Vol. 5(18) Special Issue, Dec. 2015, PP. 2664-2673

Contents list available at IJMEC

International Journal of Mechatronics, Electrical and Computer Technology (IJMEC)

Journal Homepage: www.aeuso.org

PISSN: 2411-6173 - EISSN: 2305-0543

Studying the deficiencies and problems of different architecture in developing distributed systems and analyse the existing solution

Masoud Rafighi[1] Yaghoub Farjami[2] and Nasser Modiri

[1,2]*Department of Computer Engineering and Information Technology, University Of Qom*

[3] *Department of Computer Engineering and Information Technology, Zanjan Azad University*

***Corresponding Author's E-mail:** Masoud_r62@yahoo.com*

Abstract

In this paper, we examine and compare various aspects of the three proposed architectures in producing and the development of distributed software. We are discussed about the need to create an ideal model for an optimal architecture studying the aggregation problem that the architectures are not responsive. We are supposed to reach some results such as: which is the best and completely software architecture for enterprise resource planning that have enough flexibility, exchangeability, reliability and usability.

Keywords: *Three-tier architecture, architecture Component-based, Service-oriented architecture, distributed systems.*

1. Introduction

Customers' requirements control the creation and deployment of software. Customers demand more and better functionality, they want it tailored to their needs, and they want it "yesterday." Very often, large shops prefer to develop their own in-house add-ons, or tweak and replace existing functions. Nobody wants to reinvent the wheel, but rather to integrate and build on existing work, by writing only the specialized code that differentiates them from their competition. Newer enterprise-class application suites consist of smaller stand-alone products that must be integrated to produce the expected higher-level functions and, at the same time, offer a consistent user experience. The ability to respond quickly to rapid changes in requirements, upgradeability, and support for integrating other vendors' components at any time all create an additional push for flexible and extensible applications[31]. Down in the trenches, developers must deal with complex infrastructures, tools, and code. The last thing they need is to apply more duct tape to an already complex code base, so that marketing can sell the product with a straight face. Software Architecture [31; 32] describes the high-level structure of a system in terms of components and component interactions. In design, architecture is widely recognized as providing a beneficial separation of concerns between the gross system behavior of interacting components and that of its constituent components. Similarly this separation is also beneficial when considering deployed systems and evolution as it allows us to focus on change at the component level rather than on some finer grain[32]. For instance, previous work described some of the issues involved in specifying a limited form of dynamic software structure for distributed systems in which the set of components and their interaction change as execution progresses and the system evolves [33]. A change to the software architecture could occur either as the result of some computation performed by the system or as a result of some external management action such as to insert a new component and to change those connections within the system to accommodate the new component. Management actions are performed by a configuration manager [34]. Which maintains an overall view of the structure of a system in terms of components and their interconnections and performs changes in the context of that view. In essence, the

Article History:

Received Date: Aug. 17, 2015

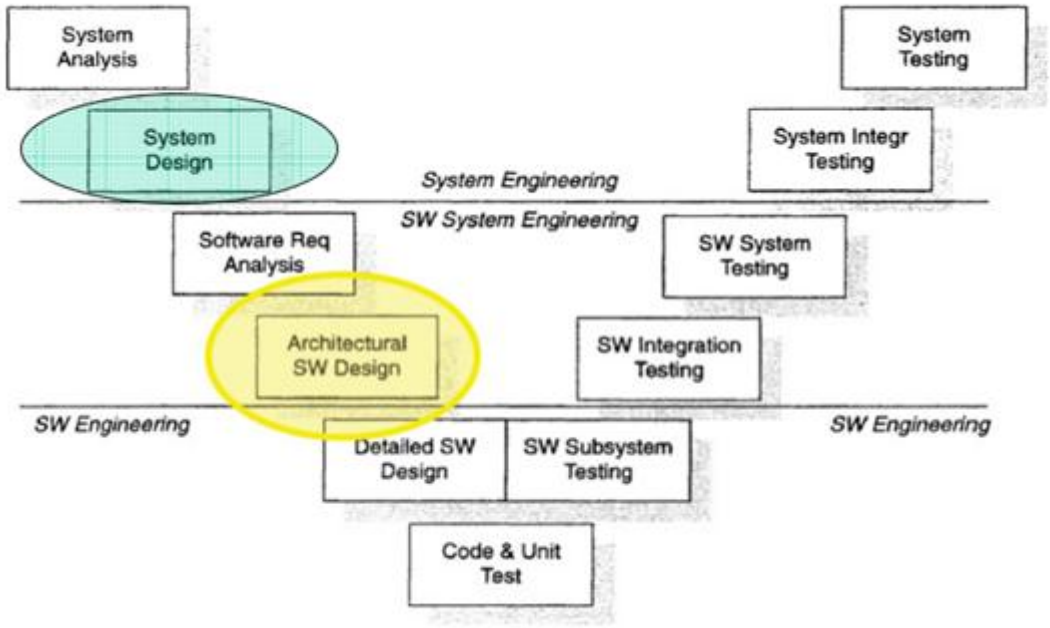
Accepted Date: Dec. 23, 2015

Available Online: Jan. 06, 2016

configuration manager is responsible for ensuring that an executing system conforms precisely to its architectural specification. This approach can however be too restrictive for current dynamic, open systems[34].

2. Software architecture

Architecture, the fundamental organization of a system consisting of components, each of which is associated with each other and with the system and the principles governing its design and evolution is. Software architecture is the choice of a general structure for implementing a software project based on a set of user requirements and business of software systems which it can implement our requires, optimized the software quality, production and maintenance and accelerate it. Nowadays due to the development of distributed systems that are constantly changing need for a flexible architecture can be felt more than ever [28].



Architecture: place in system development cycle [28].

3. Distributed systems

A distributed system is essentially a computer system where components of the system are held on physically separated, autonomous computers. These machines communicate through the use of a computer network, either a fixed or, in the case of mobile applications, a wireless network. The distributed systems appear to users as a single, integrated computing facility.

In recent years, distributed systems have become increasingly popular and important in modern computing. They provide opportunities for increasing the reliability, availability and performance of applications. However, perhaps the most important feature of a distributed system is that it allows the integration of existing systems. Companies do not wish to rewrite large numbers of legacy applications and a distributed system allows these applications to be integrated in a relatively straightforward manner[4,5].

A distributed system may comprise components written in a number of different programming languages, running on different operating systems on a variety of computer architectures [4,29].

In many cases, a distributed system may be cheaper than a single, centralized system. A large number of small, low-power systems may prove cheaper to purchase than a single mainframe or supercomputer. This is the approach employed in Beowulf clusters, which allow a collection of computers to act as a single large computer[4,5].

There are obviously many significant disadvantages to distributed systems. They are much more complicated to design, build and maintain than an equivalent centralized system. There are a large number of possible failures that could occur in a distributed system, far more than would be found in a centralized system. Because of this, a distributed system will have multiple points of failure, increasing the likelihood of the system not functioning correctly. Communication over a network will always be far slower and less reliable than communication over a local bus, which has a significant effect on the performance of a distributed system [4,5,29].

4. Distributed systems architectures

- Client--server architectures
  - Distributed services which are called on by clients. Servers that provide services are treated differently from clients that use services.
- Distributed object architectures
  - No distinction between clients and servers. Any object on the system may provide and use services from other objects [30].
  -

5. Architectures for development of Software distributed

A. Service-oriented architecture

An approach for building distributed systems that provide software functions in the form of services. In fact, automate and manage processes will provided in a distributed environment. The service can be used to call other applications and construction the new services. Service oriented architecture is a flexible set of design principles which used in development steps and Integration of computing. System that is based on service-oriented architecture, function packaged as a set of consistent services that can be used separately from the several different business domains [16]. Service oriented architecture has major impact on the development of software systems duo to the ability to increase the organization's agility, adaptability applications, integration, interoperability between systems and reuse of assets inherited organization. In service oriented architecture software sources are packaged as services which are well-defined and self-contained modules that support functionality of standard business and the content are not related to other services. Services with a standard definition language described, they have published interfaces, services interact together through Demand procedures In order to support joint a common business activity or process. The services that benefit from the web services standards such as WSDL, SOAP, UDDI. [17, 18].

