Middle-East Journal of Scientific Research 13 (11): 1471-1482, 2013
ISSN 1990-9233
© IDOSI Publications, 2013
DOI: 10.5829/idosi.mejsr.2013.13.11.1394

Applying Software Architecture Comparison Analysis Method for a Critical System (Sacamcs) Based on Sacam[1] (Concept and Case Study)

Shady Gomaa Abdulaziz and Zarinah Mohd Kasirun

Department of Software Engineering, Faculty of Computer Science and IT,
University of Malaya, Kuala Lampur, 50603 KL, Malaysia

Abstract: Comparing the processes of architecting software for large scale systems is a difficult task that depends on different factors and various environments. However, software architecture are structured according to particular specifications, constraints and rules and are oftentimes badly documented. Therefore, with the lack of information about it, there is a need to develop a software architecture comparison analysis method for critical systems, based on a comparative analysis between software systems products viewed as a black box. Because such a system depend on various criteria, including the ones that derive from the reference organization's business goals, its architectures can be compared based on a set of criteria, including criteria derived from those business goals. Through a literature review on this research area, we explored the most popular methods for comparing software architecture and found it to be the Software Architecture Comparison Analysis Method (SACAM) developed by the Software Engineering Institute (SEI). SACAM compares the architectures of software systems (not the implementation codes) according to their metrics; it does not, however, address enterprise architecture issues, such as implemented software evolution and maintenance. This paper attempts to fill the above gaps by proposing the method called Software Architecture Comparison Analysis Method for Critical Systems (SACAMCS), which is based on multi-criteria decision analysis. The proposed method has been validated using a suitable case study to compare among two critical systems (check-in systems) used in international airports.

Key words: Software Architecture · SAEMs · SACAM · SACAMCS · Chick-In system · CUTE · CUSS Kiosk

INTRODUCTION

The software architecture of a system is defined as the structure of the system comprising software components, the externally visible properties of those components and the relationships among them [1].

This definition focuses only on the internal aspects of a system and most of the analysis methods are based on it. Another brief definition given by Garlan and Perry [2,3] establishes software architecture as the structure of components in a program or system, their interrelationships and the principles and guides that control the design and evolution in time. This process-centered definition is used by Software Architecture Evaluation Methods (SAEMs[2]) because it takes into account the presence of principles and guidelines in the architecture description; thus, we can deduce that software architecture is the draft presenting the software structure in details. By definition, "the software architecture of a computing system is the set of structures needed to reason about the system, which comprise software components, relations among them and properties of both"[4].

Critical systems are technical or socio-technical systems that people or businesses depend on. If these systems fail to deliver their services as expected, then serious problems and significant losses may result.

[1]Software Architecture Comparison Analysis Method
[2]Software Architecture Evaluation Methods

Corresponding Author: Shady Gomaa Abdulaziz, Department of Software Engineering, Faculty of Computer Science and IT, University of Malaya, Kuala Lampur, 50603 KL, Malaysia.

Typically, there are three main categories of critical systems: Safety-critical systems, Mission-critical systems and Business-critical systems [5]. Socio-Technical systems are those that include hardware, software components, procedures and operational processes. The most important emergent property of a critical system is its dependability. Dependability has been proposed to cover the related system attributes of availability, reliability, safety and security [6]. Reliability and availability are usually considered to be the most important dimensions of dependability. Critical systems are complex ones that usually have long lifetimes [3]. Examples include check-in systems at airports, heart-lung machines, nuclear reactors, braking systems and air traffic control systems. Such critical systems have undergone continued improvements and modifications to adapt to new generation technologies, such as by using new operating platforms, high-level programming language and so forth. With the passage of time, the critical software system becomes legacy software, because it has been developed in a totally different environment "in the past," making them obsolete forms of technology.

Legacy software often refers to business-critical software that is maintained, because replacing it requires high business risk operation. "Many legacy systems are critical to the operation of the organization which uses them. They embed business knowledge and procedures which have emerged over the lifetime of the system. The risk of scrapping and rewriting these systems is very high; organizations cannot afford to make their legacy systems obsolete" [5].

- Legacy systems are considered a high business risk for two reasons:

The first reason is that they summarize a great deal of the business's embedded knowledge, which is practically very difficult to replace. They cannot risk losing their data, which is why so many legacy systems are still in use today. This heavy reliance on the system itself is a major risk, especially since the failure of the system can lead to severe financial implications for the business. The second reason is that maintaining these systems is a very costly and painful business and becomes a major risk due to its inability to adapt to modern business requirements.

Modernizing a legacy system should be based on a realistic assessment, through which it can be determined the most suitable strategy for evaluating and improving these legacy systems. These assessments evaluate the legacy systems based on business value assessment and system quality assessment. This is done by measuring the system's technical quality factors, such as performance, interoperability, failure rate and maintenance costs.

