Developing Software Architecture Comparison Analysis Method for Critical Socio-Technical Systems

Ahmed El-Abbassy[1]
Higher Institute of Computer Science & Information Technology, El-Shorouk Academy

Shady Gomaa Abdulaziz[2]
College of Computing & Information Technology, Arab Academy

Abdelfatah A. Hegazy[3]
College of Computing & Information Technology, Arab Academy

Summary
Software Architecture Comparison Analysis Methods provide organizations with a rationale for an architecture selection process by comparing the fitness of software architecture candidates for required systems. Comparing software architectures for any nontrivial system is a difficult task. Software architectures are designed with particular requirements and constraints, and are often poorly documented. With the lack of data about software architecture, developing comparison methods based on a black box approach is considered very helpful, and architectures can be compared based on a set of criteria derived from the business goals of an organization. A popular method for comparing software architectures as black box is the Software Architecture Comparison Analysis Method (SACAM) developed by the Software Engineering Institute (SEI). SACAM compares the architectures of software systems and not the implementation code. SACAM does not address enterprise architecture issues such as implemented software evolution and maintenance. This paper discusses and presents a proposed adaptation of SACAM to be applied in the context of critical socio-technical systems where issues of architecture evolution and maintenance are considered important factors in selecting a strategy to software modernization. The proposed method is called software Architecture Comparison Analysis Method for Critical Systems (SACAM-CS). SACAM-CS is an architecture selection method based on multi-criteria decision analysis. The proposed method has been validated using a suitable case study to compare among two check-in systems used in international airports.

Key words:
Software Engineering, Software Architecture Comparison, Software Evolution, Legacy Software, Socio-Technical Systems, Airport Check-in Systems.

1. Introduction

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. Typically there are three main categories of critical systems: Safety-critical systems, Mission-critical systems and Business-critical systems[1]. Socio-Technical systems are systems that include hardware, software components, procedures and operational processes.

The most important emergent property of a critical socio-technical system is its dependability. Dependability was proposed to cover the related system attributes of availability, reliability, safety and security[2]. Reliability and availability are usually considered to be the most important dimensions of dependability.

Critical Socio-Technical systems are complex systems usually with a long lifetime[3]. Its development continues throughout their life with changes to accommodate new requirements, new operating platforms, and so forth. With time Socio-Technical software becomes legacy software that has been developed in the past using older or obsolete technology. Legacy software is often business-critical software, maintained because it is too risky to replace it. Modernizing legacy software should be based on periodical assessment in order to decide the most appropriate strategy for evolving these systems. Assessment should include both business value assessment and system quality assessment by measuring factors such as performance, interoperability, failure rate and maintenance costs. As illustrated in figure 1 system stakeholders have typically four strategic options (choices) when considering software modernization.

