Performance in Layered Software Architectures: The case of customized organizational software

Yonah Etene[1], Josphat M. Karani[2]
[1]Department of Computer Science, Kibabii University,
[2]Department of Information Technology, Kirinyaga University

Abstract: *Software architecture is the defining and structuring of a software solution that is capable of meeting technical and operational requirements. It is essential in the organization of a system into its components and helps guide the design of the overall system to ensure functionality of the components to achieve its effective performance.When components are layered, it makes it difficult for an organization to choose and customize the standard package software. This paper examined performance in layered software architecture and how to best achieve effective customization without affecting on the operations of an organization.*

Keywords: *Software architecture, Performance, Customization*

I. INTRODUCTION

Software architecture is the fundamental organization of a software system represented in its components, their relationships to each other, to the environment, and the principles controllingits design and evolution [6]. In addition to the software architecture components and their relationships, this definition also covers architecture rules and principles like architectural styles or the use of particular conventions during the software development, maintenance, and evolution life cycles. Ref [21] notes that software architecture is the carrier of system qualities such as performance, modifiability, and security, of which none can be achieved without a unifying architectural vision.On the other hand layered software architecture, like all architectural structures, reflects a division of the software into units. These units are layered and each layer represents a virtual machine that enable provision of cohesive set of services that other software can use without knowing how those services are implemented[21].

Many today organizations are building application-specific data structures within general-purpose database management systems for either tracking customers or to manage better workflows. Even consumer-focused software, such as email or instant messaging, is increasingly being customized by individual users for visual appearance, greetings, and actions as in [13]. It is apparent that software is among the most pervasive forms of mass customization that is reaching into our lives and that such customization is different by degree than the more standardized forms of variety that is found in automobiles, consumer electronics, or interior furnishings as in [18].The purpose of this study is to investigate the shortcomings of customized software products, the level of performance in standardized forms and suggest ways of improving performance in customizable layered software products in organizations.

II. METHODOLOGY

The study employed desktop research. It surveyed and analyzed existing literature on software layered architectures used for organizational systems and how customization affects their performance. Software customization can create high level of complexity and cost, unless an organization takes time to create and periodically renew a well-defined, layered architecture for its software products [12]. Itcan offer at times the ability to obtain competitive advantagevis-à-vis companies using only standard features as in [9]. However, this also comes with a cost and associated risks. Hence the assessment of software with a view to customize is complex and should be only undertaken with a trade of in mind.

There are three categories of customization: configuration, process and technical customization. Each level of customization has an effect on the layered software architecture which then impacts on the overall performance of the system in service delivery of the organization. A point in case is the bolt-on configuration, a type of configuration customization that affects all the software layers.

III. FINDINGS AND DISCUSSION

The findings of this study are presented and discussed in this section. The findings are a result of document analysis.

The short comings of customizable software products in organizations

Today, most organizations perform a great deal of their work online that requires use of Information Systems. Though the availability of the ready-to-use software applications have greatly reduced the acquisition times of these systems,

organizations still spend a great amount of time and resources in their implementation and management. These applications are only offered as a part of generic software packages, and after their purchase, organizations still have to customize them according to their unique requirements as they are designed in such a way as to allow the organization to configure its software with hundreds of options as in [15]. This phase also requires substantial amount of learning, as the local IT team has to depend on the external consultants and that requires consulting fees. It may also take years for the customization phase to complete and the total costs can add up to millions of dollars. The pricing of these applications (licensing fee) is very complex as it depends on numerous factors such as the components purchased, consultation fees and the size of the organization among other factors as in [16].

Software reuse or commercial off the shelf software (COTS) can lower cost, but only partially. COTS application software most often satisfies less than 40 percent of the functionality of an application. Even when functional requirements are reasonably satisfied well, critical nonfunctional requirements such as security, reliability, and performance have to be addressed and this results in schedule and cost impacts. If the functional or interface requirements are not satisfied, wrappers (additional code required to make the new development able to use the existing software) must be planned, designed, developed, and tested. In all instances, the system or software architecture must be sufficiently mature to allow the detailed design of critical interfaces and the conduct of reasonable trade-offs to enable the evaluation, selection, acquisition, and integration of the capability into the system or software architecture. Only when components are produced like hardware chips, that is, components that are designed for reuse, that includes appropriate inputs and outputs, and have been fully tested for the environment can the risk of reusing someone else's code be reduced [1].

