Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 3, March 2014, pg.519 – 529

RESEARCH ARTICLE

Agile Programming and Design Patterns on Web Applications Using J2EE and Ruby on Rails –A Case Study

Vedavyas J[1], Kumarswamy Y[2]
Asst. HOD, Department of MCA, BITM, Bellary, VTU
Prof. and HOD, Department of MCA, Dayanandasagar College of Engineering, Bangalore, VTU
[1] vedavyasjamakhandi@gmail.com
[2] ysklds wamy2@yahoo.com

Abstract

Agile Programming methodologies reduce the risk of longer time consumption for the software applications development. They give much importance to the real-time communication and priority for the satisfaction of stake holders. From the object oriented world the Design patterns have taken a lot of effort for the design reuse. Design patterns provide solutions and also make easy to common design problems by reusing successful designs and architectures. By merging the application and implementation methodology lots of design patterns can be formed for to help the new designers. This paper emphasizes on the Agile methodology, Design patterns for the web applications using J2EE and Ruby on rails with respective case study design pattern example.

1. Introduction

Agile helps the software business to be quite ready for the unpredictability. It acts like an alternate for the project management. (Agile is mainly suitable for the) It mainly concentrates on how to achieve the goal by working together, to plan, to build and to deliver software.

Winston Royce's Waterfall model

As per the Royce waterfall model the size and the complexity are the two essentials for the program developments. Other than the analysis and coding step he introduced 5 steps which are program design, code design, plan-control and monitor testing.

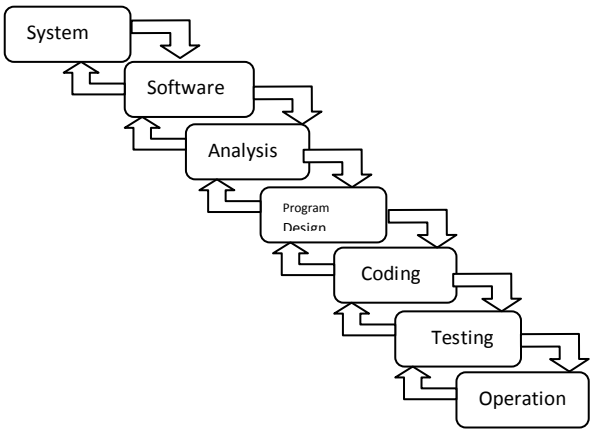

Fig 1. Winston Royce's Waterfall Model

Each phase should move to the next phase twice before it comes out. But in real time the design iterations are never bounded to the successive steps.

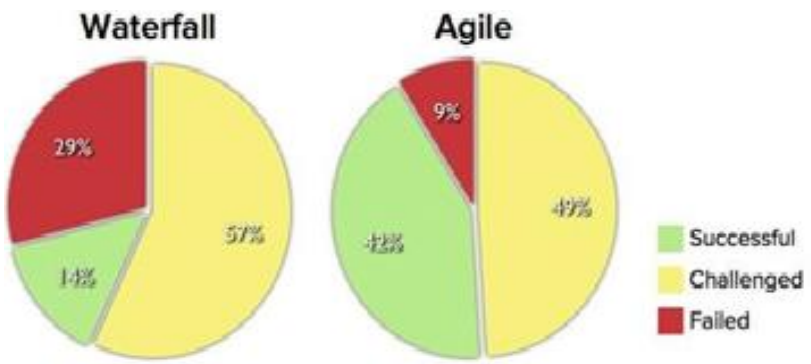

Fig 2. Waterfall and Agile model Comparison

2. Agile Development Methodologies

Agile development methodologies include

1. Scrum
2. XP (Extreme Programming)
3. Crystal
4. FDD
5. DSDM

Agile provides the following advantages

1. Improved quality.
2. Opportunities for mid-course corrections.
3. Improved customer business satisfaction.
4. Better coalition between business and IT.
5. Improved market time.