B. Three-tier architecture

This architecture consists of three layers: presentation, application and database which increase the scalability, flexibility and balance the load. There is no limitation for increasing the number of users, changing the database used, as well as flexibility in handling program in this kind of architectural. All these achievements are effective to ensure an appropriate architecture [23].

C. **architecture Component-based**

Software components are a comprehensive and extendable piece which its function is well defined. Through interfaces with other components can be combined and interact with each other. Three goals of component-based architecture include: Overcoming complexity, manage the changing, re-usability [9,10].

6. Measurement and analysis of the architecture criteria

A. **Layout of components**

Components as the original block and computational entities participating in the construction of system throw internal computation and external communication do their choruses. Every component communicates with environment by one or more port. A user interface can be a common variable; the name of a procedure which calls from other component; it is a set of events that can occur as a component and other mechanisms. Properties of a component, the data for analysis and software implementation specifies.

***B*. Create**

Configuration is a connected graph which sometimes referred to as the topology which is composed of components and connectors and describes the structure of architecture.

***C*. Connection**

When connector makes a connection between two components, component defines an interface. And every component can have several interfaces. An interface is concerned to just one component and every interface of one component can connect to several interfaces in other components. For example in Bus-Oriented architecture the interface of every component is connected to the bus connector and so it will connected to several interface in other components. Attributes can also be indicated by some of the feature, such as communication, buffering capacity and so on

***D.* Development**

Develop and promote will be Causes the development and software update in computer systems so an important metric that can be considered in the selection of the architecture is extensible metric. The software architecture must be Extensibility. We evaluate it since this metric is a major role in architecture.

***E*. The main advantage**

Each of software architecture has advantages compared to other architectures. The software architecture eliminates defects in other architectures and complement previous architectures.

***F.* The main problem**

Although each software architectures try to be the best and perfect, but with the development of information systems and their development is still facing problems and in some cases, the problems faced. These criteria were chosen only for the problems and shortcomings of Distributed software development architectures and of course there are other factors and criteria that are not effective in this research. To see a full description and explanation of software metrics can be [M. Shaw and D. Garlan, 1996] presented [27].

7. Compare architectures

TABLE I. Compare architectures

Architecture / Criterion	THREE- TIER	Component	SOA
Layout	Data layer Middle layer(logic) Presentation layer [25,24].	Components are integrated and modular, A unit is independent establishment and is independent of other components A unit of independent deployment [12,14].	the services defined on the data, Object and component associated with their service Data definition: XML Process based, Message- Oriented [22].
Creation	Data is read from the database and Middle layer sends the data to presentation layer by control and management processes [25].	Special languages for defining interfaces, (IDL)[11,12]. There's a lot of software components 1.Input/output types 2.Functional behavior 3.Concurrent behavior 4.Timing behavior 5.Resource usage 6.Security[11,15].	Making the change Web service search:UDDI Messaging: SOAP Web service interface: WSDL Transactions: WS-Transaction Service composition: WS-BPEL [22].
Connection	Layers are not dependent on each other and Each doing its job and the various modules on the operating system are stored separately [25].	IPC protocol IIOP (Internet Inter-ORB Protocol)[12,11]. Not context dependent Not related to a specific area and can be used in the system [15].	Loosely coupled communication between software components so that is independent of the Lower layer protocols Web connection: HTTP[21].
Development	Avoiding dependence on storage mechanism allows to update, change the application layer or using even a conscious change, without having influenced on client layer [25].	Components are interchangeable for example component B can be replaced with component A Compassable with other components A good combination of mechanisms is used [14,11].	Each service can guarantee that service can change information with any information on network without human intervention [19,20].
Elected or a combination of other architectures	Client/server[26].	The conference was published in Germany in 1968,is not selected from other architectures [9,10].	It is not selected from other architectures, but it uses from other technologies such as WSDL ،UDDI و ، SOAP[22].
The main advantage	Increasing efficiency, flexibility, Reusability [25].	Reuse of software in order to reduce development costs Variability, performance, Support for parallel distributed systems on runtime [15].	Independent of the platform [18].
The main problem	If system complexity will increase Tracing the flow of data through a three-tier system is challenging and difficult	Data integration components can be hard to combine[12].	The solutions of all problems are not related to software development. Problems include finding the required services, present the acceptable performance, security, Realization of Transaction to protect themselves services even from the outside, changing or closing integrated services are remain[21].