Reuse involves three activities, each of which has a price: redesign, reimplementation, and retesting. Redesign arises because the existing functionality may not be exactly suited to the new task; it likely will require some rework to support new functions, and will likely require reverse engineering to reveal its current operation. Some design changes may be in order. This will result also in reimplementation, which generally takes the form of coding changes. Whether or not redesign and reimplementation are needed, there should be a plan to conduct some retesting to be sure the preexisting software operates properly in its new environment. The findings are summarized in Table 1.

Table 1: Comparison of Types of Reusable Software

	COTS	**COTS**	**Planned Reuse**	**Incidental Reuse**
Ready to use and documented	Yes	Sometimes	Often	Sometimes
Allows programs to offsetrising development costs	Often	Often	Often	often
Tends to follow open standards, making migration easier	Often	Sometimes	Sometimes	Occasionally
Designed for reuse, generalized and well tested	Usually	Often	Sometimes	Occasionally
Often updated and improved	Usually due to competitive pressure	Occasionally	Sometimes	Seldom

Table 1 reveals that though commercial of the shelf software (COTS) may sound a better deal in most organizations because of reduction in the overall cost of the software and development time, it still has other limitations as summarized in Table 2.

Table 2: Limitations of Cots

Advantages	Disadvantages of COTS
Quicker time to market	Use involves learning curve, need for integration and further customization
Better reliability	May not meet all user requirements because it is intended for general use
More end user functionality when compared to custom-developed components	Can be difficult to support because source code may not be provided
Support for components across different hardware and environments	Vendors may discontinue support or cease business
Stricter requirements because of its release for general use	

Table 2 reveals that reliability of the software can be greatly compromised when an organization opts to use customized of the shelf software. Support of the software may proof also to be difficult because of the unavailable source code and also incase vendors cease their support or product on the market.

The level of performance in standardized forms of software

Performance is a pervasive quality of software systems; everything affects it, from the software itself to all underlying layers, such as

operating system, middleware, hardware, communication networks, etc. Software performance is a serious problem in significant fraction ofsoftware projects. This may cause delays, cost overruns, failures on deployment, and even abandonment of projects, but such failures are seldom documented.

By adopting standard packages, organizations substantially reduce the costs, risks and delays associated with custom software development, and benefit from the on-going support services provided for packages by vendors. These packaged solutions come with built-in assumptions and procedures about organizations' business processes. The basic purpose of standardization is to achieve the most efficient use of resources. Performance measures are recognized as an important element of all Total Quality Management programs. According to the IEEE Standard Glossary of Software Engineering Terminology [5], the quality of software products is defined as: the degree to which a system, component or process meets specified requirements and the degree to which a system, component or process meets the needs or expectations of a user of which performance of the software is one of the user needs.With regard to planning and engineering, performance can be equated with minimizing time and costs.

A standardized system interface, strictly object-oriented working, and centralized data management mean data consistency across all planning steps including automatically updated system documentation. However, process control also becomes more complex as the multi-layer nature of automation engineering increases and it merges more and more with information technology [17]. For example, enterprise resource planning (ERP) systems are packaged software applications, and the majority (~60%) of project cost is devoted to setup, installation and customization of the software, services that are typically provided by outside consultants such as Andersen Consulting or EDS as in [4],[14]. Their Success or failure hinges on the effective collaboration among these teams, the business knowledge of internal business experts and the technical skills of outside IT consultants [14].

Ways of improving performance in customizable layered software products

Changing packaged software to meet user needs is the essence of customization. Packaged solutions normally involve software or services that are tailored to achieve a specific scope of work and are intended to meet a broad-spectrum needs of a class of organizations, rather than the unique needs of a particular organization, as is the case in custom software development.