A. Extreme programming

Most of the project teams adopt Extreme Programming practices. XP is an elementary and concrete practices used in the agile development process. Agile is a classification whereas XP is a discrete method and one of the principles of Agile.

XP is the one of the well-defined and broad in scope when compared to the other methods of the agile. Scrum is close equivalent to the XP but the difference is that scrum is a subset of XP.XP is the only one method which gives inscrutable and fundamental disciplines for the agile developers.

Fig.3 Extreme Programming Practices

B. Scrum

As per the agile software development framework scrum is the iterative and incremental sprints model for the application development. Development team works as a unit to reach common goal by focusing the strategies like flexible and holistic rather than traditional sequential approach.

Scrum principles give

1. Iterative and incremental development- which optimizes the predictability and control risks.
2. Empiricism- inspects, adapt and transparency
3. Continues improvement
4. Teamwork-divides into small teams, self-organizing teams and cross-functional teams.
5. Scrum Master or Servant leadership

Scrum gives the answers for the questions like

1. Why it takes so long time?
2. How much effort required for to do stuff?
3. How long and how much effort things will take?

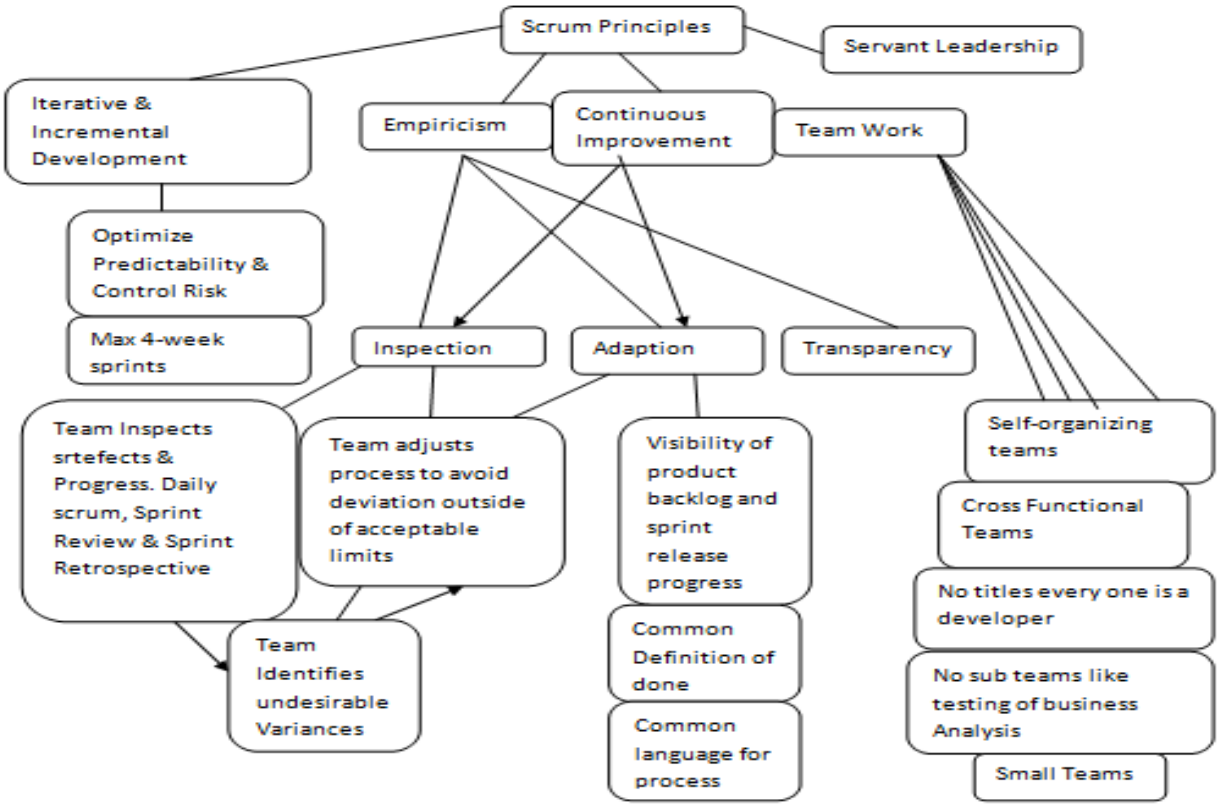

Fig.4 Scrum Principles

C. Modern Web Development Method

With agile development, team will be able in building the website within a couple of weeks of starting the project, and be getting real-time feedback from real website users within a month.