8. Problems architectures

The feasibility survey was conducted for exploring attitudes of the users and potential customers. It showed that main obstacles which hinder usage of service are related to possible cloud service termination or failure and vendor lock-in [1]. The rule engine component enables to inform the customer. If he can retrieve the data bach from cloud in the required format and ensures possibility to use the backup data with the local system of the customer and prevent from vendor lock-in situation [1]. Availability, data lock-in, data confidentiality and auditability are the obstacles which affect adoption of cloud computing [2]. Although cloud computing

providers are facing several architecture and design challenges, however, security concerns, interoperability, data lock-in are on top of those challenges. Most of the clouds are vendor-locked, as several cloud providers offer APIs (application programming interfaces) that are well-documented, but are mainly proprietary and exclusive to their implementation and thus not interoperable[3].

For 20% of the respondents, risk of vendor lock in, loss of control, and security were sources of concern. The ability to meet government and industry standards was not seen as a concern, as none of the respondents selected this option.[6] Now, certain characteristics of this alternative make it attractive for SMEs: greater adaptability, no vendor lock-in, property of the source code, and cost comparable to other alternatives [7]. This last problem has been further pursued by IS researchers who have looked at package customization and organizational adaptation as alternative ways of resolving such misalignment [7].

At present, there are many companies implement Enterprise Resource Planning (ERP), some companies choose to buy the ERP software directly, or hire the professional group coding software for the companies. However, due to the poor flexibility of the system, and not very appropriate for business processes and management concepts, Some companies hitch have lots of profits choose to self-development the ERP system [8]. ERP system change the business process of the enterprises, and it is difficult to personnel adapt to the new system, as a result, it will also prolong the whole time in ERP implementation [8]. In this condition, the system can better focus on needs of users. How to solve these business problems and technical details will be completed through the conversion tool. Although the definition of the conversion is difficult, when business needs changes, it can be used again. In the long run, this effort has positive effect to the rapid development [8].

By analyzing the existing system and the resources in the world have pointed to the problems, problems that are not responsive architectures are as follows:

1. Extensibility problem involving (the laws have changed, change in data, the changes in the organization, integration, change in operations, changes in systems, developing new systems).

2. Problem of imprisonment or trapped data.

3. Programmer lock-in problem, only the programmer can develop the system further.

To solve the above problems, there are solutions which are listed below:

One effective way to make your application extensible is to expose its internals as a scripting language and write all the top level stuff in that language. This makes it quite modifiable and practically future proof (if your primitives are well chosen and implemented). A success story of this kind of thing is Emacs. I prefer this to the eclipse style plugin system because if I want to extend functionality, I don't have to learn the API and write/compile a separate plugin. I can write a 3 line snippet in the current buffer itself, evaluate it and use it. Very smooth learning curve and very pleasing results [8,34].