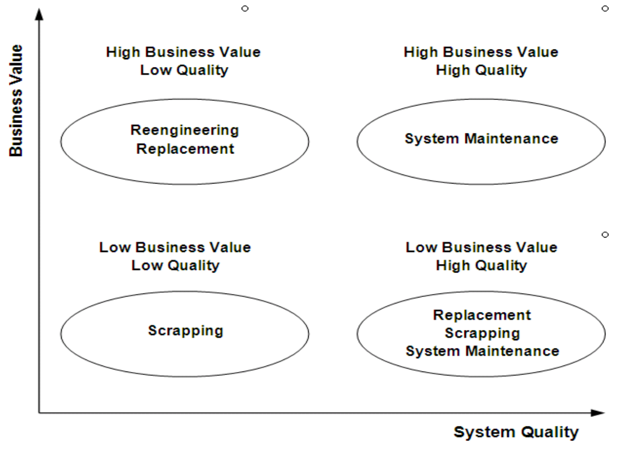


Figure 1: Legacy system clusters & modernization options

Manuscript received November 5, 2010
Manuscript revised November 20, 2010

The Software Architecture Comparison Analysis Methods are used in the analysis and comparison among software products, and are used in 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[4].
Software architectural evaluation can be conducted at different phases of software life cycle. Software architecture can be evaluated before its implementation (early evaluation), or after its implementation (late evaluation) [4, 20, 23]. Early software architectural evaluation can be conducted on the basis of the specification and description of the software architecture, and other sources of information, such as interviews with architects. Late software architectural evaluation is performed based on metrics[20, 22, 25] and can be used for evaluation of existing systems before future maintenance or enhancement of the system as well as for identifying architectural drift and erosion[20, 22]. Software architectures are often poorly documented and with the lack of data about software architecture, developing evaluation methods based on a black box approach is considered very helpful[24].
For both early and late software architectural evaluations, two basic categories are suggested[15]: qualitative evaluation and quantitative evaluation. Qualitative evaluation generates qualitative questions about software architecture to assess any given quality, whereas quantitative evaluation uses quantitative measurements to be taken from software architecture to address specific software qualities.
Techniques for generating qualitative questions include scenarios, questionnaires, and checklists. Scenarios appear to be the most utilized form for acquiring data [16]. There are also empirically-based approaches [17, 21] that define some relevant metrics for software architecture evaluation. The metrics are defined based on the goal of the evaluation.
A popular Scenario-based, early evaluation method is the Software Architecture Comparison Analysis Method (SACAM) developed by the Software Engineering Institute (SEI)[5]. SACAM uses several architecture techniques developed by SEI to compare the architecture candidates. SACAM compares the architectures of software systems and not the implementation code.
Software can be compared on several different levels. It is possible to compare the requirements, but that does not address how well the software actually realizes the requirements. On the other hand, at the implementation level, it is clear how well requirements are fulfilled, but comparing different software is almost impossible because of the huge amount of information. Comparing software architectures provides a manageable level of information.
SACAM does not address enterprise architecture issues such as implemented software evolution and maintenance. These issues are discussed in this paper where a proposed adaptation of SACAM is presented. The proposed method is called Software Architecture Comparison Analysis Method for Critical Systems (SACAM-CS). SACAM-CS will be applied in the context of critical socio-technical systems where architecture evolution and maintenance are considered important factors in selecting a strategy to software modernization. SACAM-CS is an architecture selection method based on multi-criteria decision analysis. The proposed method has been validated using a suitable case study to compare between two check-in systems used in international airports.
The rest of this work is structured as follows: Section 2 presents an overview of SACAM. In section 3 SACAM-CS is discussed before describing the case study and the experiment in Sections 4, 5 and Section 6 describes the conclusion and future work on this topic.

2. SACAM Overview

SACAM is an early evaluation, Scenario-based method that was created to provide the rationale for an architecture selection process by comparing the fitness of architecture candidates for required systems[5].
The comparison is performed in a series of steps as follows:
Step 1 **(Preparation)** examines the available inputs to prepare a successful application of the method. The inputs to the method include: (1) **architecture candidates** – the architectures that should be compared; (2) **business goals** – the source of the comparison criteria.
Step 2 **(Criteria Collation)**, 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**) determines the architectural views, tactics, styles, and patterns that are looked for during the following extractions to find supporting evidence for the scenarios of Step2.
Step 4 (**View and Indicator Extraction**) extracts the architectural views for each candidate according to the extraction directives from step 3; detects indicators that support the quality attribute scenarios from Step 2; Architecture recovery techniques may be needed to generate relevant views.
Step 5 (**Scoring**), 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 2. 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)** summarizes the results of the analysis.

SACAM uses techniques as illustrated in Figure 2. Each technique contributes to the comparison of software architectures.

