Approaches to Increasing the Quality and Reliability in the Field of Design Patterns

Dimitrichka Zheleva Nikolaeva and Violeta Todorova Bozhikova

Abstract – **The report presents two techniques in software development based on Design Patterns: programming, providing reliable functioning of the programming systems (PPRF) and reuse of software production (RSP). These are the findings of an analysis made in the field of Design Patterns. The report explores six indicators: the actuality of the topic, analysis of the approaches and methods for the development of Design Patterns, determining the advantages and disadvantages of implementing Design Patterns, study of Design Patterns according to their use and according to the purpose of software development, analysis of the programming languages used to develop Software Templates. They are important for the development of Software Based on Software Templates. In conclusion the advantages of using Design Patterns are summarized and an application is made for future projects.**

Keywords – **Software Design Patterns, Software Re-Use, Software Quality**

I. Introduction

Over the past 80 years, information technologies have been developing rapidly. Programming systems and products have become increasingly complex and this is a prerequisite for creating of new efficient technologies for development. In the seventies of the last century the development of software based on templates arise. [20] They have a concept for solving common problems in the field of object-oriented modeling. They are designed to provide standardized and efficient solutions for architectural and conceptual problems in computer programming.

The report consists of four parts. The second provides analysis, which includes a study of six indicators: the actuality of the topic, analysis of the approaches and methods for the development of Design Patterns, determining the advantages and disadvantages of implementing Design Patterns, study of Design Patterns according to their use and according to the purpose of software development, analysis of the programming languages used to develop Software Templates. The results are summarized graphically and tabularly. The third part discusses the techniques used in software development based on Software Template. The conclusion supports the need to study and implement approaches in the field of Design Patterns for improving the quality and reliability of software systems and products.

D. Nikolaeva is with the Department of Computer Science and Technologies, Faculty of Computer Technique and Automation, Technical University - Varna, 1 Studentska str., 9010 Varna, Bulgaria, e-mail: dima.nikolaeva@abv.bg

V. Bozhikova is with the Department of Computer Science and Technologies, Faculty of Computer Technique and Automation, Technical University - Varna, 1 Studentska str., 9010 Varna, Bulgaria, e-mail: e-mail: vbojikova2000@yahoo.com

II. Analysis of the Literature Linked to the Software Patterns Design.

The report presents the results of ongoing research of the first author in the field of "Software development using design patterns." Until now, 107 literature sources have been analyzed, the most important of which are cited in this publication. The large number of publications on the subject clearly demonstrates the topicality of the subject (Fig.1. Grouping of literature and Fig.2 Literature issued/ published in the period 1993-2014 in field of Design Patterns).

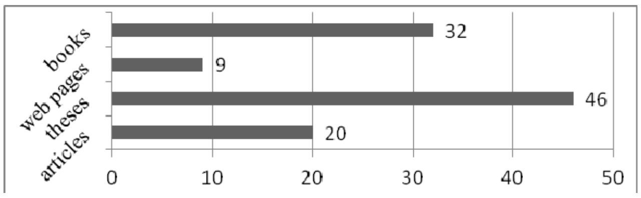


Fig.1. Grouping of literature

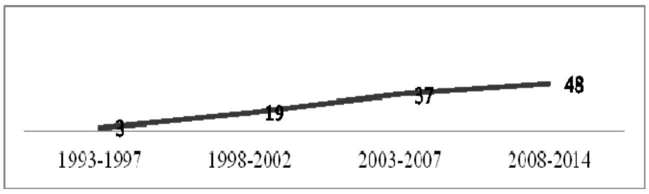


Fig.2. Literature issued/ published in the period 1993-2014 in field of Design Patterns

Apart from, confirming the actuality of the topic, the indicators subject of research aim to:
1) Analysis of approaches and methods for the development of Design Patterns;
2) Determining the advantages and disadvantages of implementing Design Patterns;
3) Study of Design Patterns according to their use;
4) Study of Design Patterns according to the purpose of software development;
5) Analysis of the programming languages used to develop Software Templates.