Layered architectures are commonly used and recognized in software development. Layering improves maintainability and testability of software-intense systems. The layered architecture for software systems should be anchored in system architecture, in order to have a common layered architecture of all the (software) components in the system, thus enabling to easily connect them and to be able to foresee Test Access Mechanisms [11] for system on system architecture level and optimize them according to the layers of software architecture.

Software as a Service (SaaS) has become a focus of many enterprises as it provides software application as Web based delivery to serve many customers. The sharing of infrastructure and applications provided by SaaS has a great benefit to customers, since it reduces costs, minimizes risks, improves their competitive positioning, as well as seeks out innovative[3].

Although SaaS application are generally developed with standardized software functionalities to serve many customers as possible, many customers often ask to change the standardized format provided functions according to their specific business needs, and this can be achieved through the configuration and customization provided by the SaaS vendor. Ref [19], asserts that this new form of software distribution is called Software as a Service (SaaS).

Even though, vendors adapt the best practices modelled in the ERP system, in order that the provided functions can be used by a huge number of customers, still many customers ask to tailor some functions to their business needs thus narrowing the gap between company-specific business processes and system-embedded best practices.

Vendors provide their customers tools to do their own customizing and configuration, by using Multi-tenancy Architectures (MTA) [8]. This is an architectural pattern in which a single instance of the software is run on the service provider's infrastructure, and multiple tenants access the same instance. On the other hand, in single-tenant environment every tenant has his own customized application instance.There are: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), as in Figure 1.

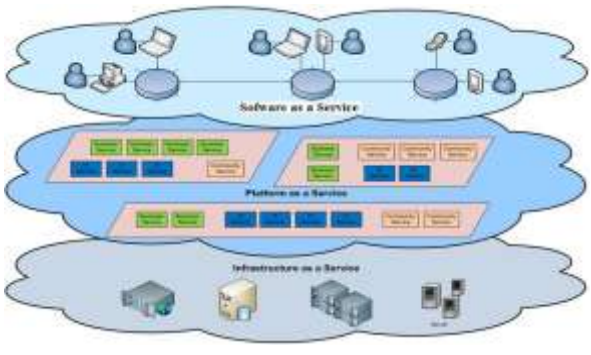

Figure 1: Layered architecture

In multi-Tenancy SaaS environment, without a good architecture and strategy the application performance is a big issue for service providers. To enable a good performance to clients SAP is uses the techniques such, Tenant configuration isolation. The system first checks in the central multi-tenant configuration file where the configuration tenant file is located, and then reads the file. In this case the system only deals with a small configuration files. The other technique is Tenant database isolation in which having different database for each tenant speeds up the read of the configuration which are saved in the database side.

The multi-layer architecture such asSAP web application is based on a multi-layer architecture and the configuration of the application is often organized in a way that changes in one layer thus will speed up the load of the application changes [3]. The building blocks for customizations consist of modular features with characteristics of a software system or systems in a domain as in [10]. These features define both common facets of the domain as well as deference's between related systems in the domain. They make each system in a domain deferent from others. Ref [12], asserts that customization when used can create high levels of complexity and cost, unlessa firm takes the time to create and periodically renew a well-defined, layered architecture for its software products. Therefore in modeling an ICT-intensive organization, software Infrastructure are often considered as in Figure 2.

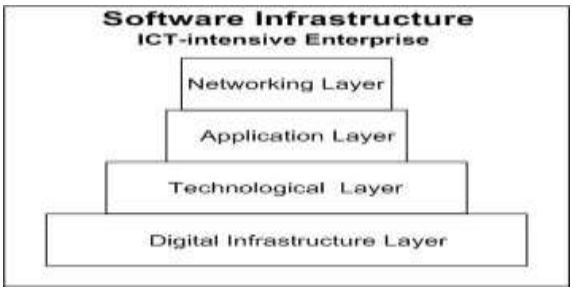

Figure2: Software Infrastructure in an ICT-intensive Enterprise

The Figure 2, shows the digital infrastructure as the basic digital platform with PCs, standalone or linked in networking and operating systems and technological layer as an architecture which is open, flexible and service-oriented. All software tools have a modular structure that recalls useful services for a particular business goal. The application layer is the set of software applications that supports different functions and business processes. The networking layer includes the software that interacts with external environment: customers, suppliers, partners and all market players.

