LATVIJAS UNIVERSITĀTES RAKSTI. 2008, 733. sēj.:
DATORZINĀTNE UN INFORMĀCIJAS TEHNOLOĢIJAS 53.–71. lpp.

Use of Design Patterns in PHP-Based Web Application Frameworks

Andris Paikens, Guntis Arnicans

Department of Computing University of Latvia
Raiņa blvd 19, Rīga, Latvia LV-1586

Andris.Paikens@di.lv, Guntis.Arnicans@lu.lv

Abstract. It is known that design patterns of object-oriented programming are used in the design of Web applications, but there is no sufficient information which data patterns are used, how often they are used, and the level of quality at which they are used. This paper describes the results concerning the use of design patterns in projects which develop PHP-based Web application frameworks. Documentation and source code were analysed for 10 frameworks, finding that design patterns are used in the development of Web applications, but not too much and without much consistency. The results and conclusions can be of use when planning and developing new projects because the existing experience can be taken into account. The paper also offers information which design patterns are not used because they may be artificial or hard-to-use in real projects. Alternatively, developers may simply lack information on the existence of the design patterns.

1 Introduction

The rapid increase in the number of Web users over the last two decades, the expanded opportunities and accessibility of software design, and the greater demand for such applications – all of this has contributed to an enormous increase in the number of people who are working on the design of Web applications. Web applications used to be nothing more than an add-on to some other serious system for a period of time not so long ago; and design of these web applications involved people with a great deal of experience in other areas of software. But today eager young people begin designing Web pages without being aware of even the simplest principles of software design. The complexity of applications has increased, and their use has become more serious.

The authors of this paper are interested in the use of design patterns in frameworks related to Web application design, because the true use of design patterns in actual projects has not been yet described to any great extent, which means that it is not yet clear whether this approach is of use for the relevant assignments.

2 Problems in the Use of Design Patterns

2.1 Design patterns

One can agree with Jason E. Sweath [1], who has argued that many software design problems are resolved again and again, and many fundamental solutions are established with the goal of speeding up the development processes, reducing the amount of work that has to be done, and improving the quality of software and, by extension, that of the resulting products. Many of these solutions can be brought together under the heading of "design patterns".

How to describe the concept of design patterns in one sentence? Design patterns represent theoretical material collected by experts in the field regarding the principles which should be used in solving a specific problem, the issues that should be taken into account, and the issues that are unimportant in the context of the relevant problem.

Use of design patterns may seem quite natural when working with a language that supports the object-oriented software design and programming approach (OOP), but it is a fact that languages which support OOP are not mandatory. However, even if a language that does not support OOP principles is used, it is necessary to be familiar with the foundations of OOP to understand the principles of design patterns and their use.

The literature about design patterns offers several definitions [2], [3], [1]. If we correlate these, we come up with a new definition of design patterns:

Design patterns describe a specific design problem which is repeatitive and which appears in the context of a specific design, offering a proven and general scheme to solve the problem. The solution scheme is detailed through a description of its components, their responsibilities and relationships, and the way in which they work together.

Here we present the three-part scheme which is at the foundation of every design pattern [9]:

1. *Context* – the situation in which the problem occurs;
2. *Problem* – the problem which repeatedly occurs in this context;
3. *Solution* – a proven resolution to the problem.

The context of a design pattern refers to the environment in which a problem arises. Context also covers all steps that have been taken before the specific problem occurred. Context can also include the steps that will have to be taken after the problem occurs.

Problem in this regard is an issue which must be handled by implementing specification requirements and by taking the context into account. When the problem is identified, its overall specifications must be designed, and its essence must be understood – what is the specific design problem that has to be solved?

Problem can be divided into several sections:

1. Requirements for the solution, e.g., it must be effective;
2. Issues that must be taken into account, e.g., which standards should be observed;
3. The desired properties.

Solution in a design pattern shows how to resolve the repeated problem and how to balance out related issues. There are two parts to the solution – the structural part,

which is the static part of the solution, and the performance part, which is the dynamic part of the solution.

2.2 Properties of Design Patterns

Each design pattern is different from others, because each is intended for the solution of a different problem, or for a different solution to one and the same problem. At the same time, all design patterns have certain properties in common.