The conclusions made are important for the development of Software Based on Software Templates. The results of the analysis are presented in table and graphically as follows:

TABLE 1.SUMMARY ANALYSIS OF DESIGN PATTERNS

1) Analysis of approaches and methods for the development of Design Pattern

- Strategies for tolerating software errors.
- A structural approach to the assessment of the design of model-oriented and object-oriented projects [1]
- Parametric Approach and approach check
- Creating a methodology for reuse, storage and application of code
- Development of components for reuse (componentization) [2]
- Development of libraries for re-use and easy configuration [10]
- Methodology for building dynamic binding of components [16]
- An approach for modeling complex business domains [5]
- An approach for detecting design patterns to support reverse engineering [4], [6], [9]
- A method of creating software models, leading to repeatedly used and cost-efficient software, the basis for this process are the so called mind maps
- An approach to develop Detector clones as a prefix to the Eclipse IDE

2) Determining the advantages and disadvantages of implementing Design Patterns

Advantages

- Lead to automating applications[1]
- Create and maintain complex, large-scale, flexible systems [17]
- Lead to re-use and develop components and libraries for re-use
- Simplify design and optimize code
- Develop a high structural level
- Increase the level of conceptual thinking [19]
- Lead to better decisions
- Create a common basis for comparison and detection of Design Patterns [11]
- Applicable (give a quick access to the database, help to develop games, etc.).
- Used for exchange of experience (help programmers, designers, architects and analysts for the successful use of Design Patterns in combination with a wide range of programming languages)
- Lead to the integration of the Object-Oriented, Event-Based and Aspect-Oriented Programming

Disadvantages

- The architectures of some models are not optimal
- It is not clear which model should be applied in which situations
- Novice designers err in their application
- The documentation describing the patterns is not accurate, leading to different interpretations by the designers who use it
- Lack adequate functionality, which limits the use of design patterns within a session
- Similar code segments that appear in the source code, increase both the productivity and the probability of error propagation

3) Study of the Design Patterns according to their application (covering all literary sources):

A. total (for all Design Patterns) and

B. in groups according to the purpose that the relevant Design Patterns (**B.1.** total and **B.2.** in percent)

3)A.

Design Patterns (DP)	Use of DP	DP according to purpose
Memento	23	Behavioral patterns
Flyweight	24	Structural patterns
Iterator	28	Behavioral patterns
Chain of Responsibility	29	Behavioral patterns
Interpreter	29	Behavioral patterns
Builder	31	Creational patterns
Visitor	32	Behavioral patterns
Prototype	33	Creational patterns
Bridge	35	Structural patterns
Decorator	35	Structural patterns
Mediator	35	Behavioral patterns
Façade	36	Structural patterns
State	38	Behavioral patterns
Command	40	Behavioral patterns
Template method	40	Behavioral patterns
Composite	42	Structural patterns
Adapter	43	Structural patterns
Proxy	43	Structural patterns
Abstract Factory	46	Creational patterns
Observer	47	Behavioral patterns
Singleton	48	Creational patterns
Factory method	49	Creational patterns
Strategy	49	Behavioral patterns

DP according to the purpose	**3) B.1.**	**3) B.2.**
Behavioral patterns	390	46 %
Structural patterns	258	30 %
Creational patterns	207	24 %

4) Study of the Design Patterns according to the purpose of software development (only theses discussed):

A. Delivering reliable operation; [7], [14], [16]
B. Achieving quality; [8], [11], [12]
C. Reuse of software production; [3]
D. Evaluation of the final software [19]

5) Analysis of programming languages for the development of Software Templates:

