Pattern Methodology Of Documenting And Communicating Domain Specific Knowledge

Dr.Hari Ramakrishna[1] and Dr.K.V Chalapathi Rao[2]
[1]Chaitanya Bharathi Institute of Technology, Department of CSE, Hyderabad, A.P., India
Email: dr.hariramakrishna@rediffmail.com
[2]CVR College of Engineering, Department of CSE, Ibrahimpatan, R.R.District, A.P., India
Email: chalapatiraokv@gmail.com

***Abstract*— the main objective of this paper is to present pattern methodology of documenting and communicating domain specific knowledge along with a brief historic review of the domain. This paper presents pattern methodology of documenting domain specific knowledge along with few simple framework samples.**

***Index Terms*—pattern, pattern language, frameworks, pattern frames, Graphic frameworks.**

I. INTRODUCTION

Patterns for software development are one of the latest trends to emerge from the object oriented approach. Fundamental to any science or engineering discipline is a common vocabulary for expressing its concepts, and a language for expressing these interrelationships. The goal of patterns within the software community is to create a body of literature to help software developers resolve recurring problems encountered throughout all of software development. Patterns help create a shared language for communicating insight and experience about these problems and their solutions.

The current use of the term *pattern* is derived from the writings of the architect *Christopher Alexander* who has written several books on the topic as it relates to urban planning and building architecture. Although these books are ostensibly about architecture and urban planning, they are applicable to many other disciplines, including software development.

Alexander proposes a paradigm for architecture based on three concepts namely *Quality*, *Gate* and the *Way*. Quality is freedom, wholeness, completeness, comfort, harmony, habitability, durability, openness, resilience, variability and adaptability. The gate is the mechanism that allows us to reach the quality. And the Way is progressively evolving an initial architecture, which then flourishes into a live design possessing the quality. [1]

In 1987 *Ward Cunningham* and *Kent Beck* were working with Smalltalk and designed user interfaces. They decided to use some of Alexander's ideas to develop a small five-pattern language for guiding novice Smalltalk programmers. They presented the results at OOPSLA-87 in Orlando in the paper "*Using Patterns Language for Object-Oriented Programming*". Soon after, Jim Coplien (referred to as Cope) began compiling a catalog of C++ idioms. These are one kind of patterns. Later they are published as "*Advanced C++ Programming Styles and Idioms"*. [2]

From 1990 to 1992 the members of GOF (Erich Gamma, Richard Helm, Ralph Johnson and John Glissades frequently referred as *GOF* Gang of Four) had done some work compiling a catalog of patterns. Discussions of patterns abounded at OOPSLA-91 workshop conducted by *Bruce Andersen.* This was repeated in 1992. Many pattern domain experts participated in these workshops including *Jim Coplien*, *Doug Lea*, *Desmond D'Souza,Norm Kerth*, *Wolfgang Pree* and others.

In August 1993, *Kent Beck* and *Grady Booch* sponsored a mountain retreat in *Colorado*, the first meeting of what is now known as the *Hillside Group*. Another pattern workshop was held at OOPSLA –93 and then in April of 1994, the Hillside Group met again to plan the first conference on Patterns Languages for Program Design (referred as PloP or PloPD). Shortly thereafter the GOF released a book on Design Patterns [3]. Most of the patterns presented in that book are from Erich's Ph.D thesis. Several conferences are continuously organized on this domain.

Software patterns first became popular with the wide acceptance of the book '*Design Patterns: Elements of Reusable Object-Oriented Software'*. Patterns have been used for many different domains ranging from organizations and processes to teaching and architecture. At present the software community is using patterns largely for software architecture, design, development processes and organizations. Though several conference proceedings and books are available on this domain , other books that helped popularize patterns are '*Pattern-Oriented Software Architecture: A system of Patterns* ' (also called as POSA book) by Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal (referred as Gang of Five GoV). Collection of selected papers from the first and second conferences on Patterns Languages for Program Design is released as a book namely "*Pattern Languages of Program Design"*.