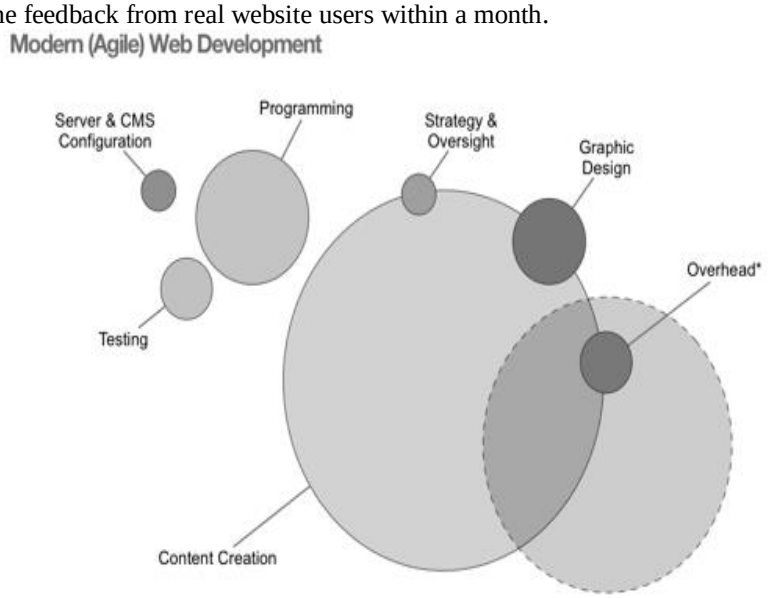

Fig.5 Modern Web development

Your internal team will be empowered to make these changes and will be trained in asking the key questions, which are:

1. Does this change help us reach our goals?
2. Do these images and words support our brand?
3. Is this solution better than what's there now?
4. What's the worst that could happen if we tried this for a week?
5. Once the week is done, how should we decide if we leave it or change it back?

3. Design Patterns in Web Applications

In recent epoch designers are migrating to the non-web based applications which are complicated and time consuming in the software development and also face challenges like handling of data, structuring and organizing the web applications. So the solution for the above said is making use of object oriented technologies like design patterns or simply patterns.

Design patterns are the basic building blocks for the developers which give birth to the new concept of pattern based web applications, which promotes reusability and consistency. All the common practices are captured or documented for the future purpose. The good practices are known as patterns and the bad ones are known as anti-patterns.

Based on the survey made during this paper, the pattern based web applications are having some menace in the process like

1. Large number of patterns exist choosing the right one for the context is enormously difficult
2. Each and every code should be considered for the review which is associated with the design patterns at different constraints.
3. Patterns are the helpful only for the advanced/experienced users.
4. Patterns are not suitable for the often changing web applications

