International Journal of Business, Human and Social Sciences
ISSN: 2517-9411
Vol:2, No:9, 2008

A Study on using N-Pattern Chains of Design Patterns based on Software Quality Metrics

Niloofar Khedri, Masoud Rahgozar, and MahmoudReza Hashemi

***Abstract*—**Design patterns describe good solutions to common and reoccurring problems in program design. Applying design patterns in software design and implementation have significant effects on software quality metrics such as flexibility, usability, reusability, scalability and robustness. There is no standard rule for using design patterns. There are some situations that a pattern is applied for a specific problem and this pattern uses another pattern. In this paper, we study the effect of using chain of patterns on software quality metrics.

***Keywords*—**Design Patterns, Design patterns' Relationship, Software quality Metrics, Software Engineering.

I. Introduction

TODAY, design patterns are wildly used in various software domains such as design, implementation, development, test and reengineering. Design patterns [12] are high level building blocks that promote elegance in software by ordering proven and timeless solutions to common problems in software design. Design patterns convey the experience of software designers. Applying design patterns in software design and implementation have important effects on software quality metrics such as flexibility, usability, reusability, scalability and robustness [5]. In [8] design patterns' relationship are classified in 6 categories and then a new way for applying patterns in the software system is given. As we know studying software characteristics in the software design is an essential content but no consideration on applying patterns based on their expected software metrics in [5] has been studied yet.

In this paper, we investigate the situations in applying design patterns where the first pattern uses the second one and the second pattern uses the third one. Also we investigate the situations where the first pattern uses the second one.

Firstly we talk about the quality metrics which design patterns are expected to bring. Secondly, we classify the design patterns based on their relationships. Then we study the "use" relationship and compare their design patterns' software quality metrics.

N. Khedri is with the Database Research Group, faculty of ECE, School of Engineering, University of Tehran, Tehran, Iran (e-mail: n.khedri@ece.ut.ac.ir, niloofar_khedri@yahoo.com).

M. Rahgozar is with the Control and Intelligence Processing Center of Excellence, Faculty of ECE, School of Engineering, University of Tehran, Tehran, Iran (e-mail: rahgozar@ut.ac.ir).

M. R. Hashemi is with the Database Research Group, Faculty of ECE, School of Engineering, University of Tehran, Tehran, Iran (e-mail: hashemi@comnete.com).

II. Software Quality Characteristics of Design Patterns

A. Quality Characteristics related with Design Patterns

Design Patterns are solutions for reoccurring problems, applying design patterns in software design and implementation have effect on software quality metrics such as flexibility, usability, reusability, scalability and robustness. Gamma et al. in "Design Patterns: Elements of Reusable Object-Oriented Software" [12], define design patterns as: "Patterns specify design problems and make object-oriented more flexible, elegant and ultimately reusable" and Design patterns help you chose design alternatives that make a system reusable and avoid alternatives that compromise reusability.

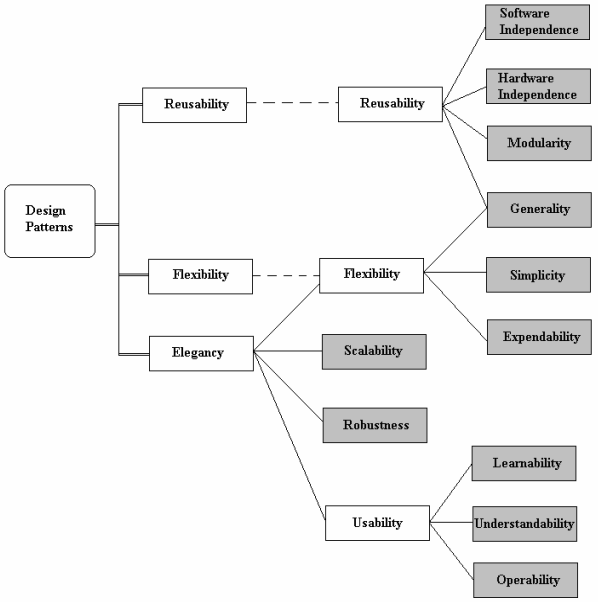

Fig. 1 Design Patterns and their Software Quality Characteristics

Software elegancy is defined as maximizing the information delivered through the simplest possible interface. When considering these definitions, design patterns are expected to bring:

- Flexibility: "Effort required modifying an operational program" [4].
- Elegancy: Issues of elegance in software are reflected to robustness, scalability, flexibility, and usability.