At present Patterns are adopted into application domains for example: [4].

i) Patterns in software development generally, including software design, software engineering, and software architecture
ii) Process patterns for management and development processes
iii) Patterns for human-computer interaction (user-interface patterns, or novel modes of interaction)

iv) Patterns for education (ranging from professional training to classroom teaching)
v) Patterns for business and organizations
vi) Modeling patterns, analysis patterns, design patterns
vii) Patterns for object-oriented design, aspect-oriented design, and software design generally
viii) Patterns to describe libraries, frameworks, and other reusable software elements
ix) Patterns for middleware, including distribution, optimization, security, and performance improvement
x) Domain specific patterns and technology specific patterns, as well as generic patterns
xi) Patterns for refactoring and reengineering
xii) Formal models and type systems for patterns
xiii) Programming environments, software repositories, and programming languages for patterns
xiv) The use of patterns to improve quality attributes such as adaptability, evolvability, reusability and cost-effectiveness

II. Patterns And Pattern Languages

Pattern can be defined as *"Reusable solution for recurring problem"*. *Patterns Dirk Riehle and Heinz Zullighoven* define pattern as, *"the Abstraction from a concrete form, which keeps recurring in specific non-arbitrary contexts"*. Another definition of pattern for software community is given as, *"a named nugget of insight that conveys the essence of a proven solution to a recurring problem within a certain context amidst competing concerns".*

Each pattern is a three-part rule, which expresses a relation between a certain context, and a certain system of forces, which occurs repeatedly in that context, and a certain software configuration, which allows these forces to resolve themselves. Alexander defines three-part rule, as *"Each pattern is a three-part rule, which expresses a relation between a certain context, a problem and a solution."*

Cope says that a good pattern does the following:

1) It solves a Problem
2) It is a proven concept
3) The solution is not obvious
4) It describes a relationship
5) The pattern has a significant human component (like comfort, quality of life)

If something is not a pattern, it doesn't mean that it is not good. Similarly if it is a pattern, hopefully it is good but need not be always. Many of the initial patterns focused in the software community are design patterns. The patterns in the GOF book are Object Oriented Design Patterns [3]. There are many other kinds of software patterns beside design patterns, analysis *patterns* published by Marin Fowler and other patterns like *organizational patterns* are also available.

Architectural patterns express a fundamental structural organization or schema for software systems. Design Patterns provide a schema for refining the subsystems or components of a software system or the relationships between them. They describe commonly recurring structure of communicating components that solve a general design problem within a particular context.

Idioms are low-level patterns specific to a programming language. An idiom describes how to implement particular aspects of component or the relationship between them using the features of the given language. The difference between these patterns is in their corresponding level of abstraction.

Riehle and Zullighoven have classified patterns as Conceptual patterns, Design Patterns and Programming Patterns [12].

A collection of patterns forms a vocabulary for understanding and communicating ideas. A *pattern language* is such collection skillfully woven together into a cohesive "whole" that reveals the inherent structure and relationships of its constituent parts towards fulfilling a shared objective [1]. If a pattern is a recurring solution to a problem in a context given by some forces, then a pattern language is a collection of such solutions, which at every level of scale, work together to resolve a complex problem into an orderly solution according to a predefined goal.

Cope defines a pattern language as a collection of patterns and the rules to combine them into an architectural style. Pattern languages describe software frameworks or families of related systems [5, 6, and 7]. In some other context, Cope defines pattern language as a structured collection of patterns that build on each other to transform needs and constraints into architecture.

III. Domain Specific Pattern Languages

The following is a description of view on domain specific patterns and pattern languages as per the domain experts.

The domain-specific patterns are confidential –they represent a company's knowledge and expertise about how to build particular kinds of applications; so references to them are not generally available. One can however more and more of this knowledge will become public over time. In the long run, sharing experience is usually more effective for everyone than trying to hold onto secrets.

The development of complete pattern language is an optimistic but worthwhile goal. Such a language provides solutions to all design problems that can occur in the respective domains. Christopher Alexander claims to have done this in the domain of *Architecture*. Pattern language already exists for small sub-domains of software design, for example the CHECKS pattern language for information integrity [8].

It would be very beneficial to have a pattern language that covers a substantial part of the design space of the respective domains.

IV. Frameworks

The software frameworks are closely related to design patterns and object-orientation. A software framework is

a reusable mini-architecture that provides the generic structure and behavior for a family of software abstractions, along with a context of mimes/metaphors that specify their collaboration and use within a given domain. The following presents views of Brad Appleton on frameworks.

