A Perception on Programming Methodologies for Software Development

P.K. Singh
AMITY University, Noida, U.P.,

Parag Mittal
JRE Group of Institutions, Gr. Noida, U.P.,

Lakshay Batra
JRE Group of Institutions, Gr. Noida, U.P. l

Utkarsh Mittal
JRE Group of Institutions, Gr. Noida, U.P.,

ABSTRACT
In this paper some of the basic programming methodoligies are covered that are used in day to day life for developing various programs. It describes about what each paradigm is all about, what are the advantages and disadvantages of using that paradigm, it occupies a major part in describing features that will make the reader more comfortable in choosing the platform on which he wishes to develop his program. Moreover it even include list of various programming languages that follow a particular methodology. The main purpose of this paper is to help the reader in choosing the most appropriate platform or methodology for developing his program. It will even benefit the readers by creating interest in exploring new paradigms by giving them a small yet relevant synopsis about the paradigm.

GENERAL TERMS
Software Development Models, Programming Methodologies, Software Development, Agent Oriented Softwares, Aspect Oriented Softwares

KEYWORDS
Software Methodologies, Software Development, AOSD, Agent Oriented.

1. INTRODUCTION
A software development methodology or system development methodology in software engineering is a method which is used to design and develop a mechanism of fostering an information system. The software development methodology (also known as SDM) framework didn't emerge until the 1960s. According to Elliott (2004) the systems development life cycle (SDLC) can be considered to be the oldest formalized methodology framework for building information systems [26]. The primary motive behind SDLC has been to carry out the advancement of information systems in a calculated manner and procedural way, demanding every phase of life chain from birth of the belief to dispatch of the terminating system, to be carried out thoroughly and gradually" within the ambience of the scheme being implemented. The essential concept behind this approach scheme in the 1960s was to establish enormous scale operative business arrangement in an era of extensive business group. Today an abundance of software structures are present from structure programming to object oriented, components, aspect oriented and agile methodologies. Each one of them is having its own role and specific to different types of software applications. Major emphasize in this paper is to investigate the main role of each methodologies in detail with its usage related to specific applications. This paper is organized as follows: Section 2 provides basic research procedure used to conduct this analysis. Section 3 reports the detail of each software methodologies in details with their associated impacts on software applications. Major findings with conclusion and future scope are presented in Section 4.

2. RESEARCH METHODOLOGY
Defining an adequate search string is quiet difficult for analysis. Most of times identify the string for search relies on the experience of the involved researchers [30]. According to our research survey, we defined the following string:

(Software Methodologies) OR (Programming Methodologies) OR (Software Frameworks)

The sources of primary studies vary from indexed repositories (IEEE, ACM Digital Library, Elsevier, Science Direct, ICST, IJCSE, ICIIP, IJCA etc) to general purpose search engines(Google and Scirus). We have downloaded 105 papers out of which, we have considered the related papers on software methodologies. We have limited to our findings for research papers because we want to analysis the features of several methodologies. Each paper is analyzed based on the related methodologies and finally conclusion is drawn based on the literature reported in the published papers.

3. SOFTWARE METHODOLOGIES
There is number of software methodologies reported in the literature, we have considered the most popular ones for analysis in this research paper.

3.1 Structured Oriented
Structured based programming aims clearly on improving the developing time, quality and clarity of the program by using sequence, selection and repetition statements. This includes use of statements like if-else, endif, loops etc. which make the program complex and difficult to maintain and is commonly called as spaghetti code. A language is described as structured when it’s part of syntax is enclosed in between curly braces that are preceded by a Keyword. The programmer basically breaks his source code into logically structured chunks. Writing a structured program requires more time and reusability is not provided, which leads it’s to more maintainable software’s. ESPOL was reported one of the first structure programming language, developed by Burroughs Corporation in 1961. Afterward PL/I, PL 360 then C came into existence in 1969 developed by Dennis Ritchie.