In [9], [1], [10], and other sources, the following properties of design patterns are mentioned; to a greater or lesser extent, all design patterns have them in common:

- A design pattern focuses on a design problem which is repeatitive and which occurs in specific design situations, offering a solution to the problem;
- Design patterns document existing, proven, and applied design solutions;
- Design patterns identify and make more precise abstractions, which is more than just defining an individual class, instance, or component;
- Design patterns ensure common vocabulary and understanding regarding the relevant design principles;
- Design patterns are a way of documenting software architecture;
- Design patterns support the development of software with certain desirable properties;
- Design patterns help to design complicated and heterogeneous software systems;
- Design patterns help to put reins on the complexity of software.

2.3 The Specifics of Web Applications

Web applications are quite different from other types of software. The execution process is one of the main differences – most of the processing is carried out on the server, the result is sent to the user's browser, and then the user can engage in other activities related to the application. For that reason, it is harder to work with objects – the PHP software design language, for instance, establishes an object which exists only for a short period of time – while the code is responsible for handling the particular request execution. Not all software design languages that are used in the design of Web applications are supported by OOP, and that makes it more difficult to deal with more complicated problems and to use design patterns.

The most important difference between Web applications and others is that most people have to use a Web browser to employ the software, and that covers some of the relevant functionality. Web applications are also affected by the fact that they are stored on servers, are called up, and are carried out both on the server and on the workstation of the user.

2.4 Positioning the Problem

While researching the use of design patterns in Web applications, we sought to learn the current situation in terms of whether design patterns are used in Web applications at this time at all and, if so, to what extent.

Theories about software design are good if they can be sooner or later applied in practice. Much time has passed since the first publications about design patterns, and design patterns also are of use in the design of Web applications. Hence, review of several projects can lead to the discovery of overall trends in the use of the design patterns.

There are many different design patterns, but many popular OOP design patterns must be viewed differently when it comes to Web applications. From all available design patterns, these were reviewed: Abstract Factory, Application Controller, Active Record, Adapter, Builder, Command, Composite, Custom Tag, Data Mapper, Decorator, Dependency Injection, Domain Model, Factory Method, Front Controller, Handle-Body, Iterator, MockObject, Model-View-Controller, Mono State, Observer, Proxy, Registry, Server Stub, Singleton, Specification, State, Strategy, Table Data Gateway, Template View, Template Method, Transform View, ValueObject, View Helper, Visitor. For detailed description of these design patterns see Appendix A.

Some of these have emerged in a natural way. Others are artificial and seldom used in real life. The review of the situation told us which design patterns are more popular and how many projects use them. The results could be used as hypotheses regarding the purposes toward which design patterns are used – are they used to reuse fragments of source code? Are they used to divide up the project into logical components to reduce the complexity of understanding and development? Or are they merely a fashion statement aimed at advertising one's own product?

2.5 Benefits from Researching the Existing Situation

The benefits of learning about the existing situation in the use of design patterns for the development of Web applications might be the following:

- Information about the existing situation helps others to start using design patterns, achieving the level which specialists in this area achieved after years of practice in testing the proposed design patterns more quickly. A beginner can immediately use the most popular tested design patterns, which means that he or she has more time to look for other design patterns aimed at specific tasks.
- Additional information is obtained about the diversity and use of design patterns. Research cannot immediately lead to precise answers to all the possible questions, but it outlines the directions and areas in which answers could be obtained through additional research. Our research showed that not all design patterns are actually needed from the list of so many. There are a few seemingly attractive design patterns that are not used at all.
- One can draw conclusions whether design patterns fulfil their mission in the first place. Do they help people to understand software architecture and logic? To what extent do design patterns support the principle of "divide and conquer", i.e., to what extent are the logic of systems described via design patterns?

- One gets an idea about the attitude of designers vis-à-vis design patterns. Is the use of design patterns in source code displayed openly, or is it hidden and thus hard to recognise?
- There is a better understanding of problems which are not solved via the use of design patterns. Hence, existing design patterns can be reworked, and new ones can be developed. It may well be that many design patterns which are used in classical object-oriented projects are defined as inappropriate for the development of Web applications.

2.6 The Method for Studying Design Patterns

To gain an objective idea about the use of design patterns, the research was based on the following principles:

1. Several solutions in one and the same class of tasks must be studied, i.e., the relevant Web applications must be used for more or less the same task;
2. Web applications basically use one and the same software design language;
3. There is access to documentation about the application and the source code (open code software is reviewed);
4. Developers have at least minimal knowledge about design patterns, learned from available books about design patterns.

Framework development projects which are written in the PHP language corresponded to these principles quite well. All of the selected projects were based on open source software, so project documentation and source code were available.

