

Using Quality Models to Evaluate National ID systems: the Case of the UAE

Ali M. Al-Khouri

Abstract—This paper presents findings from the evaluation study carried out to review the UAE national ID card software. The paper consults the relevant literature to explain many of the concepts and frameworks explained herein. The findings of the evaluation work that was primarily based on the ISO 9126 standard for system quality measurement highlighted many practical areas that if taken into account is argued to more likely increase the success chances of similar system implementation projects.

Keywords—National ID system, software quality, ISO 9126.

I. INTRODUCTION

THE United Arab Emirates (UAE) have recently initiated a national ID scheme that encompasses very modern and sophisticated technologies. The goals and objectives of the UAE national ID card programme go far beyond introducing a new ID card document and homeland security [1]. To increase its success, the government is pushing for many innovative applications to explore ‘what can be done with the card’. Examples of such possible applications of the card ranges from using it as a physical identity document to prove identity, to linking it to wide range of government services, with the vision of replacing all existing identity documents (e.g., driving license, labour card, health card, etc.) with this new initiative. From such perspectives, it becomes critical that such systems maintain a high level of quality. Quality models can play a good role as useful tools for quality requirements engineering as well as for quality evaluation, since they define how quality can be measured and specified [2]. In fact, the literature reveals that the use of quality frameworks and models may well contribute to project success, as it enables the early detection and addressing of risks and issues of concern at an early stage of the project (see for example [3],[4],[5]). This paper attempts to provide a short evaluation of the population register software (referred to in this paper as PRIDC – population register and ID card) implemented part of the national ID card project in the UAE to pinpoint areas of possible improvements.

The paper is structured as follows. The first section provides brief background information about the concept of software quality and measurement standards, with focus on

ISO 9126 framework. The next section presents the methods employed to obtain data based on which the system was evaluated. The next few sections provide an overview of the PRIDC system, its components, its development lifecycle approach, results obtained from the previous tests, and mapping these latter set of data to ISO 9126 quality attributes. The paper is then concluded with some reflections on the areas that need to be considered when pursuing similar evaluation studies with a focus on national ID systems.

II. SOFTWARE QUALITY

It is becoming a common trend for IT projects to fail. The rate of failure in government projects is far higher than those in the private industry. One of the main causes for such failures was widely quoted in the literature to be related to poor user requirements resulting in a system that does not deliver what was expected from it (see also the statistics presented in Fig. 1 from the recent Standish Group study).

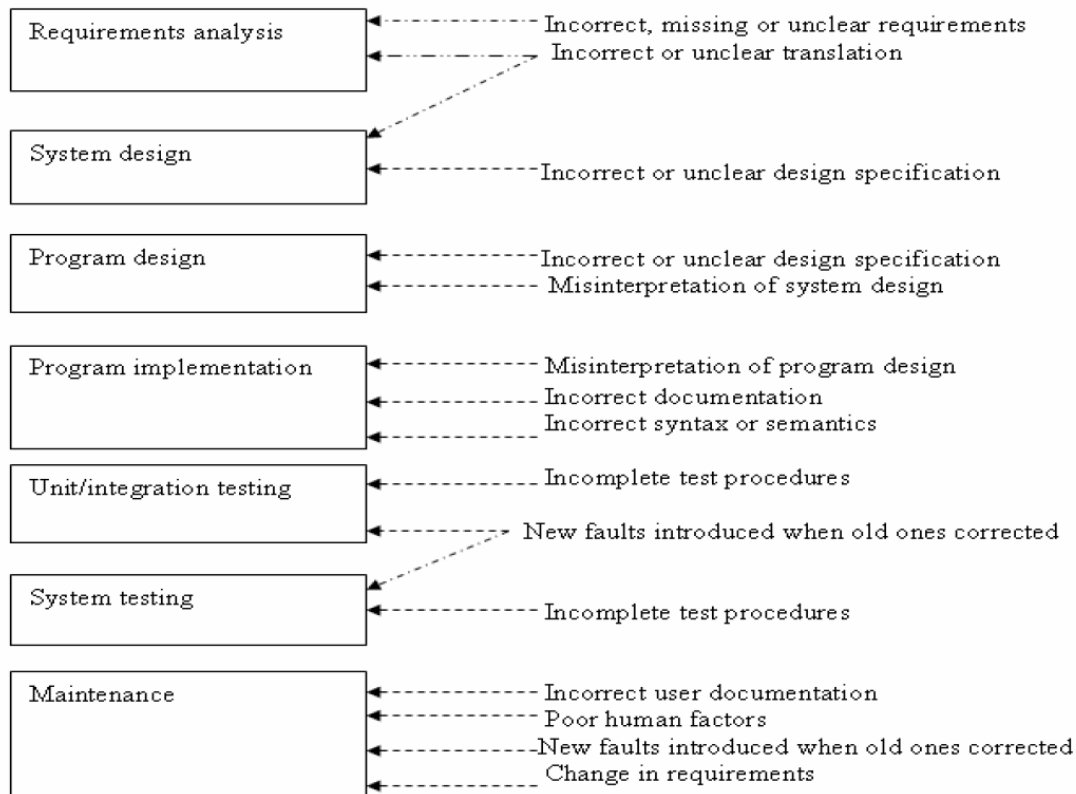
Standish Study Results:	
51%	of project failed
31%	were partially successful
Failure causes:	
13.1%	In complete requirements
12.4%	Lack of user involvement
10.6%	Inadequate resources
9.9%	Unrealistic user expectations
9.3%	Lack of management support
8.7 %	Requirements keep changing
8.1%	Inadequate planning
7.5 %	System no longer needed