- Robustness: "Robustness is the degree to which an executable work product continues to function properly under abnormal conditions or circumstances" [2]. Also, the attributes related to the correct functioning of a software product in the case of invalid inputs or under stressful environmental conditions [10].
- Scalability: "Scalability is the ease with which an application or component can be modified to expend its existing capacities" [2], [9], [6].
- Flexibility: "Effort required modifying an operational program" [4].
- Usability: "The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions" [3], [2], [11].
- Reusability: "Reusability is the ease with which an existing application or component can be reused" [2], [7].

So design patterns are expected to increase the following quality characteristics: Flexibility, Reusability, Robustness, Scalability, and Usability.

Flexibility consists of the following quality characteristics:
- Expendability: "The degree to which architectural, data or procedural design can be extended" [11].
- Generality: "The breadth of potential application of program components" [11].
- Modularity: "The functional independence of program components" [11].

Reusability consists of the following quality characteristics:
- Generality
- Hardware independence: "The degree to which the software is decoupled from the hardware on which it operates" [11].
- Modularity
- Software system independence: "The degree to which the program is independent of nonstandard programming language features, operating system characteristics, and other environmental constraints" [11].

Usability consists of the following quality characteristics:
- Learnability: "The capability of the software product to enable the user to learn its application" [3], [1].
- Operability: "The capability of the software product to enable the user to operate and control it" [3]. Also, the ease of operation of a program" [11], [2], [1].
- Understandability: "The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use" [3].

Fig. 1 shows the main characteristics and sub characteristics of the design patterns quality.

B. Quality Evaluation of Design Patterns

Khosravi and Gueheneuc in "A Quality Model for Design Patterns" [5] studied design patterns and evaluated manually their quality characteristics using five-levels scale (Excellent, Good, Fair, Bad and Very bad. Also, they used N/A for characteristics not applicable to some design patterns [5].

III. Effect of Chain of Patterns on Software Quality Characteristics

A. Classification of Design Patterns Relationships

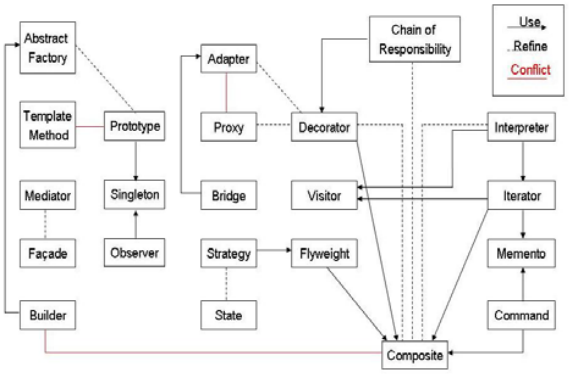

Fig. 2 Graphical illustration of patterns relationship [8]

According to [8] design patterns relationships can be classified through 6 categories:

1. Use: A pattern uses another pattern.
2. Refine: A more specific pattern refines a more general and abstract pattern.
3. Conflict: One pattern conflicts with another pattern when they both provide mutually exclusive solutions to similar problems.
4. Similar: This relationship is often used to describe patterns which are similar because they address the same problem. The similarity relationship seems to be much broader than just conflicts and as it is also used to describe patterns which have a similar solution technique such as Strategy and State.
5. Combine: Two patterns are combining to solve a single problem.
6. Require: One pattern requires a second pattern if the second pattern is a prerequisite for solving the problem addressed by the first pattern.

In software system, during the applying design patterns to a specific problem, there are some cases to apply patterns that use another pattern in its implementation.

In next part, pattern chains in which first pattern used the second one are considered to study the effect of applying chain of patterns.