The primary cons that are present in structured programming are absence of encapsulation, information hiding, and recurrence of code hence comprising an extensive arrangement than is needed. Governing a fault may be difficult to handle; debugging attempts can be hamper since the obstacle code will look correct and even execute appropriately in one instance of the program but not elsewhere. Another disadvantage is that the extensive use of GoTo statement i.e. the fact of their jumping around in the code is mainly dependent on the line number [1], if you add a line in between it can destroy the working of your code. The main advantages of using structured programming are that it more secure and reliable codes. C is one of most popular structures programming language and best suitable language for system programming till date. Whenever it comes to performance, C is unbeatable. C provides you the direct access to memory of your CPU that’s why most of the device drivers are written in C language. Major parts of the Windows, Unix and Linux are still written in C, which makes it still one of the best language for system programming.

3.2 Object Oriented

All the major concepts were developed in the 1960s as part of a language called Simula. Alan Kay and his group developed a programming language named Smalltalk in the 1970s. Bjarne Stroustrup developed an extension to the C language that eventually evolved to the language C++. Explosion of the research in object-oriented programming techniques began in the first major conference on object-oriented programming in 1986, there were dozens of languages like Eiffel, Objective-C, Actor, Object Pascal, and various Lisp dialects etc. Later to this Java emerge as the most effective and popular object oriented language. The Java compilers, virtual machines, and class libraries were originally developed by Sun Microsystems from 1991 and first released in 1995 [2, 28]. As of May 2007, in yielding with the condition of the Java Community Process, Sun reauthorized most of its Java technologies under the GNU General Public License. Java initially developed based on the key features security, architecture neutral, portable, high performance, multi threading and dynamic programming. This is based on the byte code concept with the JVM (Java Virtual Machine) concept. It released in various version starting from JDK 1.0 to Java SE 7 till 2011. Object oriented methodologies was developed to overcome the problems of structured programming techniques. One of the most important features of objected oriented approach is the option to modify existing code or solution to solve different problems. Some of the basic concepts of object oriented programming are objects, attributes, methods, events, abstraction and classes, constructors. There are several features that are makes its one of the popular language among programmers such as inheritance, polymorphism, abstraction, encapsulation and reusability. (i)Inheritance: It is the process of deriving a new class from existing class. it allows reusability and extension. The sub class or derived class gets all the properties of super class or parent class. It helps in reducing code size. (ii) Polymorphism and Overloading: Polymorphism is ability of different objects to respond differently to identical messages. It is a implementation concept related to objects. Overloading is a kind of polymorphism. (iii) Encapsulation: It is the process of binding data and functions together in a object [3]. Its basic aim is providing data hiding and ensures security. (iv) Data abstraction: It helps to highlight or point out the essential aspects of an application. It also facilitates reusability. (v) Reusability: Reusability is re- usage of structure without changing the existing one but adding new features or characteristics to the existing structure. The design procedure that is used in procedural programming is Top down design [4]. However, the design procedure used in object oriented programming is bottom up and top down both. Java is one of the accepted programming languages today for many reasons. It is a co-ordinate language with a strong library of reusable software components. The programs in Java can be executed on many different computer architectures and operating systems due to of the use of the JVM (Java virtual machine). It is also called as code portability or WORA (write once, run anywhere). Java is expected to be tutored in university computer science classes. Multiple computer science books written in the past decade use Java in the code references. Therefore acquiring knowledge of Java syntax is beneficial even though you never code in it. Java advantages include WORA, acceptability and their major disadvantage is that it is lagging than naturally compiled languages.

The aim that software must be divided into different components – structured from prefabricated components - first became noticeable with Douglas McIlroy's talk at the NATO conference on software engineering in Garmisch, Germany, 1968, labeled Mass Produced Software Components [32]. McIlroy's further attachment of pipes and filters into the UNIX operating system was the first implementation of an infrastructure for this idea. Brad Cox of step stone explained the modern concept of a software component. The infrastructure and market for these components was created and set up by Brad Cox by inventing the Objective-C programming language and called them Software ICs. This view was summarized in a book on OOP - An Evolutionary Approach 1986. In early 1990s IBM guided the path with their System Object Model (SOM). In response, the actual deployment of component software with COM and OLE was paved by Microsoft. As of 2010 many successful software component models exist. Development based on Component accentuate on the separation of concerns in regard with the wide functionality present end to end in a software system. For defining, implementing and composing loosely coupled independent components into systems a reuse-based approach is implied. Both short term and long term benefits for software are achieved via this practice .It employs principle of object oriented programming. Properties of object are encapsulated by each component. A component framework defines dynamically loadable and independently developed components, instead of classes that are colligated together.