Fig. 1 Standish group study results

The CHAOS survey of 8000+ projects found that of the eight main reasons given for project failures, five are requirements related. Getting the requirements right is probably the single most important thing that can be done to achieve customer satisfaction. Fig. 2 depicts further reasons for such failures [6]. Many of these failures are argued to could have been prevented with requirements verification and the adoption of quality assurance frameworks [4],[7].

Manuscript received March 27, 2007.

Ali M. Al-Khouri is with Emirates Identity Authority, Abu Dhabi, United Arab Emirates (phone: +97150-613-7020; fax: +9712-404-6661; e-mail: alkhouri@emiratesid.ae).



Source: Adopted from Pfleeger (2001)

Fig. 2 Causes of faults during development

In general terms, there are three different approaches to system quality assurance:

1. *Product Certification*

An independent party (or a QA company) conduct a limited exercise in verification, validation and / or test of the software components.

2. *Process Audit:*

An independent party conduct and assessment of the development process used to design, build and deliver the software component.

3. *User Satisfaction:*

Analysis of the actual behaviour of the software.

Since the objective of the evaluation study in this paper here is to judge whether the given implemented system has met the requirement of product quality, the third category approach was defined as the boundaries for the evaluation taken place in this study.

A. Quality Measurement Standards

Software quality assessment is attracting great attention as the global drive for systemic quality assurance continues to

gather momentum e.g., pressures of consolidations, mergers, and downsizing, emergence of new technologies [8]. Of the very initial works conducted in the field of software quality assessment was done by B. Boehm and associates at TRW [9] and incorporated by McCall and others in the Rome Air Development Center (RADC) report [10]. The quality models at the time focused on the final product and on the identification of the key attributes of quality from the user's point of view. The assessment framework was later improved; consisting of quality attributes related to quality factors, which were decomposed into particular quality criteria and lead to quality measures (see Fig. 3).

Attempted standardisation work over the intervening years resulted in the Software Product Evaluation Standard, ISO-9126 (ISO/IEC, 1991). This model was fairly closely patterned after the original Boehm structure, with a six primary quality attributes that were subdivided into 27 sub-characteristics as illustrated in Fig. 4.

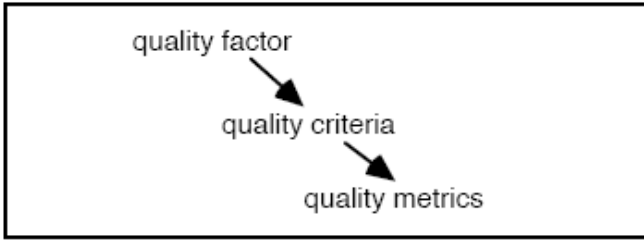


Fig. 3 Boehm quality model

However, the standard was criticised to provide very general quality models and guidelines, and that they are very difficult to apply to specific domains such as components and CBSD (see for example: [11],[12]. However, this is believed by others to be in fact one of its strengths as it is more adaptable and can be used across many systems [13],[14]. To solve this problem ISO/IEC 9126 has been revised to include a new quality model which distinguishes between three different approaches to product quality:

- (1) External Quality metrics – ISO TR 9126 -1: a result of the combined behaviour of the software and the computer system and can be used to validate the internal quality of the software;
- (2) Internal Quality metrics – ISO TR 9126 – 3: a quantitative scale and measurement method, which can be used for measuring an attribute or characteristic of a software product;
- (3) Quality in use metrics – ISO TR 9126 – 4: is the effectiveness, productivity and satisfaction of the user when

carrying out representative tasks in a realistic working environment. It can be used to measure the degree of excellence, and can be used to validate the extent to which the software meets user needs. Fig. 5 depicts the relationship between these approaches.

