
# Design Patterns

## in

# Software Engineering

Student: *Ehsan Feghhi*

Professor: *Kanita Hadziabdic*

Course: *CS504 Software Engineering*

*Sarajevo 2017*

INTERNATIONAL UNIVERSITY OF SARAJEVO

# Abstract

In this research paper design patterns in software engineering will be explored. In fact, design patterns are the output of programmers' experience in many years and of course, because of the progress of the science of programming, the process of patterns also continues and even some of them have lost their usage. We should also notice that as the most real-world issues can have different solutions, problems in programming also can also be solved in different ways and each design pattern is an optimal solution to a repetitive problem in object oriented programming.

Benefits of studying design patterns is that it makes the person's mind more familiar with issues and problems with the object-oriented programming framework. And after that, a programmer is familiar fairly with the optimal solutions to the repetitive issues. The benefit of using design patterns in coding is that since design patterns are well-known among programmers everywhere, this creates a common language. The usefulness of this common language is particularly evident in team work programming.

# 1. Introduction

The use of patterns came to the mind of an architect name “Christopher Alexander”. He was born in 1936 in Vienna and grew up in Oxford. He studied architecture and mathematics at Cambridge University in England at undergraduate level and then he went to the United States and received doctoral degree from Harvard University. His theories about the nature of human-centered design have affected fields beyond architecture, including urban design, software, sociology and others. In software, alexander is regarded as the father of the pattern language movement.

Alexander faced an issue and that was how should be a good and high quality design for constructing a building. In order to solve his problem, he studied buildings, streets, settlements and any other places that was made by human. He discovered that well-designed buildings have common features and they have similar characteristics. He called these similar characters and features, a pattern.

Each pattern represents a problem and a solution to that problem which can be repeated over and over again and you can use this solution for millions of times without doing it in the same way twice. In the early 1990s, some software developers were struggling with Alexander's work. They were confronted with the question that if architectural design patterns are correctly answered in this field, can patterns be created for software design.

What are the issues in the software that occur frequently and can be solved with almost identical methods?
Is it possible to use the concept of patterns in software design, can we create solutions based on patterns after identifying patterns?

They were questions that developers proposed and found their answers. The answer was yes. The next step was identifying patterns and developing standards for documenting patterns.

In the early 1990s, many people worked on design patterns but four persons had the most impact in this regard. “Erich Gamma”, “Richard Helm”, “Ralph Johnson” and “John Vlissides” wrote a book called “*Design Patterns: Elements of Reusable Object-Oriented Software”*. These four writers are famous for the Gang of Four. In this book, they used the patterns in software design, and created a standard format for documenting patterns. They categorized 23 types of patterns and over time, standard formats have been proposed to document the patterns.

# 2. Pattern structure

Initially, several infrastructure design patterns were introduced in the field of computer engineering, which their number was about 20. But now the number of design patterns have reached more than 100. It is important to note that the design patterns help solve the problem, but they do not give us the full solution. Patterns in engineering were introduced first time in building architecture, but the first person who used the patterns for generating the software was a Swiss computer scientist "Erich Gamma". Gamma together with three others formed the **GoF** group and described the patterns in their book. The book was welcomed by the public, and since then GoF design patterns have come to fame, and have been used in many different fields of software engineering.

The efficiency and appropriateness of the GoF design patterns are so high that today professional tools support them directly. Among these tools we can name "Borland together", "Rational XDE" and "Rational Rose". GoF's basic patterns are divided into three categories in terms of purpose:

| | |
|---|---|
| **Creational Pattern** | They are used in the process of creating objects and include:<br>- Abstract factory<br>- Builder<br>- Factory method<br>- Prototype<br>- Singleton |
| **Structural Pattern** | They are used in combination of classes and objects and include:<br>- Adapter<br>- Aggregate<br>- Bridge<br>- Composite<br>- Decorator<br>- Extensibility<br>- Façade<br>- Flyweight<br>- Marker<br>- Pipes and filters<br>- Opaque pointer<br>- Proxy |
| **Behavioral Pattern** | Discuss interactions between classes or object as well as how they distribute responsibilities and include:<br>- Chain of responsibility<br>- Command<br>- Interpreter<br>- Iterator<br>- Mediator<br>- Memento<br>- Null object<br>- Observer<br>- Protocol stack<br>- Scheduled-task |

| | |
|---|---|
| | - Single-serving visitor<br>- Specification<br>- State<br>- Strategy<br>- Template method<br>- Visitor |

Here we are going to describe some of these patterns which are commonly used.

