THE CURRENT STATE OF SCALABILITY: WHAT IS AND WHAT SHOULD BE

MOHAMED E. FAYAD *, SHIVANSHU K. SINGH **, RAFAEL CAPILLA ***

Traditional approaches of architecting software are incapable of providing all the solutions for developing scalable architectures. Uncertainty or lack of knowledge about which steps or guidelines to use, in order to obtain a good modularization of the architecture, is one of the major problems that keeps us from realizing a truly scalable architecture.

INTRODUCTION

The previous column in this series on scalability, discussed the current state of Scalability in Software Architecture. It discussed the problems with the way software is architected today and introduced a new way of looking at scalability, i.e. from the view of the Software Stability Model [1]. Here, the authors take a look at how the current approach to architecting software is not the best way to ensure scalability from architecture and requirements engineering perspective. The current approaches to the software architecture introduce way to many dependencies into the system (e.g. tight coupling and high cohesion, no proper identification of layers of functionalities or lack of adequate modularization), such that managing requirements over time becomes a big hassle and a tedious task, when it comes to adding/removing functionality.

Regardless of effectiveness, there are cases where Scalability may be implemented using a conventional approach, as described in the previous column. However, these cases may not provide the most suitable implementation for scalability. Therefore, the resulted architecture would be struggling in reaching a high level of consistency, becoming a perfect facilitator of a series of drawbacks through time. Such pitfalls encourage the utilization of an innovative approach, the Software Stability Model (SSM) [1], which generates architectures capable of evolving through time without the concern of potential collapses. These evolving architectures are referred as Timeless architectures. Timeless architectures are well-designed architectures, whose structures remain constant, and are able to evolve proportionally with the appearance of new requirements over time for a long time [2]. In other words, they are flexible enough to scale the scope of their elements, methods, etc. when handling more demanding requirements. For instance, gluing together two or more architectures that were used separately to perform one or more common tasks (Horizontal Scalability), or adding new functionalities to architecture's structure to address more users needs (Vertical Scalability). These benefits will turn the architecture not only into a more stable and reusable architecture, but also into a more scalable one. Timeless and scalable architectures must survive with minimum changes over time as much as possible, and evolve accordingly to new requirements and business goals in a flexible and elastic manner. Featuring this kind of architectures will be discussed more in the upcoming columns.

Scalable Architectures with Traditional Design Approaches

At its essence, a traditional approach is incapable of providing all the solutions for developing Timeless and Scalable architectures. A major difficulty of this approach lies in the factors, such as uncertainty or lack of knowledge on which steps or guidelines to use, in order to obtain a good modularization of the Architecture. For instance, when dealing with Vertical Scalability is not clear what elements or layers to keep and what to drop to reach an efficient result [3]. From the Horizontal Scalability perspective, traditional model approach does not facilitate a specific idea on which points a particular architecture would be bound with/unbound from other external and heterogeneous architectures. In this way, the

* *Department of Computer Engineering, Charles W. Davidson College of Engineering, San Jose State University, San Jose, USA.*

** *Software/Data Engineering and DevOps, Athos, San Francisco, California, USA.*

*** *Associate Professor at the University Rey Juan Carlos of Madrid (Spain), USA.*

well-designed reference architectures with enough flexibility capable to integrate new functional pieces without much effort can be considered as suitable candidates to achieve horizontal stability. These factors will incapacitate the ideal architecture not only on its deployment but also in its capacity to adapt to multiple environments (Constrained and Unconstrained). Additionally, these factors will turn the resulted architecture into an excellent host for serious contamination problems when scaling it vertically and/or horizontally. Since this occurrence becomes more prominent through the entire architecture, the harmful result of the corresponding ripple effects [4] would get magnified. To see how the ripple effects are propagated throughout the entire architecture, refer to Figure 1.

There are further difficulties experienced when using a traditional approach. They are listed using a detailed "Cause and Effect" table. The purpose of this table is to detail the effects of the usage of a conventional approach over the implementation and deployment of Timeless and Scalable Architectures. The authors would concentrate not only in the characteristics of this approach, and how it deals with Scalability, but also in showing the effects in implementation process and in the resulted architecture itself. Described below are the various problems that exist with the current approaches to designing a software architecture, following traditional approaches to software development, along with their effects on horizontal and vertical scalability with respect to the software's architecture.

Problems/Causes and Effects of using Traditional Approaches to develop Timeless & Scalable Architectures

Cause 1. Unavailability of well defined guidelines to factor scalability into any software architecture and unsystematic processes.

Effects:

- *Vertical Scalability:*

High impact on implementation, impossible test cases and a high chance of stability and scalability problems to occur with respect to the architecture may lead to architecture collapse.

- *Horizontal Scalability:*

Same as the impacts on Vertical Scalability; High impact on implementation, impossible test cases and a High chance of stability and scalability problems to occur with respect to architecture, includes a high chance of architecture collapse.