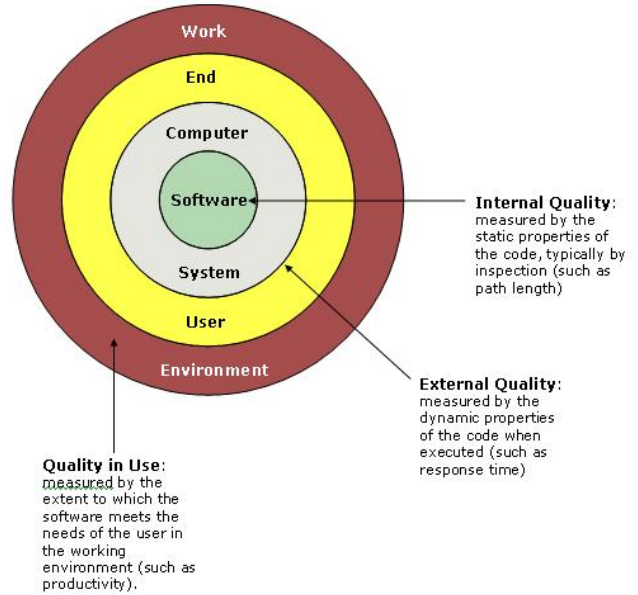


Fig. 5 Relationship between internal quality, external quality and quality in use

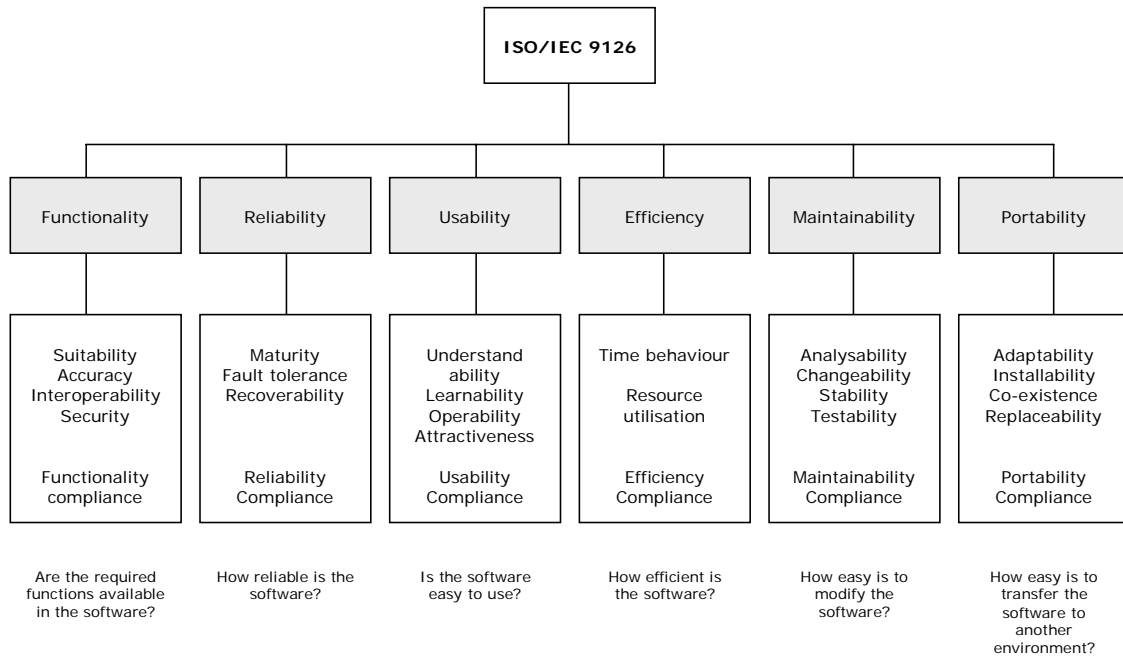


Fig. 4 ISO/IEC 9126 standards characteristics

In brief, internal metrics measure the software itself, external metrics measure the behaviour of the computer-based system that includes the software, and quality in use metrics measure the effects of using the software in a specific context of use. Appropriate internal attributes of the software are prerequisites for achieving the required external behaviour, whereas external behaviour is a prerequisite for achieving quality in use (see also Fig. 6).