One application which I've extended a little is Trace. It has a component architecture which in this situation means that tasks are delegated to modules that advertise extension points. You can then implement other components which would fit into these points and change the flow [8].
But due to the distributed systems need database, these solutions can't be hopeful way. Like most things in life, taking the time to plan ahead when building a web service can help in the long run understanding some of the considerations and tradeoffs behind big websites can result in smarter decisions at the creation of smaller web sites. Below are some of the key principles that influence the design of large-scale web systems:

- **Availability:** The uptime of a website is absolutely critical to the reputation and functionality of many companies. For some of the larger online retail sites, being unavailable for even minutes can result in thousands or millions

of dollars in lost revenue, so designing their systems to be constantly available and resilient to failure is both a fundamental business and a technology requirement. High availability in distributed systems requires the careful consideration of redundancy for key components, rapid recovery in the event of partial system failures, and graceful degradation when problems occur [8,34].

- **Performance:** Website performance has become an important consideration for most sites. The speed of a website affects usage and user satisfaction, as well as search engine rankings, a factor that directly correlates to revenue and retention. As a result, creating a system that is optimized for fast responses and low latency is the key [8,34].
- **Reliability:** A system needs to be reliable, such that a request for data will consistently return the same data. In the event the data changes or is updated, then that same request should return the new data. Users need to know that if something is written to the system, or stored, it will persist and can be relied on to be in place for future retrieval [8,34].
- **Scalability:** When it comes to any large distributed system, size is just one aspect of scale that needs to be considered. Just as important is the effort required to increase capacity to handle greater amounts of load, commonly referred to as the scalability of the system. Scalability can refer to many different parameters of the system: how much additional traffic can it handle, how easy is it to add more storage capacity, or even how many more transactions can be processed [8,34].
- **Manageability:** Designing a system that is easy to operate is another important consideration. The manageability of the system equates to the scalability of operations: maintenance and updates. Things to consider for manageability are the ease of diagnosing and understanding problems when they occur, ease of making updates or modifications, and how simple the system is to operate. (I.e., does it routinely operate without failure or exceptions?) [8,34].
- **Cost:** Cost is an important factor. This obviously can include hardware and software costs, but it is also important to consider other facets needed to deploy and maintain the system. The amount of developer time the system takes to build, the amount of operational effort required to run the system, and even the amount of training required should all be considered. Cost is the total cost of ownership [8,34].

Each of these principles provides the basis for decisions in designing distributed web architecture. However, they also can be at odds with one another, such that achieving one objective comes at the cost of another. A basic example: choosing to address capacity by simply adding more servers (scalability) can come at the price of manageability (you have to operate an additional server) and cost (the price of the servers) [8,34].

When designing any sort of web application it is important to consider these key principles, even if it is to acknowledge that a design may sacrifice one or more of them [8,34].

When it comes to system architecture there are a few things to consider: what are the right pieces, how these pieces fit together, and what the right tradeoffs are. Investing in scaling before it is needed is generally not a smart business proposition; however, some forethought into the design can save substantial time and resources in the future [8,34].

This section is focused on some of the core factors that are central to almost all large web applications: services, redundancy, partitions, and handling failure. Each of these factors involves choices and compromises, particularly in the context of the principles described in the previous section [8,34].

When considering scalable system design, it helps to decouple functionality and think about each part of the system as its own service with a clearly defined interface. In practice, systems designed in this way are said to have a Service-Oriented Architecture (SOA). For these types of systems, each service has its own distinct functional context, and interaction with anything outside of that context takes place through an abstract interface, typically the public-facing API of another service [8,34].

Deconstructing a system into a set of complementary services decouples the operation of those pieces from one another. This abstraction helps establish clear relationships between the service, its underlying environment,

and the consumers of that service. Creating these clear delineations can help isolate problems, but also allows each piece to scale independently of one another. This sort of service-oriented design for systems is very similar to object-oriented design for programming [8,34].