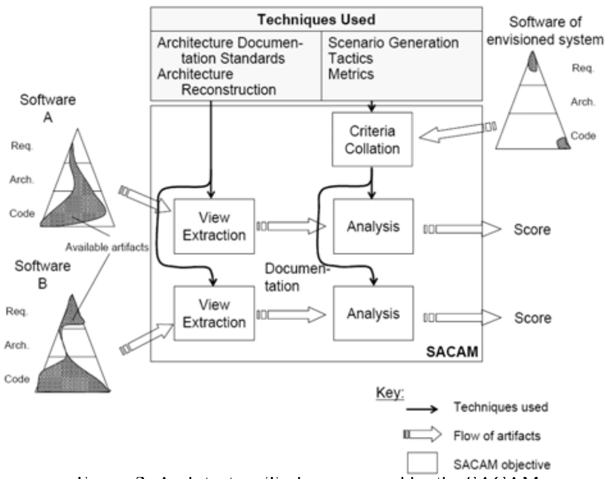


Figure 2: Architecture Techniques used by the SACAM

These techniques are used to generate the necessary artifacts that are then analyzed to provide the final scores for each architecture. SACAM techniques are summarized hereafter.
1) Scenario Generation: SACAM requires criteria that are 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 scenarios generation are the architecture tradeoff method[6] and the quality attribute workshop(QAW)[7].
2) Tactics: To achieve particular qualities that are addressed with scenarios, the notion of tactics strategies to achieve quality attribute goals is introduced[8]. SACAM uses tactics in the analysis 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[9].
3) Metrics: Metrics support quantitative analysis that provides useful indicators of overall complexity where change might be difficult or most likely. Metrics are used in SACAM on the code level, if available, or on a detailed design level[10, 11].
4) 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. SACAM uses the "views and beyond" architectural documentation approach[12].
5) 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 not all architectural views have to be reconstructed, only the relevant views. For example if the intension is to find the architecture that is 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 [13].

The outputs of SACAM include a recommendation for the decision-making process, the scores and the related reasoning for each candidate, and the generated artifacts such as architectural views, tactics and scenarios.

SACAM is a standard framework that allow for comparing several architectures. This important feature permits with some adaptation to use SACAM in both early architecture evaluation (before its implementation) and late architecture evaluation (after its implementation).
The proposed adaptation is described in the next section.

3. SACAM-CS

SACAM-CS is a proposed adaptation of SACAM that permits to apply SACAM not only as an early architecture evaluation method, but also as a late architecture evaluation method.
Late software architecture evaluation can use data measured on the implementation of software architecture, and metrics can be used to reconstruct the actual software architecture, allowing it to be compared to a planned architecture[17, 18, 19].
SACAM-CS complements the steps and the techniques used by SACAM in order to allow the comparison among implemented software products, and consequently could be used in software modernization situation assessment.
AS illustrated in figure 3, the evolution/maintenance history is included in the comparison and analysis of candidate architectures. The rational for including the evolution/ maintenance history is that implemented Software systems undergo constant change causing the architecture of the system to degenerate over time.

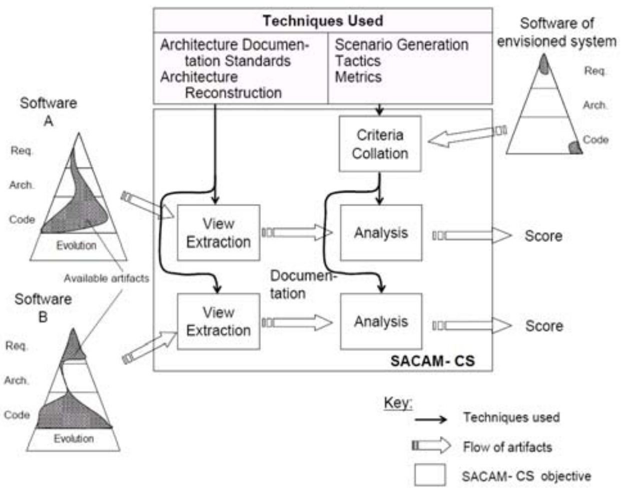


Figure 3: Architecture Techniques used by the SACAM

In what follows the complementary techniques proposed by SACAM-CS are presented.

