number is in turn correlated to the other variables. To improve this analysis, object-oriented software metrics like the Chidamber and Kemerer's Metrics Suite (1994) can be used. These metrics can reflect differences in the complexity of classes and their interactions better than simple LOCs. However, it was shown by Masuda et al. (1999) and by Reisinger (2001) that the introduction of design patterns can increase the complexity measured by object-oriented metrics by using additional classes, more inheritance, and increased coupling between the classes the pattern is composed of. Since this would create an additional artificial relationship between complexity and the usage of design patterns we restrict our analysis here to the simple LOCs-oriented analysis made possible by the output of the CVS tool.

To analyze the application of patterns in more detail we cross-tabulate team size by developers using patterns (see Table 5). A simple assumption would be that the application of patterns is independent from the projects and there exists a fixed subset of developers who know and use patterns. Under this assumption we can estimate the proportion of developers using patterns from the projects with a team site of 1 in Table 5. We found patterns in 11.4% of the projects with one developer. If the usage of patterns is independent of the projects, this percentage can be used as an estimate of the total proportion of developers using patterns. With this estimate we can calculate the probability that we see at least one developer using a pattern in teams of sizes 2 and up. This simple model largely overestimates the observed proportion of projects with developers who document changes with patterns (compare in Table 5). The observed proportion of projects with developers who use patterns stays almost constant around 20% for team sizes 2 to 9. Only for team sizes 10 and larger the proportion jumps to over 60% which is still considerably below the estimate. Therefore, the assumption that the developers who use patterns are evenly distributed over all projects does not hold.

Table 5: Cross-tabulation of team size and the number of developers who use patterns

		Team size						Total
		1	2	3	4	5-9	10+	
Developer using patterns	0	226	107	43	26	29	5	436
	1	29	21	8	6	6	2	72
	2		2		1	2	3	8
	3						1	1
	4						1	1
	10						1	1
Total		255	130	51	33	37	13	519
Observed projects with patterns		11.4%	17.7%	15.7%	21.2%	21.6%	61.5%	16.0%
Estimated projects with patterns[a]		11.4%	21.5%	30.4%	38.3%	50.8%	82.2%	18.4%

[a] Using team size=1 to estimate propability of a developer using patterns

Next, we only look at the projects where patterns are used (rows with developers using patterns greater than 0 in Table 5). Interestingly, we found in almost all these projects only one developer who used patterns for documentation no matter what size the team was. Again only at a team size of 10 and larger we found more developers using patterns. The projects with team sizes 1 through 9 all had on average similar sizes and number of check-ins. For the projects with team sizes 10 and higher the average of all variables (except the project status) was by the factor 2 to 4 higher which seems to make them different. However, the fact that for almost all projects only one developer uses design patterns indicates that there must be factors that influence the usage of design patterns which are not dependent on the team size and other characteristics of the project but on the developer and his or her position inside the project.

An interesting fact is that the development status of the project has only very small significant correlation (between .13 and .24) with the other variables and no significant correlation with the indicators of pattern usage. This means we have very different projects in terms of size in the data set (ranging from small tools to large applications) and that we cannot find evidence that design patterns are used more often in later stages of the life-cycle to refactor code (replace code and design with more flexible design provided by a design pattern) as suggested by Gamma et al. (1995). To analyze this in more detail we split the projects into two groups. The first group contains project that are still in an early phase (277 projects with status 1-3) and projects that are more mature (242 projects with status 4-6). If design patterns are used frequently for refactoring, the more mature projects should contain significantly more different design patterns or a higher proportion of check-ins containing patterns. We used the non-parametric Mann-Whitney U test to compare the two groups of projects. We found that project size, team size and number of check-ins is significantly higher for the more mature projects ($p < .0033$, significant at the .01 level using Bonferoni correction for 3 independent tests). But there is no significant difference between the groups for the indicators of pattern usage ($p > .9$). This either results from the fact that design patterns are not used for refactoring and documenting these changes in our data set or that if design patterns are used at all they are already applied in the initial stages of development to create new design. For OSS development the second explanation seems appealing since in the early stages of the project developers need to create and communicate the design of a system using code (Vixie, 1999). And supporting communication is a major advantage attributed by experts to design patterns.

Analysis of Differences Between Developers

In this section we use the developer as the main unit of analysis. We analyze the usage of patterns for each developer and compare it with observed characteristics of the developer to investigate if there exists a relationship. As the developer's characteristics we use the number of projects a developer participates in, the LOCs he or she modified in the analyzed period, the increase of LOCs and the check-ins the developer produced. As the indicator of pattern usage we use the number of different patterns used by a developer. To exclude developers who did not apply patterns only because of their small contribution of code to the analyzed projects we restrict our analysis to the 761 of the 1,487 developers who added or changed more than 1,000 LOCs.