Another key part of service redundancy is creating a shared-nothing architecture. With this architecture, each node is able to operate independently of one another and there is no central "brain" managing state or coordinating activities for the other nodes. This helps a lot with scalability since new nodes can be added without special conditions or knowledge. However, and most importantly, there is no single point of failure in these systems, so they are much more resilient to failure [8,34].

As they grow, there are two main challenges: scaling access to the app server and to the database. In a highly scalable application design, the app (or web) server is typically minimized and often embodies a shared-nothing architecture. This makes the app server layer of the system horizontally scalable. As a result of this design, the heavy lifting is pushed down the stack to the database server and supporting services; it's at this layer where the real scaling and performance challenges come into play [8,34].

Finally, another critical piece of any distributed system is a load balancer. Load balancers are a principal part of any architecture, as their role is to distribute load across a set of nodes responsible for servicing requests. This allows multiple nodes to transparently service the same function in a system. Their main purpose is to handle a lot of simultaneous connections and route those connections to one of the request nodes, allowing the system to scale to service more requests by just adding nodes. Load balancers are an easy way to allow you to expand system capacity, and like the other techniques in this article, play an essential role in distributed system architecture. Load balancers also provide the critical function of being able to test the health of a node, such that if a node is unresponsive or over-loaded, it can be removed from the pool handling requests, taking advantage of the redundancy of different nodes in your system [8,34].

Conclusion

The comparison of these methods with parameters (layout, create, connect, development, main advantage, the main problem) to the conclusion that although each of these architectures claim that are suitable for distributed system or changing, but in practice they aren’t responsive these changes in system.
A repeating theme in my development work has been the use of or creation of an in-house plug-in architecture. I've seen it approached many ways - configuration files (XML, .conf, and so on), inheritance frameworks, database information, libraries, and others. In my experience:

- A database isn't a great place to store your configuration information, especially co-mingled with data
- Attempting this with an inheritance hierarchy requires knowledge about the plug-ins to be coded in, meaning the plug-in architecture isn't all that dynamic
- Configuration files work well for providing simple information, but can't handle more complex behaviors
- Libraries seem to work well, but the one-way dependencies have to be carefully created

These examples seem to play to various language strengths. Is good plugin architecture necessarily tied to the language? Is it best to use tools to create plugin architecture, or to do it on one's own following models?

Can you say why plugin architecture has been a common theme? What problems does it solve, or goals does it address? Many extensible systems/applications use plugins of some form, but the form varies considerably depending upon the problem being solved.

That's a great question. I'd say there are some common goals in an extensible system. Perhaps the goals are all that's common, and the best solution varies depending on how extensible the system needs to be, what language the system is written in, and so on. However, I see patterns like IOC being applied across many languages, and I've been asked to do similar plugins (for drop in pieces responding to the same functionality requests) over and over again. I think it's good to get a general idea of best practices for various types of plugins. In a world of increasingly complex computing requirements, we as software developers are continually searching for that ultimate, universal architecture that allows us to productively develop high-quality applications. This quest has led to the adoption of many new abstractions and tools. Some of the most promising recent developments are the new pure plug-in architectures. Mentioned pathologic problems still remain open in systems and not with these architectures resolve common problems and the need for a revision in architectural theory.

References

[1] Ahmed Elragal , Moutaz Haddara, "The Future of ERP Systems: look backward before moving forward, CENTERIS, prise Information Systems / HCIST, International Conference on Health and Social Care Information Systems and Technologies, ELSEVIER, Procedia Technology5(2012)21 – 30

[2] Bernard Manouvrier , Laurent Ménard, First published in France in 2007 by Hermes Science/Lavoisier entitled "Intégration applicative EAI, B2B, BPM et SOA"

[3] Baolin Xu, Chunjing Lin, "An extended practical three-tier architecture based on middleware," 978-1-4673-5000-6/13/$31.00 ©2013 IEEEs

[4] Channabasavaiah, Holley and Tuggle, "Migrating to a service-oriented architecture," *IBM DeveloperWorks*, 16 December 2003.