3.1 Scenario Generation

The SACAM-CS will be applied to compare between two implemented systems. The goal of the scenario is to show which system is more reliable, efficient and achieved all quality attribute conditions by using quality attribute scenarios such as:

(1) Availability is concerned with system failure and duration of system failures.

(2) Modifiability is concerned with cost of change, both in time and money.

(3) Performance is concerned with response time.

(4) Security is the ability of the system to prevent or resist unauthorized access while providing access to legitimate users. An attack is an attempt to breach security.

(5) Testability refers to the ease with which the software can be made to demonstrate its faults or lack thereof. To be testable the system must control inputs and be able to observe outputs.

(6) Usability is how easy it is for the user to accomplish tasks and what support the system provides for the user to accomplish this.

3.2 Metrics

When you are 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, you should make measurements of the system and its maintenance processes. Examples of data that may be useful include the costs of maintaining the system hardware and support software, 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, you have to assess a range of factors that are primarily related to the system dependability and the system documentation. You may also collect quantitative system data that will help you judge the quality of the system. Examples of data that might be collected are: 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 to determine whether the system should be replaced, transformed or maintained.

4. 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. 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 [14].

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.

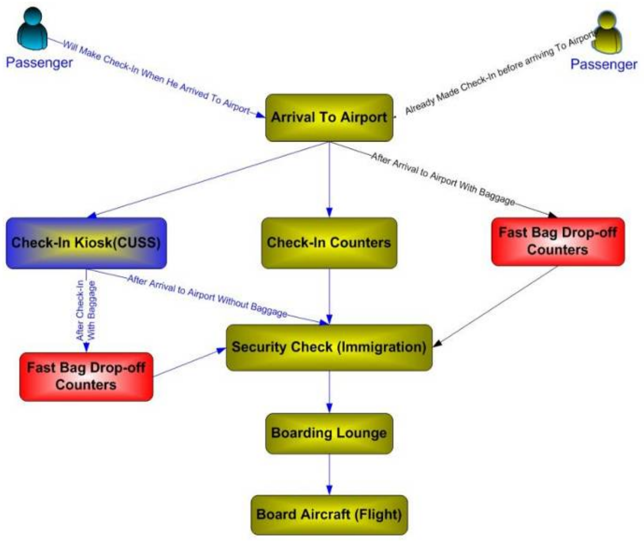

Passenger
Will Make Check-In When He Arrived To Airport
Passenger
Already Made Check-In before arriving To Airport
Arrival To Airport
After Arrival to Airport With Baggage
Check-In Kiosk(CUSS)
Check-In Counters
Fast Bag Drop-off Counters
After Check-In With Baggage
After Arrival to Airport Without Baggage
Fast Bag Drop-off Counters
Security Check (Immigration)
Boarding Lounge
Board Aircraft (Flight)

Figure 4: Check-in Context Diagram

4.1 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. 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.

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.

4.2 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.

5. 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 4. 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:

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

5.1 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 2: Rate of Process Speed

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

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.

Table 3: Rate of passenger satisfaction

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

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.

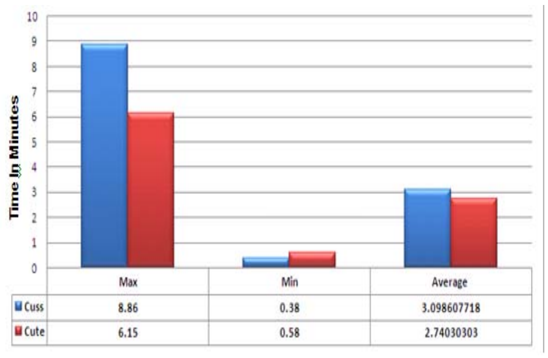


Figure 5: Comparison between CUSS with Bags and CUTE

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.