Abstract factory: a method for summing up a group of distinct factories which have the same structure but are composed from different classes.

Adapter: it allows the interface of existing class to be used as another interface. Usually this pattern is used to make current classes work without changing the source code. It also converts an interface on one class to another interface which is expected by the customer. This pattern allows collaboration between object that previously could not work together because of incompatible interfaces.

Bridge: it separates an abstract concept from its implementation so that both can independently change.

Builder: this pattern unlike the abstract factory pattern and the factory method pattern which their intent is to observe the phenomenon of polymorphism, was introduced to solve another problem of constructing and arranging objects in programming. The problem is that sometimes it is necessary to deliver a large number of parameters to our constructor for constructing an object, which reduces the programming stuff and the readability of the program. In order to solve this problem, we use the builder pattern.

Chain of responsibility: By giving the chance to more than one object to respond to a request, it prevents the sender and receiver from intermingling. In this way, the objects that receive the request are considered in a chain and pass the request through this chain until one of the objects answers it.

Command: it encapsulates a request as an object. Therefore it makes possible to parameterize customers with different requests, sort requests and provide reversible actions.

Composite: This pattern allows customers to process single and compound objects in the same way.

Decorator: it makes ability to add behavior to an object dynamically or statically without changing the behavior of other objects of the same class.

Facade: it is usually used in object-oriented programming. Its name is an analogy to the architectural façade. The façade is an object that has a convenient interface for accessing a large and complex part of the code, such as a class library.

Factory method: it defines an interface for constructing objects, while allowing subclasses to decide on which object they are going to take sample. In fact, this pattern allows subclasses to do sampling.

Flyweight: flyweight is a type of object which saves memory usage. A flyweight object share information with other similar objects as much as possible. When one type of objects in large number share common information, that common part is repeated in all those examples which cause a high consumption of memory that is unacceptable. To solve this problem, we usually put the shared part in another structured data that all flyweight objects will have access to it.

Iterator: Provides a method for sequencing the elements of a complex object without exposing it.

Mediator: this pattern defines an object which determine the relationships of a set of objects. This pattern is known as a behavioral pattern because it can change the behavior of the program during its implementation. This pattern, by avoiding direct references between a set of objects, encourage a minimal connection between them and allows us to change them independently.

Memento: Without breaking the encapsulation, it captures and stores the internal state of an object so that the object can return to that state later.

Observer: it defines one type of one-to-many relationship between objects, so that when an object has changed its state, all affiliated objects get to know to coordinate with it.

Prototype: Using an object as an example, specify the type of new objects that should be created and create those objects by making new copies of this sample.

Proxy: It creates a successor or place to control access to an object. Proxy, in general, is an intermediate class for doing something else.