A. (Java);
B. (C++);
C. (C#);
D. (Xml);
E. (Other)

Design Patterns	**4)**				**5)**				
	A	B	C	D	A	B	C	D	E
Abstract Factory	1	1	6	2	26	13	21	5	3
Factory method	2	1	7	1	27	9	22	9	7
Builder	1	0	4	3	18	7	14	4	4
Prototype	1	1	3	1	16	7	15	5	1
Singleton	1	2	8	2	23	12	19	9	7
Adapter	2	1	7	1	23	13	19	6	6
Bridge	1	2	4	1	16	11	17	6	2
Composite	2	1	6	2	25	11	21	7	4
Decorator	1	1	5	1	22	11	19	4	2
Façade	1	1	6	1	20	10	17	6	4
Flyweight	0	1	4	1	13	8	12	5	2
Proxy	1	1	6	1	23	11	17	6	5
Chain of Responsibility	0	1	4	1	18	10	15	3	1
Command	2	1	7	2	25	13	15	6	5
Interpreter	1	1	3	1	17	6	13	4	2
Iterator	0	0	5	1	15	8	12	3	2
Mediator	0	2	5	2	22	11	15	6	3
Memento	1	1	3	1	13	7	11	3	1
Observer	1	1	6	3	27	13	21	7	3
State	1	1	7	1	20	10	15	5	2
Strategy	2	2	6	2	26	15	24	10	4
Template method	1	1	7	1	24	10	17	5	3
Visitor	0	2	4	1	16	11	15	5	1

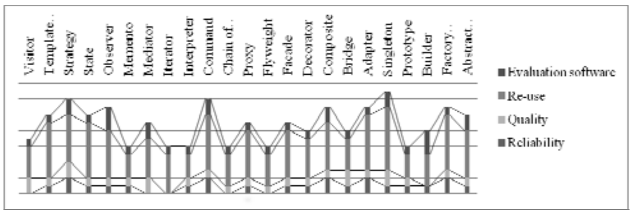


Fig.3. Use according to the purpose of the software development

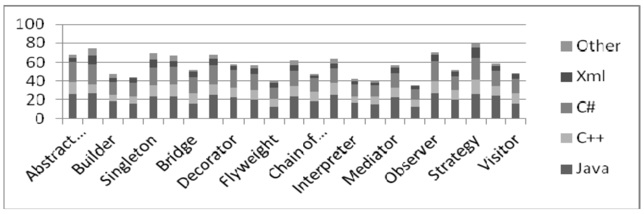


Fig.4. Use of programming languages

The results are important for the development of an approach based on the Design Patterns. After summarizing the data several conclusions can be made:

1) The issue of software templates is up to date, due to the increased interest in recent years.

2) Software Based on Software Templates is practically-oriented and despite its shortcomings, the software templates are preferred to create reliability and quality software. [10]

3) Software Templates allow for re-use, reducing the time and money to develop. Through software programming templates automate labor.

4) The most used by Software Templates: Strategy, Factory method and Singleton, but at least are: Memento, Flyweight and Iterator. The most used group of Design Patterns: Behavioral patterns - 46%, followed by Structural patterns - 30%, and the smallest is the percentage of Creational patterns - 24%.

5) Java is the most used programming language for creating Factory method and Observer. Programming languages: C++, C # and Xml are most often used to create Strategy. The language most used to create applications with Design Patterns in Java, followed by C # and C ++, the least used is Xml.

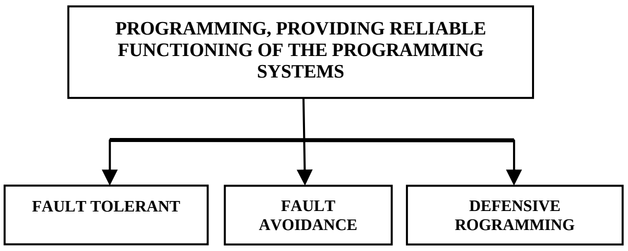
From the conclusions in 2) and 3) we can conclude that there are two main techniques for developing software-based software templates: programming, providing reliable functioning of the programming systems (PPRF) [1], [4], [6], [9] and re-use in the software production (RSP) [2], [5], [10], [16].

III. Techniques Used in Software Development Base Design Patterns

The development of quality software is associated with construction and validation processes. Construction processes are linked to Fault avoidance (prevention): avoid defects and Fault tolerance: software development with an acceptable level of error. The validation processes are related to the validation of software created and accordingly to: Fault removal: detection and elimination of errors and Fault / failure forecasting giving a forecast. Such processes also take place in the creation a Software Based on Software Templates.