Web application frameworks are ideologically close to design patterns because they handle many of the same functions, albeit at a different level. This means that the developers of web application frameworks are far better informed and knowledgeable about design patterns than is the average Web application designer; and this is very believable. The first review of project documentation confirmed it – design patterns were discussed to a sufficient degree.

It was decided that the research would be based on the price/performance ratio, thus obtaining information about the existing situation via sensible use of resources. Without preliminary knowledge, it is hard to choose the frameworks which are needed for the research if more than 40 frameworks are available. The research was essentially based on reviewing 10 PHP frameworks that were mentioned in the article [5]. On the basis of various criteria, these are among the best and most popular frameworks. In terms of projects, researchers studied project documentation, project Web page, and the source code.

Many design patterns which are frequently mentioned in the literature were chosen. The researchers checked whether each selected design pattern is mentioned in some way in documentation, the homepage, and the source code. The source code was examined to look for the names of the methods as well as comments.

The information allowed researchers to determine whether the relevant design pattern was being used for project implementation. Unclear situations were also noted. The researchers did not analyse whether the design patterns were used in strict accordance with the principles defined in the literature.

Initially, the goal was to interview the organiser of each project, but that would have required greater resources and would not have had all that much effect on the

results. We wanted to learn about the most important trends in the use of design patterns, and small omissions in the process could not have had a major effect on the results.

3 The Research Object

3.1 Design Patterns Used in the Research

Information on design patterns can be found in many books, for example [6] and [7], and we used many resources in our work. We would particularly like to point out three resources and the information about design patterns contained therein.

First, the book [1], which contains many design patterns and also explains the implementation of these patterns in PHP (*Value Object, Factory, Singleton, Registry, Mock Object, Strategy, Iterator, Observer, Specification, Proxy, Decorator, Adapter, Active Record, Table Data Gateway, Data Mapper, Model-View-Controller*).

A clear and understandable description of design patterns is found in www.dofactory.com – *Factory Method, Adapter, Builder, Bridge, Proxy, State, Strategy, Template Method, Visitor*.

Very useful information is also found in Wikipedia (http://en.wikipedia.org), although that resource does not provide information on all specific design patterns (*Dependency Injection, Abstract Factory, Mock Object, Singleton, Composite, Decorator, Chain-of-Responsibility, Observer, Iiterator, Active Record, Model-View-Controller*).

For detailed description of these design patterns see Appendix A.

3.2 Frameworks Used in the Research

"*A framework is a basic conceptual structure used to solve a complex issue. This very broad definition has allowed the term to be used as a buzzword, especially in a software context.*

A software framework is a re-usable design for a software system (or subsystem). A software framework may include support programs, code libraries, a scripting language, or other software to help develop and glue together the different components of a software project. Various parts of the framework may be exposed through an API." [8]

The frameworks used in the research are defined in [5], which has all-encompassing information on each of them – see Table 1. Some of the information is unquestionably out of date, but we must remember that many frameworks are constantly being developed and improved, and that makes it difficult to complete the list.

Table 1 is presented not to define the best framework, but instead to give the reader a sense of what each framework does. The functions could influence the

decision to use or not to use a specific design pattern. Detailed description of frameworks listed below can be found in Appendix B.

Table 1: Comparison of 10 different frameworks

	PHP4	PHP5	MVC	Multiple DB's	ORM	DB Objects	Templates	Caching	Data Validation	Ajax	Authentification module	Modules
Zend Framework		+	+	+	+	+	+	+	+		+	+
CakePHP	+	+	+	+	+	+	+	+	+	+	+	
Symfony Project		+	+	+	+	+		+	+	+	+	
Seagull Framework	+	+	+	+	+	+	+	+	+		+	+
WACT	+	+	+	+		+	+		+			
Prado		+		+			+	+	+	+	+	+
PHP on TRAX		+	+	+	+	+			+	+		
ZooP Framework	+	+	+	+		+	+		+	+	+	+
eZ Components		+		+		+		+	+			+
CodeIgniter	+	+	+			+	+	+	+			+

3.3 The Results of the Study of Frameworks

Before we analyse the results in depth, we must stress once again that using or not using design patterns cannot be the indicator to determine the best or worst framework. The comparison of two different frameworks only makes it possible to draw conclusions about the methods used by their developers. It is not possible to decide that one is better than the other – unless, of course, the comparison is based on the use of a specific design pattern in both frameworks, also looking at other indicators such as the amount of time that is needed to update the project code and to make the relevant changes in designing the project. However, these indicators are not really appropriate for comparison because if a project represents more than the strict implementation of a specific design pattern, then it includes several design patterns or, perhaps, solutions which have nothing to do with design patterns. Business logic and databases are mutually related, and it is all but impossible to measure the amount of time spent on the implementation of a single design pattern.