B. Use Relationship and Software Metrics

In the "use" relationship a pattern uses another pattern. Fig. 2 shows the graphical illustration of patterns. Also it shows that the use relationship has the most frequency among patterns relationships. The largest chain of patterns has 3 patterns. In this part firstly, the chains of patterns that have 3 patterns are investigated and secondly the chains of patterns that have 2 patterns are investigated.

1. Three-Pattern Chains and Software Quality Metrics

There are six usage chains of patterns that have 3 patterns as listed below:

1. Builder → Abstract Factory → Template Method
2. Chain of Responsibility → Decorator → Composite
3. Interpreter → Iterator → Composite
4. Interpreter → Iterator → Visitor
5. Interpreter → Iterator → Memento
6. Strategy → Flyweight → Composite

There is a question here: is it useful to use the three-pattern chain, according to software metrics? We assume that if the first pattern of a chain is ranked "good "and the second one is ranked "fair", the chain of these two patterns is ranked "fair". The rank of the chain is always set to the lowest rank of the corresponding patterns; it means that, if one of the chain patterns is ranked N/A (not Applicable) the chain rank is set to the N/A. The quality characteristics of the 6 three-pattern chains are shown in Table I and Table II. According to Table II we can not achieve software independence and hardware independence by applying chain of patterns.

- Chain 1 (Builder → Abstract Factory → Template Method) is the only chain that has the best result in quality characteristics; but also it is the only chain in which modularity is not applicable. Chain

TABLE I
DETAILS OF QUALITY CHARACTERISTICS IN THREE-PATTERN-SIZED CHAINS

Cycle No	Design Patterns	Expendability	Simplicity	Generality	Modularity	Software Independence	Hardware Independence	Learnability	Understandability	Operability	Scalability	Robustness
1	Builder	Good	Good	Fair	Fair	N/A	N/A	Fair	Good	Fair	Good	Good
1	Abstract Factory	Excellent	Excellent	Good	Good	Fair	Fair	Good	Good	Good	Good	Good
1	Template Method	Excellent	Good	Fair	N/A	N/A	N/A	Good	Good	Good	Good	Good
1	Result	Good	Good	Fair	N/A	N/A	N/A	Fair	Good	Fair	Good	Good
2	Chain of Responsibility	Good	Good	Good	Bad	N/A	N/A	Fair	Fair	Good	Bad	Fair
2	Decorator	Excellent	Excellent	Good	Fair	Good	N/A	Good	Good	Good	Good	Fair
2	Composite	Fair	Fair	N/A	Fair	N/A	N/A	Fair	Good	N/A	N/A	Good
2	Result	Fair	Fair	N/A	Bad	N/A	N/A	Fair	Fair	N/A	N/A	Fair
3	Strategy	Good	Fair	Bad	Fair	Fair	N/A	Bad	Bad	Fair	Bad	Fair
3	Flyweight	Bad	Bad	Fair	Good	N/A	N/A	Good	Bad	Fair	Good	Good
3	Composite	Fair	Fair	N/A	Fair	N/A	N/A	Fair	Good	N/A	N/A	Good
3	Result	Bad	Bad	N/A	Fair	N/A	N/A	Bad	Bad	N/A	N/A	Fair
4	Interpreter	Good	Fair	Good	Fair	N/A	N/A	Fair	Fair	Good	Good	Fair
4	Iterator	Excellent	Excellent	Good	Fair	Good	N/A	Good	Fair	Fair	Good	Good
4	Composite	Fair	Fair	N/A	Fair	N/A	N/A	Fair	Good	N/A	N/A	Good
4	Result	Fair	Fair	N/A	Fair	N/A	N/A	Fair	Fair	N/A	N/A	Fair
5	Interpreter	Good	Fair	Good	Fair	N/A	N/A	Fair	Fair	Good	Good	Fair
5	Iterator	Excellent	Excellent	Good	Fair	Good	N/A	Good	Fair	Fair	Good	Good
5	Memento	Good	Fair	Fair	Very Bad	N/A	N/A	Bad	Fair	Good	Fair	Bad
5	Result	Good	Fair	Fair	Very Bad	N/A	N/A	Bad	Fair	Fair	Fair	Bad
6	Interpreter	Good	Fair	Good	Fair	N/A	N/A	Fair	Fair	Good	Good	Fair
6	Iterator	Excellent	Excellent	Good	Fair	Good	N/A	Good	Fair	Fair	Good	Good
6	Visitor	Excellent	Good	Good	Fair	Good	N/A	Good	Bad	Fair	Good	Fair
6	Result	Good	Fair	Good	Fair	N/A	N/A	Fair	Bad	Fair	Good	Fair