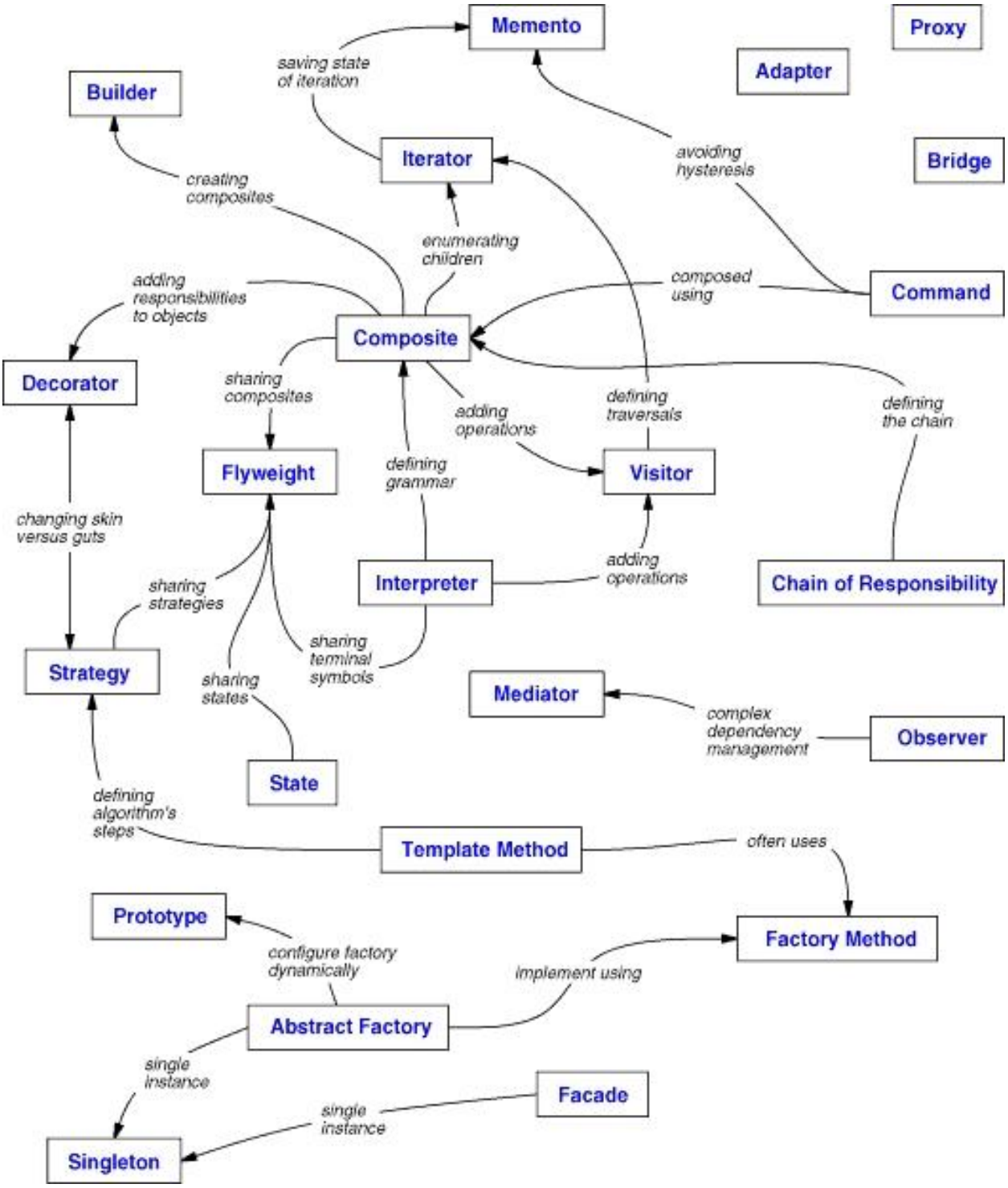
Singleton: it ensures that a class has only one object and provides access to that instance.

State: it allows object to change its behavior when its state changed. Objects change class behavior in which they are located.

Strategy: it defines, encapsulate and replace a family of algorithms. The strategy allows algorithm no matter where it is used.

Template method: it defines an algorithm skeleton and devolve implementation some of its steps into subclasses. This pattern allows subclasses to modify some of the steps of an algorithm without changing the general structure of the algorithm.

The following figure, represent the design patterns relationship.



Based on design patterns scope, we can divide them again in two different types. Class patterns deals with relationships between classes and it is static during compile time, while Object patterns deal with objects that can be changed during implementation and are more dynamic. Among the patterns which were mentioned so far, only the Adapter, Interpreter, Factory method and Template method fall into the class patterns and the rest of the pattern are in the Object patterns.

| | | **Purpose** | | |
|---|---|---|---|---|
| | | Creational | Structural | Behavioral |
| **Scope** | Class | Factory method | Adapter | Interpreter<br>Template method |
| | Object | Abstract factory<br>Builder<br>Prototype<br>Singleton | Bridge<br>Composite<br>Decorator<br>Flyweight<br>Façade<br>Proxy | Chain of responsibility<br>Command<br>Iterator<br>Mediator<br>Memento<br>Observer<br>State<br>Strategy<br>Visitor |

We describe design patterns based on a consistent form. Each pattern is divided into sections based on the following template. The template gives the same structure information so that a pattern can be easily understood, compared, and used.

| | |
|---|---|
| Name | A good and useful name for the pattern<br>The name of the model carries the nature of the pattern in a nutshell. |
| Intent | A brief and concise statement about what the pattern does<br>A short sentence that answers the following questions: What does this template do? What is its logic and purpose? |
| Alias | Other names which pattern is known with them |
| Motivation | A scenario that describes a problem and how to solve the problem by the structure of classes and objects in the pattern. |
| Applicability | Under what circumstances can this design pattern be applied? What are some examples of poor design that the pattern can address them? |
| Structure | A graphical representation of the pattern |

| | |
|---|---|
| Components | Classes and objects that exist in the pattern |
| Collaborations | How components work together to carry out their duties |
| Results | Results of using the desired pattern |
| Sample code | A piece of code to implement a sample |

# 3. Reasons to choose pattern

Earlier when the first software was written, people made everything from scratch, because there were no software experiences and they had to create new ideas for software creation. After a few years of software companies' empowerment in the late 1980s, software engineers began to think of using their previous experiences instead of reinventing the wheel. Therefore, patterns were drawn from past experiences to become available to everyone as "a library of useful experiences" to be no need for people to discover them again.