In other words, we evaluated the use of design patterns, not the frameworks within which the design patterns were used.

The research allowed us to create several result tables which have been combined for the purposes of this article in Table 2. The following notations are used:

"d" – the design pattern is mentioned in project documentation or on the homepage;

"c" – the design pattern is mentioned in the comments on the software code;

"m" – the design pattern's name is part of the name of the software design method;

Table 2: Research results

	Zend Framework	CakePHP	Symfony Project	Seagull Framework	WACT	Prado	PHP on TRAX	ZooP Framework	eZ Components	CodeIgniter	"d"	"c" and "m"	"c" and "m" and "d"
Abstract Factory	c											1	1
Application Controller	c	c		c	d		cd	c		mc	2	6	7
Active Record		d				cmd	mcd			dc	4	3	4
Adapter	mdc		cm	c		m				c	1	5	5
Builder	mc		mc	c	m	d			cm		1	5	6
Command	?	?	mc	?		?	?	?	?	?		9	9
Composite			mc	cd	?	m					1	4	4
Custom Tag													
Data Mapper		d				cm					1	1	2
Decorator	mc		cmd	cd	m	cm					2	5	5
Dependency Injection													
Domain Model					d						1		1
Factory Method	c		c	dc	c	c					1	5	5
Front Controller	cmd	d	d	c	cd					c	4	4	6
Handle-Body													
Iterator	cm		cm	d	c	m			c		1	5	6
MockObject				c	c							2	2
Model-View-Controller	d	dc	d	cd	d	c	d	d		cd	8	4	9
Mono State													
Observer	cm	m	cm	cd	m		m				1	6	6
Proxy	cmd		?	cmd							2	3	3
Registry	m	mc	m	cd	c					c	1	6	6
Server Stub						c						1	1
Singleton	c	c	cm	?d		c	c	c	c	c	1	9	9
Specification	?		?		?	?						4	4
State	?		?	?	?	?		?				6	6
Strategy	cm			?d	?						1	3	3
Table Data Gateway													
Template View				d	d						2		2
Template Method					c							1	1
Transform View													
ValueObject	mc			mc								2	2
View Helper	d	c									1	1	2
Visitor				c					c			2	2
"d"	5	4	3	11	5	2	3	1		2			
"c" and "m"	17	7	13	18	13	13	5	2	5	8			
"c" and "m" and "d"	19	10	15	19	16	14	6	5	5	7			

"?" – the design pattern's name is mentioned in the name of the class or method, but without detailed analysis, so it cannot be understood whether the method or class implements the design pattern, or whether the name of the design pattern simply coincides with the name of the method or class.

3.4 Mention of Design Patterns in Documentation

Before examining the overall situation, let us look at those design patterns which are mentioned in documentation. Documentation for us meant the homepage, the user instructions, or any other documentation offered by the project developer (except for software comments).

The frequency of use is seen in column "d", but it has to be remembered that this is perhaps not a depiction of the actual situation. Still, it does point to overall trends. Some of the design patterns might be a marketing trick with which the project developers try to attract users. Some design patterns may be mentioned but not really used, being planned only for the next project versions. Many of the indicated design patterns are used, but it is hard to tell the extent to which they are. An additional survey of developers would not change the results substantially because we assume that there are no cardinal differences between the documentation and reality.

It is also impossible to determine whether the noted design patterns have been the only ones used in each specific project because it is possible that a specific design pattern was used but is not mentioned in the documentation. Some projects had fairly incomplete information sections in terms of homepages and documentation. However, the general trends can be determined with certainty.

Table 2 shows that the *MVC (Model-View-Controller)* design pattern has been used most often in frameworks – in 8 projects in total. Other frequently used include *Front Controller* and *Active Record* (4 times). Still other design patterns are used in just one or two projects, and there are some design patterns which are not used at all.

Table 2 shows that the framework which uses most design patterns is *Seagull* – 11 different design patterns are used in it.

3.5 Mention of Design Patterns in Source Code

When it comes to the design patterns mentioned in documentation, it can be asserted that they have been used in the implementation of frameworks, or at least the developers have wanted to use them but have not done so for various reasons. The use of design patterns described in comments and methods is perhaps more questionable because of the unambiguous way in which methods and classes are named and comments about them are developed. Only in the guidelines to the *Zend Framework* there is a mention of the fact that the name of a design pattern must be included in the name of a class or method if it is used therein.