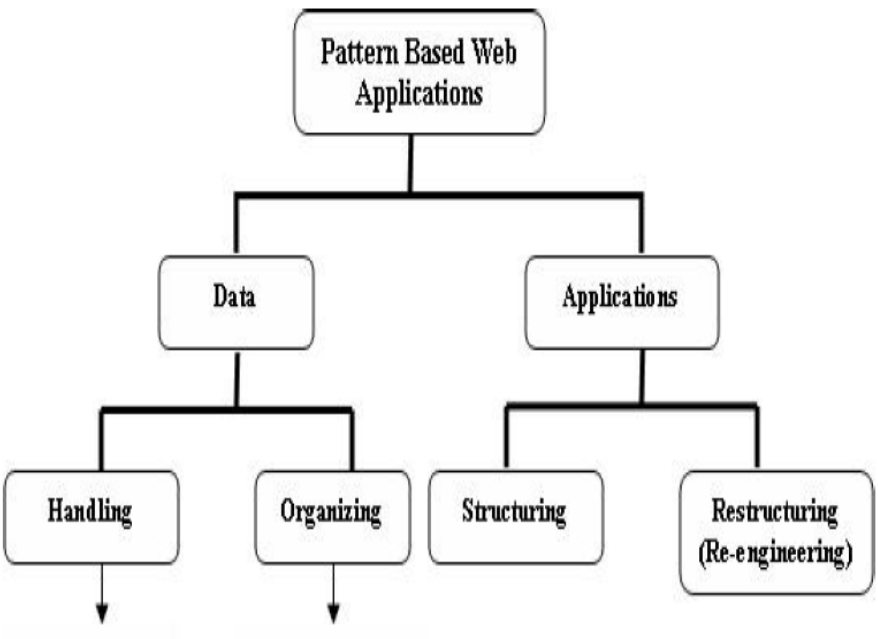

Fig:6 Pattern Based web application

4. Case Study of Design Patterns on J2EE Web Applications

A. J2EE Web Application: An Example on Partitioning

Let us consider the web-application where a client wants to fetch information about a company's employees in a simple way by executing two operations.

1. By supplying a name, and clicking on a "search" button, search the employee directory "by name". The search returns the set of employees that match the search criteria in a format that displays an abbreviated employee record for each member of the returned set.
2. By clicking on a "details?" button, get detailed information about a specific employee. Implementation in a stand-alone, single address-space, environment, is straightforward. From the perspective of the MVC design pattern

The Model consists of the records in the employee directory. There are four Views: a "search" panel; a display of abbreviated information about a set of employee records; a display of detailed information about a specific employee; and a report that no employees match the search criteria.

There are two Controllers: one that, given a "search" directive, drives the process of querying the Model and returns a result set; and one that, given a "details" directive, queries the Model to get the full set of information about the specified employee. Implementation as a web-application in a server environment raises the issue of partitioning which is conceptually irrelevant to, but in practice complicates, the MVC design pattern. Naively, as there are two Controllers, the application can be implemented in one of four ways. Either both Controllers execute exclusively on the client or server, or one Controller executes on the client and the other executes on the server. Each partitioning decision greatly affects the way that the application is implemented. For example, if both Controllers run on the client (the "fat-client" approach), the entire Model must be downloaded to the client -- which is often impractical. If both Controllers run on the server (the "thin-client" approach), two round trips between client and server must be performed each time that the client searches for an employee and then asks for more detail about that employee.