First, we look at the number of projects a developer participates in (see Figure 6). More than 80% of all developers only participate in one project. Although participation in several projects could indicate more experience no significant correlation was found between the number of projects a developer participates in and the indicator of design pattern usage.

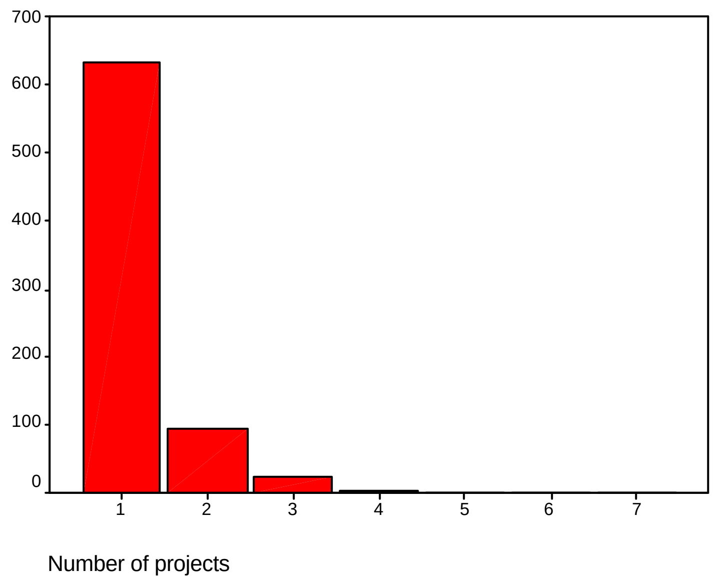

Figure 6: Number of projects a developer contributes to

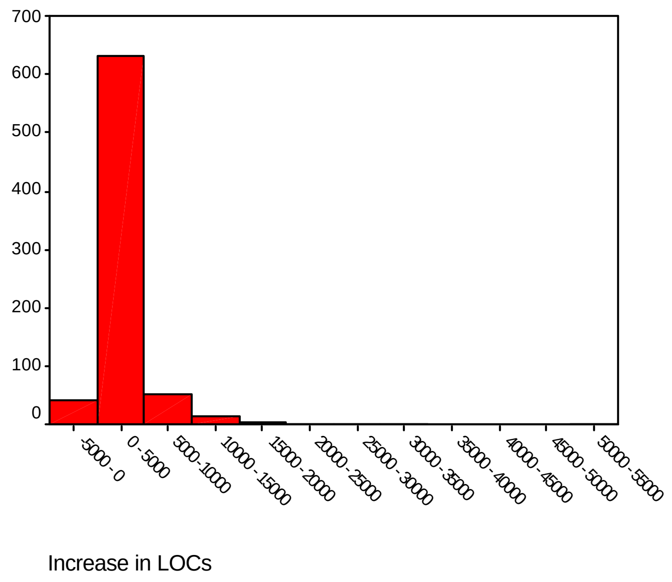

Figure 7: Histogram of the increase of LOCs by developer.

The increase in LOCs produced by developers was also used by Mockus at al. (2000) and Koch and Schneider (2002) to analyze two big open source projects. A main finding of these studies was that there exists a relatively small very active group of *core developers* who typically produce more than 80% of the total code. This group is responsible for implementing most of the new functionality which involves also making design decisions. The histogram of the increase in LOCs by developer for the analyzed projects is depicted in Figure 7. Only 278 of the 1,487 developers are responsible for 80% of the total increase in LOCs for all projects. With 18.7% of the total developers the proportion is similar to the proportion found for the gnome project analyzed in Koch and Schneider (2002). A observation that also fits well the Pareto Principle (also known as the 80/20 rule) which was found to hold in many different settings and disciplines. The 278 most dedicated developers can be seen as the core developers in the development community of the analyzed projects where depending on the projects size none (for very small projects), one or several core developers work together with less active developers on a project.

We use rank order correlations to check if the usage of design patterns is related to indicators of activity of different developers. Significant correlation coefficients were found between the number of different patterns he or she uses and the LOCs added or changed (.19), the increase in LOCs (.20) and the number of check-ins conducted (.27). All three indicators are significantly intercorrelated (with correlation coefficients between .57 and .63). These correlations could mean that developers who work more intensely on a project (more check-ins) and therefore increase the code size are more likely to document their changes using design patterns. But this is a very tentative interpretation given the low correlation coefficient and the influence check-ins have on the opportunity to use a design pattern and the probability of detecting a design pattern with the used heuristic.