Rationale of Effects: The lack of adequate guidelines to achieve a concrete level of stability in any architecture may block the architecting process and the assumptions made and lead to a proper design. Hence, bad design decisions may cause an unstable design.

Cause 2. Unclear Direction of Scalability and Ad-Hoc Processes.

Effects:

- *Vertical Scalability:*

High impact on implementation, Low Cohesion and an Unstable Structure and a High chance of stability and scalability problems to occur with respect to architecture, includes a high chance of embedded containment.

- *Horizontal Scalability:*

High impact on implementation, Low Cohesion and an Unstable Structure and a High chance of stability and scalability problems to occur with respect to architecture, includes a high chance of embedded containment.

Rationale of Effects: Wrong or inadequate design decisions to achieve a more scalable design may damage the structure of the architecture, to provoke mismatches and inconsistency problems.

Cause 3. No clear way of 'Layer' Identification.

Effects:

- *Vertical Scalability:*

High impact on implementation, not cost / time effective and a High chance of stability and scalability problems to occur with respect to architecture, including a high chance of ripple effects.

- *Horizontal Scalability:*

High impact on implementation, not cost / time effective and a High chance of stability and scalability problems to

occur with respect to architecture, including a high chance of ripple effects.

Rationale of Effects: A bad identification of the layers that would be added or removed in the architecture may boost the ripple effects in the design of bad decisions.

Cause 4. Shortage of Layer Boundaries.

Effects:

- *Vertical Scalability:*

High impact on implementation and a Medium-High effort required during implementation, with medium-high degree of difficulties associated with scaling up. Unclear borders between layers make more difficult to assign architectural modules to a given layer.

- *Horizontal Scalability*

n/a

Rationale of Effects: The difficulty to clearly identify the boundaries of architecture layers and assign each architectural component to the right layer leads to unclear borders, where the architect has difficulties when assigning functionality to a given layer.

Cause 5. Undefined Connection/Interfacing points between Layers.

Effects:

- *Vertical Scalability:*

High impact on implementation and a High effort required during implementation, with high degree of difficulties in scaling down.

- *Horizontal Scalability:*

n/a

Rationale of Effects: Wrong definition of architectural interfaces makes the design less scalable and complicates the programming issues.

Cause 6. Problems in Scaling out, as again no clear architectural interfaces defined.

Effects:

- *Vertical Scalability:*

n/a

- *Horizontal Scalability:*

High impact on implementation and a Medium-High effort required during implementation, with medium-high degree of difficulties in scaling out.

Cause 7. Problems in Scaling in, as no proper guidelines are there to reduce or remove any architecture from the existing system.

Effects:

- *Vertical Scalability:*

n/a

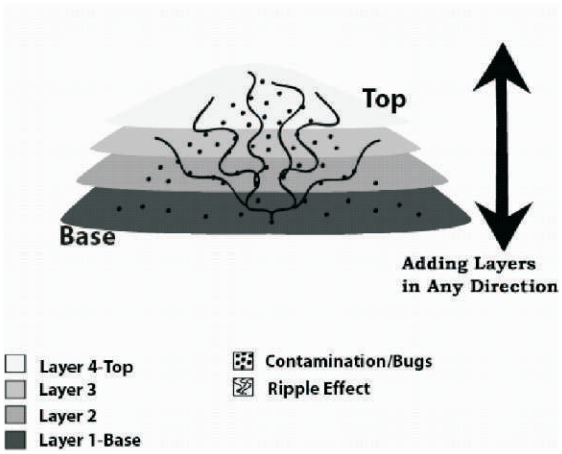
- *Horizontal Scalability:*

High impact on implementation and a High effort required during implementation, with high degree of difficulties in scaling in.

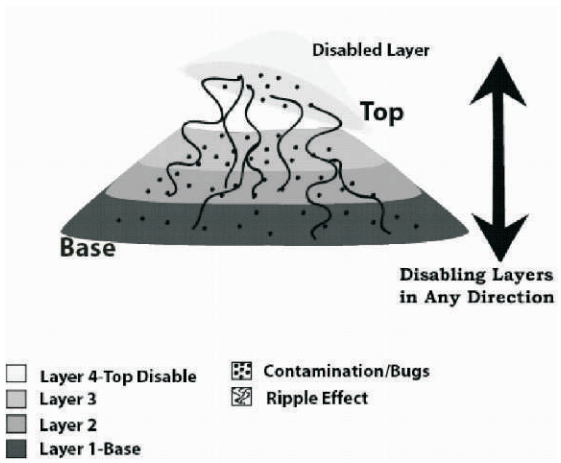
In order to come up with a formal way of evaluating and considering scalability requirements for any software, attempts have been made in the past [5]. These methods involve frameworks that let you evaluate the scalability requirements, decide the various conflicts and priorities and help in coming up with the necessary set of guidelines to incorporate the required scalability into the system. However, it should be noted that, these approaches are still approaching the issue of scalability at the deployment and configuration level than doing so when coming up with the core architecture. Also, the term architecture as taken by these approaches [5] refer more to the way the various component s of the software system are deployed and configured than anything else. While these approaches might be helpful and would surely come into play when a system has to be actually deployed, they do not address the concern of scalability in a holistic way and so scaling with the changing functional requirements is just not possible if those approaches are to be followed in an isolated way.