TABLE II
QUALITY CHARACTERISTICS IN THREE-PATTERN-SIZED CHAINS

Cycle No	Expendability	Simplicity	Generality	Modularity	Software Independence	Hardware Independence	Learnability	Understandability	Operability	Scalability	Robustness
1	Good	Good	Fair	N/A	N/A	N/A	Fair	Good	Fair	Good	Good
2	Fair	Fair	N/A	Bad	N/A	N/A	Fair	Fair	N/A	N/A	Fair
3	Bad	Bad	N/A	Fair	N/A	N/A	Bad	Bad	N/A	N/A	Fair
4	Fair	Fair	N/A	Fair	N/A	N/A	Fair	Fair	N/A	N/A	Fair
5	Good	Fair	Fair	Very Bad	N/A	N/A	Bad	Fair	Fair	Fair	Bad
6	Good	Fair	Good	Fair	N/A	N/A	Fair	Bad	Fair	Good	Fair

can be ranked good for Expendability, Simplicity, Understandability, Scalability and Robustness. The three remain quality characteristics may be ranked fair. This chain has good results to gain Flexibility.

- Chain 2 (Chain of Responsibility → Decorator → Composite) Generality, Software Independence, Hardware Independence, Operability, and Scalability are not applicable.
- Chain 3 (Interpreter → Iterator → Composite) and chain 4 (Interpreter → Iterator → Visitor) are not rank good.
- Chain 5 (Interpreter → Iterator → Memento) promotes only Expendability and we can not achieve Modularity, Learnability and Robustness.
- Chain 6 (Strategy → Flyweight→ Composite) has the better results than chain 2, 3, 4 and 5. Expendability , Generality and Scalability can be ranked "good"; but we can not achieve Understandability. This chain has good results to gain Flexibility.

It seems that more studies are required to analyze chain 1 and chain 6.

2. Two-Pattern Chains and Software Quality Metrics

We assume that if the first pattern of a chain is ranked "good "and the second one is ranked "fair", the chain of these two patterns is ranked "fair". The rank of the chain is always set to the lowest rank of the corresponding patterns; it means that, if one of the chain patterns is ranked N/A (not Applicable) the chain rank is set to the N/A. The quality characteristics of the 16 two-pattern chains are shown in Table IV and Table V. According to Table IV we can not achieve hardware independence by applying chain of patterns.

It is important that we can achieve software independence only in cycle 13 (Iterator → Visitor)

We can not gain Generality, Operability and Scalability in cycles 5, 7, 8 and 11 which have Composite patterns.

- Chain 1 (Abstract Factory → Template Method) is the only chain that has the best result in quality characteristics. Chain can be ranked good for Simplicity, Learnability, Understandability, Operability, Scalability and Robustness. Also chain can be ranked excellent for Expendability. We can not achieve Modularity by using this chain. This chain has good results to gain Flexibility and Usability.

TABLE II
CHAIN OF USE RELATIONSHIP IN PATTERNS

Pattern	**Relationship**	**Pattern**
Abstract Factory	uses	Template Method
Bridge	uses	Adapter
Builder	uses	Abstract Factory
Chain of Responsibility	uses	Decorator
Command	uses	Composite
Command	uses	Memento
Decorator	uses	Composite
Flyweight	uses	Composite
Interpreter	uses	Iterator
Interpreter	uses	Visitor
Iterator	uses	Composite
Iterator	uses	Visitor
Iterator	uses	Memento
Observer	uses	Singleton
Prototype	uses	Singleton
Strategy	uses	Flyweight

- Chain 2 (Bridge → Adapter) promotes only Modularity and Scalability. We can not achieve Modularity by applying this chain. Other quality characteristics are ranked fair in this chain.
- Chain 3 (Builder → Abstract Factory) promotes Expendability, Simplicity, Generality and Operability. This chain has good results to gain Flexibility and Usability.
- Chain 4 (Chain of responsibility → Decorator) promotes Expendability, Simplicity, Understandability, Scalability and Robustness. This chain has good results to gain Flexibility and Usability. We can not achieve Modularity and Scalability by using this chain. This chain has good results to gain Flexibility.