Today in most of the world's universities, design patterns are taught in advanced programming, so that the mentality of pattern be embedded in programming from the very early stages of the university. In many universities, a variety of patterns are taught in the field of software engineering to provide visualization to students. When people found this vision, in dealing with specific issues in one area, they will try to publicize them and convert them to patterns.

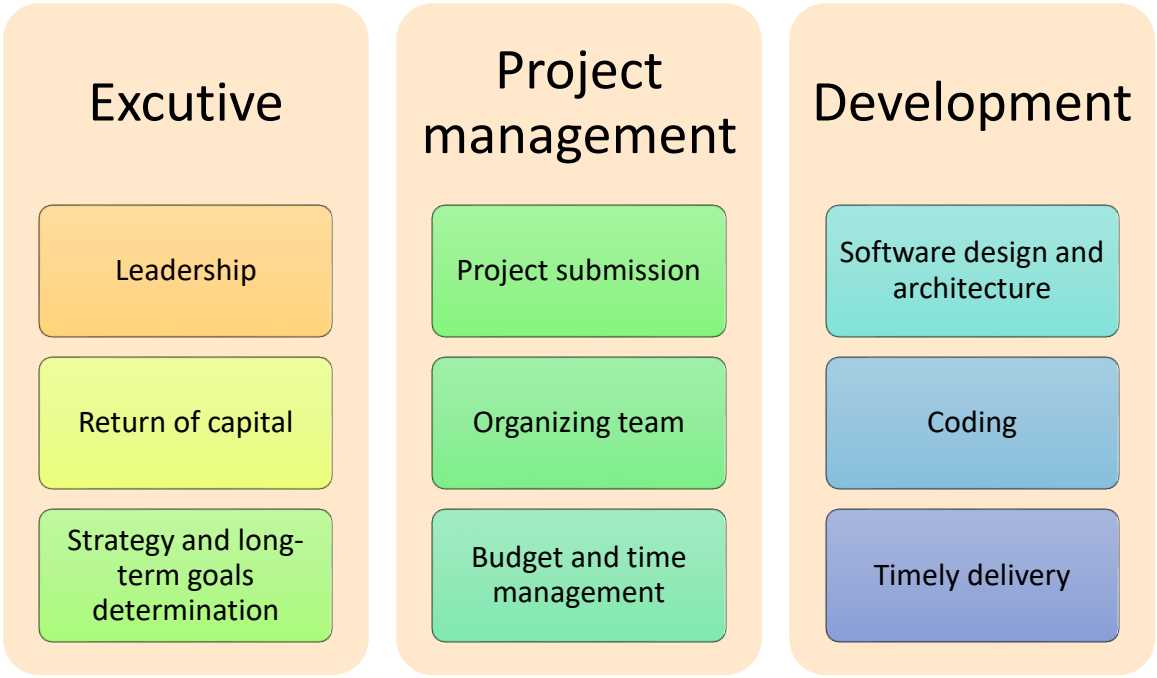
# 4. Pattern's selection criteria

The following are involved in choosing a suitable design pattern suitable for a software:

- Who wants to use the software or who is the stakeholder of the software?
- How much does it cost to generate software?
- In which area the software is going to be used?
- How long the life of the software should be?
- How much time is required to generate software?
- Is there any prediction of future software changes?
- Is it the commercial, research or service software?

All of the mentioned points are involved in choosing a good design pattern and even a good and result-oriented architecture for producing software. In my opinion, a good pattern or architecture is not the one that has the best available methods, but it is the one that can deliver the best result based on project conditions.

# 5. Rate of efficiency




# 6. Pattern's definitions

## 6.1 Comprehensive Definition

A general approach to solving a particular issue in a few cases is called a design pattern. In fact, design patterns are not dedicated to the software world, and in some industries they may also use models and patterns. In software systems, a design pattern is publicly accepted only when it was resultful at least in three cases of the same problem. Although it is easy to guess that the number of design patterns is high, but only a few of these patterns have come to be known worldwide. Because other patterns are either very simple that you cannot name them the design pattern or they are very large and complex which cannot be used mostly. Also, in many cases, companies have their own design patterns for their specific workflow, which the developers of that software are bound to observe.