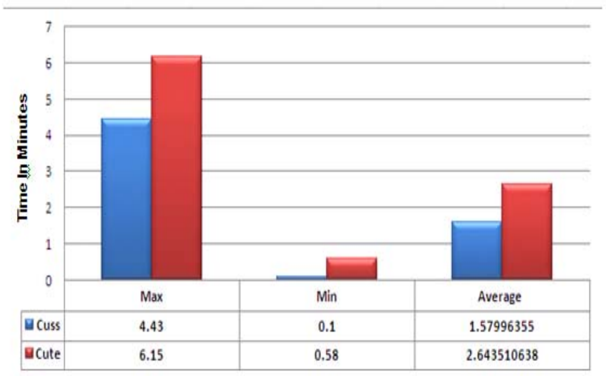


Figure 6: Comparison between CUSS without Bags and CUTE

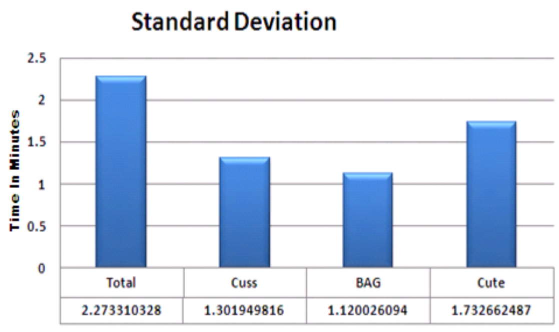


Figure 7: Standard Deviation of Process Time

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.

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

	0 - 1	1 - 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

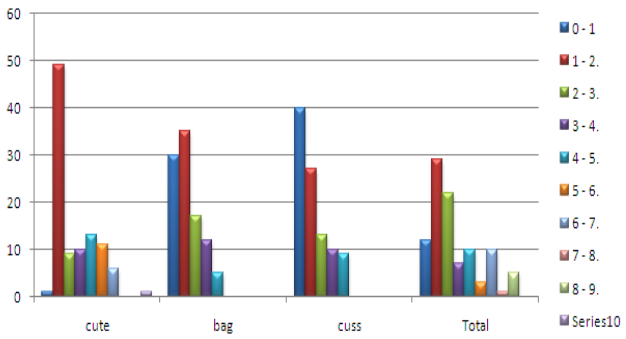


Figure 8: All ranges of processing time for all Check-In Systems

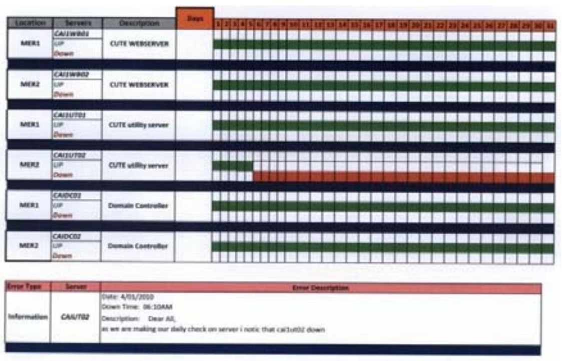
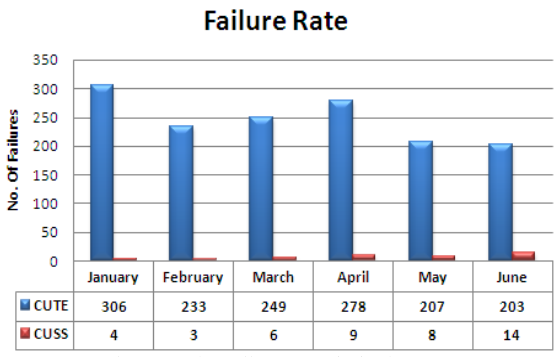


Figure 9: Time Chart for CUTE/CUSS System in January 2010

5.2 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 λ (lambda) and is important in reliability engineering.
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 8.



	January	February	March	April	May	June
CUTE	306	233	249	278	207	203
CUSS	4	3	6	9	8	14

Figure 8: The Failure rate of Check-In System

5.3 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 9.

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.