The framework accomplishes patterns by hard coding the context into a kind of "virtual machine" (or "virtual engine"), while making the abstractions open-ended by designing them with specific *plug-points* (also called *hot spots*). These plug-points (typically implemented using callbacks, polymorphism, or delegation) enable the framework to be adapted and extended to fit varying needs, and to be successfully combined with other frameworks. A framework is usually *not* a complete application: it often lacks the necessary application-specific functionality. Instead, an application may be constructed from one or more frameworks by inserting this missing functionality into the plug-and-play "outlets" provided by the frameworks. Thus, a framework supplies the infrastructure and mechanisms that execute a policy for interaction between abstract components with open implementations [9].

The GOF also defines object-oriented frameworks as: *"a set of cooperating classes that makeup a reusable design for a specific class of software"*. A framework provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. A developer customizes a framework to a particular application by sub classing and composing instances of framework classes. For example Microsoft Application framework belongs to this kind.

A framework dictates the architecture of the application. It will define the overall structure, such as partitioning into classes and objects, the key responsibilities, how the classes and objects collaborate, and the thread of control

A framework predefines these design parameters so that the application designer/implementer can concentrate on the specifics of the application. The framework captures the design decisions that are common to its application domain. Frameworks thus emphasize *design reuse* rather than code reuse, though a framework will usually includes concrete subclasses that work directly.

The difference between a framework and an ordinary programming library is that a framework employs an *inverted flow of control* between itself and its clients. When using a framework, one usually just implements a few callback functions, or a few specialized classes, and then invokes a single method or procedure. At this point, the framework does the rest of the work, invoking any necessary client callbacks or methods at the appropriate time and place. For this reason, frameworks are often said to abide by the Hollywood Principle *"Don't call us, we'll call you."* or the Greyhound Principle *"Leave the driving to us."*

Design patterns may be employed both in the design and the documentation of a framework. A single framework typically encompasses several design patterns. In fact, a framework can be viewed as the implementation of a system of design patterns. Despite the fact that they are related in this manner, it is important to recognize that frameworks and design patterns are two distinctly separate entities: a framework *is executable software*, whereas design patterns represent knowledge and experience *about software*. In this respect, frameworks are of a physical nature, while patterns are of a logical nature: frameworks are the *physical realization* of one or more software pattern solutions; patterns are the instructions for *how to implement* those solutions.

The major differences between design patterns and frameworks are as follows: [3].

Design patterns are more abstract than frameworks. Frameworks can be embodied in code, but only examples of patterns can be embodied in code. Strength of frameworks is that they can be written down in programming languages and not only studied but executed and reused directly. In contrast, design patterns have to be implemented each time they are used. Design patterns also explain the intent, trade-offs, and consequences of a design.

Design patterns are smaller architectural elements than frameworks. A typical framework contains several design patterns. The reverse is never true, but one can build patterns for frameworks. Several domain specific patterns are available for designing frameworks. [10]. The San Francisco Frameworks are example of popular frameworks available in the software market [11].

V. FRAMEWORKS SAMPLES

Typical framework samples starting from representation of a framework in UML, Figure 1 presents the UML building block for representing a framework.

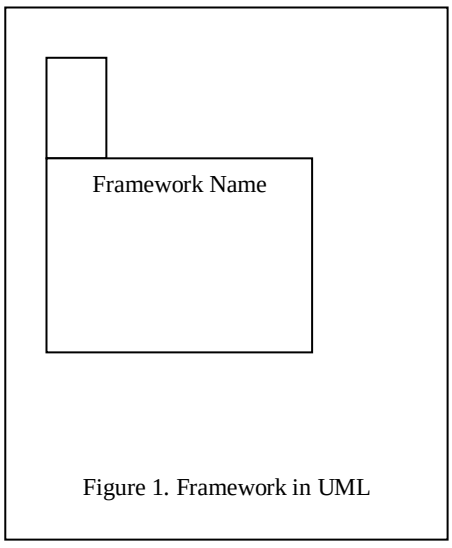

Figure 1. Framework in UML

A simple hello world program in Java uses java framework. This example can be implemented using java Applet. The Java Applet is a part of Java Framework. This in turn depends on AWT and Java language. The following Figure 2 represents a hello world Applet component structure in UML using Java frameworks.