[5] Dalia Kriksciuniene, Donatas Mazeika, "Cloud Computing and the Enterprise Needs for Data Freedom, the Third International Conference on Future Computational Technologies and Applications," FUTURE COMPUTING 2011

[6] Dr. Ashu Khanna, Dr. Vivek Kumar and Indu Saini, "ERP SYSTEMS: PROBLEMS AND SOLUTION WITH SPECIAL, REFERENCE TO SMALL & MEDIUM ENTERPRISES," International Journal of Research in IT & Management," IJRIM, Volume 2, Issue 2 (February 2012)

[7] D. Garlan, D. Perry, "Introduction to the Special Issue on Software Architecture. IEEE Transactions on software Engineering," 1995. 21(4): p. 269-274.

[8] Bennouar, T. Khammaci and A. Henni, "A new approach for component's port modeling in software architecture, Journal of Systems and Software," Volume 83, Issue 8, August 2010, Pages 1430-1442, Distributed Systems Architectures, "Based on Software Engineering," 7 Th Edition by Ian Sommerville

[9] E. Perry, A. L. Wolf, "Foundations for the Study of Software Architectures, ACM SIGSOFT Software Engineering Notes," Vol. 17, No. 4, pp. 40-52, 1992.

[10] Eduardo B. Fernandez, Mihai Fonoage, Michael VanHilst and Mirela Marta, "The Secure Three-Tier Architecture Pattern, International Conference on Complex, International Conference on Complex, Intelligent and Software Intensive Systems," 0-7695-3109-1/08 $25.00 © 2008 IEEE

[11] Eckerson, Wayne W, "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications," Open Information Systems 10, I (January 1995): 3(20)

[12] Mentzas, A. Friesen, "Semantic Enterprise Application Integration for Business Processes: Service-Oriented Frameworks," IGI Global 2010

[13] Hoyer, Volker; Stanoesvka-Slabeva, Katarina; Janner, Till; Schroth, Christoph; (2008). "Enterprise Mashups: Design Principles towards the Long Tail of User Need," Proceedings of the 2008 IEEE International Conference on Services Computing (SCC 2008). Retrieved 2008-07-08.

[14] https://f5.com/fr/resources/white-papers/soa-challenges-and-solutions, Updated mars 18, 2007

[15] Jacek Lewandowski, Adekemi O. Salako, Alexeis Garcia-Perez, "SaaS Enterprise Resource Planning Systems: Challenges of their adoption in SMEs, IEEE 10th International Conference on e-Business Engineering," 2013

[16] J. Magee, J. Kramer, "Dynamic Structure in Software Architectures, 4th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE 4)," San Francisco, California, USA, 21, pp. 3-14, October 1996.

[17] Liu Chen, Liu Xinliang, "Self-development ERP System Implementation Success Rate Factors Analysis," IEEE XPLORE Symposium on Robotics and Applications(ISRA) 2012

[18] McIlroy, M. Douglas, "Mass produced software components," Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968. Scientific Affairs Division, NATO. p. 79.

[19] Majdi Abdellatief, Abu Bakar Md Sultan, Abdul Azim Abdul Ghani, Marzanah and A. Jabar , "A mapping study to investigate component-based software system metrics, Journal of Systems and Software," Volume 86, Issue 3, March 2013, Pages 587-603

[20] Manuel Oriol, Thomas Gamer, Thijmen de Gooijer, Michael Wahler, Ettore and Ferranti, "Fault-tolerant fault tolerance for component-based automation systems, in: Proceedings of the 4th International ACM SIGSOFT Symposium on Architecting Critical Systems (ISARCS 2013)," Vancouver, Canada, 2013. M. Shaw, D. Garlan, "Software architecture: perspectives on an emerging discipline," Prentice Hall, 1996.