Table 5: All Servers of Check-In Systems

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

$$\frac{\mathbf{1}\ \boldsymbol{Failure}}{\mathbf{840.72}\ \boldsymbol{Hours}} = \mathbf{0.001198}$$

$$\boldsymbol{or}\ \mathbf{0.001198}\ \boldsymbol{failures\ for\ every}\ \mathbf{744}\ \boldsymbol{hours\ of\ operation}$$

5.4 Queuing Time

Queuing theory deals with the mathematics involved in the study of waiting lines, or queues. The basis of the problem is a customer that arrives and joins a queue waiting for service, and then he receives the service and exits. To study queuing systems, information is required on the arrival process, the service process and the queue discipline. The arrival process deals with how arrivals are distributed over time. Most of the work in queuing theory has revolved around a Poisson distribution; this means that the inter-arrival time between successive arrivals is unrelated and exponentially distributed.
The service process requires information on the service time, again a Poisson distribution is commonly used, the number of servers and whether the servers are in series or parallel.
The queue discipline deals with how items are chosen from the queue for service. Common rules are FIFO (First In First Out) and LIFO (Last In First Out). Other

considerations are the capacity of the queue, customers avoiding the queue due to length and customers leaving the queue due to waiting too long.
Queuing theory answers questions about the system such as: a customer's mean waiting time in the queue, a customer's mean time in the system, and the length of the queue or server utilization. With this knowledge changes to the system can be investigated, such as implementing additional servers, prioritizing customers or adjusting the size of the waiting area.

5.4.1 M/M/1 Model

M/M/1 is Kendall's notation of this queuing model. The first part represents the input process, the second the service distribution, and the third the number of servers as illustrated in figure 10.

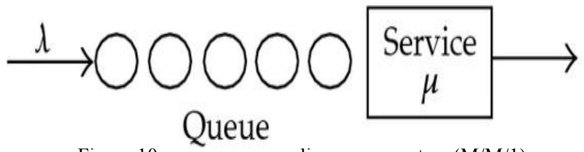


Figure 10: one server one line queue system (M/M/1)

The M represents an exponentially distributed inter-arrival or service time; specifically M is an abbreviation for Markovian. The M/M/1 Waiting line system has a single channel, single phase, Poisson arrival rate, exponential service time, unlimited population, and First-in First-out queue discipline. For an M/M/1 queue where λ is average arrival rate into the system and μ is the average service rate, it is simple to calculate questions about the system. The average number of customers in the system is given by (1), and the total time in the system is given by (2). Similar equations have been developed for variations on the M/M/1.

$$\frac{\rho}{\rho - 1} \quad \text{Where} \quad \rho = \frac{\lambda}{\mu} \qquad (1)$$

$$\frac{1}{\lambda - \mu} \qquad (2)$$

The input parameters to the queuing system is illustrated in table 6.

Table 6: Input parameter to queuing Time system

Input Parameter	The value
Arrival Rate (λ)	6
Service Rate (μ)	8
Experiment duration	3600
Maximum queue length	100

This simulated experiment examines the MM1 queuing system and here is the explanation for the above table input data:

1) Arrival Rate (λ) = 6 Passengers per Minute.
2) Service Rate (μ) =8 Serviced Passengers per Minute.
3) Experiment Duration= 3600 Minute = 60 Hour
4) Maximum Queue Length= 100 Passenger in System Waiting "Area" Queue
The results are illustrated in table 7.

Table 7: Basic Result

Result	Computed value	Simulated value
Customers in system (L_s)	3	3.0102030
Customers in queue (L_q)	2.25	2.2567900
Time in system (W_s)	0.5	0.5022366
Time in queue (W_q)	0.375	0.3765161
Idle probability (p_0)	0.25	0.2496987
Server utilization (ρ)	0.75	0.7534130

The computed results came from the formulas of MM1 queuing system. And the simulated results come by experimental approach from random sample of space.
And the results were satisfying mostly if we knew that the:
1) Passengers in system (Ls) =3.01at the same time taking service
2) Passengers in queue (Lq) =2.256 passenger waiting to start being serviced.
3) Time in system (Ws) =0.502 to finish the service.
4) Time in queue (Wq) =0 .376 waiting time inside the passengers waiting area.
5) Idle probability (p0) =0.249 the probability that the system server is being idle.
6) Server utilization (ρ) =0.753 which lead to a great performance on the other hand.