Table 6: Differences between developers who use patterns and who do not

Uses patterns		Number of projects	LOCs added/changed	Increase in LOCs	Number of checkins
no	N	667	667	667	667
	Minimum	1	1001	-4157	1
	Maximum	7	173869	52797	2430
	Median	1.00	3585.00	1068.00	49.00
	Mean	1.22	7988.10	2117.14	97.26
	Variance	.382	213576579.378	15432652	36096.841
yes	N	94	94	94	94
	Minimum	1	1251	-3485	5
	Maximum	6	60955	32970	1665
	Median	1.00	6866.00	2858.00	123.50
	Mean	1.37	12022.61	4371.15	210.72
	Variance	.731	177046413.317	31091095	67273.148

To analyze this further, we divided the population of developers (the developers who added/changed more than 1,000 LOCs) into a group of developers for whom we never found patterns in the analyzed projects and a group of developers for whom we did. Table 6 contains the statistics for both groups. Except for the number of projects the medians and means of all variables are about double as high for the group which uses patterns. The Mann-Whitney U Test confirms that the location of the distributions of LOCs added or changed, the increase in LOCs and the number of check-ins differ significantly (applying Bonferroni correction for 4 independent tests which reduces the maximum accepted *p*-value to .0025 at the .01 level). This suggests that there is a highly significant difference between developers who use patterns for documentation and developers who do not. Developers who use patterns tend to create more new code (increase of LOCs) and therefore are also involved in creating new design.

Table 7: Cross-tabulation for normal developers and core developers who are responsible for more than 80% of the projects code base with pattern usage

			Uses patterns		
			no	yes	Total
Core developer	no	Count	447	36	483
		%	92.5%	7.5%	100.0%
	yes	Count	220	58	278
		%	79.1%	20.9%	100.0%
Total		Count	667	94	761
		%	87.6%	12.4%	100.0%

In Table 7 we show the cross-tabulation for developers (core developers identified above in this section and other developers who added or changed at least 1,000 LOCs) and pattern usage. On average, 12.4% of developers use pattern names in the log messages. The table shows a big difference (a significant relationship, $p < .01$ using the chi-square test) in pattern usage between core developers (20.9%) and other developers (7.5%). The observation of more core developers commenting their changes with patterns could have the following reasons:

1. Core developers simply check-in more changes and therefore have more opportunities to use patterns and document these changes with the names of the used design patterns.
2. Design patterns represent an efficient way to apply best practices in the form of flexible and robust design and to communicate these design decision (Beck et al., 1996). This is needed more often by experienced developers who create most of the new design.
3. The open source community has a reputation-based culture (Raymond, 1999; Feller & Fitzgerald, 2000). Visible pattern adoption by core developers advertises competence (Seen et al., 2000) and can therefore be used to strengthen their position in the project team and in the open source community.

It seems reasonable to believe that all three reasons contribute to adopting design patterns to document changes in source code. However, it is not possible to quantify the extend to which each reason influences the observed pattern usage based on historic version control data alone. Another study which includes interviews of developers who use patterns is necessary.

SUMMARY OF RESULTS AND CONCLUSION

Open source software development represents a new and efficient way to produce high quality software. Many traditional software engineering practices are adopted by open source developers. However, different software engineering practices support the collaborative and implementation-driven development style of OSS better than others. In this chapter we analyzed the adoption of design patterns by open source developers. Design patterns are interesting for OSS development since their adoption does not require an infrastructure investment and they can be used by a developer without interfering with the development style of others. For almost 1,000 open source projects using Java we checked if the 23 original design patterns introduced by Gamma et al. (1995) were used to document changes in the code. There are 3 main results of analyzing the factors that influence pattern usage:

1. The characteristics of projects have little influence on the developers' adoption of design patterns. For most team sizes the percentage of projects where patterns are used for documentation is around 20% in the data set. Only the projects with the largest team size (10+) show a significant higher proportion. These projects differ from the rest by a much higher level of activity (number of check-ins, increase and changes of LOCs are on average 2-4 times higher than the other projects). Also we discovered that for most projects where design patterns were used only one developer used them (again except the few most active projects).

2. For the usage of design patters in a project and the project's phase in the software life-cycle no relationship was found. Therefore, in the analyzed OSS projects we found no evidence that design patterns are widely used for refactoring in later stages of the software development. A reason for this finding could be that the analyzed projects are still too early in their life-cycle to make major restructuring necessary. Another explanation could be that open source development generally favors a more flexible design by already using patterns for developing new code and frequent modifications and expansion of the code. This constant change would reduce the need for an explicit refactoring in later phases of the life-cycle.

3. There exist differences between developers who use patterns and developers who do not. There are significant differences in the level of activity of developers in the analyzed projects. These differences indicate that the small number of developers that create most of the code (for OSS projects often called core developer) are more likely to use design patterns. About 20% of these developers used the 23 design patterns to document changes.