As shown in Figure 1 decision makers or "stakeholders" typically have the following strategic options (decisions) when modernizing a legacy system:

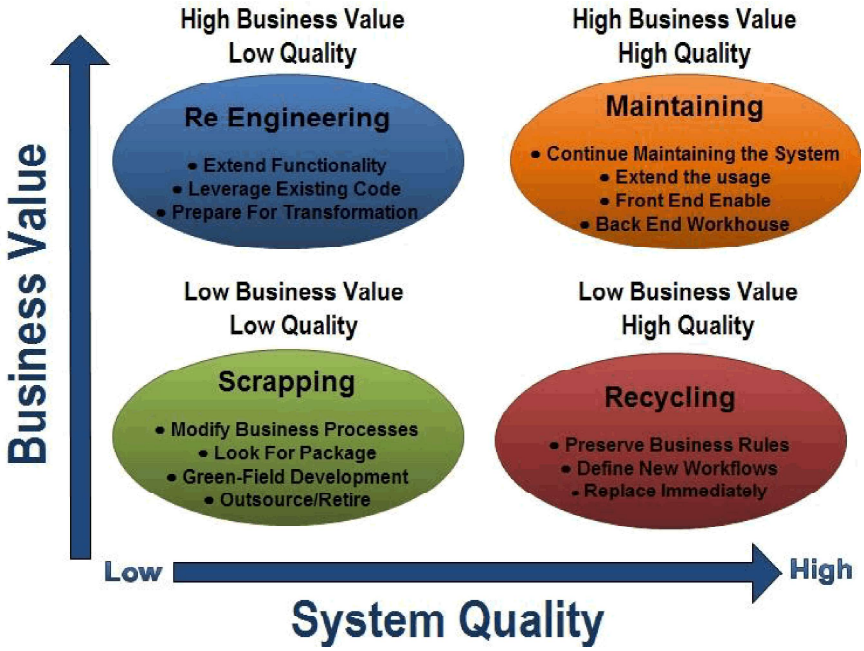

Fig. 1: Four Strategies 'Decisions' for modernize a Legacy System

- Scrap the system completely and modify business processes so that it is no longer required;
- Continue maintaining the system;
- Transform the system by re-engineering it to improve its maintainability; or
- Replace the system with a new system.

Software Architecture Comparison Analysis Methods (SACAM) are used in the analysis and comparison of software products and are used in a software modernization situation assessment. These methods provide organizations with a rationale for an architecture selection process by comparing the fitness of software architecture candidates being used in envisioned systems [7]. Software architectural evaluation can be performed at two phases of the software life cycle. Software architecture can be evaluated before its implementation (early evaluation) or after its implementation (late evaluation) [7-10] Early software architectural evaluation can be performed based on the specification design, description of the software architecture and other sources of information, such as interviews with architects. Late software architectural evaluation is conducted on the basis of the metrics [8,11,12] and can be used for evaluating the existing systems before future maintenance or improvement of the system as well as for identifying architectural strengths and weaknesses points [8,11].

Related Work: There is little research related to SACAM. Stoermer, Bachmann and Verhoef identify and characterize a software architecture comparison analysis method for use in conducting comparisons among software architectures and extracting the advantages and limitations in the process of developing or establishing a new software system [13]. In addition, they also provide the generalized version of the SACAM, which is built based on a process of comparing architecture candidates and then selecting one from these. However, this is different from comparing real case studies of software architecture in practical life to achieve the business goals of organization, which is the basis for the comparison criteria. As such, SACAM is not suitable for a critical system environment.

Furthermore, it provides a system of scoring consisting of three value ranges: not satisfied, fully satisfied, or satisfied under certain conditions in general. These issues are discussed in this paper. The improved version or proposed adoption of SACAM in the present work is called Software Architecture Comparison Analysis Method for Critical Systems (SACAMCS). This aims to help decision makers who select software architectures to explore the fitness of the current existing software architectures for taking a decision about the required software, i.e. the "software of the envisioned system", based on the comparison among software architectures.

The rest of this work is structured as follows: Section 3 presents an overview of SACAM. In section 4 SACAMCS is discussed before describing the case study and the experiment in Sections 5, 6 and Section 7 describes the conclusion and future work on this topic.

SACAM: The SACAM was first developed by the Software Engineering Institute (SEI) for making comparisons between software architectures depending on multi-criteria extracted from an organization's business goals.

It provides organizations with a rationale for an architecture selection process by comparing the fitness of software architecture candidates being used in envisioned systems [13].

Performing the Comparison: The comparison is performed using a series of steps as described below:

- Step 1 (Preparation). In this step, the available inputs to prepare a successful application of the method are prepared. These include (1) architecture candidates (those that should be compared) and (2) business goals (the source of the comparison criteria).
- Step 2 (Criteria Collation). Here, a set of criteria for the architecture comparison is identified. A criterion formulates a requirement for the architecture to support the organization's business goals. Criteria are refined into quality attribute scenarios.
- Step 3 (Determination of Extraction Directives). In this step, the architectural views, tactics, styles and patterns the evaluators are looking for during the following extractions are determined to find supporting evidence for the scenarios of Step 2.
- Step 4 (View and Indicator Extraction). Here, the architectural views for each candidate are extracted according to the extraction directives from Step 3; this step also detects the indicators that support the quality attribute scenarios from Step 2. Architecture recovery techniques may be needed to generate relevant views.
- Step 5 (Scoring). In this step, each criterion is scored for an architecture candidate. The scoring is based on the evidence provided by Step 4 and the quality attribute scenarios determined during step.