The consequences of traditional approaches to software architecture and how they affect scalability can be seen in a simplified way using Figures 1 and 2.

These illustrations describe in a very simple way the ripple effects experienced by the developed architecture using the Traditional approach. Adopting this approach will require a constant involvement from developers trying to safeguard the architecture from collapsing. Unfortunately, these traditional approaches do not

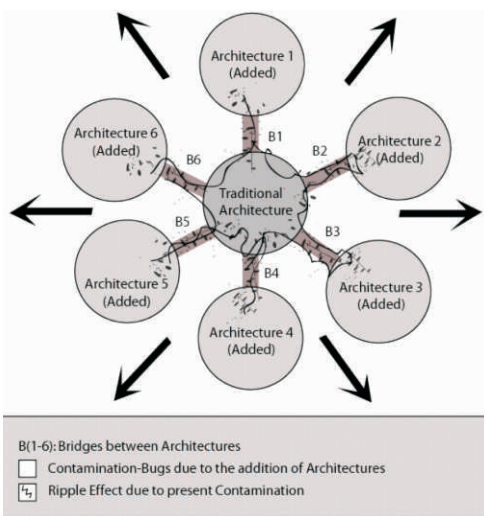


(a) Upward



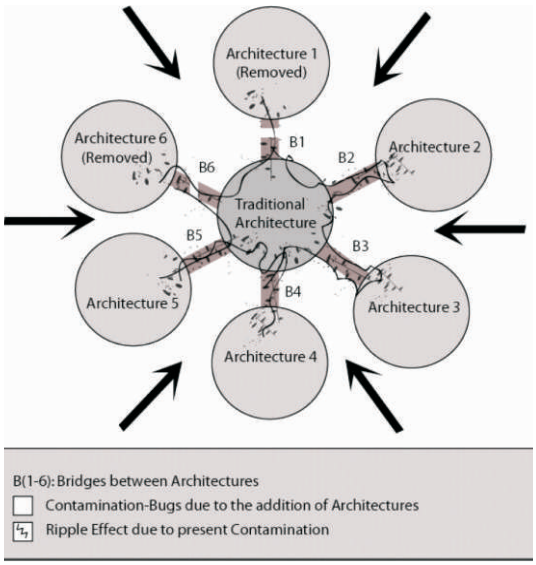
(b) Downward

Figure 1. Vertical Scalability Ripple Effects



(a) Extensibility

provide any help to the latter case. They do not offer the right tools or processes to accomplish Scalability when addressing large changes in problem size [3].



(b) Reduction

Figure 2. Ripple Effects of Horizontal Scalability in Traditional Systems

Additionally, due to its lack of flexibility, traditional approaches are unable to handle the diversity of employed methods encountered within Scalability's implementation when incurring in different problem domains [3].

These fallouts strongly enforce the necessity of an approach which can provides good mechanisms for implementing a Timeless & Scalable Architecture, such as partitioning, composition, and visibility control [3]. In the searching of accomplishing such mechanisms, some essential questions should be answered: How are we going to design a stable architecture, in such way, that the increasing loads or demands will be manageable over time? How can we assure that the software system will operate at the same capacity in a more constrained environment? What are the elements or blocks necessary to reach a grade of stability in our architecture design? How do we ensure the necessary (high – as high as possible) degree of modularization to achieve effective scalability in all directions?

SSM is an answer to this problem. Concepts such as EBTs and BOs [1, 2], when used in architecting software, help in goal realization in the sense that we have a clear distinction between the sets of functionality. The concept of Knowledge Maps, working in conjunction with the SSM

concepts, provides the necessary modularity that helps us in realizing a truly scalable system [6]; scalable with the changing functional and non-functional requirements of the system.

Conclusion

The traditional approaches to scalability in software do not address the essential need to provision this ability in the very core, the architecture of the software. There have been multiple attempts in which, formal ways of approaching and evaluating software scalability have been proposed [5] but these approaches miss out the fact that the issue of scalability is to be addressed at the architecture of the software and not only at the deployment of it. Requirements change over time and so the architecture should be able to evolve with those changes without causing contamination and faults. The objective of this column is to provide an answer for those essential questions. The proposed solution is based on the idea that if a software architecture is designed properly, reaching certain grade of stability, it can be then scaled according to future requirements without the danger of collapsing or redesigning the entire architecture from scratch. Therefore, knowing if this architecture reaches a desired stability level is a major question software architects may raise. Software stability concepts and approaches [1, 3, 6, 7] would be introduced as the main facilitator for this goal implementation.