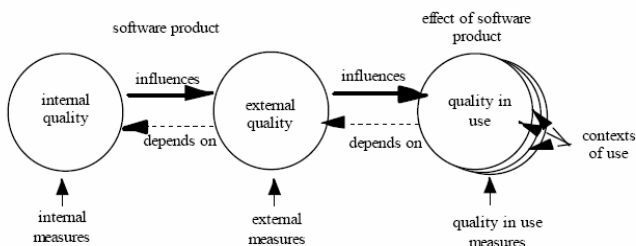


Fig. 6 Approaches to software product quality

It is also worth to mention that a new project was launched called SQuARE - Software Product Quality Requirements and Evaluation (ISO/IEC 25000, 2005) - to replace the above but follow the same general concepts of 9126 standard (see also Fig. 7).

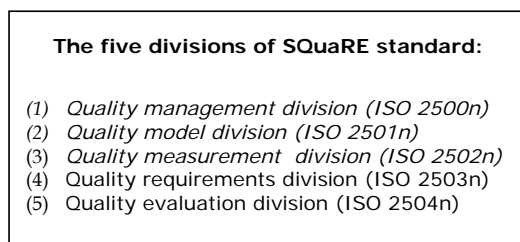


Fig. 7 SQuARE standard

Nonetheless, research and practical work shows that the assessment of the quality of a software component is in general a very broad and ambitious goal [11]. Recent research also shows that these characteristics and sub-characteristics covers a wide spectrum of system features and represent a detailed model for evaluating any software system as Abran et al. [15] explain:

"...ISO 9126 series of standards even though it is not exhaustive, this series constitutes the most extensive software quality model developed to date. The approach of its quality model... is to represent quality as a whole set of characteristics... This ISO standard includes the user's view and introduces the concept of quality in use."

III. METHODOLOGY

"If you chase two rabbits, both will escape."

Chinese Proverb

ISO 9126 quality characteristics and sub-characteristics were used to evaluate the national ID system. In this investigation several evaluation methods were employed. Following were the prime sources of information for the evaluation study:

1. information gathered from the test sessions took place during the acceptance of the project deliverables;
2. observation of the system environment (both at the central operational and registration centres);
3. by means of recording the author's own experience as the Director of the Central Operations sector, and head of the technical committee overseeing the implementation of the programme.

In general, the evaluation was qualitative in nature. In carrying out the evaluation and recording the findings, the PRIDC system went through two types of testing; functional and technical.

A. Functional Testing

This is an application level testing from business and operational perspective. It is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. Often called black box testing, this type of tests is generally performed by QA analysts who are concerned about the predictability of the end-user experience. During the deliverables acceptance, the national ID system was tested with black box testing procedures (that focuses on testing functional requirements and does not explicitly use knowledge of the internal structure) as per the test plan designed by the solution provider. No change was allowed by the vendor to the test plan as they wanted to narrow down the scope of testing, and limit it to the test cases developed by them.

B. Technical Testing

This is the system level testing. It tests the systems, which are supporting or enabling to run the Functional Applications. With general perception of QA, the COTS are not seen to be required to test but they need to be audited for the configuration and deployment set-up.

Generally, white-box testing (also called as glass, structural, open box or clear box testing) was considered here by the technical team to test the design of the system that should allow a peek inside the 'box', as this approach focuses specifically on using internal knowledge of the software to guide the selection of test data. White-box testing requires the source code to be produced before the tests can be planned and is much more laborious in the determination of suitable input data and the determination if the software is or is not correct. It is worth mentioning that a failure of a white box test may result in a change which requires all black-box testing to be repeated and the re-determination of the white box paths. For this obvious reason there was always negligence from the vendor to initiate a white-box testing.

It must also be heeded that neither black nor white box testing can guarantee that the complete specifications have

implemented and all parts of the implementation have been tested. To fully test a software product, both black and white box testing are required. While black-box testing was limited by the test plan documents provided by the vendor, the white box testing was not possible to perform since the source code was still not handed-over to the client at the time of writing this study. However, all the architectural component of the national ID Sub-systems which were selected and assembled from the COTS were assessed and audited to their configuration and deployment set up. Having addressed the evaluation methods, the following sections describe the details of the work carried out in this study.

IV. PRIDC SYSTEM AS A COMPONENT-BASED SYSTEM

“For more than a decade good software development practice has been based on a “divide and conquer” approach to software design and implementation. Whether they are called “modules”, “packages”, “units”, or “computer software configuration items”, the approach has been to decompose a software system into manageable components based on maximizing cohesion within a component and minimizing coupling among components.” (Brown and Wallnau, 1996, p.414)

A. What is Component-Based Software (CBD)?

Component-based software development (CBD) is an emerging discipline that promises to take software engineering into a new era [16]. Building on the achievements of object-oriented software construction, it aims to deliver software engineering from a cottage industry into an industrial age for Information Technology, wherein software can be assembled from components, in the manner that hardware systems are currently constructed from kits of parts (ibid).