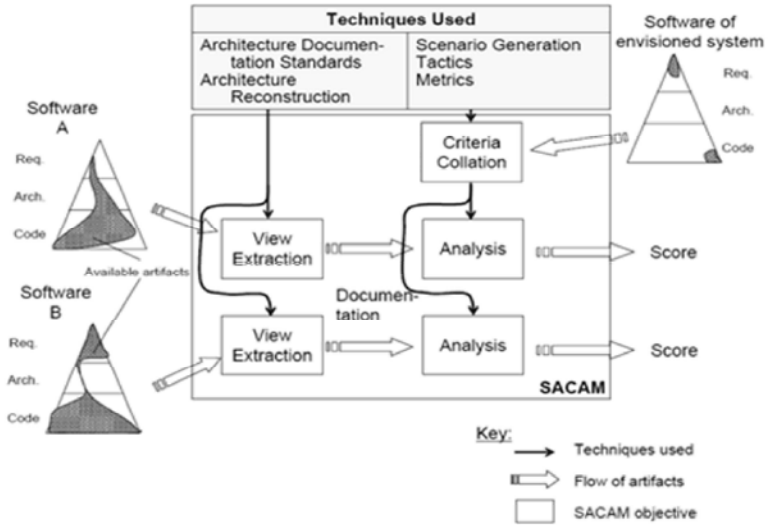

Fig. 2: Software Architecture Comparison Analysis Method (SACAM)

The scoring might consider weights that are provided by stakeholders for the criteria. The scoring provides the reasoning and the resulting score for how well the criteria scenarios are supported by a candidate.

- Step 6 (Summary). This step summarizes the results of the analysis.

In order to generate the necessary artifacts, that is the architectural documentations that are then analyzed to provide the final scores for each architecture, SACAM uses the techniques that are listed in Figure 2.

Sacam Techniques

Scenario Generation: SACAM requires criteria derived from the business goals of an organization for an envisioned system. The criteria are articulated in quality attributes that are further refined into quality attribute scenarios. Scenario generation is a technique for capturing quality attributes and refining them into quality attribute scenarios. SEI methods that incorporate scenario generation are the architecture tradeoff method [10] and the quality attribute workshop (QAW) [14].

Tactics: To achieve particular qualities that are addressed with scenarios, the notion of tactics strategies to achieve quality attribute goals is introduced [15]. In the analysis, SACAM uses tactics as indicators to evaluate if the extracted views support a criterion articulated as a quality attribute scenario. Collections of tactics are available for a variety of quality attributes [1].

Metrics: Metrics support quantitative analysis that provides useful indicators of overall complexity, where change might either be difficult or most likely. Metrics are used in SACAM on the code level, if available, or on a detailed design level [16,17].

Architectural Documentation Standards: SACAM requires the availability of architectural documentation to perform the comparison criteria analysis. Experience shows that architectural documentation across system is heterogeneous. For example, there are differences in notations, stakeholders, level of documentation detail and scope. One of the SACAM's challenges is to obtain comparable architectural documentation. It uses the "views and beyond" architectural documentation approach [18].

Architecture Reconstruction: The architectural documentation used to perform the comparison might be unavailable, insufficient, or out of date. In these cases the architecture has to be reconstructed. However, only the relevant views have to be reconstructed. For example, if the intention is to find the architecture best suited to support modifiability, views showing module dependencies are more important than process views. This goal-oriented approach is used in the Quality-Attribute-driven Software Reconstruction (QADSAR) method[19].

The Findings of SACAM: SACAM is an analytical framework that allows the comparison of several architectures without specific criteria, especially at the scoring phase. The SACAM may be used with some

adaptations or enhancements in both early software architectural evaluation (before its implementation) and late software architectural evaluation (after its implementation). Scores are given to each software architecture candidate, the generated artifacts such as scenarios, tactics and architectural views, from which recommendations for the decision making process are made. The improved version or proposed adoption of SACAM is described in the next section

Proposed Adoption Method "SACAMCS": The SACAMCS is an improved version of SACAM for a critical system environment. We propose the application of SACAMCS as an early and late SAEMs. SACAMCS has been developed in a technical reuse context of the business-critical system to provide decision makers with answers to a question like: "What is the process that we should enact for selecting the rationale architecture from the current environment?" (Selection process). It is as follows, the answer that SACAMCS gives to such a question: “In order to enact the Selection process, you are expected to proceed by comparing the "suitability" or structure of available software architectures”.

Hence, on one side SACAMCS can help those organizations, which already examine and analyze the common architectural features and differences (e,g. for exploring the software architecture design for software product line architecture), in deriving a suitable decision about selecting the proper strategy to software modernization. On the other side SACAMCS can help organizations to work on existent software architectures, by evaluating them based on the available implementation data, and using metrics to reconstruct the actual software architecture, so allowing comparison with a planned architecture [20-22].

In other words, SACAMCS supplements the steps and the techniques used by SACAM in order to allow comparisons between implemented software products; thus, it can be used in software modernization situation assessments. As shown in Figure 3, the evolution (maintenance history), existing architectural documentation and interviews (user satisfaction) are included in the comparison and analysis of candidate architectures. The logical reason for including the evolution (maintenance history) is that implemented software systems are subject to constant change, which cause changes in the architecture of the system with over time.