References

[1]. M.E Fayad, (2002). "Accomplishing Software Stability". *Communications of the ACM*, Vol.45, No.1.

[2]. M.E. Fayad, H.S. Hamza, and H. A. Sanchez, (2004). "Towards Scalable Software Architectures". *IEEE International Conference on Information Reuse and Integration*, Las Vegas, NV.

[3]. Mauri Laitinen, M.E. Fayad, and Robert P. Ward, (2002). "The Problem with Scalability". *Communications of the ACM*, Vol.43, No.9.

[4]. N.H. Madhavji, J. Fernández-Ramil, and D. Perry, (2006). *Software Evolution and Feedback: Theory and Practice*. John Wiley & Sons.

[5]. Duboc L., Rosenblum D., and Wicks T., (2007). "A framework for characterization and analysis of software systems scalability". in *Proceeding of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, *ESEC/FSE'07*, pp.375-384.

[6]. M. E. Fayad, H. A. Sanchez and Shivanshu K. Singh, (2010). "Knowledge Maps – Fundamentally Modular Approach to Software Architecture, Design, Development and Deployment". in *Proceedings of the 19th International Conference on Software Engineering and Data Engineering*, pp.127-133.

[7]. M.E. Fayad and S. Wu, (2002). "Merging Multiple Conventional Models into One Stable Model". *Communications of the ACM*, Vol.45, No. 9.

ABOUT THE AUTHORS

Dr. Mohamed E. Fayad is a Full Professor of Computer Engineering at San Jose State University from 2002 to present. He is one of the founders and president of Arab Computer Society (ACS) from April 04 to April 2007. Dr. Fayad is a known and well recognized authority in the Domain of Theory and the Applications of Software Engineering. Fayad's publications are in the very core, archival Journals and Conferences in the Software Engineering field. Dr. Fayad has published more than 218 high quality papers, that includes more than 40 profound Reports in reputed Journals, and 90 advanced articles in refereed Conferences, more than 25 Journal Columns, 11 well-cited theme issues in prestigious Journals and flagship Magazines, 24 different Workshops in very respected Conferences, over 125 tutorials, seminars, and short presentations in 20+ different counties, NASA Red Team Review of QRAS and NSF-USA Research Delegations' Workshops to Argentina and Chili and eight authoritative books, of which three of them are translated into different languages such as Chinese. Dr. Fayad received an MS and a Ph.D. in computer science from the University of Minnesota at Minneapolis. He is the lead author of several classic Wiley and CRC books.

Shivanshu Singh is working at Software/Data Engineering and DevOps, Athos, San Francisco, California, USA. His research is focused on the areas of Unified Software Engines, Software Architecture, Architectural and Stable Patterns, Knowledge Maps, Spatiotemporal Databases, Software Engineering Processes, Requirements Engineering, Collaborative Systems and more. Shivanshu received his Bachelor of Technology degree in Information and Communication Technology from Dhirubhai Ambani Institute of Information and Communication Technology, Gandhinagar, India in 2007 and has years of professional experience in software engineering, research and development and teaching. He is involved in the development of some major Journals in the field of software engineering and multiple new business developments. He has multiple Journal, Conference and Column publications in his name and is the Editor in Chief for the International Journal of Unified Software Engines (IJUSE). He is in the process of writing one book on Unified Software Engines (USEs). Shivanshu is the Lead of Research of a team of about ten graduate students at San Jose State University, researching on various topics in Software Engineering. He is also a member of IEEE, IEEE Computer Society and the ACM, as well as an invited member of the Phi Kappa Phi Honor Society, for academic excellence; Shivanshu is currently working towards his Master of Science degree in Software Engineering at San Jose State University, San Jose, California and he is working on a PhD degree at Carnegie Mellon University, Doctor of Philosophy (Ph.D.), Software Engineering.

Dr. Rafael Capilla is an Associate Professor at the University Rey Juan Carlos of Madrid (Spain), USA. His research focuses on Software Architectures and Architectural Design Decisions, Product Line Engineering and Software Variability, and Dynamic Service Binding Among others. He co-authored over 79 publications in International Conferences, Journals and Book Chapters and he regularly serves as reviewer. He participates in several Spanish and European research projects and currently is the Head of the Software Architecture & Internet Technologies (SAIT) research group. Dr Capilla has been a visiting researcher in several Spanish and European universities and research center and is a member of IEEE CS since 1998. He also chaired and co-chaired International Conferences and Workshops and serves as an active PC member in several Software Conferences.