Component frameworks can be used without access to their source code and can be extended through composition thus Component frameworks are also known as black-box frameworks.There are various facilities provided by component based software’s such as (i) Information Hiding: The most important characteristic is that components completely hide their implementation (ii) Context Independence Components can be transformed into different application contexts. Components have to be self-contained elements, independent from other components[11] (iii) Implicit invocation: Components address each other not directly but indirectly, a central registry stores information about components as well as interfaces [12].

3.3 Agent Oriented

The concept of Agent-oriented programming (AOP) and the guidelines of incorporating software around the concept of agent was first used by Yoav Shoham in year 1990 in his Artificial Intelligence studies [29, 31]. AOP is a sub set of OOP. AOP agents show commitments, alternatives, beliefs and communicate with one another via constrained set of speech type acts like notify, promise, request and decline the state of the agent is its intellectual state. The basic unit of agent oriented programming is Agent. The following properties are observed in an Agent (i) They are autonomic (ii) They share and communicate with other agents (iii) They can sense the environment and according to change in environment they change their responses. Three main components of Agent-oriented are (i) Formal language with clear syntax and semantics for describing mental state (ii) Interpreted language in which to define and program agents(iii)An "identifier" to convert neutral devices into programmable agents [5]. This approach is based on concept of software agent which represents the central meaning of abstraction in AOSE. [6]. Multi-agent systems tend, by their very nature, to be distributed — the idea of a centralized multi-agent system is an oxymoron [7,8]. Table 1 represents the basic features if OOP and Agent Oriented methodologies.

Table 1. Features of Agent Oriented Methodologies

Features	OOP	Agent Oriented
Basic Unit	Object	Agent
Parameters defining state of basic unit	Unconstrained	Beliefs, commitments, capabilities, choices…
Types of messages	Unconstrained	Inform, request, offer, promise, decline…

3.4 Aspect oriented Programming

Aspect oriented programming is a methodology with multiple crosscutting concerns or aspects. The idea behind AOP is that it separates the concern (similar to encapsulation). The explicit concept of AOP was developed by Gregor Kiczales and colleagues at Xerox PARC, and this led with the AspectJ AOP extension to Java [9, 32]. AOP provide better modularity in programs, which is basic prerequisite in software engineering discipline and it can also reduce the development effort, testing time and provide better reusability and maintenance as compare to OOP in several aspects [30]. It's built upon the existing programming methodologies. It expresses each concern in its own module i.e. called as Aspect. Here concern is a particular goal or area of interests that must satisfy the overall system goal. CrossCutting Concerns are the concerns that cannot be implemented without scattering codes (Duplicate/Complementary code blocks). Aspect is a modular unit designed to implement a concern. There are series of languages belong to AOSD i.e. AspectJ, AspectJ, CeaserJ and Aspect C++ etc. AspectJ is the most popular language and used by most among the aspect family of languages. AspectJ is widely accepted by researchers because it is mature and code is available to prove some framework proposed on AOSD. Some of the similarities with OOP are shown in Table 2.

Table 2. Similarities between OOP and AOP [10]

OOP	AOP
Class – encapsulates methods and attributes	Aspect – encapsulates pointcuts, advice & attributes
Method signature – defines entry points for execution of methods	Pointcuts – defines the set of entry points in which advice is triggered
Method bodies – implementation of primary goals	Advice – implementation of crosscutting concerns
Compiler – converts source code to byte code	Weaver – instruments code with advice

In spite of the popularity of Aspect Oriented methodologies, it is still immature and need more investigation. Aspect is not the popular choice among software industry because some the issues related to the maintenance, performance, reusability are still open. Still there are some drawbacks associated with it such as poor tool supports, debuggers, profilers etc. Maintenance and debugging is too complex and difficult to adapt but still it have several advantages of AOP (i)Reduces technical complexity (ii)Easy to Use once get use to (iii) Allows codes and functions to bind together in one block (iv)Useful in solving problems like logging, security.