SACAMCS is performed in the context of the business-critical systems, in which architecture evolution and maintenance are considered important factors in choosing one of four strategies to software modernization according to Figure 1. SACAMCS is an architecture selection method based on a multi-criteria decision analysis. According to this example, there is one a legacy system in an organization and there is one or more further,actual or potential, critical systems; then, the decision makers can identify which of them is the best by using the SACAMCS in order to choose the right software modernization strategy. Thus, SACAMCS provides the decision makers with four options (and not just one, as when SACAM is used).

SACAMCS uses scenarios, existing architectural documentation of the potential architecture candidates and existing maintenance reports of these systems.

The Complementary Techniques Proposed by Sacamcs: The complementary techniques proposed by SACAMCS are presented as follows:

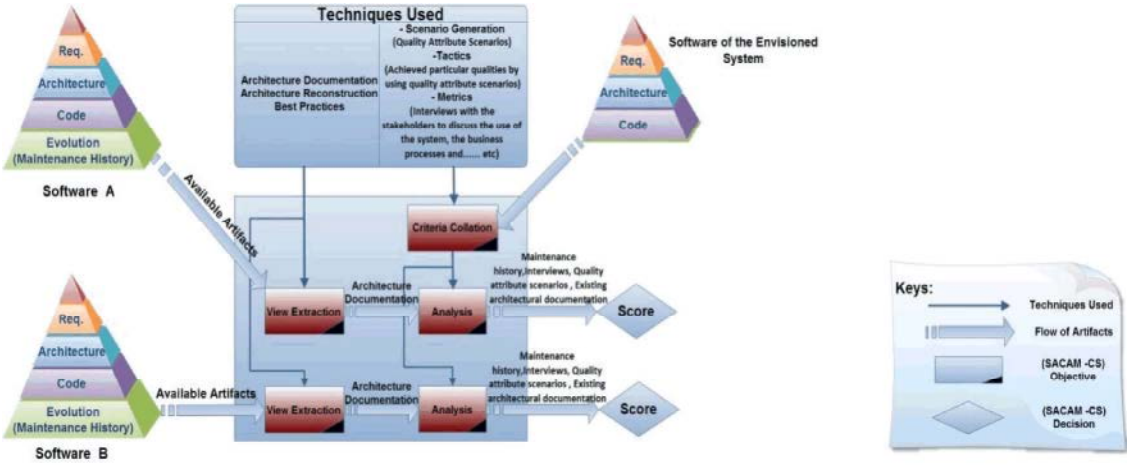

Fig. 3: Software Architecture Comparison Analysis Method for Critical System (SACAMCS)

Scenario Generation: Let SACAMCS be applied to compare two implemented systems. The goal of the scenario will be to show which system is more efficient and reliable by using the following quality attribute scenarios such as:

- Availability- concerned with mean time to failures, fault detection and repair time.
- Modifiability- concerned with the changes of platforms, environment and requirements; these cost money and time.
- Performance- concerned with timing in the system and response time.
- Security- concerned with the ability of the system to prevent unauthorized usage, such as attacks to access data and system service as well as hinder normal system operation.
- Testability- refers to the process of finding and solving errors and how easy it would be to test the system so as to monitor control inputs and outputs; such testing uses up to 40% of system cost.
- Usability- refers to the process of minimizing user error and using the system efficiently to accomplish tasks.

Tactics: To achieve particular qualities that are mentioned in the scenarios, an evaluation of the performance of the systems and strategies to achieve quality attribute goals has been presented in a previous work[15].

Metrics: In assessing implemented software products, we should consider both product business value and product technical quality. To assess the business value of a system, stakeholders are interviewed to discuss the use of the system, the business processes that are supported, the system dependability and the importance of the system outputs. To assess a software system from a technical perspective, we need to consider both the application system itself and the environment in which the system operates.

The environment is important because many system changes result from changes to the environment, such as upgrades to hardware or operating system. If possible, in the process of environmental assessment, we should monitor the system and its maintenance processes. Examples of data that may be useful include the costs of maintaining the system, the number of hardware faults that occur over some time period and the frequency of patches and fixes to the system support software. To assess the technical quality of an application system, there is a need to assess a range of factors that are primarily related to the system dependability and documentation. Quantitative system data must also be collected to help decision makers evaluate system quality.

Examples of data that might be collected are as follows: the number of system change requests, the number of user interfaces and the volume of data used by the system. The business value of a legacy system and the quality of the application software and its environment should be assessed in order to determine if a system should be replaced, transformed, or maintained.

Case Study Background: The aviation industry has grown at an unprecedented rate. To cope up with the growth airports have to expand the terminal facilities and meet new standards of operational efficiency. Airports need software systems that support their ability to evolve in response to their rapidly changing environment. Legacy systems that limit a business's adaptability are seen as significant problems [23]. In response to this situation, airports have started implementing new technologies at the terminals for convenience of the passenger. The new solutions strive to improve operational efficiency and reduce queues at the airport. The new technologies like self-service and web check-in are being installed at many airports to increase Check-in capacity. There are two major check-in systems commonly in use by international airports: The Common User Terminal Equipment (CUTE) system; and the Common Use Self Service (CUSS) system [24].