The names of some of the design patterns may coincide with the names that have been chosen by the developers for methods designed for very different purposes. It is impossible to make the design patterns table more precise because that would require not only a complete study of all the design patterns, which is very difficult,

but also an in-depth insight into all of the frameworks. That is almost impossible considering how many different frameworks there are. The task would be unrealistic for a few people who have a few months to spend on the task. Certainly the amount of resources that would be needed is not justified in terms of the results that could be obtained.

Table 2 suggests that the scene is very different if we compare the design pattern usage in code (column "c and m") to the mention of design patterns in documentation (the column "d"). There is no distinct leader here – the *Singleton* design pattern is used most often (9 times). The *Command* design pattern is used with equal frequency, but this fact is quite questionable because the design pattern bears a name which is also used for many software design methods; hence it may well be that in some cases the *Command* design pattern was not used at all. The design patterns *Application Controller* and *Observer* are used six times apiece. Others are used quite often but with less frequency. Still other design patterns are not used at all.

Among the frameworks in terms of the use of design patterns, *Seagull Framework* is the leader with 18 design patterns.

3.6 The Timeframe of the Research and Evolution of Frameworks

Initial research took place in late 2005 and at the beginning of 2006. Up to date results were collected until October 2007. Several things have changed since the authors of this paper investigated the subject for the first time, but these changes are not shocking as evolving products are upgraded time after time. The authors upgraded the original research by refreshing information on the usage of design patterns and related conclusions; some information was added in separate chapters of this paper so that readers could get an insight of how various frameworks have been developed and improved. As it can be observed, most of the products have not changed much over this period of time.

It is interesting how different frameworks have evolved over time. In Table 3, information is presented about the versions that were available in the spring of 2006 and the versions that are available in the autumn of 2007.

Improvements over time are not an unusual phenomenon if we speak about software. Changes show us the activities and intensity of development. The maturity level of the software can also be observed from such record of change as products do not change too much if a stable version is reached and no new functionality is demanded. Table 4 contains differences between the use of design patterns in spring 2006 and autumn 2007. Use of patterns mentioned here is implemented in recent versions of these frameworks and was absent in early 2006.

As we can see from Table 4, there are no many changes.

Table 3: Framework version evolution

Framework	Spring 2006	Autumn 2007
Zend Framework	**Preview 0.1.3**	**1.0.2**
CakePHP	**1.0.1.2708**	**1.2.0**
Symfony Project	**Beta 0.6.2**	**1.0.7**
Seagull Framework	**0.6.0RC2**	**0.6.2**
WACT - Web Application Toolkit	**0.2 alpha**	**0.2alpha**
Prado	**3.0.0 RC2**	**3.1.0**
PHP on Trax	**0.13.0**	**0.14.0**
ZooP Framework	**1.1**	**1.3**
eZ Components	**1.0.1**	**1.3.1**
CodeIgniter	**1.3.2**	**1.5.4**

Table 4: Changes in the use of design patterns in the period 2006-2007

	Zend Framework	CakePHP	Symfony Project	Prado	PHP on TRAX	ZooP Framework	eZ Components	CodeIgniter
Abstract Factory	c							
Application Controller	c							
Active Record				cmd				d
Adapter	c							
Builder	mc			d			cm	
Command	?	?		?		?	?	?
Data Mapper				cm				
Decorator	c							
Factory Method	c							
Front Controller	cm							
Iterator	c							
Model-View-Controller		c						
Observer	cm							
Proxy	cm							
Registry								c
Server Stub				c				
Singleton	c				c	c		c
Specification			?					
Strategy	c							
Visitor							c	

3.7 Conclusions on the Use of Design Patterns

For a more complete understanding, the two aforementioned results were joined together to get an overall sense of the main trends in the use of design patterns in frameworks (column "c+m+d"). It cannot be said that the result is dramatically different, but the numbers are adjusted to a certain extent here.

The leading design patterns are *MVC* and *Singleton*, which are used in 9 of 10 frameworks. Next on the list is *Application Controller*, which is used 7 times. *Builder*, *Front Controller, Iterator, Observer, Registry*, and *State* are following with 6 times. *Adapter*, *Decorator*, and *Factory Method* are used 5 times each.

The following design patterns were not used in any projects among the studied: *Abstract Factory*, *Custom Tag*, *Dependency Injection*, *Handle Body*, *Mock Object*, *Mono State*, *Table Data Gateway* and *Transform View*. Apparently the problems which are resolved by these design patterns are not all that important for the designers of frameworks.