3.5 Goal Oriented

Goal oriented programming methodology makes use of languages that have very high level of abstraction. These are typically used as professional programmer's productivity tool. This type of paradigm is restricted to only a specific type of application thus referred to as Goal Oriented. GOAL first appeared in the books based on investigation studies on primary school children during 1970s and 1980s (as cited in Payne, Youngcourt, and Beaubien (2007)). Incorporating Atkinson's theory of achievement motivation (1964), researchers were biased in demonstrating the differences in classroom learning styles. It focus on developing software agents that perform in a goal directed way and are capable to dynamically switch from one character to another, to dodge failure in accomplishing their own goals or to meet requirements in a quality way [13]. Goal-oriented programming is targeted at dynamic domains such as agent based systems in which the programmer does not have full control over all aspects of system behavior [14]. Goal-oriented programming is abounding. Monitoring the execution for goal achievement does not add anything as the program was developed to satisfy the goals. In the Goal-Oriented Execution model, an agent A is defined through the following tuple: A = (KB; SS; CP; GS) [15] where: KB is the agent Knowledge Base, SS describes Set of Services offered by the agent. The agent uses these services to achieve its goal; these services can be shared by other agents too for accomplishing of the goal. CP is the set of Compiled Plans given by the agent to achieve its goals.SG represents the Goals that are to be achieved by the agent.It adds flexibility in handling failures.Goal-oriented Requirements Language (GRL), is an i*-based modeling language applied in system development, is executed to strengthen goal-oriented modeling and reasoning about both functional and non-functional requirements. It contributes constructs for exhibiting various types of concepts that appear during the specification process [33, 34].

3.6 Service Oriented

Service-oriented architecture (SOA) is a software developing methodology built on structured collections of discrete software modules, known as services, that together provide the complete functionality of a huge software application [16]. As from 2008 network managers are applying the principles of SOA in their area. Various remarkable examples of service-oriented network management architectures include M.3060: Principles for the Management of Next Generation Networks recommendation from the ITU-T and TS 188 001 NGN Management OSS Architecture from ETSI. SOA is a technique of software architecture for building software in the form of interoperable services. To interface with SOA XML and JSON are used. Loose coupling of services with operating systems and other application governing technology is required by SOA. SOA divides functions into distinct blocks, or services, those developers make accessible over a network in order to allow multiple users to combine and reuse them in the development of applications. These resources and their corresponding consumers interact with one another by transferring (passing) data in a well-defined, shared format, or by handling an activity between two or more resources [17]. Major requirements for adequate usage of SOA are to be fulfilled are as follows: (i) Interoperability passing data in a well-defined, shared format, or by coordinating an activity between two or more services [17]. Major requirements for adequate usage of SOA are to be fulfilled are as follows: (i) Interoperability among different systems and programming languages (ii) Need of creating a federation of resources, data flow must be secured and maintained to a federated database system which as a result permits new functionality augmented to point out a common business format for each data element [17].

The following principle for service-oriented design was given by The Microsoft Windows Communication Foundation team [17]. Services are autonomous, Boundaries are explicit, services share contract and schema, not class, service compatibility is based on policy define by Thomas Erl of SOA Systems Inc [18],it explained some specific service-orientation principles that are common to all SOA platforms. The principles stated by Thomas Erl were published in "Service-Oriented Architecture: Concepts, Technology, and Design": Standardized service contract, Service loose coupling, service abstraction, service reusability, Service abstraction, Service reusability, Service autonomy, Service statelessness, Service discoverability, Service composability. Authors also included the following principles: Service granularity, Service normalization, Service optimization, Service relevance, Service encapsulation, Service location transparency. There are various types of SOA; Service Architecture, Service Compostion Architecture, Service Inventory Architecture, Service-oriented enterprise Architecture. It provides several benefits; SOA helps businesses responding more quickly and makes it more cost-effective to the changing market conditions [19], SOA mainly promotes its ease of reuse, SOA helps in attaining the objective of separating users. This helps in maximizing the reuse of services; It reduces the interconnection to—and usage of—existing IT legacy. Some other usages may be considered as SOA quite adoptive in designing are as follows: (i) Loose coupling; language independence, helps interface with legacy system. (ii)Distributed; manage load, failover for reliability (iii) Asset management; Leverage existing resources, creates assets, separates team[20]. It offers all these benefits along with some restrictions such as SOA is not always the best architectural choice because best utilization of SOA needs additional development and designs along with infrastructure which increases the cost. Web Services and Service Oriented Architecture is not suitable for applications because of the following reasons: (i) Stand alone, non distributed applications which do not require application or component integration (ii) Applications which have limited scope (iii) Applications in which loose coupling is not required and one way asynchronous communication is required or desired iv)Homogenous application environments for example, an environment in which all applications were made by using J2EE components. In the instances mentioned, it is recommended to use XML over HTTP for inter-component communications instead of utilizing Java remote method invocation (v) Applications which require GUI based functionality. Such an application is not suited for heavy data exchange that is service based. It is widely accepted in several applications like i) E-commerce (ii) E-business (iii) M-commerce (iv) E-entertainment (v) E-learning (vi) E-government (vii) E-health etc. Various tools which are used for managing SOA infrastructure are: (i) HP Software & Solutions (ii) HyPerformix IPS Performance Optimizer (iii) IBM Tivoli Framework (iv) Red Hat JBoss Operations Network (v) Oracle SOA Management Pack Enterprise Edition.