As illustrated in figure 4, with the CUTE system the passenger arrives at the airport and approaches the check-in counter.The check-in process is a one-step process where he/she can interact with the check-in agent and decide on seats and drop bags.With the CUSS system passengers with baggage can drop the bags at the baggage drop-off and proceed to the security check.

The CUTE System: CUTE was 1st implemented in 1984 for the Los Angeles Summer Olympic Games. From 1984 until the present, approximately 400 airports worldwide have installed some level of CUTE[24]. CUTE systems allow an airport to make gates and ticket counters in common use. These systems are known as "agent- facing" systems, because they are used by the airline agents to manage the passenger check-in and boarding process. Whenever an airline agent logs onto the CUTE system, the terminal is re-configured and connected to the airline's host system. From an agent's point of view, the agent is working within his airline's information technology (IT) network [24].

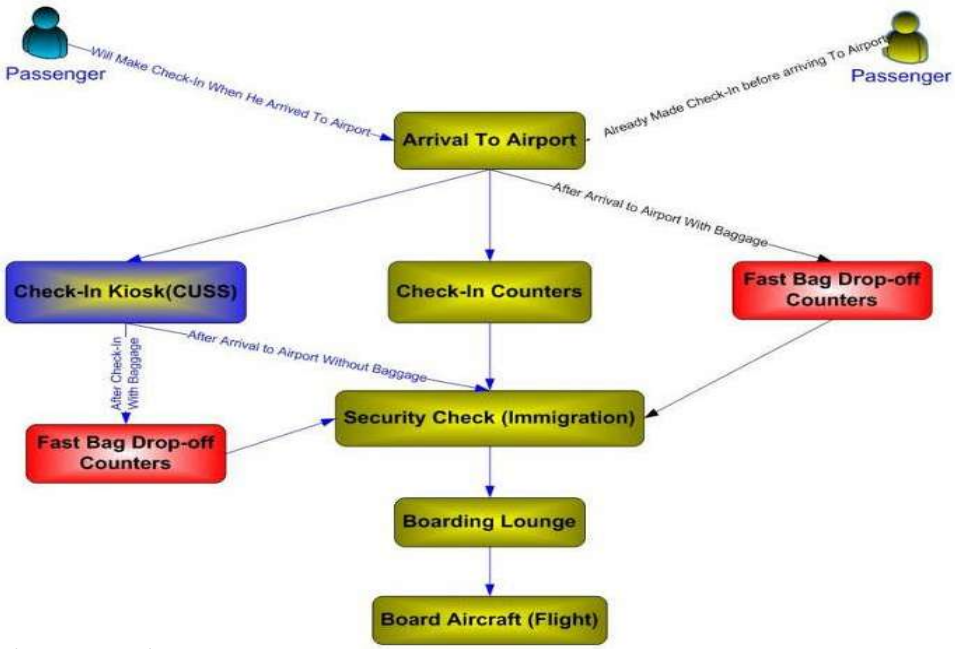

Fig. 4: Check-in Context Diagram [24]

The Air Transport Association (IATA) describes the factors to be considered for the design of the check-in area for the check-in desks with CUTE. IATA provides some standard thumb rules based on the queuing theories and which are very useful in sizing the overall terminal at the initial stages.

The CUSS System: This evolving pattern enables passengers to obtain boarding passes, check baggage and conduct other transactions at times and places of their convenience. Passenger check in procedures will gradually shift from check in procedures performed at check in counters, to check in procedures performed at home from the internet, by mobile phone, or through self service check in facilities at the airport such as CUSS Kiosk. The trend is towards common use equipment which may consist of free standing column type or counter type workstations with built-in Automated Ticket and Boarding pass (ATB) printer. The CUSS provides ticketed passengers the ability to perform many tasks, not limited to, check-in for flights, select or change a seat assignment and obtain a boarding pass for their departures. The CUSS will be used by self-service passengers to check-in, seat allocation, boarding pass printing and baggage check-in in a common use environment. Self-service is becoming the common check-in mechanism in Europe, US and in many airports. In the MEA-Middle East Area region it started as a dedicated self-service and the first CUSS kiosks have been installed at Cairo Airport International TB3. The CUSS will be designed for the use of different types of passengers with or without luggage where passengers with luggage could use the new use facility of the Common Use Baggage System. The CUSS platform software is responsible for managing the entire Kiosks System, The final configuration of the CUSS kiosk will vary depending on airport operational and security requirements. The equipment required for CUSS consists of two redundant servers (usually the same servers used for CUTE system), located in the MER – Main Equipment Room and self service kiosks[24].

Experience at Cairo International Airport: We have developed an experience at Cairo International Airport (CAI) to compare between the two check-in systems (the CUTE system and the CUSS system) discussed in section 5. To assess the business value and the technical quality, a survey has been elaborated and operational metrics have been collected for both systems that are deployed by Egypt Air Airlines. The assessed quality attributes and the metrics used are listed in table 1.

Measuring Customer Satisfaction: From the survey the majority of passengers consider the CUSS System process is faster than the CUTE System as shown in Table 2 with 68% strongly agree versus 10.75% for the CUTE System.