Therefore an ICT-intensive enterprise must have a robust, integrated, interoperable and intelligent infrastructure where software modules can exchange data to automate, in an efficient and effective manner as in [4].Ref [7], notes that heap layers are more flexible and efficient infrastructure that can be used for building custom and general-purpose allocators. This infrastructure is based on a combination of C++ templates and inheritance called mixins. Mixins are classes whose superclass may be changed. Using them allows the programmer to code allocators as compassable layers that a compiler can implement with efficient code. This technique allows programmers to write highly modular and reusable code with no abstraction penalty.

IV. CONCLUSIONS

Performance properties include involve quantities such as: time durations, occurrence frequencies, probabilities, repetitions and data sizes. Performance analysis normally yields information like average response time of components or systems, mean throughput capacity, resource utilization or probabilities of missing delay targets. Therefore, software architecture plays an important role in determining quality attributes, such as modifiability, reusability, reliability, and performance, of a layered software system with focus on customized software systems. While a good architecture cannot guarantee attainment of quality goals, a poor architecture can also prevent their achievement. While decisions made at every phase of the development process can impact on quality of software, architectural decisions have the greatest impact on quality attributes such as modifiability, reusability, reliability, and performance.Most performance failures are due to a lack of consideration of performance issues early in the development process and in the architectural design phase. Further, poor performance is more often the result of problems in the architecture rather than in the implementation. Performance in layered software is therefore a function of the frequency and nature of inter-component communication, in addition to the performance characteristics of the components themselves, and hence can be predicted by studying the architecture of a system

REFERENCES

[1] K. Bradford, &L. Vaughan, Improve Commercial-off-the-Shelf (COTS) IntegrationEstimates.2004
[2] C.Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. Hart, EnablingMultitenancy: An industrial experience report. 2010
[3] Z. Djamal, Configuration in ERP SaaSMulti-Tenancy. 2009
[4] R. Dolmetsch, T. Huber, E. Fleisch, and H. österle, Accelerated SAP.1998
[5] IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990
[6] IEEE 1471-2000 standard
[7] B. Gilad, and C. William, Mixin-based inheritance. 1990
[8] D. Jacobs, and S. Aulbach, Ruminations on multi-tenant databases. 2007
[9] D. Jyotirmoy Dutta nd. Understanding PLM System customization
[10] K.Kang, S.G.Cohen, J.A. Hess, W.E. Novak, A.S.Peterson, Feature-Oriented DomainAnalysis (FODA) Feasibility Study.1990

[11] J.Lamm, A. Espinoza, and A.K. Berg, System tests for reconfigurable signal processingSystems. 2009
[12] C. Mann, 'Why software is so bad', *Technology Review*.2002
[13] H. Marc, "Modular, layered architecture: the necessary foundation for effective massCustomization in software". 2005
[14] H. Osterle, E. Feisch, and R. Alt, R. "Business Networking: Shaping Enterprise Relationshipson the Internet
[15] N. Pollock &J. Cornford, "Customizing Industry Standard Computer Systems for Universities. 2004
[16] K. Rajeev, "Efficient Customization of Software Applications of an Organization". 2013
[17] AG. Siemens. "The SIMATIC PCS7, Process Control System. 2013
[18] T. Simpson, K. Umapathy, J. Nanda, S. Halbe, and B. Hodge,

'Development of a frameworkfor web-based product platform customization', Journal of Computing and InformationScience in Engineering. 2003
[19] M. Turner, D. Budgen,P. Brereton, "Turning software into a service".2003
[20] P. C. Clements, "Coming Attractions in Software Architecture". 1993
[21] P.Clement, F. Bechmann, L.Bass, D.Carlan, J. Ivers, R. Little, P. Merson, R.Nord andJ.Stafford. Documenting Architectures Layers (CMU/SEI-2000-SR-004)