3.7 Autonomic Computing

Autonomic Computing refers to the self-management of computing systems, adapting to unpredictable changes while hiding intrinsic complexity to operators and users. It was started by IBM in 2001.The main purpose of Autonomic Computing was to make computer systems skilled enough for self-management to remove the increasing complexity of computing systems management. An autonomic system is a system which is capable of making decisions on its own, using high-level policies; it will continuously keep checking and optimizing its status and automatically adapt itself to changing conditions [22]. Central idea is based in Context-oriented Programming. IBM defined the following four functional areas: (i) Self-configuration: Components are configured automatically, (ii) Self-healing: Faults are automatically detected and corrected (iii) Self-optimization: Resources are controlled and monitored automatically to confirm the proper functioning as per exact requirements (iv) Self-protection: Recognition and shield from arbitrary attacks. IBM defined five evolutionary levels in the automatic deployment model: Level 1 presents the ongoing situation where systems are to be configured manually. Levels 2, 3 & 4 basically define the functions for automated management and level 5 presents the goal of autonomic computing or self-managing computing systems. The design complexity of Autonomic Systems can be uncomplicated by using design patterns such as the model-view-controller (MVC) pattern to enhance concern separation by encasing functional concerns [21]. This approach can be integrated in distributed systems so that the main controllers (human) do not have to worry about the small errors and issues. System should be capable of solving as many problems as possible on its own. Therefore we need a system which could ease the management of operations automatically. Autonomic Computing is one of the best answers in this area [27].

The actual idea used in Autonomic Systems is Closed Control Loops. This idea evolves from the Process Control Theory. A closed control loop in an Autonomic System invigilates the hardware or software component and automatically keeps its parameters in the defined range. It posses several characteristics such as; The behavior of autonomic systems differ from system to system but even then the system should at least illustrate some properties to achieve its motive (ii) Automatic: This means to be able of controlling the internal functions as well as various operations (iii) Adaptive: It should be capable of changing the operation as per requirement (i.e. configuration, state and functions) (iv) Aware: An autonomic system must be able to monitor (sense) its operational context as well as its internal state in order to be able to assess if its current operation serves its purpose [22]. It offers several benefits; (i) The most important feature of autonomic computing is reduced TCO (Total Cost of Ownership) (ii) Maintenance cost is scaled down as there are less breakdowns (iii) Less number of people or IT personnel will be needed to handle the computing systems (iv) Reduced deployment (v) Stability of IT systems is increased by Automation or Autonomic Computing (vi) Autonomic Computing provides maximization of system availability, thus minimizes human effort to manage large servers and also cost of maintenance. The major characteristics of this computing associated with its metrics are presented in Table 3.