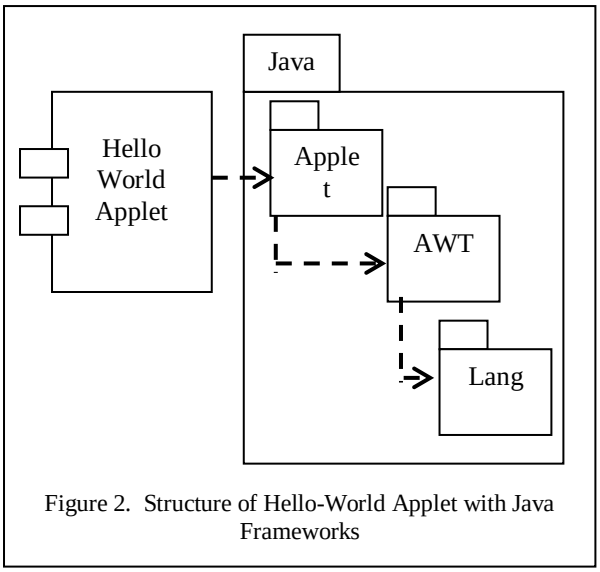

Figure 2. Structure of Hello-World Applet with Java Frameworks

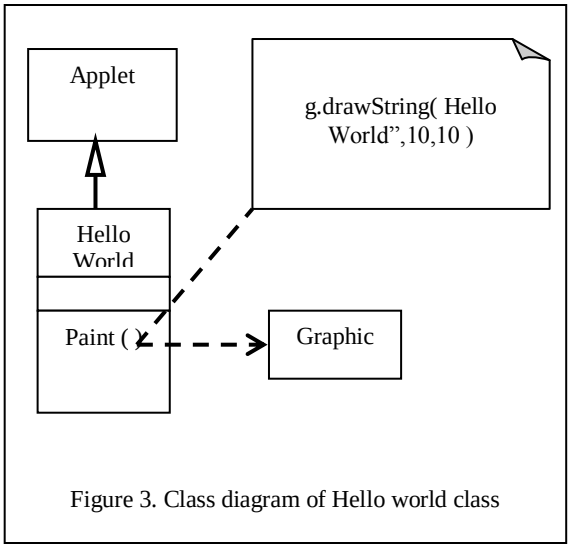

Figure 3. Class diagram of Hello world class

From this diagram, it can be observed that Hello-world applet is depending on Java Applet. The HTML client or Java frame which includes the Hello-World Applet gets Hello-World Interface through the Java Applet. Message from client application will invoke the Java Applet that in turn sends them to the Hello-world Applet. For displaying the hello world message the Hello-world applet implementation again depends on Java graphic library. The class diagram of the hello world example is displayed in the following Figure.3

The following application which uses application wizard of Microsoft demonstrates a better view of anther framework. This application depends and reuses Microsoft Document View Architecture. Microsoft provides Application Wizard for using the framework and class wizard for managing the applications. A simple MFC based application structure in UML is presented in the following Figure 4.