The technique PPRF, used for developing software-based Software Templates is associated with an assessment of: Reliability - frequency of system failures; the collapse of the system - working situation unusual response software; defect - a programming error in the input data leading to collapse. PPRF affects three modern approaches to develop Software Templates (Fig.5):

- ✓ Development of software to minimize the defects in it (Fault avoidance);
- ✓ Software development with an acceptable level of error (Fault tolerance);
- ✓ Defensive programming.



PROGRAMMING, PROVIDING RELIABLE FUNCTIONING OF THE PROGRAMMING SYSTEMS
FAULT TOLERANT
FAULT AVOIDANCE
DEFENSIVE ROGRAMMING

Fig.5. Approaches used in the technique PPRF

The essence of the first approach (Fault avoidance) is to reduce errors in software development. Fault tolerance is software that is designed to keep working even if there are errors. Protective programming (Defensive programming) is an approach that incorporates mechanisms to detect, assess and eliminate potential errors. Validation processes: Fault / failure forecasting and Fault removal are features of Fault tolerance, while Fault removal is a function of Defensive programming, due to the specifics in the design of software. Fault tolerance principles are: Reliable operation - the principle of repetition of elements; a different number of components can be applied for the same activity each one of them performing certain functions, the results are compared and the program continues with the frequent result. The principle is also known as N-version programming in software results; Recovery Blocks - software units comprising an alternative to re-code execution failover; Exception handlers - components for handling exceptions /a message to suspend the process. Features Fault tolerance: Forecast and detect defects (Fault / failure forecasting and Fault removal); Assess the damage after a system crash; Disaster recovery; Locate and remove defects causing the collapse of the system. Principles of Fault-free software are: Create precise specifications; Use of a design allowing the encapsulation of the information; Mechanisms for assuring quality software; Planning and system testing.

The second technique RSP is a combination of planned and systematic activities aimed at using existing software components. Their practical implementation would be successful if they met certain requirements: program components must be designed so that they are readily adjustable to consider the cloning process to regulate the mechanism for (establishing the variable) compiling the names of the change to prevent errors, components should be portable. As a result of the analysis in Table 1**.**Summary analysis of Design Patterns we can conclude that in the second RSP technology to develop software based on the Software Templates (reuse) four groups of trends have be registered:

- ✓ Design of libraries of components for re-use;
- ✓ Create reusable components;
- ✓ Establishment of standards for integrated library usage of the elements for reuse;
- ✓ Create models for the process of re-use and appropriate software tools that support the development of and / or reuse.

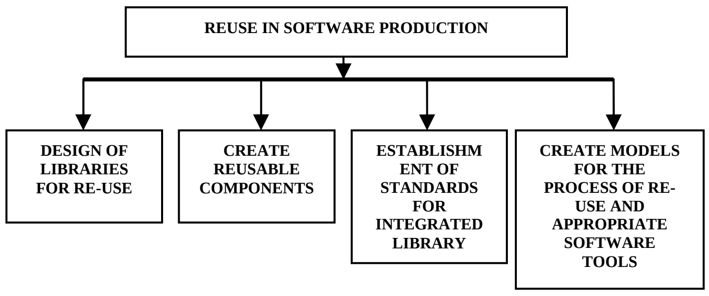

REUSE IN SOFTWARE PRODUCTION
DESIGN OF LIBRARIES FOR RE-USE
CREATE REUSABLE COMPONENTS
ESTABLISHMENT OF STANDARDS FOR INTEGRATED LIBRARY
CREATE MODELS FOR THE PROCESS OF RE-USE AND APPROPRIATE SOFTWARE TOOLS

Fig.6. Trends used in RSP technique

The reuse of existing software components has several advantages: increasing the productivity of programmers work to improve the quality of the developed software, quickly create new products as it reduces development time. The success of this approach is due to the techniques of storing and retrieving components. Search mechanisms, are usually associated with a keyword search, text descriptions of natural language faceted technique by drawing descriptions from a different perspective. The principle of operation of the techniques is different, but what unites them is the ability to improve the quality and reliability of software. In PPRF technique this is achieved through different approaches to reduce software errors, while in RSP technique re-use of software components is implemented whose properties are already checked. The choice of approach depends on the application domain and the specific requirements for the created software.