Table 3. Autonomic Computing Characteristics [23]

Characteristics	Metrics
Self-configuration	Maintainability, Usability, Functionality, and Portability
Self-healing	Reliability and Maintainability
Self-optimization	Efficiency, Maintainability, and Functionality
Self-protection	Reliability and Functionality
Self-awareness	Functionality
Openness	Portability
Context-awareness	Functionality
Anticipatory	Efficiency and Maintainability

There are number of applications reported in literature, most important are; (i) It simply reduces the complexity of computing management systems, (ii) It provides the basis for Grid Computing and E-Sourcing (iii) It balances the server load (iiv) Also, it helps in process allocation, memory error-correction, automatic updating of drivers and related software, monitoring power supply, pre-failure warning, automated system recovery and backup etc. Context-oriented Programming (COP) joins layers; intense activation based on context, and scoped activation. Various combinations of these approaches are already there, some of them are very old also. The separation of programs into layers is a approach that goes back to early experiments made in Smalltalk [35]. There, layers were accepted as a component to express different design options of the same software, and permit the developer to apply various combinations thereof. Delegation layers and Slate are very close to COP in that they are based on layers, allow sending messages in the scope of a specific layer, and even allow manual dynamic composition of layers. ContextL, ContextJ, ContextS are popular variants of COP. These language abstractions enable non-trivial and efficient implementations.

3.8 Agile Methodologies

Agile methodology is basically used in software development in software industry worldwide. It helps teams react to uncertainty through additional, repeated work cadences, known as sprints [24]. Lightweight agile software development methods came up in the mid-1990s as a reaction against the heavyweight waterfall-oriented methods, which were described by their experts as being heavily regulated, controlled, micromanaged and excessively additional approaches to development. Early practices of agile methods include Rational Unified Process (1994), Scrum (1995),Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These practices are now altogether referred to as agile methodologies, after the Agile Manifesto was stated in early 2001. This methodology is a substitute to waterfall or the traditional sequential development technology. It is a cluster of software development methods that are based on repeated and additional development, where necessities and answers come in existence through association between self-organizing, cross-functional teams. Some basic key features it provides are delivering frequently more iterations, less defects, test frequently, collaborative approach and maximum ROI [25]. There are number of advantages with agile;(i)Agile methodology is the saving of time and money(ii) No detail requirement needed(iii) Early benefit to the user/business(iv) Face to face communication (v) Agile facilitates smooth flow of knowledge sharing(vi) Less time to market(vii) Less cost to customer and high quality. It come with some limitations as follows: **(**i) Smaller Planning Horizon (ii) Lesser design and documentation (iii) Need clear customer vision (iv) Necessity of experienced and senior resources. Well-known agile software development methods include; (i)Agile Modeling (ii) Agile Unified Process (AUP) (iii) Crystal Clear(iv) Crystal Methods (v) Dynamic Systems Development Method (DSDM) (vi) Extreme Programming (XP) (vii) Feature Driven Development (FDD) (viii) GSD (ix) Kanban (development) (x) Lean software development (xi) Scrum (xii) Velocity tracking. It is widely accepted methodologies among the software developers because of its flexibility features during development and manages the projects easily through it usages. However, according to the researcher the agile methods seem far better and outdo other methods for developmental and non-sequential projects. It is believed by many organizations that agile methodologies are too extreme and follow a hybrid approach that fuses elements of agile and plan-driven approaches.

4. CONCLUSION AND FUTURE SCOPE

In this paper several methodologies starting from the structural programming, object oriented methodologies to finally agile software development with its associated pros and cons were reported. Number of benefits offered by these frameworks along with their limitations is presented in this paper. Major emphasize is given on identifying the key concepts behind each framework and its usages. Data analysis based on their usages related to several applications may be carried out to know its better effectiveness and acceptability among worldwide developers is part of future work. This paper will helps the researchers on providing the brief idea on software development frameworks and provide an overview of each methodologies strengths and its usages specific to several applications.

5. ACKNOWLEDGMENTS

We would like to express our deepest thanks to **AMITY University, Noida** and **JRE group of Institutions, Greater Noida, Uttar Pradesh** for providing us research facilities and environment.

6. REFERENCES

[1] http://wiki.answers.com/Q/What_are_the_advantages_and_disadvantages_of_structured_programming

[2] Byous, Jon. "Java technology: The early years." Sun Developer Network 1998.