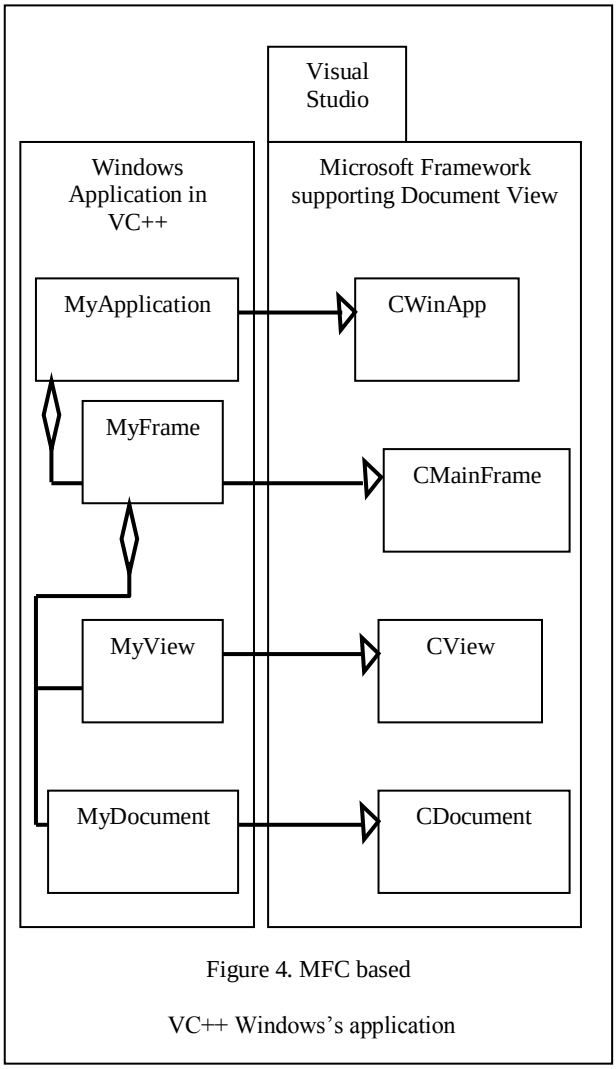

Figure 4. MFC based

VC++ Windows's application

This UML Diagram represents a logical structure of Document View Architecture. The application class of the client module is inherited from CWinApp, a class of Microsoft MFC framework. In fact the Application wizard will decide from which class of CWinApp class group the MyApplication class should be inherited, depending on the requirement of the client specified through application wizard. The user requirements are collected in six steps at the time of creating an application (project workspace) in VC++ through application wizard. The type of project workspace also will change the aggregation-combination, depending on how the user is exporting his functionality.

The ATL technology of Microsoft also provides similar frameworks for supporting Automation layer; component technology and web based computing. Several commercial frameworks available in the market use Microsoft frameworks. These frameworks are referred to as Middleware integration frameworks.[10] Microsoft Visual studio present frameworks for providing web services and other features. These enable the client to use the latest technology without having detailed

knowledge of the hidden technology. Such frameworks enable common use to use complex technology.

Benefits of Object Oriented Application Frameworks are Modularity, Reusability, Extensibility, and Inversion of control. Some of the Challenges of Object Oriented Application Frameworks are Development efforts, Learning curve, Integratability, Maintainability, Validation and defect removal.

VI. Object Oriented Graphic Frameworks

Several pattern languages are available for handling the problems and for documenting and communicating the skill set of a Graphic and CAD developer. A simple graphic framework known as pattern-frame for integrating existing graphic libraries with Microsoft ATL framework is presented briefly as an example.

Name: Middleware integration pattern-frames

Intent: To export object oriented framework into component oriented framework using middleware frameworks.

Motivation and Applicability: The object oriented graphic frameworks require to be ported into new technology *component-oriented technology* for providing better interfaces to the client.

In this *pattern frame*, the object oriented frameworks is ported into component-oriented frameworks through integrating middleware frameworks for supporting component technology.

Structure: The structure of these frameworks is presented in the following UML diagram presented in the Figure 5.

Participants and collaboration: Figure 5. presents mainly three participants.

1. The object oriented frameworks: They are some of the object oriented frameworks, which need to be ported to component technology. In this example a three dimensional object oriented graphic framework with all user required graphic functionality (domain specific functionality) is selected for porting in to COM technology.
2. The Middleware Integration frameworks: These frameworks defines interface for exporting the functionality of object oriented framework and aggregate the object oriented frameworks to form black box. They will depend on Middleware component frameworks like Microsoft ATL, and Java beans frameworks. They basically provide automation layer. Few VB command are presented in table 1.
3. The Middleware component framework: This is a framework used to build components, supporting component technology. Examples of middleware component frameworks are Java Beans development kit, Microsoft Active Template Libraries.