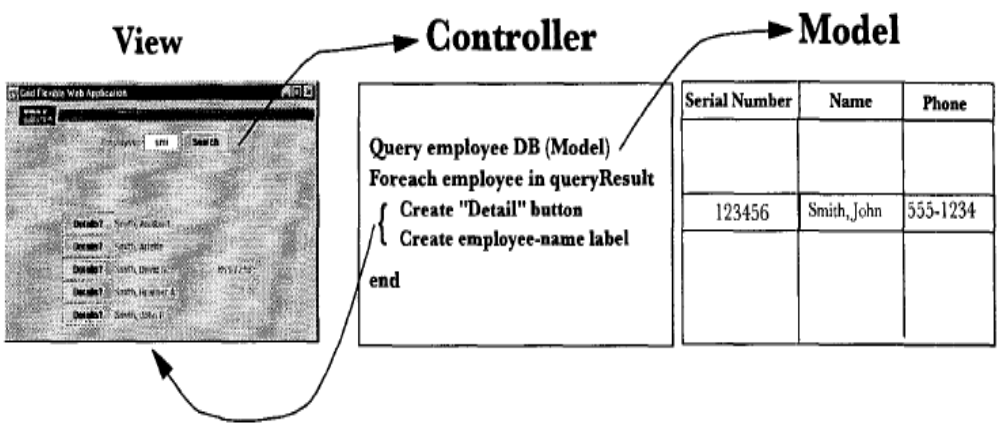

Fig 7. MVC for Employee Record

In fact, for many environments, either the thin-client or the fat-client is ideal. Instead, using a *dual-MVC* approach**,** we partition the Controllers between the client and server. Specifically, the "search" Controller executes on the server in association with a Model consisting of the complete employee directory. However, when returning relatively small sets of employee records, the Controller *also* returns the full record for each of the employees, so that they can be maintained in the client-side Model. The dual-mvc approach allows requests for detailed employee information to be served by the client, thus eliminating a client/server interaction. (This implementation is beneficial only when application scenarios typically consist of a preliminary search for an employee using a "partial name", followed by request for more information after the specific employee is determined by inspection. Remember: this is only a motivating example!)

Of course, what we really want is to do *avoid parti**tioning*** while implementing the application, since the correct partitioning decision depends on factors that are not necessarily determined until actual deployment. For example, if the

employee directory is relatively small, the "fatclient" approach with both Controllers executing on the client makes sense and would provide better performance.

Conversely, if the application is deployed in a "internet" environment in which users want minimal customization of their environment, the "thin-client" approach may be the only solution possible. Delaying application partitioning for as long as possible is even more attractive because partitioning gets in the way of designing the Views and developing the business logic needed by the Controllers. Flexible web-application partitioning addresses these needs. In fact, flexible web-application partitioning goes further, allowing partitioning decisions to vary *dynamically*, during application execution.

The *J2EE* programming model explicitly supports the MVC design pattern, and enables programs executing in MVC mode to execute in a single address space. When deployed, these programs can be flexibly partitioned without changing the source code used during smvc development. We refer to such *J2EE* applications as *J2EElications.*

B. Design Patterns on Ruby on Rails as A MVC with Abstract Factory Design Pattern

Ruby on Rails is a web framework that was written to emphasize several out-of-control-awesome design patterns, including active record, convention over configuration, and model-view-controller (MVC). For creating families of related objects together abstract factory is the common interface. The client calls the method provided by the common interface but doesn't build any objects directly.

In this paper we have implementation of the abstract factory using ruby is shown below for the category of courses in the university as model classes

```
#courses.rb
Class Course
attr_accessor:title
def initialize(title)
      @title=title
      end
      end

      class PG < Course
      def description
      puts " I am PG graduate named#{@title}"
      end
      end

      class UG < Course
      def description
      puts " I am UG graduate named#{@title}"
      end
      end
```

We can see, both models derive from a common superclass Course. Let's define the Factories delegate to build these objects:

```
#factories.rb
module MyAbstractCourseFactory
  def create(title)
    raise NotImplementedError, "You should implement this method"
  end
end
```



```
class PGFactory
  include MyAbstractCourseFactory
  def create(title)
    pg.new title
  end
end

class UGFactory
  include MyAbstractCourseFactory
  def create(title)
    ug.new title
  end
end
```

Note that we have defined the abstract factory (MyAbstractCourseFactory) as a module: it defines the abstract method that must be implemented by the class that includes PGFactory and UGFactory represent the two concrete factories responsible to build, respectively, UG and PG courses.