- Chain 5 (Command → Composite), Chain 8 (Flyweight → Composite) and Chain 11 (Interator → Composite) only promotes Robustness.
- Chain 6 (Command → Memento) promotes Expendability and Operability.
- Chain 7 (Decorator → Composite) only promotes Understandability.
- Chain 9 (Interpreter → Iterator) and Chain 10 (Interpreter → Visitor) promotes Expendability, Generality and Scalability.
- Chain 12 (Interator → Memento) only promotes Expendability.
- Chain 13 (Interator → Visitor) can be ranked good for Simplicity, Generality, Software Independence, Learnability and Scalability. Also chain can be ranked excellent for Expendability. We can not achieve Understandability by using this chain. This chain has good results to gain Flexibility.
- Chain 14 (Observer → Singleton) promotes Scalability and Robustness.
- Chain 15 (Prototype → Singleton) promotes Modularity, Scalability and Robustness.
- Chain 16 (Strategy → Flyweight) is the worst chain among 2-pattern chains and we can not achieve any software quality characteristics by applying this chain.

TABLE III
DETAILS OF QUALITY CHARACTERISTICS IN TWO-PATTERN-SIZED CHAINS

Cycle No	Design Patterns	Expendability	Simplicity	Generality	Modularity	Software Independence	Hardware Independence	Learnability	Understandability	Operability	Scalability	Robustness
1	Abstract Factory	Excellent	Excellent	Good	Good	Fair	Fair	Good	Good	Good	Good	Good
1	Template Method	Excellent	Good	Fair	N/A	N/A	N/A	Good	Good	Good	Good	Good
1	Result	Excellent	Good	Fair	N/A	N/A	N/A	Good	Good	Good	Good	Good
2	Bridge	Good	Fair	Good	Good	N/A	N/A	Fair	Fair	Good	Good	Good
2	Adapter	Fair	Fair	Bad	Good	Good	N/A	Good	Fair	Fair	Good	Fair
2	Result	Fair	Fair	Bad	Good	N/A	N/A	Fair	Fair	Fair	Good	Fair
3	Builder	Good	Good	Fair	Fair	N/A	N/A	Fair	Good	Fair	Good	Good
3	Abstract Factory	Excellent	Excellent	Good	Good	Fair	Fair	Good	Good	Good	Good	Good
3	Result	Good	Good	Fair	Fair	N/A	N/A	Fair	Good	Fair	Good	Good
4	Chain of Responsibility	Good	Good	Good	Bad	N/A	N/A	Fair	Fair	Good	Bad	Fair
4	Decorator	Excellent	Excellent	Good	Fair	Good	N/A	Good	Good	Good	Good	Fair
4	Result	Good	Good	Good	Bad	N/A	N/A	Fair	Fair	Good	Bad	Fair
5	Command	Good	Bad	N/A	N/A	N/A	N/A	Bad	Very Bad	Good	Good	Good
5	Composite	Fair	Fair	N/A	Fair	N/A	N/A	Fair	Good	N/A	N/A	Good
5	Result	Fair	Bad	N/A	N/A	N/A	N/A	Bad	Very Bad	N/A	N/A	Good