Table 1: Experiment quality attributes and used metrics

Quality Attribute Used Metrics	Passenger Satisfaction - Process Time
Queuing Time	System reliability Failure rates during 6 months
Service availability	Uptime during one month

Table 2: Rate of Process Speed

	Process speed				
	0% to 49% Strongly Disagree	50% to 64% Somwhat Disagree	65% to 74% Natural	75% to 84% Somewhat Agree	85% to 99% Strongly Disagree
CUTE	23.42%	18.99%	25.32%	21.52%	10.75%
CUTE	0%	0%	8%	24%	68%

Table 3: Rate of passenger satisfaction Figure 5: Comparison between CUSS with Bags and CUTE

	Change / Select Seat				
	0% to 49% Strongly Disagree	50% to 64% Somwhat Disagree	65% to 74% Natural	75% to 84% Somewhat Agree	85% to 99% Strongly Disagree
CUTE	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
CUTE	0%	0%	4 %	4 %	92. %

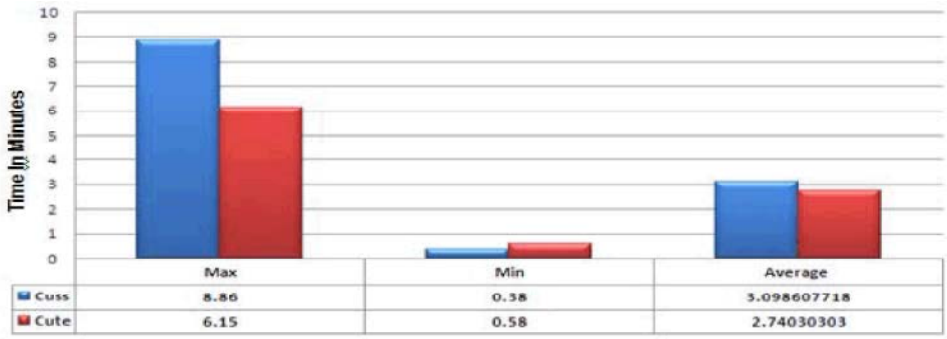

Fig. 5: Comparison between CUSS with Bags and CUTE

Fig. 6: Comparison between CUSS without Bags and CUTE

Also the survey indicates that the majority of passengers are satisfied with the CUSS "Change/Select seat" service as shown in Table 3 with 92% strongly agree versus 0% for the CUTE.

Check-in counters were used by Egypt Air Airlines and there were a maximum of six counters open at the time of observation. As shown in figure 5, the average process time per passenger for the CUTE System is 2.74 minutes.

The CUTE average process time is smaller than the CUSS System. This is a result of the efficiency of the check-in agent and the interaction with the passenger. This human element causes significant variations in check-in times that are shown in figure 5. However the comparison of CUSS without bags and CUTE indicates clearly that the CUSS process time is faster than the CUTE as illustrated in Figure 6. The standard deviation in process time is shown in Figure 7.

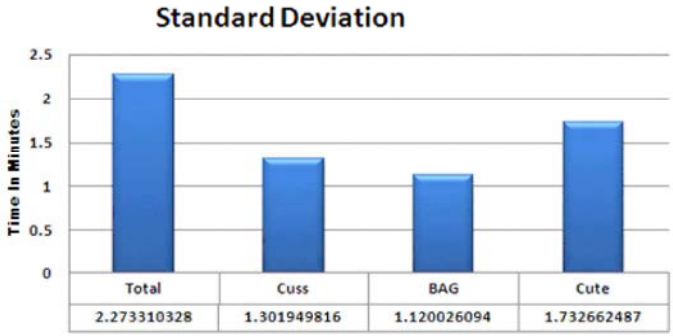

Fig. 7: Standard Deviation of Process Time

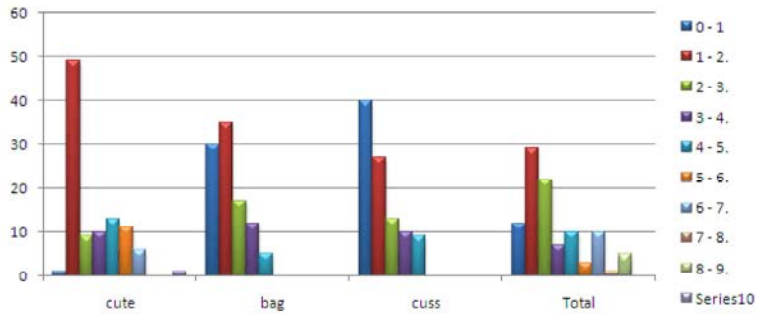

Fig. 8: All ranges of processing time for all Check-In Systems

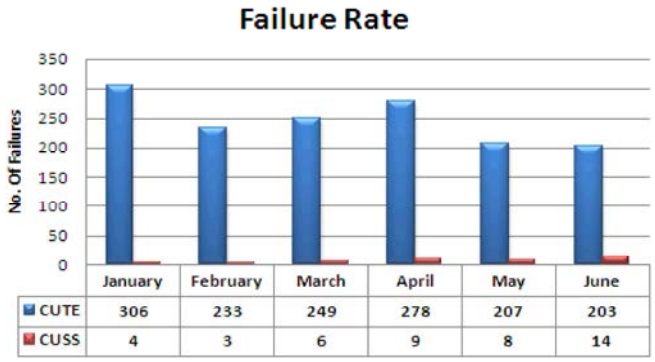

Fig. 9: The Failure rate of Check-In System

Table 4: All ranges of processing time for all Check-In System

	0-1	0-2.	2-3.	3-4.	4-5.	5-6.	6-.7	7-8.	8-9.
CUTE	1	49	9	10	13	11	6	0	0
CUSS	40	27	13	10	9	0	0	0	0
Bag	30	35	17	12	5	0	0	0	0
Total	12	29	22	7	10	3	10	1	5