5.5 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 customer.In the airline industry, this control comes in the form of enabling the customer to select their own seat, request an upgrade, or change flights.
Further to this conclusion, the following was observed:

1) 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.

2) The processing times for the passenger who had some experience of using a kiosk was significantly less than average.
3) Most of the passengers need assistance in completing the process and there were two roving agents helping passengers.
4) The location of the kiosks made them very accessible and easily visible before the passengers could see the check-in counters.

6. Conclusion and Future Work

This paper discusses the need to adapt the Software Architecture Comparison Analysis Method –SACAM in order to use it in legacy software modernization context, where candidate implemented software products are evaluated.
The main contribution of this paper is proposing an adaptation by considering the evolution/maintenance history in architecture evaluation, and introducing complementary techniques that allow assessing software products from both business value perspective and technical quality perspective. The proposed adaptation was demonstrated by conducting a comparison between two check-in systems used in international airports as a case study.
The future work includes extending the method to apply in other system modernization situations such as reengineering and the comparison between an existing product and a planned one.

References

[1] Ian Sommerville, (2007), Software Engineering Eighth Edition, Addison-Wesley(Ch. 3).
[2] Laprie, J.C., (1995), Dependable Computing: Concepts, Limits, Challenges, Proc. 25th IEEE Symposium on Fault-Tolerant Computing.
[3] Thayer, R.H., (2002), Software System Engineering: a tutorial IEEE Computer (Ch.2).
[4] Banani Roy and T.C. Nicholas Graham, (2008), Methods for Evaluating Software Architecture: A Survey, Technical Report No. 2008-545, School of Computing, Queen's University at Kingston, Ontario, Canada
[5] Christoph Stoermer, Felix Bachmann, Chris Verhoef, (2003), SACAM: The Software Architecture Comparison Analysis Method, TECHNICAL REPORT, CMU/SEI-2003-TR-006, ESC-TR-2003-006.
[6] Clements, P.; Kazman, R.; & Klein, M., (2002), Evaluating Software Architectures. Reading, MA: Addison Wesley.
[7] Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W., (2003), Quality Attribute Workshops (QAWs), Third Edition (CMU/SEI-2003-TR-016).
[8] Bachmann, F.; Bass, L.; & Klein, M., (2003), Deriving Architectural Tactics: A Step Toward Methodical Architectural Design (CMU/SEI-2003-TR-004, ADA413644
[9] Bass, L.; Clements, P.; & Kazman, R., (2003), Software Architecture in Practice, Second Edition. Reading MA: Addison Wesley.
[10] Arora, V.; Kalaichelvan, K.; Goel, N.; & Munikoti, R., (1995), Measuring High-Level Design Complexity of Real-Time Object-Oriented Systems, Proceedings of the Annual Oregon Workshop on Software Metrics.
[11] Faust, D. & Verhoef, C., (2003), Software Product Line Migration and Deployment, Software: Practice and Experience, Volume 33, Issue 10.
[12] Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J., (2002), Documenting Software Architectures Views and Beyond. Reading, MA: Addison Wesley.
[13] Stoermer, C.; O'Brien, L.; & Verhoef, C., (2003), Moving Towards Quality- Attribute-Driven Software Architecture Reconstruction, Proceedings of the 10th Working Conference on Reverse Engineering (WCRE) Victoria, Canada.
[14] Shady G. Abdelaziz, Abdelfatah A .Hegazy and Ahmed Elabbassy , (2010), Study of Airport Self-service Technology within Experimental Research of Check-in Techniques Case Study and Concept, IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 3, No 1.
[15] G. Abowd, L. Bass, P. Clements, Rick Kazman, L. Northrop, and A. Zaremski, (1996), Recommended Best Industrial Practice for Software Architecture Evaluation (CMU/SEI-96-TR-025). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University.
[16] M. A. Babar, L. Zhu and R. Jeffery, (2004), A Framework for Classifying and Comparing Software Architecture Evaluation Methods, In the Proceedings on Australian Software engineering, pp. 309-318.
[17] M. Lindvall, R. T. Tvedt and P. Costa, (2003), An empirically-based process for software architecture evaluation, Empirical Software Engineering VOL.8, No.1.
[18] R.T. Tvedt, M. Lindvall, and P. Costa, (2002), A Process for Software Architecture Evaluation using Metrics, In the proceedings of 27th Annual NASA Goddard/IEEE.
[19] Davide Falessi1, Muhammad Ali Babar2, Giovanni Cantone1, Philippe Kruchten3, (2009), Applying Empirical Software Engineering to Software Architecture: Challenges and Lessons Learned, Technical Report 09.80, Dipartimento Di Informatica, Sistemi E Produzione, Universita Degli Studi Di Roma "Tor Vergata".
[20] Eric Bouwers, Joost Visser, Arie van Deursen, (2009), Criteria for the Evaluation of Implemented Architectures, ReportTUD-SERG-2009-018, Delft University of Technology, Software Engineering Research Group. Accepted for publication in the Proceedings of the International Conference on Software Maintenance (ICSM), 2009, IEEE Computer Society
[21] Muhammad Ali Babar, Patricia Lago, Paris Avgeriou, (2009), Empirical Assessment in Software Architecture: Importance and Challenges,Wicsa7 Workshop:Empirical Assessment in Software Architecture.
[22] Marwan Abi-Antoun, Talia Selitsky, Thomas LaToza, (2010), Developer Refinement of Runtime Architectural Structure, SHARK'10 May 2, 2010, Cape Town, South Africa.