This first study of the adoption of design patterns by open source software developers has many limitations, e.g., information on the quality of the produced code is not included, no differentiation between types of changes (bug fixes, new features, etc.), the complexity of the projects is not analyzed using object-oriented metrics, only a very limited number of known design patterns and only one programming language is used, and no further information from the developers (actual effort used for changes, reasons for using patterns) is incorporated. Each of these limitations provides a direction for further research. Even with these limitation, the results of this study show that design patterns are adopted for documenting changes and thus for communication in practice by many of the most active open source developers.

REFERENCES

Atkins, D., Ball T., Graves, T., & Mockus, A. (1999). Using version control data to evaluate the impact of software tools. Proceedings of the 21st International Conference on Software Engineering, 324-333.

Antoniol, G., Fiutem, R., & Cristoforetti, L. (1998). Design pattern recovery in object-oriented software. Proceedings of the 6th Workshop on Program Comprehension (WPC), 153-160.

Beck, K., Coplien, J.O., Crocker, R., Dominick, L., Meszaros, G., Paulisch F., & Vlissides, J. (1996). Industrial experience with design patterns. Proceedings of the 18th International Conference on Software Engineering, 103-114.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). Pattern-oriented software architecture, a system of patterns. Chichester, England: John Wiley & Sons Ltd.

Chidamber, S.R. & Kemerer, C.F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering, 20(6),* 476-493.

Coplien, C.O. & Schmidt, D.C. (1995). Pattern languages of program design. Reading, Massachusetts: Addison-Wesley.

Cook, J.E. & Votta, L.G. (1998). Cost-effective analysis of in-place software processes. *IEEE Transactions on Software Engineering, 24(8),* 650-663.

Dempsey, B.J., Weiss, D., Jones, P., & Greenberg, J. (2002). Who is an open source software developer? Profiling a community of linux developers. *Communications of the ACM, 45(2),* 67-72.

Feller, J., & Fitzgerald, B. (2000). A framework analysis of the open source software development paradigm. Proceedings of the 21st Annual International Conference on Information Systems, 58-69.

Fogel, K. (1999). Open source development with CVS. Scottsdale, AZ: CoriolosOpen Press.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design patterns: elements of reusable object-oriented software. New York: Addison-Wesley.

Goldfedder, B., & Rising, L. (1996). A training experience with patterns. *Communications of the ACM, 39(10),* 60-64.

Helm, R. (1995). Patterns in practice. Proceedings of the 10th Conference on Object-Oriented Programming, Systems, Languages, and Applications, 337-341.

Jorgensen, N. (2001). Putting it all in the trunk: Incremental software engineering in the FreeBSD open source project. *Information Systems Journal, 11(4),* 321-336.

Koch, S., & Schneider, G. (2002). Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal, 12(1),* 27-42.

Martin, R.C., Riehle, D., & Buschmann, F., (1998). Pattern languages of program design 3. Reading, Massachusetts: Addison-Wesley.

Masuda, G., Sakamoto, N., & Ushijima, K. (1999). Evaluation and analysis of applying design patterns. Proceedings of the International Workshop on the Principles of Software Evolution (IWPSE99).

Mockus, A., Fielding, R., & Herbsleb, J. (2000). A case study of open source software development: the Apache server. Proceedings of the 22nd International Conference on Software Engineering, 263-272.

Mockus, A., & Votta, L.G. (2000). Identifying reasons for software changes using historic databases. International Conference on Software Maintenance, 120-130.

Perpich, J.M., Perry, D.E., Porter, A.A., Votta, L.G., & Wade, M.W. (1997). Anywhere, anytime code inspections: using the Web to remove inspection bottlenecks in large-scale software development. Proceedings of the 19th International Conference on Software Engineering, 14-21.

Prechelt, L., Unger-Lamprecht, B., Philippsen, M., & Tichy, W.F. (2002). Two sontrolled experiments assessing the usefulness of design pattern information in program maintenance. *IEEE Transactions on Software Engineering, 28(6),* 595-606.

Raymond, E.S. (1999). The cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary. Sebastopol, California: O'Reilly and Associates.

Reißing, R. (2001). The impact of pattern use on design quality. Position paper, Beyond design: Patterns (mis)used, Workshop at the 18th Conference on Object-Oriented Programming, Systems, Languages, and Applications.

Seen, M., Taylor, P., & Martin, D. (2000). Applying a crystal ball to design pattern adoption. 33rd Technology of Object-Oriented Languages and Systems (TOOLS-33 2000), 443-454.

Vixie, P. (1999). Software engineering. In C. DiBona, S. Ockman & M. Stone (Eds.), *Open Sources: Voices from the Open Source Revolution.* Cambridge, Massachusetts: O'Reilly and Associates.

Vlissides, J. (1998). Pattern hatching: Design patterns applied. Software Patterns Series. New York: Addison-Wesley.

Vlissides, J., Coplien, J. O., & Kerth, N. L. (1996). Pattern languages of program design 2. Reading, Massachusetts: Addison-Wesley.