The standard deviation for the CUTE System is 1.7 while the CUSS without is 1.3 bag and the Total (CUSS+BAG) is 2.3.

As shown in Table 4 and figure 8, the processing time for the CUTE is often between one and two minutes per passenger while the CUSS takes less than a minute. That Passengers using kiosks need to go to baggage drop-off if required to check in bags. The processing time for each process is shown in Table 4. The characteristics for each method are discussed in detail in this section.

Analyzing Failure Rates: Failure rate is the frequency with which an engineered system or component fails, expressed for example in failures per hour. It is often denoted by the Greek letter ë (lambda) and is important in reliability engineering [25]. The failure rate of a system usually depends on time, with the rate varying over the system age. The measurement of failure rates (hardware, software) for all workstation using CUTE or CUSS System for 6 months is shown in figure 9.

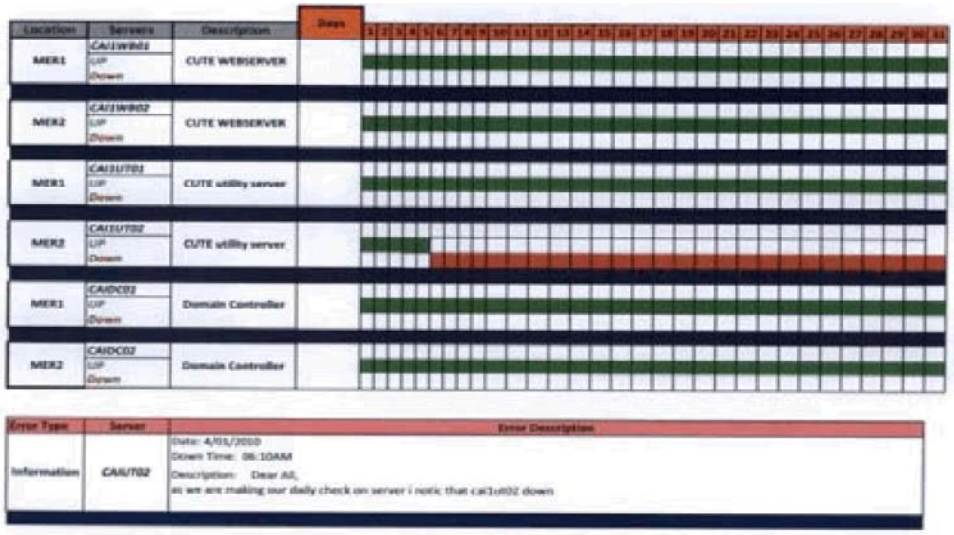

Fig. 10: Time Chart for CUTE/CUSS System in January 2010

Table 5: All Servers of Check-In Systems

Ststem	Server Name	Hour	Failure
CUSS	CAI1WB01	744	No Failure
CUSS	CAI1WB02	744	No Failure
CUTE	CAI1UT01	744	No Failure
CUTE	CAI1UT02	96.72	Failure

Analyzing System Service Availability: Availability means that a system is on-line and ready for access. A variety of factors can take a system off-line, ranging from planned downtime for maintenance to catastrophic failure. The goals of high availability solutions are to minimize this downtime and/or to minimize the time needed to recover from an outage. Exactly how much downtime can be tolerated will dictate the comprehensiveness, complexity and cost of the solution. The following steps are used to measure availability. Step1: Collecting and Presenting system service availability data for each check-in system (CUTE and CUSS) for one month as shown in Figure 10.

Step2: Calculate the failure rate data for each server in each system in each month as shown in table 5. These systems require processing, 24 hours a day, 7 days a week; it means these systems require 24 × 7 availability with 99.9% uptime.

Summary: From the results mentioned above, we can deduce the following: The passenger is satisfied with the process of self-service check-in using CUSS Kiosk, the process of self-service is faster, accurate and easier to understand. Self-service puts control into the hands of the passenger. In the airline industry, this control comes in the form of enabling the passenger to select their own seat, request an upgrade, or change flights. Further to this conclusion, the following was observed:

- The minimum time was 0.38 minutes and on average it takes 3.1 a minute to complete a transaction and print a boarding pass from CUSS System.
- The processing times for the passenger who had some experience of using a kiosk was significantly less than average.
- Most of the passengers need assistance in completing the process and there were two roving agents helping passengers.
- The location of the kiosks made them very accessible and easily visible before the passengers could see the check-in counters.