The overall view regarding frameworks is quite similar to the previous ones, but there are slightly higher number of design patterns that are used. *Seagull Framework* remains the leader, sharing first position with *Zend Framework*.

The most accurate idea about the use of design patterns in Web application frameworks is seen in the gray-shaded boxes of Table 2, as they indicate that the design pattern is mentioned both in the documentation and in source code.

3.8 Additional Observations

During our basic research, we noticed a few other important issues that have to do with the subject discussed here.

Our first observation is displayed in Table 2 – documentation and the actual software code are often not in line with one another. This does not necessarily mean that most frameworks are poorly documented, but the fact is that in most cases, there is no unambiguous link between the documentation and the source code. What is more, the design patterns that are used probably are cited in the documentation of the developers, but they are not noted in the code. This may occur because the use of a design pattern often involves development of several classes, each of which is well commented. However, the source code does not mention the design pattern which all of the classes have in common.

We already mentioned that documentation can be affected by a marketing trick – the desire to present wishes as facts. It is also possible that the opposite is true – developers do not want their competitors to learn about the most important elements of their project, although they can be studied through the analysis of the source code.

What might it mean?

1. If the basic goal is to ensure understanding of an open source project by dividing it up into logical segments in which design patterns are one of the instruments, then the goal has not been achieved. Documentation and the source code are incomplete and sometimes contradictory.
2. If the developers offer their own frameworks and ask others to trust them, then they should also popularise more standard approaches, i.e., the use of design patterns.

3. More precise use of design patterns would make it easier for new participants to become involved in projects.

Neither should we forget about the psychology of software designers. It is hard to place them in strict frameworks. Each software designer likes to reinvent the bicycle. The use of design patterns is something of a manufacturing line, and the creativity of the software designer can apply only to the details. Open source projects are based on enthusiasm and freedom of participation, and it is quite possible that the developers have greater knowledge about design patterns than is seen in their projects. However, they want to design software without any limitations on what they are doing.

It must also be remembered that large projects involving large or medium-sized groups of developers will inevitably move away from the intended route sooner or later. Over the course of time, people will forget agreements about standards related to coding and preparation of commentary, production of test examples, and procedures related to the submission of code. This creates a certain disorder as well as problems with documentation. Only strict control can prevent this; it is most possible if overall responsibility is taken by a small group of people such as Zend in the case of the "Zend Framework."

4 Conclusions

It may seem that the task of researching the use of design patterns in the design of Web applications is simple, but as soon as the work begins, it becomes clear that the situation is far more complicated than it has been expected. Some frameworks have good homepages, which most framework developers have not really ensured. Hence, finding information about projects can be very complicated or even impossible. Most developers have taken the time to prepare sensible comments in relation to their software codes, but sometimes it is quite hard to understand whether the method has a name because a specific design pattern has been used, or the developer has simply decided on what he or she considers to be the most appropriate name without even thinking about the design pattern.

After collecting all the information, we found that *MVC* and *Singleton* are still the leading design patterns. That is logical because the separation of data, business logic, and visualisation are the basic ideas in designing complicated systems.

Other design patterns which are frequently used include *Application Controller*, *Builder*, *Front Controller, Iterator, Observer, Registry*, and *State*. However, these are not the only design patterns used, which reminds us once again how very diverse the frameworks are – each uses a different set of design patterns which suggests that developers have encountered various problems during the development process.

Further research could focus on drafting the design for a framework which uses as many design patterns as possible, taking into account the experience in the use of design patterns in the surveyed projects. This could show the strength of design patterns in the way that most of the functionality could be described through known design patterns while new design patterns could be developed for the rest. Alternatively, researchers might find out that not everything can be described with the help of design patterns.

Still other alternative might be the attempt to create and study a completely functional open code project such as the *E-store*, which is based on a certain framework. The analysis of such a project might allow researchers to identify the real problems that can be resolved or hindered by the use of design patterns.

When we analysed the use of design patterns, we found it necessary to group all design patterns into logical groups because it is hard to manage a large number of design patterns which each is completely independent; therefore, it is also necessary to indicate the links and interaction among various design patterns.

In conclusion, we must also say that the success of a project does not depend on the use or omission of a design pattern. There are frameworks which use very few design patterns while others use many of them. It can never be claimed that this fact accounts for one framework being better or worse than other.