[23] Michael Mattsson, Håkan Grahn, and Frans Mårtensson, (2006), Software Architecture Evaluation Methods for Performance, Maintainability, Testability, and Portability, Second International Conference on the Quality of Software Architectures (QoSA 2006), Västerås, Sweden.

[24] Antonia Bertolino1, Henry Muccini2, and Andrea Polini1, (2006), Architectural Verification of Black-box Component-Based Systems, RISE'06 Proceedings of the 3rd international conference on Rapid integration of software engineering techniques.

[25] Komaldeep Purewal, Dr. Lili Yang, Dr. Alan Grigg, (2009), Quantitative Assessment of Quality Attributes in Systems Architecture using Evidential Reasoning, 7th Annual Conference on Systems Engineering Research (CSER 2009).

Ahmed Elabbassy received the Ph.D. Degree Computer Sciences from ENSAE, France, 1979, now is a professor in the Department of Computer Science, Elshourouk Academy, Egypt, His research interest includes: Software Engineering, Operating Systems and Information Management.

Shady G. Abdelaziz is a graduate student in the College of Computer Science at Arab Academy for Science Technology and Maritime Transport, where he is studying basic and applied research in Software Engineering and Aviation Information Technology. His current research interests include Aviation Information Technology and system analysis/synthesis.

Abdelfatah A .Hegazy received the B.E. degrees, from the Military Technical Collage, Cairo, Egypt, 1978. In 1982 he received the M.Sc. In Computer Sciences from George Washington University, USA. Dr. Hegazy received the Ph.D. Degree Computer Sciences from George Washington University, USA, in 1985. After working as an assistant professor (from 1985) in the Dept. of computer engineering operation research, the Military Technical Collage., and an associate professor (from 1990), he has been a professor at College of Engineering at the Arab Academy for Science and technology. Since 1998. His research interest includes: Information Systems Planning; E-Commerce, E-Government, Information Systems Security, network security, knowledge Management, Web Intelligent Systems and Enterprise Resource Planning Systems. He is a member of IEEE, ACM, AIS, AANIS, and CSS-Computer Scientific Society Egypt.