Conclusion and Future Work: This paper discussed and presented the need to produce an improved version of SACAM in order to use it in a critical system environment. This work also presented the strategies to modernize obsolete critical systems. These needs come from SACAM's limits that are summarized in the following:

- SACAM uses the metric only in code level if available and not in the software level as a whole.
- SACAM compares the architectures of software systems and not the design and the implementation code, so that the SACAM deals with software architecture as a black-box.

- SACAM allows to compare the requirements, but this does not address how the software actually realizes the requirements. On the other hand, it is at the design and the implementation levels that becomes clear how well requirements are fulfilled. However, in order to compare different systems, their codes a d low level design documents are not enough, but comparing different software is almost impossible, because of the massive amounts of information and poor software architecture documentation found.
- SACAM seems not suitable for a critical system environment. In fact, while SACAM is a general tool, which allows the comparison of candidate architectures, from which the ideal strategy is selected, it does not take in count life to achieve the organization's business goals, hence eventually it does not allow to compare software architectures, as they are in practical life.
- SACAM does not take in count including already implemented software evolution and maintenance, hence it does not help address enterprise architecture issues in coming up with suitable score-based architectural decisions.

The main contribution of this paper is the proposed improved version of the SACAM for use in a critical system environment. The improvements include the consideration of the evolution (maintenance history) in architecture evaluation and introduction of complementary techniques that allow software product assessments from both the perspectives of the business value and the technical quality.

In conclusion, the contribution of this paper should be evaluate in the context of a critical system environment, where issues of architecture evolution and maintenance are considered important factors in selecting a strategy for software modernization. Future works are expected to include detailed explanations and illustrations about the SACAMCS analysis criteria. We also expect to validate the proposed extension when SACAMCS is applied to a critical system environment using an industrial case study.

REFERENCES

1. Bass, L., P. Clements and R. Kazman, Software architecture in practice. 2003: Addison-Wesley Longman Publishing Co. Inc.
2. Perry, D.E. and A.L. Wolf, 1992. Foundations for the study of software architecture. ACM SIGSOFT Software Engineering Notes, 17(4): 40-52.
3. Shaw, M. and D. Garlan, 1996. Software architecture: perspectives on an emerging discipline. 1996.
4. Jones, L., 2011. An Architecture-Centric Approach for Acquiring Software-Reliant Systems, 2011, DTIC Document.
5. Sommerville, I., 2011. Software engineering. Pearson.
6. Laprie, J.C., 1995. Dependable computing: Concepts, limits, Challenges.
7. Roy, B. and T.C.N. Graham, 2008. Methods for evaluating software architecture: A survey. School of Computing TR, 545: 82.
8. Bouwers, E., J. Visser and A. Van Deursen, 2009. Criteria for the evaluation of implemented architectures. 2009. IEEE.
9. Mattsson, M. H. Grahn and F. Mårtensson, 2006. Software architecture evaluation methods for performance, maintainability, testability and portability.
10. Clements, P., R. Kazman and M. Klein, 2002. Evaluating software architectures: methods and case studies. Addison-Wesley.
11. Abi-Antoun, M., T. Selitsky and T. La Toza, 2010. Developer refinement of runtime architectural structure. ACM.
12. Purewal, K., L. Yang and A. Grigg, 2009. Quantitative assessment of quality attributes in systems architecture using evidential Reasoning.
13. Stoermer, C., 2003. SCAM: The Software Architecture Comparison Analysis Method, 2003, DTIC Document.
14. Barbacci, M., *et al.* 2003. Quality attribute workshops (QAWs)(CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, 2003, Carnegie Mellon University.
15. Bachmann, F., Deriving architectural tactics: A step toward methodical architectural design, 2003, DTIC Document.
16. Arora, V., *et al.* 1995. Measuring High-Level Design Complexity of Real-Time Object-Oriented Systems.
17. Faust, D. and C. Verhoef, 2003. Software product line migration and deployment. Software: Practice and Experience, 33(10): 933-955.
18. Clements, P., *et al.* 2010. Documenting software architectures: views and beyond.: Addison-Wesley Professional.
19. Stoermer, C., L. O'Brien and C. Verhoef, 2003. Moving towards quality attribute driven software architecture reconstruction. IEEE Computer Society.
20. Lindvall, M., R.T. Tvedt and P. Costa, 2003. An empirically-based process for software architecture evaluation. Empirical Software Engineering, 8(1): 83-108.

21. Tvedt, R.T., M. Lindvall and P. Costa, 2002. A process for software architecture evaluation using metrics.. IEEE.
22. Falessi, D., *et al.* 2010. Applying empirical software engineering to software architecture: challenges and lessons learned. Empirical Software Engineering, 15(3): 250-276.
23. Greenwood, R.M. *et al.* 2006. The Impact of Software-Architecture Compliance on System Evolution. Software Evolution and Feedback, pp: 69-280.
24. Abdelaziz, S.G., A.A. Hegazy and A. Elabbassy, 2010. Study of Airport Self-service Technology within Experimental Research of Check-in Techniques Case Study and Concept. International Journal of Computer Science Issues (IJCSI), 7: 3.
25. Medhat, N., *et al.* 2008. Medical Equipment Quality Assurance by Making Continuous Improvement to the System. 2008. IEEE.