[3] http://www.exforsys.com/tutorials/programming-concepts/features-of-oop.html

[4] http://www.ctp.bilkent.edu.tr/~russell/java/LectureNotes/1_OOConcepts.htm

[5] www.cs.ucf.edu/~lboloni/Teaching/EEL6938_2007/AOP

[6] Mubarak, Hisham. "Developing flexible software using agent-oriented software engineering." Software, IEEE 25.5 (2008):12-15.

[7] Wooldridge, Michael, and Nicholas R. Jennings. "Pitfalls of agent-oriented development." Proceedings of the second international conference on Autonomous agents. ACM, 1998.

[8] http://en.wikipedia.org/wiki/Agent-oriented_programming

[9] http://en.wikipedia.org/wiki/AOP

[10] http://www.authorstream.com/Presentation/aSGuest11429-142463-aspect-oriented-programming-entertainment

[11] http://en.wikibooks.org/wiki/Computer_Programming/Component_based_software_development

[12] http://en.wikibooks.org/wiki/Computer_Programming/Component_based_software_development

[13] Morandini, Mirko, Frédéric Migeon, Marie-Pierre Gleizes, Christine Maurel, Loris Penserini, and Anna Perini. "A goal-oriented approach for modelling self-organising MAS." In Engineering Societies in the Agents World X, pp. 33-48. Springer Berlin Heidelberg, 2009.

[14]]http://awesome007.disi.unige.it

[15] Palanca, Javier, et al. "Distributed goal-oriented computing." Journal of Systems and Software 85.7 (2012): 1540-1557.

[16] Velte, Anthony T. (2010).Cloud Computing: A Practical Approach. McGraw Hill. ISBN 978-0-07-162694-1.

[17] http://en.wikipedia.org/wiki/SOA

[18] Microsoft Windows Communication Foundation team (2012). "Principles of Service Oriented Design". msdn.microsoft.com. Retrieved September 3, 2012.

[19] C. Koch, "A new blueprint for the enterprise." CIO Magazine 5.4 (2005): 1-8.

[20] http://cs.simpson.edu/files/DAMA_Presentation.pdf

[21] Xiang-xi Meng; Ya-sha Wang; Lei Shi; Feng-Jian Wang, "A Process Pattern Language for Agile Methods," Software Engineering Conference, APSEC 2007., pp.374-381, 4-7 Dec. 2007.

[22] http://en.wikipedia.org/wiki/Autonomic_computing

[23] Nami, Mohammad Reza, Koen Bertels, and Stamatis Vassiliadis. "Autonomic Computing Systems: Issues and Challenges." 17th Annual Workshop on Circuits, Systems and Signal Processing, published in 2006.

[24] http://agilemethodology.org/

[25] http://www.rsrit.com/Documents/AgileMethodology_ReliableSoftware.pdf

[26] J. Strachan & G. Elliott 2004 Global Business Information Technology. pp.87.

[27] Pradeep Kumar Singh, Arun Sharma, Amit Kumar and Ayush Saxena "Autonomic Computing-A Revolutionary Paradigm for Managing Self Managing Systems", in the proceeding of IEEE, Dec. 2011.

[28] Object-oriented programming "The History of Java Technology". Sun Developer Network. ca. 1995. Retrieved 2013-04-30.

[29] Shoham, Y. (1990). Agent-Oriented Programming (Technical Report STAN-CS-90-1335). Stanford University: Computer Science Department.

[30] P.K.Singh, O.P.Sangwan and Arun Sharma, "A Systematic Review on Fault Based Mutation Testing Techniques and Tools for Aspect-J Programs", published in IACC-2013, India, February 22-23, 2013.

[31] Shoham, Y. (1993). Agent-Oriented Programming. Artificial Intelligence.pp.51-92. Cite SeerX: 10.1.1.123.5119

[32] M. Douglas, McIlroy (January 1969). "Mass-produced software components." Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany. sn, 1968.. pp. 79.

[33] E. S. Yu, "Towards modelling and reasoning support for early-phase requirements engineering."Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on. IEEE, 1997.

[34] L. Chung, J. Mylopoulos, & E. Yu, (1999). From object-oriented to goal-oriented requirements analysis. Communications of the ACM, 42(1), 31-37.

[35] Hirschfeld, Robert, Pascal Costanza, and Oscar Nierstrasz. "Context-oriented programming." Journal of Object Technology 7.3 (2008).