References

1. Jason E. Sweat. PHP | Architect's Guide to PHP Design Patterns. Marco Tabini & Associates, Inc., 2005
2. Dirk Riehle, Heinz Züllighoven. Understanding and Using Patterns in Software Development. Theory and Practice of Object Systems 2, 1, 1996
3. Richard P. Gabriel. Patterns of Software: Tales from the Software Community. Oxford Univ. Pr., April 1, 1996
4. Christopher Alexander. The Timeless Way of Building. Oxford University Press, 1979
5. Dennis Pallett. Taking a look at ten different PHP frameworks. [online] www.phpit.net [referenced 13.10.2007]. Available online: http://www.phpit.net/article/ten-different-php-frameworks
6. Martin Fowler. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002
7. Design Patterns – Elements of Reusable Object-Oriented Software. Erich Gamma, Richard Helm, Ralph Johnson, John Vissides, Addison Wesley Professional, March 1995
8. http://en.wikipedia.org/wiki/Framework [referenced 30.11.2007]
9. Tony Marston. Design patterns – a personal perspective. [online] [referenced 14.05.2006]. Available online: http://www.tonymarston.net/
10. Brad Appleton. Patterns and Software: Essential Concepts and Terminology. [online] [referenced 15.05.2006]. Available online: http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html

Appendix A: Design Patterns

The description of all design patterns mentioned in this table was taken from book [1].

Design pattern	Description
Abstract Factory	*"Facilitates the building of families of related objects"*
Application Controller	*"A central point for handling navigation for an application, typically implemented in an index.php file dispatching based on URL query parameters."*
Active Record	*"Creates an object that wraps a row from a database table or view, provides database access one row at a time, and encapsulates relevant business logic."*
Adapter	*"Allows classes to support a familiar interface so you can use new classes without refactoring old code."*
Builder	*"Facilitates the initialization of complex object state."*
Command	*"Encapsulates a request as an object."*
Composite	*"Manages a collection of objects where each "part" can stand in as a "whole". Typically organized in a tree hierarchy. "*
Custom Tag	*"Improves presentation separation by encapsulating components to appear as new HTML tags."*
Data Mapper	*„An object that acts as a translation layer between domain objects and the database table that contains related data."*
Decorator	*"Attaches responsibilities to an object dynamically. Can simplify class hierarchies by replacing subclasses."*
Dependency Injection	*"Constructs classes to accept collaborators through the constructor or setter methods, so that a framework can assemble your objects."*
Domain Model	*"An object model of business logic that includes both data and behavior."*
Factory Method	*"Facilitates the creation of objects."*

Front Controller	*"A controller that handles all requests for a web application."*
Handle-Body	*"A collective name for design patterns that hold a reference to a subject object (for example, Proxy, Decorator, and Adapter)."*
Iterator	*"Easily manipulates collections of objects."*
MockObject	*"Supplies a stub that validates whether certain methods were or were not called during testing."*
Model-View-Controller	*"An application layering pattern that separates concerns between your domain model, presentation logic and application flow.* *The Model-View-Controller (MVC) pattern organizes and separates your software into three distinct roles:* *• the Model encapsulates your application data, application flow, and business logic;* *• the View extracts data from the Model and formats it for presentation;* *• the Controller directs application flow and receives input and translates it for the Model and View."*
Mono State	*"Allows all instances of an object to share the same state."*
Observer	*"Registers objects for later callback. Event-based notification. Publish/Subscribe."*
Proxy	*"Provides access to an object through a surrogate object to allow for delayed instantiation or protection of subject methods."*
Registry	*"Manages references to objects through a single, well-known object."*
Server Stub	*"Simulates a portion of your application for testing purposes."*
Singleton	*"Provides global access to a single instance of an object."*

Specification	*"Flexible evaluation of objects against dynamic criteria."*
State	*"Has an object change its behaviour depending on state changes."*
Strategy	*"Allows for switching between a selection of algorithms by creating objects with identical interfaces."*
Table Data Gateway	*"An object that acts as a gateway to a database table or view, providing provide access to multiple rows."*
Template View	*"Renders a page by replacing embedded markers with domain data."*
Template Method	*"Defines an algorithm with "hook" methods allowing subclasses to change the behavior without changing the structure."*
Transform View	*"Processes domain data sequentially to transform it to some form of output."*
ValueObject	*"Handles objects whose equality is determined by the value of the objects' attributes, not by the identity of the objects."*
View Helper	*"A class that helps the View by collecting data from the Model."*
Visitor	*"Defines an algorithm as an object that "visits" each member of a aggregate performing an operation."*

Appendix B: PHP Frameworks

Descriptions of the frameworks are taken from their own homepages.