IV. Conclusion and Future Progress

The report summarizes the techniques for developing software based on Design Patterns (PPRF and RSP). For this purpose, a large amount of literature in the field of Software Design Patterns has been examined. Multi criteria analysis has been conducted and results interesting for further study have been received and presented. It must be emphasized that the use of Software Design Patterns on the one hand saves time and resources for development, increasing productivity of programmer’s work and on the other it leads to minimizing programming errors both resulting in quality and reliability of the developed software. Our research in the field of Design Patterns continues. The idea is to explore and develop an approach to software development based on a hybrid software template. The creation of such an approach is related to solving a number of research problems, both methodological and practical.

References

[1] Yacoub, S.M., *Pattern-Oriented Analysis and Design (POAD): A Methodology for Software Development* - Department of Computer Science and Electrical Engineering, West Virginia University Morgantown, 1999

[2] Althammer, Egb., *Reflection Patterns in the Context of Object and Component Technology* - der Universität Konstanz, Mathematisch-Naturwissenschaftliche Sektion, Fachbereich Informatik und Informationswissenschaft Konstanz, 2001

[3] Jamal, S., *Pattern-Based Approach for Object Oriented Software Design* - Department of Computer Science K.U.Leuven, Leuven, Belgium, August 18, 2003

[4] Maggioni, St., *Design Pattern Detection and Software Architecture Reconstruction: an Integrated Approach based on Software Micro-structures*, Dipartimento di Informatica, Sistemistica e Comunicazione, Universita degli Studi di Milano-Bicocca, 2008

[5] Hoftmann, Kl. B., *Domain-driven design in action*, Designing an identity provider, Department of Computer Science, University of Copenhagen, spring 2009

[6] Rasool, G., *Customizable Feature based Design Pattern Recognition Integrating Multiple Techniques,* Doctoral dissertation Fakultät für Informatik und Automatisierung Technische Universität Ilmenau, October, 2010

[7] Qi, X., *Language Support For Reliable, Extensible Large-Scale Software Systems*, Doctoral dissertation, Cornell University, 2010

[8] Abramov, J., *A Patten Based Approach for Design and Implementation of Secure Databases*, Doctoral dissertation, BEN-GURION UNIVERSITY OF THE NEGEV, November, 2010

[9] Dong, J., Zhao, Y., & Peng, T., A review of design pattern mining techniques. *International Journal of Software Engineering and Knowledge Engineering*, *19*(06), 823-855, 2009

[10] BYNENS, M., A System of Patterns for the Design of Reusable Aspect Libraries, Doctoral dissertation, © Katholieke Universiteit Leuven – Faculty of Engineering, Belgium, 2011

[11] Binun, A., *High Accuracy Design Pattern Detection,* Doctoral dissertation, Universitäts-und Landesbibliothek Bonn, 2012

[12] Ferenc, R., *Advances in Software Product Quality Measurement and its Applications in Software Evolution,* Doctoral dissertation, University of Szeged, 2014

[13] Ebnenasir, A., & Cheng, B. H. *A Pattern-Based Approach for Modeling and Analysis of Error Recovery*, 2007

[14] Pawson, R., *Naked objects*, Doctoral dissertation, Trinity College, Dublin June 2004

[15] Zibran, M. F., *Management Aspects of Software Clone Detection,* Doctoral dissertation, University of Saskatchewan Saskatoon, June 2014

[16] Rasool, G., A. I., & At. M., *A Comparative Study on Results of Design Patterns Recovery Tools World Applied Sciences Journal*, *28*(9), 2013

[17] Vernon, V., *Implementing domain-driven design*. Addison-Wesley, 2013

[18] Khwaja, S. A., *Towards Design Pattern Definition Language (DPDL),* Doctoral dissertation, King Fahd University of Petroleum and Minerals, 2010

[19] Trad, A., & Trad, C., *Audit, Control and Monitoring Design Patterns (ACMDP) for Autonomous Robust Systems (ARS)*, *International Journal of Advanced Robotic Systems*, *2*(1), 25-38, 2005

[20] Erich Gamma et al, Design Patterns: Elements of Reusable Object-Oriented Software, ISBN: 0201633612, Addison-Wesley Publ. Co., January 15, 1995