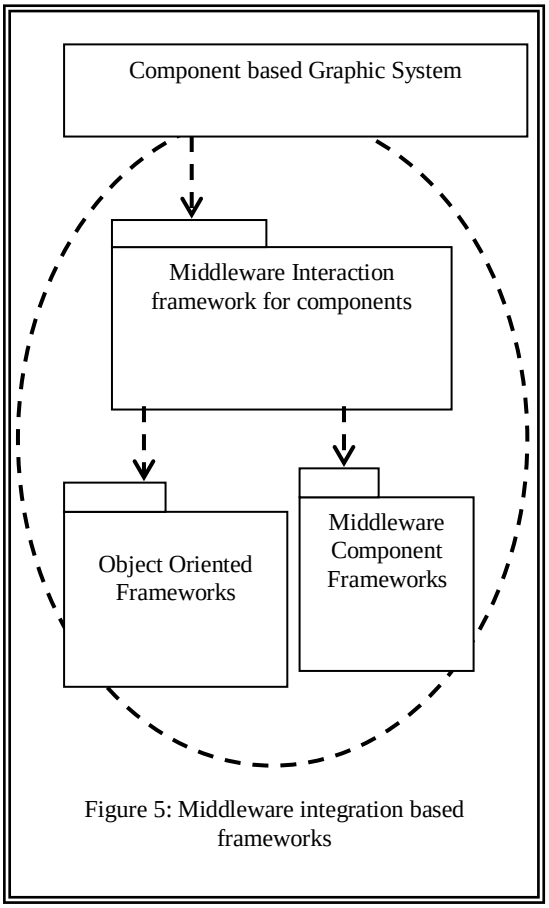

Figure 5: Middleware integration based frameworks

A. Implementation and Code:

The selected object oriented graphic framework can be exported to VB client using Active X controls. Building a graphic component is possible by integrating traditional graphic framework with Active X controls, which is a middleware framework. This will export traditional graphic framework functionality to VB client. Table1 shows sample VB front-end application commands used to control the graphic component frameworks. Sample view of the e VB application is presented in the Figure 6. HGP3D1 is the name of the framework component.

Table I.
Few VB Statements used in the application

HGP3D1.Sp3d Text1.Text, Text2.Text, Text3.Text, Text4.Text
HGP3D1.RoteteSegmentAbs Text7.Text, Text4.Text, Text5.Text, Text6.Text
HGP3D1.ShowAll
HGP3D1.CloseSegment Text7.Text
HGP3D1.RotateSegRel Text7.Text, Text4.Text, Text5.Text, Text6.Text

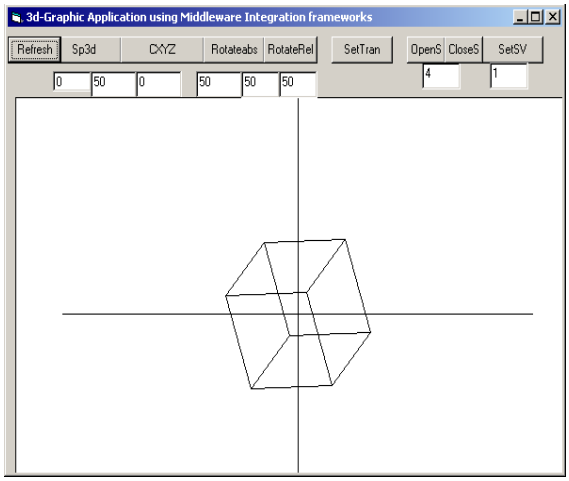

Figure: 6 A Three Dimensional graphic Framework

A sample output of the application used to simulate PCB board using same framework is presented in the following Figure 7.

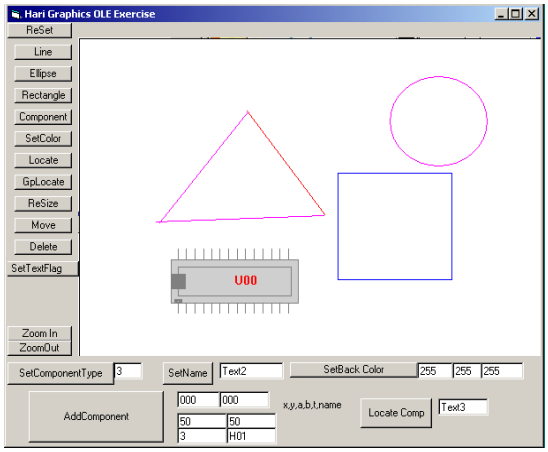

Figure 7. A graphic Frameworks for simulating PCB

Several framework patterns for the development of Graphic frameworks are presented in the following section.

VI. PATTERN LANGUAGE FOR GRAPHIC FRAMEWORK

Problems in evolving Graphic frameworks can be documented adapting pattern approach for provide solutions. Sixteen typical common design and implementation issues are identified and they are classified into three groups depending on nature of the issues namely *object oriented*, *component oriented* and *distributed & web based pattern frames*. The following are a catalog of few pattern frames [10].

A. Object Oriented pattern Frames

These are based on simple object oriented patterns. They make lightweight frameworks. They provide solutions for under engineering problems for developing frameworks.

Table II.