Zend Framework

Extending the art & spirit of PHP, Zend Framework is based on simplicity, object-oriented best practices, corporate friendly licensing, and a rigorously tested agile codebase. Zend Framework is focused on building more secure, reliable, and modern Web 2.0 applications & web services, and consuming widely available APIs from leading vendors like Google, Amazon, Yahoo!, Flickr, as well as API providers and cataloguers like StrikeIron and ProgrammableWeb.

Expanding on these core themes, we have implemented Zend Framework to embody extreme simplicity & productivity, the latest Web 2.0 features, simple

corporate-friendly licensing, and an agile well-tested code base that your enterprise can depend upon.

http://framework.zend.com [referenced 03.11.2007]

CakePHP

Cake is a rapid development framework for PHP which uses commonly known design patterns like ActiveRecord, Association Data Mapping, Front Controller and MVC. Our primary goal is to provide a structured framework that enables PHP users at all levels to rapidly develop robust web applications, without any loss to flexibility.

http://www.cakephp.org/ [referenced 03.11.2007]

Symfony

Based on the best practices of web development, thoroughly tried on several active websites, symfony aims to speed up the creation and maintenance of web applications, and to replace the repetitive coding tasks by power, control and pleasure.

http://www.symfony-project.org/ [referenced 03.11.2007]

Seagull

Seagull is a mature OOP framework for building web, command line and GUI applications. Licensed under BSD, the project allows PHP developers to easily integrate and manage code resources, and build complex applications quickly.

Many popular PHP applications are already seamlessly integrated within the project, as are various templating engines, testing tools and managed library code. If you're a beginner, the framework provides a number of sample applications that can be customised and extended to suit your needs. If you're an intermediate or advanced developer, take advantage of Seagull's best practices, standards and modular codebase to build your applications in record time.

http://seagullproject.org/ [referenced 03.11.2007]

Wact

The Web Application Component Toolkit is a framework for creating web applications. WACT facilitates a modular approach where individual, independent or reusable components may be integrated into a larger web application. WACT assists in implementing the Model View Controller pattern and the related Domain Model, Template View, Front Controller and Application Controller patterns.

The WACT framework is developed with the philosophy of continuous refactoring and Unit Testing. WACT encourages these activities in applications based on the framework. WACT uses Simple Test as a unit testing framework.

http://www.phpwact.org/ [referenced 03.11.2007]

Prado

PRADO is a component-based and event-driven framework for rapid Web programming in PHP 5. PRADO reconceptualizes Web application development in terms of components, events and properties instead of procedures, URLs and query parameters.

http://www.xisc.com/ [referenced 03.11.2007]

PHP On Trax

Php On Trax (formerly Php On Rails) is a web-application and persistance framework that is based on Ruby on Rails and includes everything needed to create database-backed web-applications according to the Model-View-Control pattern of separation. This pattern splits the view (also called the presentation) into "dumb" templates that are primarily responsible for inserting pre-build data in between HTML tags. The model contains the "smart" domain objects (such as Account, Product, Person, Post) that holds all the business logic and knows how to persist themselves to a database. The controller handles the incoming requests (such as Save New Account, Update Product, Show Post) by manipulating the model and directing data to the view.

In Trax, the model is handled by what's called a object-relational mapping layer entitled Active Record. This layer allows you to present the data from database rows as objects and embellish these data objects with business logic methods.

http://www.phpontrax.com/ [referenced 03.11.2007]

ZOOP Framework

Zoop is an object oriented PHP framework. Zoop is modeled after the MVC design pattern. It is a high performance, secure, and scalable framework for PHP. It is designed to be very fast and efficient and very nice for the programmer to work with. Zoop has been built in a modular way so it is both easily extensible, and light. It has been in development and production use since 2001 and is quite mature.

http://zoopframework.com/ [referenced 03.11.2007]

eZ Components

eZ Components is an enterprise ready general purpose PHP components library used independently or together for PHP application development.

http://ez.no/ezcomponents [referenced 03.11.2007]

CodeIgniter

CodeIgniter is a powerful PHP framework with a very small footprint, built for PHP coders who need a simple and elegant toolkit to create full-featured web applications. If you're a developer who lives in the real world of shared hosting accounts and clients with deadlines, and if you're tired of ponderously large and thoroughly undocumented frameworks.

http://www.codeigniter.com/ [referenced 03.11.2007]