The code of a College, that can provide courses based on the needs of the student, will be defined as follows:

```
class College
  def initialize(number_of_courses, course_type)
    if course_type == :pg
      title = 'Post Graduation'
      course_factory = PGFactory.new
    elsif course_type== :ug
      title = 'Graduation'
      course_factory = UGFactory.new
    end

    @course = []
    number_of_courses.times do |i|
      @course<< course_factory.create("#{title} #{i+1}")
    end
  end

  def show_courses
    @course.each {|course| course.description}
  end
end
```

launching the following file main.rb

```
#main.rb
require 'courses.rb'
require 'factories.rb'
course_store = College.new(2, :pg)
course_store.show_course
course_store =College.new(5, :ug)
course_store.show_course
```


5. Conclusion

A good agile team picks and chooses the management and technical practices that best work for them. Agile programming is a software development methodology better suited to the current demands of shorter time frames and easily adaptable software. Design Patterns help in faster and more effective deign for software, with reduction in effort achieved through making use of existing/standard patterns in design. In this paper we presented the narration approach to design pattern automation by partitioning. The main characteristic of this paper is to simplify and to make intuitively appealing.

References

[1] The Standish Group, "The Chaos Report," West Yarmouth, MA: 1995.
[2] Beck .k, "Extreme Programming Explained –Embrace Change," 2/e, Addison Wesley, Boston, MA, 2005
[3] Beedle .M, Schwaber. S, "Agile Software Development with SCRUM," Prentice Hall, Upper Saddle River, NJ, 2001
[4] Poppendieck. M, Poppendieck. T, "Lean Software Development: An Agile Toolkit," Addison-Wesley, Boston, MA, 2003 International Journal of Software Engineering & Applications (IJSEA), Vol.3, No.1, January 2012-45
[5] Ned Chapin, "Agile Methods' Contributions in Software Evolution," in Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM'04)
[6] N. Chapin, et al., "Types of software evolution and software maintenance," Journal of Software Maintenance and Evolution, John Wiley & Sons, Chichester UK, January 2001, pp. 3–30.
[7] Agile Alliance, Manifesto for Agile Software Development, 2001. http://www.agilemanifesto.org/
[8] K. Beck, Extreme Programming Explained, Addison-Wesley, Boston MA, 2000.
[9] Alexander C., et al *Pattern Language: Towns, Building and Construction.* New York: Oxford University Press, 1977.
[10] Erich Gamma. , et al. "Design Patterns Elements of Reusable Object-Oriented Software," Singapore, Pearson Education, 2003
[11] Andr´e DeHon., et.al. "Design Patterns for Reconfigurable Computing". *IEEE Symposium on Field-Programmable Custom Computing Machines* (FCCM 2004), April 20–23, 2004.
[12] http://www.rubyist.net/~slagell/ruby/
[13] Sam Ruby., et al. "Agile Web Development with Rails", 4th Edition, The Pragmatic Bookshelf.
[14] http://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/
[15] Wolfgang Keller, "Object/Relational Access Layers A Roadmap, Missing Links and More Patterns" in Proceedings of the 3rd European Conference on Pattern Languages of Programming and Computing,1998.
[16] http://martinfowler.com/eaaCatalogindex.html
[17]Shuster, J., *UIML: AnAppliance-Independent XML User Interface Language*, Proceedings of the Eight International World Wide Web Conference, May, 1999,617-630.
[18] Barracuda: Open Source Presentation Framework**,** http://barracuda.enhydra.org/, 2001.
[19] Beck, K., Extreme Programming Explained: Embrace Change (XP Series)**,** Addison-Wesley Pub Co., 1999.
[20] Bennett, B. et al, A distributed object oriented framework to offer transactional support for long running business processes, IFIPIACM Intemational Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000).
[21] Bergsten, Hans, **JavaServer Pages,** O'Reilly, 2000.
[22] Betz, K., Leff, A., Rayfield, J., **Developing Highly-Responsive User Inte$aces with DHTML and Servlets",** Proceedings of the 19th IEEE International Performance, Computing, and Communications Conference -- IPCCC-2000, 2000.
[23] Buschmann, F. et **al, Pattern-Oriented Software Architecture: A System of Patterns,** John Wiley and Sons, 1996, 123- 168.
[24] Coutaz, J., **PAC, An Object-Oriented Model for Dialog Design,** Elsevier Science Publishers, Proceedings of Human-Computer Interaction - INTERACT, 1987,43 1-436.
[25] ***Enterprise JavaBeans Specipcations,*** http://java.sun.com/products/ejb/docs.html, 2001.
[26] ***JAVA PLUG-IN** 1.2 **SOFTWARE FAQ,*** http://java.sun.com/**products/plugin/l.2/plugin.faq.html** , 2001.
[27] Flanagan, David, ***JavaScript: The Definitive Guide,*** 3rd,O'Reilly, 1998.
[28] Vedavys J, "**A Study of MVC-A Software Design Pattern For Web Application Development on J2EE Architecure"** in International Journal of Computer Engineering and Technology(IJCET),ISSN 0976-6367(Print), ISSN 0976-6375(Online), Volume-4,Issue-3,May-June(2013),© IAEME
[29] Vedavys J,**Automation and Testing of Software Design Pattern for e-commerce Web Application Development using J2EE Architecture**" in International Journal of scientific and Engineering Research (IJSER), ISSN 2229-5518, Volume 4, Issue7, July 2013 Edition.
[30] Gray, G and Reuter, A. ***Transaction Processing: Concepts****and* ***Techniques,*** Morgan Kaufmann, 1993.