Object Oriented Pattern Frames

Name	Intent
TRADITIONAL GRAPHIC FRAMEWORKS	This will apply traditional graphic techniques for building frameworks
FUNCTION CLASS FRAMEWORKS	This will apply basic object oriented patterns for building configurable function classes
FOUNDATION CLASS FRAMEWORKS	This will provide hot spot object libraries for reusing most common modules of the domain
WHITE -BOX FRAMEWORKS	This will generate object library for configurable generic domain specific classes
FLYWEIGHT OBJECT FRAMEWORKS	This will decrease number of classes and number of objects in a system

B. Component Oriented Pattern Frames

These frameworks are based on Component technology. All are black box frameworks. They provide solutions for building Component based application frameworks.

Table III.

Component Oriented Pattern Frames

Name	Intent
MIDDLEWARE INTEGRATION BASED FRAMEWORKS	This will reuse middleware integration frameworks for building Enterprise frameworks
Component based frameworks	This will apply patterns defined on components for providing black box frameworks
Abstract Component frameworks	This will generate black box framework components using simple object oriented primitive patterns
Component wrapper frameworks	This will apply simple object oriented primitive patterns for using black box frameworks

C. Distributed nad Webbased Pattern Frames

These are useful for building frameworks for supporting typical distributed and web based application requirements.

Table IV.

Distributed Web based Pattern Frames

Name	Intent
DISTRIBUTED FRAMEWORKS	This will provide environment for building domain specific distributed application components
WEB ENABLED FRAMEWORKS	This will provide environment for building web enabled applications
WEB BASED FRAMEWORKS	This will provide environment for building web based applications

The catalog of frameworks listed above form a pattern language for building frameworks. Some of the frameworks are more general in the sense that they are applicable in other domains. But a few frameworks are specific to Graphic, CAD and GIS systems. This pattern language starts its journey from a simple function country to a complex component world.

CONCLUSIONS

The pattern methodology is useful for documenting, communicating skill set of expert knowledge for the purpose of reuse. Development of patterns, pattern languages and frameworks are essential for every domain for enabling complex technology useful to common user and for the reuse and communication of domain expert skill set.

Using pattern at unrequited conditions create over engineering problem. Not adapting any such methods leads to under engineering problems. Extreme Programming referred as XP provides solutions for such problems.

REFERENCES

[1] Christopher Alexander, "An Introduction for Object-oriented Design", A lecture Note at Alexander Personal web site ***www.patternlanguage.com.***

[2] Pattern Languages of Program Design. Edited by James O. Coplien and Douglas C. Schmidt. Addison-Wesley, 1995.

[3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, "Design Patterns: Elements of Reusable Software Architecture", Addison-Wesley, 1995.

[4] LNCS Transactions on Pattern Languages of Programming http://www.springer.com/computer/lncs?SGWID=0-164-2-470309-0.

[5] Foote B, Yoder J. "Attracting Reuse". Third Conference on Pattern Languages of Programs (PLoP'96), Monticello, Illinois, September 1996

[6] Foote B, Opdyke W. 'Life Cycle and Refactoring Patterns that Support Evolution and Reuse'. First Conference on Pattern Languages of Programs (PLoP'94). Monticello, Illinois, August, 1994.

[7] Roberts,Don, et Ralph Johnson, Evolving Frameworks A Pattern Language for Developing Object-Oriented Frameworks, Proceedings of Pattern Languages of Programs, Allerton Park, Illinois, September 1996 (PLoP '96), Addison-Wesley, 1997.

[8] "CHECKS Pattern Language of Information Integrity" at http://c2.com/ppr/checks.html.

[9] Durham A, Johnson R. "A Framework for Run-time Systems and its Visual Programming Language". Proceedings of OOPSLA '96, Object-Oriented Programming Systems, Languages, and Applications. San Jose, CA. October 1996.

[10] Dr.Hari Ramakrishna, "Design Pattern for Graphic/CAD Frameworks", Ph.D thesis submitted to Faculty of Engineering Osmania University March 2003, "Architecture of the San Francisco frameworks"–IEEE eeexplore.ieee.org/iel5/5288519/5387143/05387145.pdf.

[11] Dirk Riehle and Heinz Züllighoven "Understanding and Using Patterns in Software Development" http://www.ubilab.com/publications/print_versions/pdf/tapos-96-survey.pdf.