[21] M.J. Charistensen, R.H. Thayer. "The Project Manager's Guide to Software Engineering's Best Practices." Wiley, 2002.

[22] M. Shaw, D. Garlan, "Software Architecture: Perspectives on an Emerging Discipline," Prentice Hall, 96.

[23] Na Luo, Weimin Zhong, Feng Wan, Zhencheng Ye and Feng Qian, "An agent-based service-oriented integration architecture for chemical process automation, Elsevier," Chinese Journal of Chemical Engineering 23 (2015) 173–180

[24] Placide Poba-Nzaou & Louis Raymond, "Custom Development as an Alternative for ERP Adoption by SMEs: An Interpretive Case Study, Information Systems Management," 02 Sep 2013.Published online: 21 Oct 2013

[25] R. Allen, "A Formal Approach to Software Architecture," PhD Dissertation, Carnegie Mellon University, 1997.

[26] Ralf H. Reussner, Heinz W. Schmidt and Iman H. Poernomo,"Reliability prediction for component-based software architectures," Journal of Systems and Software, Volume 66, Issue 3, 15 June 2003, Pages 241-252

[27] Robert Nunn, "Distributed Software Architectures Using Middleware," 3C05 Coursework 2

[28] S. Crane, N. Dulay, H. Fosså, J. Kramer, J. Magee, M.Sloman and K. Twidle, "Configuration Management forDistributed Systems," Proc. of the IFIP/IEEE International Symposium on Integrated Network Management (ISINM 95), Santa Barbara.

[29] Séverine Sentilles, Aneta Vulgarakis, Tomáš Bureš, Jan Carlson and Ivica Crnkovic´, "A Component model for control-intensive distributed embedded systems, in:Michel Chaudron, Clemens Szyperski, Ralf Reussner (Eds.), Component-Based Software Engineering," Lecture Notes in Computer Science, vol. 5282, Springer, Berlin/Heidelberg, 2008, pp. 310–317.

[30] Schroth, Christoph ; Janner, Till; (2007). "Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services," IT Professional 9 (2007), Nr. 3, pp. 36-41, IEEE Computer Society. Retrieved 2008-02-23.

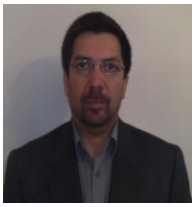
[31] William Otter, Aniruddha S. Gokhale and Douglas C. Schmidt, "Efficient and deterministic application deployment in component-based enterprise distributed real-time and embedded systems," Inf. Softw. Technol. 55 (2)(2013) 475–488.

[32] Wu Liping, Zhao Zhuo and Chen Qi, "Research and Design of System, Management Software on the Basis of Three-tier Architecture, Computer Engineering," Vol.32 No.17, 2006, 283-285.

AUTHORS

Masoud rafighi was born in Tehran, Iran on 1983/08/10. He is PHD student of Qom University. He receives M.Sc degree in computer engineering software from Azad University North Tehran Branch, Tehran, IRAN. He has recently been active in software engineering and has developed and taught various software related courses for the Institute and university for Advanced Technology, the University of Iran. His research interests are in software measurement, software complexity, requirement engineering, maintenance software, software security and formal methods of software development. He has written a book on software complexity engineering and published many papers.

Yaghoub Farjami received his PhD degree in Mathematics (with the highest honor) in 1998 from Sharif University of Technology, Tehran, Iran. He is Assistant Professor of Computer and Information Technology Department at University of Qom. His active fields of research are ERP, BI, and Information Security.



Nasser Modiri received his MS degree in Micro Electronics from university of Southampton, UK in 1986. He received PHD degree in Computer Networks from Sussex university of UK in 1989. He is a lecturer at department of computer engineering at Islamic Azad University of Zanjan, Iran. His research interests include Network Operation Centers, Framework for Securing Networks, Virtual Organizations, RFID, Product Life Cycle Development and Framework for Securing Networks