Appendix-1 Patterns with Applicable and consequences

S.No	*Pattern Name*	*Applicability*	*Consequences*	*Remark*
1	Command Processor	• Separates request for a service from its execution. • Ideal for the development of hyper-controllable applications.	• This pattern provides flexibility in handling requests and their functionalities. • It also allows commands to be executed in separate threads of control. • Implementation of indirections costs storage and time thereby leading to efficiency loss.	A reengineering strategy with Memento, Observer, Visitor and Singleton may replace the functionality of Command Processor.
2.	Document View	• Document holds the core functionality and data, View combines the 'View' and 'Presentation above'	• Suited for 2-Tier Client-Server architecture	---
3.	Document-View-Presentation	• The Document holds the core functionality and the data; view to manage the display (render + accept service request) and Presentation deals with output and user input	• Reuse the rendering output • Pluggable presentation component • Thin user interface • Ideal for multiple window based applications • Increased Complexity	---
4.	Model/View/Controller	• Dividing an application into functionality(model), display(view) and user input(controller)	• Easy maintenance of multiple views of the same model. • Synchronized and 'pluggable' views.. • Uncontrollable number of updates. • Close coupling between views and controllers	The model can be made to skip unnecessary updates.
5.	Navigation Strategy	• Ideal to apply when there is a need to establish a relation between two or more objects	• The pattern encourages dynamic creation and linking of nodes in an	The Prototype pattern can be used to overcome the barriers of the

S.No	*Pattern Name*	*Applicability*	*Consequences*	*Remark*
		at different times. • Used in situations where objects stored in a database are to be retrieved whenever an associated object raises a demand.	active hypermedia environment. • Allows one to define different kinds of links as well as end points. • Used to improve memory requirements by deferring the retrieval of the target code only when needed. • Increased number of objects and communication overhead between the classes involved.	pattern.
6.	NavigationObserver	• Maintenance of navigation history • Maintenance of different viewers for the history • Enabling the backtracking in the navigational path.	• Decouples navigation from its history and history from the display of it. • Provides application independent functionality for the style of viewing the history. • Causes overhead when attempts are made to filter certain types of nodes in the history.	A reengineering effort made by introducing an alternate architecture involving singleton or mediator.
7	Presentation/ Abstraction / Control	• Defines a structure in the form of levels of cooperative agents. Each agent is divided in to Presentation, Abstraction and Control components	• New agents can easily be added / dropped at any time. • Easy implementation of multi tasking • Increased system complexity	Provides a maintainable and extensible structure with clear separation of concepts between different system tasks.