Another positive point in design patterns is that they are the result of an experiment, not academic topics. In other words, in order to create a pattern, you don't need to look for a new model in the software engineering labs, but you need to have attention to your past and your future works. You should ask yourself if I've solved this problem before or is there a chance to I face this issue again in future.

### 6.2 Main goals

Sometimes when developers design or implement their own programs, they face classes which are not so-called classical classes. These classes create problems that can be solved with the help of patterns. In fact, the design of such classes has over the time encountered every programmer, therefore professional programmers based on their personal experiences, have decided to create patterns that will be the solution to these problems.

Currently, there may be around hundreds of patterns in this regard, but only few of them are standardized and used by designers and programmers. Design patterns play a very important role as the common language of the software development team. When you mention a pattern in your work, then you don't need to add extra explanation about your work, because that pattern is known by everyone.

Another advantage of design patterns is their unobtrusive assistance in object-oriented design training. In other words, in order to focus designers' thought on thinking object-oriented, one of the best tools is design patterns.

### 6.3 Minor goal

The design pattern is responsive to many problems in software production and the design of object-oriented programs. Object-oriented programming, although it's a good solution to reuse code and objects which are generated in software projects, but also creates limitations. For example, if you need to change the object used in the project, many parts of the written code will be changed. In this way, written codes are largely dependent on that object.

## 7. Feasibility and needs analysis

### 7.1 Main user's request

Here, the main goal is to meet the demands of customers or so-called users, and the manufacturer must do his utmost to meet these demands. One of the most important tasks in software development is the need to extract or analyze the requirements of that system. Public clients usually have a conceptual, abstract, and ambiguous conception of the end result of their demands and they don't know exactly what does the software that they are looking for. At this stage, incomplete, complex, ambiguous, and even contradictory requirements are identified by skilled and

experienced software engineers. At this point, ready-made and operational software components may help to reduce the risk and problems of the requirements. First, the general requirements are gathered from the users, and then the domain of the development and production of the software that should be produced will be identified and analyzed, and then documentation should be written transparently.

### 7.2 Determining the approximate costs

Familiarity with the actual cost of software production or the creation of a website in dedicated projects helps the employer to accurately estimate the true price of their software, and this helps in selecting the right company to do this. Just like construction engineering consultants who help the employer select the right contractor after the bidding process. In fact, the consultant engineers are fully aware of the cost of the project.

## 8. Pattern planning

### 8.1 Forming a work team

In discussing pattern design, only to mention what should be done by who (job description) is not enough, but the processes, data, goals, and roles of those who perform tasks must be consistent with the goals and strategies of the organization. This requires that the organization has a map of all its dimensions so that it can understand the relationships between the dimensions of the organization and, if necessary, adapt to the changes.

### 8.2 Gantt Chart

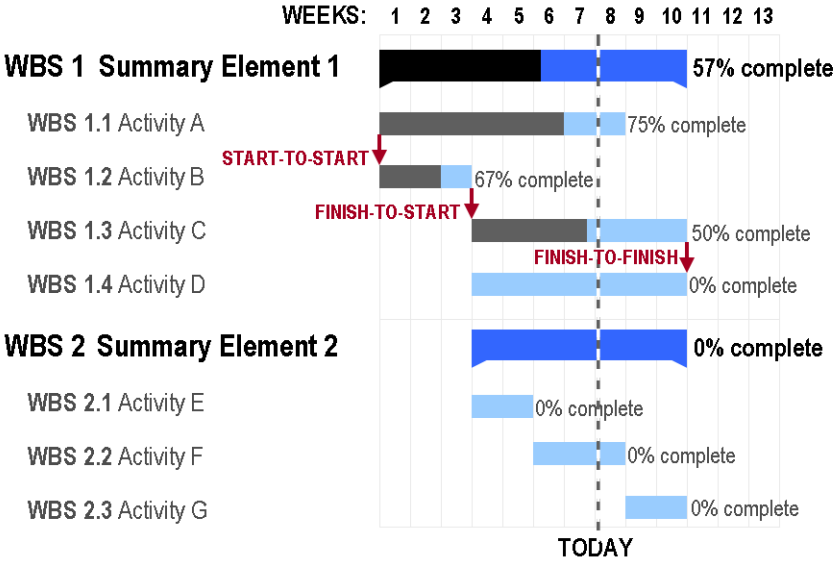
How to advance a project should be seen as important as of how it is made. The team or builders actually should know the basics of moving ahead, or they must know what they want to do, just like pairing patches of puzzle that will not show us the final image until it is not over. Obviously, all the projects which are under construction do not have infinite budgets or they do not have infinite time. All of them must be completed and enter to the market or be used for the purpose which they are defined for it. Failure to define a specific timetable in the production process of the product will eventually lead developers to face up to huge underdeveloped workflow by the middle of the project.