6	Command	Good	Bad	N/A	N/A	N/A	N/A	Bad	Very Bad	Good	Good	Good
6	Memento	Good	Fair	Fair	Very Bad	N/A	N/A	Bad	Fair	Good	Fair	Bad
6	Result	Good	Bad	N/A	N/A	N/A	N/A	Bad	Very Bad	Good	Fair	Bad
7	Decorator	Excellent	Excellent	Good	Fair	Good	N/A	Good	Good	Good	Good	Fair
7	Composite	Fair	Fair	N/A	Fair	N/A	N/A	Fair	Good	N/A	N/A	Good
7	Result	Fair	Fair	N/A	Fair	N/A	N/A	Fair	Good	N/A	N/A	Fair
8	Flyweight	Bad	Bad	Fair	Good	N/A	N/A	Good	Bad	Fair	Good	Good
8	Composite	Fair	Fair	N/A	Fair	N/A	N/A	Fair	Good	N/A	N/A	Good
8	Result	Bad	Bad	N/A	Fair	N/A	N/A	Fair	Bad	N/A	N/A	Good
9	Interpreter	Good	Fair	Good	Fair	N/A	N/A	Fair	Fair	Good	Good	Fair
9	Iterator	Excellent	Excellent	Good	Fair	Good	N/A	Good	Fair	Fair	Good	Good
9	Result	Good	Fair	Good	Fair	N/A	N/A	Fair	Fair	Fair	Good	Fair
10	Interpreter	Good	Fair	Good	Fair	N/A	N/A	Fair	Fair	Good	Good	Fair
10	Visitor	Excellent	Good	Good	Fair	Good	N/A	Good	Bad	Fair	Good	Fair
10	Result	Good	Fair	Good	Fair	N/A	N/A	Fair	Bad	Fair	Good	Fair
11	Iterator	Excellent	Excellent	Good	Fair	Good	N/A	Good	Fair	Fair	Good	Good
11	Composite	Fair	Fair	N/A	Fair	N/A	N/A	Fair	Good	N/A	N/A	Good
11	Result	Fair	Fair	N/A	Fair	N/A	N/A	Fair	Fair	N/A	N/A	Good
12	Iterator	Excellent	Excellent	Good	Fair	Good	N/A	Good	Fair	Fair	Good	Good
12	Memento	Good	Fair	Fair	Very Bad	N/A	N/A	Bad	Fair	Good	Fair	Bad
12	Result	Good	Fair	Fair	Very Bad	N/A	N/A	Bad	Fair	Fair	Fair	Bad
13	Iterator	Excellent	Excellent	Good	Fair	Good	N/A	Good	Fair	Fair	Good	Good
13	Visitor	Excellent	Good	Good	Fair	Good	N/A	Good	Bad	Fair	Good	Fair
13	Result	Excellent	Good	Good	Fair	Good	N/A	Good	Bad	Fair	Good	Fair
14	Observer	Excellent	Good	Excellent	N/A	N/A	N/A	Fair	Good	Good	Good	Good
14	Singleton	Bad	Very Bad	Fair	Excellent	Fair	Fair	Fair	Fair	Fair	Good	Good
14	Result	Bad	Very Bad	Fair	N/A	N/A	N/A	Fair	Fair	Fair	Good	Good
15	Prototype	Excellent	Good	Fair	Good	N/A	N/A	Fair	Good	Fair	Excellent	Good
15	Singleton	Bad	Very Bad	Fair	Excellent	Fair	Fair	Fair	Fair	Fair	Good	Good
15	Result	Bad	Very Bad	Fair	Good	N/A	N/A	Fair	Fair	Fair	Good	Good
16	Strategy	Good	Fair	Bad	Fair	Fair	N/A	Bad	Bad	Fair	Bad	Fair
16	Flyweight	Bad	Bad	Fair	Good	N/A	N/A	Good	Bad	Fair	Good	Good
16	Result	Bad	Bad	Bad	Fair	N/A	N/A	Bad	Bad	Fair	Bad	Fair

It seems that more studies are required to analyze chain 1 and chain 3, chain 4 and chain 13. Chains 9, 10 and 15 are also good cases for future studies.