In fact, it can be said that the history of finding this chart dates back to the First World War, where the American, “Henry Gantt”, used some timetables for the first time for planning and controlling shipbuilding projects. Henry Gantt, with the help of his innovative tool, was able to significantly reduce the time of construction of ships during the First World War. Today Gantt chart for its

simplicity and comprehensiveness is being used as an interesting and popular method in the world for controlling and managing projects.

According to a survey conducted among Microsoft Project Planning and Control Software users, 80% of executives in the world prefer to use Gantt chart to plan and control their projects. Most commonly used methods for project planning and control which are used today, are the ones that were invented and developed by the United States Department of Defense and the military industry during the 1950s and 1960s.

Gantt chart is one of the most popular and useful ways of showing activities displayed against time.




The horizontal axis represents the time that can be used either in absolute terms or in relative terms, based on the time when the project began. In the above sample image, it can be seen that each activity is at the end of the previous activity. However, these activities can also be active simultaneously at the same time.

# 9. Collecting data and information

## 9.1 Documentation in patterns

Documentation at all stages of the project, such as determining the goals of the system, maintaining and upgrading and improving the system, although the project has been completed, should be done. This documentation may also include writing the structure of the program's appearance as well as internal and external applications. It is very important that everything is documented. A design

pattern is a document that succeeds in solving a certain category of problems and is used to solve future problems. At first, an architect engineer named Christopher Alexander used design patterns in his field of work, which later expanded on the idea in other areas of work, and was especially highlighted in programming.

# 10. Anti-patterns

## 10.1 What are anti-patterns?

Professional progress is directly related to self-criticism. If a programmer wants to advance in his career, he must criticize himself and challenge his current trends and codes. Software Development Anti Patterns are usually inappropriate processes that implement programming to solve various programming issues, but later these trends create problems in the structure of the program.

Creating a good structure in the program helps develop and maintain an application. While the creation and implementation of an application is often done sporadically, and in many cases, the delivery speed of the project is superior to software architecture (especially in Agile-based projects and startups).

By identifying different patterns in programming, one can find out what patterns are wrong to make a program. For example, if a programmer wrote a class that would do a lot of work inside this class, it would lead to an anti-pattern called the God Object. This anti-pattern makes it possible, for example, to change the part of the program code, to change the other parts of the program (which is not the case in previous software engineering). Or it may no longer be possible to reuse the application code of the program.

## 10.2 God class

In short, if you have a class in the form of an object-oriented program that has a lot to do, then there's a Blob anti-pattern. This anti-pattern is also known as the God Class.

Suppose you have a program that performs various operations, such as uploading images, displaying images, managing errors, determining the user's access level to specific images, and so on. Now, suppose you have a very large class, all of which are done exclusively by the functions of this class. In fact, a Blob class has been created here that accepts excessive liability.

You might even have several other classes along with this great class that only hold data for this class. It does not matter again and may cause problems later in the software development and development process.

## 10.3Lava Flow Anti-pattern

Maybe you've worked on research projects. The main feature of these projects is that it needs more research and development (R & D) than other programming projects. In these projects, codes are often written that have a lot of complexity, and very little documentation. In these projects, the mainstream of the programming is focused on getting optimal output, which makes the programmer less concerned with the overall structure of the code and its architecture.

In many projects that require a lot of research and development, codes, classes, and complex methods are written, which is usually not followed by future programmers. These codes do not usually get handwritten because they are afraid that part of the program will be lost by manipulating these codes.

As these projects become larger, research codes such as a lava series, which have been tightening over time, are being poured over the project, and the anti-pattern of lava flows is taking place. These lavas, which have been rigid, cannot be changed, and nobody thinks of them. Especially if the original programming of these lava is separated from the project!

# Reference List

1. http://www.dofactory.com/net/design-patterns
2. https://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Architecture/Design_Patterns
3. https://medium.com/swlh/design-patterns-coming-full-circle-d8292e261dc6
4. http://fusion.cs.uni-magdeburg.de/pubs/gcse99-cdrom.pdf
5. https://sourcemaking.com/antipatterns/software-development-antipatterns