TABLE IV
QUALITY CHARACTERISTICS IN TWO-PATTERN-SIZED CHAINS

Cycle No	Expendability	Simplicity	Generality	Modularity	Software Independence	Hardware Independence	Learnability	Understandability	Operability	Scalability	Robustness
1	Excellent	Good	Fair	N/A	N/A	N/A	Good	Good	Good	Good	Good
2	Fair	Fair	Bad	Good	N/A	N/A	Fair	Fair	Fair	Good	Fair
3	Good	Good	Fair	Fair	N/A	N/A	Fair	Good	Fair	Good	Good
4	Good	Good	Good	Bad	N/A	N/A	Fair	Fair	Good	Bad	Fair
5	Fair	Bad	N/A	N/A	N/A	N/A	Bad	Very Bad	N/A	N/A	Good
6	Good	Bad	N/A	N/A	N/A	N/A	Bad	Very Bad	Good	Fair	Bad
7	Fair	Fair	N/A	Fair	N/A	N/A	Fair	Good	N/A	N/A	Fair
8	Bad	Bad	N/A	Fair	N/A	N/A	Fair	Bad	N/A	N/A	Good
9	Good	Fair	Good	Fair	N/A	N/A	Fair	Fair	Fair	Good	Fair
10	Good	Fair	Good	Fair	N/A	N/A	Fair	Bad	Fair	Good	Fair
11	Fair	Fair	N/A	Fair	N/A	N/A	Fair	Fair	N/A	N/A	Good
12	Good	Fair	Fair	Very Bad	N/A	N/A	Bad	Fair	Fair	Fair	Bad
13	Excellent	Good	Good	Fair	Good	N/A	Good	Bad	Fair	Good	Fair
14	Bad	Very Bad	Fair	N/A	N/A	N/A	Fair	Fair	Fair	Good	Good
15	Bad	Very Bad	Fair	Good	N/A	N/A	Fair	Fair	Fair	Good	Good
16	Bad	Bad	Bad	Fair	N/A	N/A	Bad	Bad	Fair	Bad	Fair

IV. CONCLUSION

In this paper, we investigated the benefits of design patterns to increase the Reusability, Flexibility, Usability, Scalability and Robustness. We focused more particularly on "use" relationship i.e. the cases that we apply chain of patterns that use each other. Our study on this cases shows that applying pattern chains with 2 or 3 patterns does not always lead us to achieve better software quality metrics. However there are some cases in which the quality characteristics are ranked "good" level and Flexibility, Scalability and Robustness are increased.

In our future work, more studies are needed to investigate other situations that using chain of patterns promotes the system quality.

REFERENCES

[1] A. A. Aaby, Software: a fine art. Jan 2004. http://cs.wwc.edu/aabyan/FAS/book.
[2] D.G. Firesmith, Common concepts underlying safety, security, and survivability engineering. December 2003. http://www.sei.cmu.edu/pub/documents/ 03.reports/pdf/03tn033.pdf.
[3] International Standard. ISO/IEC 9126-1. Institute of Electrical and Electronics Engineers, 2001. http://www.iso.ch.
[4] J. E. Gaffney, Metrics in software quality assurance. Proceedings of the ACM '81 conference, March 1981. http://portal.acm.org.
[5] K. Khosravi, Y.G. Gueheneuc, A Quality Model for Design Patterns. Summer 2004.
[6] L. G. Williams, C. U. Smith, Introduction to Software Performance Engineering. Addison Wesley, Nov 2001. http://www.awprofessional.com/articles/article.asp?p=24009.
[7] L. J. Arthur, Software evolution, the software maintenance challenge. John Wiley and sons, 1951.
[8] L. Tahvildari, K. Kontogiannis, On the Role of Design Patterns in Quality-Driven Re-engineering. Proceedings of the Sixth European Conference on Software Maintenance and Reengineering (CSMR02), 2002.
[9] M. B. Nilles, A hard look at quality management software. Quality Digest, 2001. http://www.dofactory.com/patterns/Patterns.aspx.
[10] O. Balci, Credibility assessment of simulation results. Proceedings of the 18th conference on winter simulation, 1986.
[11] R. S. Pressman, Software Engineering a practitioner's Approach. McGraw-Hill, Inc, 1992.
[12] R. Johnson, E. Gamma, R. Helm and J. Vlissides, Design Patterns Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.