Component-based software development (CBSD) shifts the development emphasis from programming software to composing software systems as it embodies the ‘buy, don’t build’ philosophy espoused by [17]. See also Fig. 8. The concept is also referred to in the current literature as component-based software engineering (CBSE) [18],[19]. It principally focuses on building large software systems by integrating different software components and enhancing the overall flexibility and maintainability of the systems. If implemented appropriately, the approach is argued to have the potential to reduce software development costs, assemble systems rapidly, and reduce the spiraling maintenance burden associated with the support and upgrade of large systems [20].

[21] define component-based software development as an approach “based on the idea to develop software systems by selecting appropriate off-the-self components and then to assemble them with a well-defined software architecture.” They state that a component has three main features:

1. is an independent and replaceable part of a system that fulfils a clear functions,
2. works in the context of well-defined architecture,
3. communicates with other components by its interface.

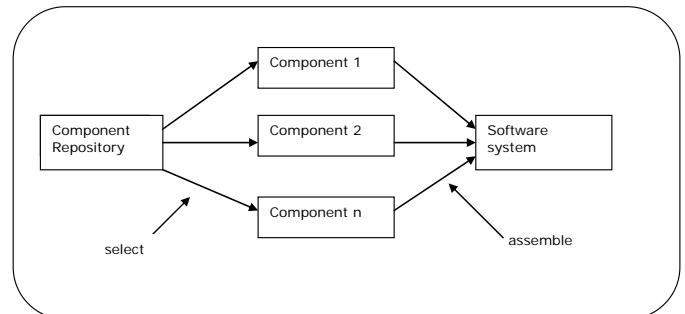


Fig. 8 Component-based software development

According to [22] two main advances are raising the profile of software components as the basic building blocks of software - *see also*: [16],[23],[24],[25],[26],[27]:

- (1) the object-oriented development approach which is based on the development of an application system through the extension of existing libraries of self-contained operating units, and
- (2) the economic reality that large-scale software development must take greater advantage of existing commercial software, reducing the amount of new code that is required for each application.

Component-based development approach introduces fundamental changes in the way systems are acquired, integrated, deployed and evolved. Unlike the classic waterfall approach to software development, component-based systems are designed by examining existing components to see how they meet the system requirements, followed by an iterative process of refining requirements to integrate with the existing components to provide the necessary functionality [22].

B. Component-Based Software Development Lifecycle

The component life cycle is similar to the life cycle of typical applications, except in implementation and acquisition phases where the two life cycles differ. The life cycle of component-based software systems can be summarised as follows:

1. *Requirement analysis*: a process of defining and understanding the activities that the information system is meant to support;
2. *Software architecture design*: a process of developing detailed descriptions for the information system;
3. *Component identification and customisation (implementation)*: a process of formalising the design in an executable way by acquiring complete applications or components through purchase, outsourcing, inhouse development, component-leasing etc;
4. *System integration*: a process of adjusting the system to fit the existing information system architecture. This

- can include tasks such as adjusting components and applications to their specific software surroundings,
5. **System testing:** a process of identifying and eliminating undesirable effects and errors and to verify the information system. This can include both user-acceptance- and application integration-tests,
 6. **Software maintenance:** a process of keeping the integrated information system up and running. This can include tasks such as upgrading and replacing applications and components in the information system. It also includes performing consecutive revisions of the integrated information system.

Having shortly highlighted some background information about the concept of component-based development and lifecycle, the next section takes a snapshot of the PRIDC system and maps it to component-based software.

C. PRIDC System Development Life Cycle

Broadly speaking, the development of the PRIDC system in general can be described to have incorporated the following two approaches:

1. the development of a uniquely tailored information system (population register) to enable the registration of population into the system in accordance to the pre-defined business requirements, and

2. the integration of several application/hardware package to achieve the desired functionality requirements e.g., biometrics, PKI, smart cards.

For the purpose of benchmarking PRIDC system development lifecycle, a framework proposed by [28] for quality assurance of component-based software development paradigm has been adopted in this study. The framework contains eight phases relating to components and systems that provide better control over the quality of software development activities and processes:

1. *Component requirement analysis.*
2. *Component development.*
3. *Component certification.*
4. *Component customisation.*
5. *System architecture design.*
6. *System integration.*
7. *System testing.*
8. *System maintenance.*

The details of this benchmarking are presented in the following tables.

TABLE I
COMPARISON OF PRIDC SYSTEM LIFECYCLE WITH COMPONENT BASED SOFTWARE APPROACH

No.	Component-based Software Phases .	PRIDC System Life cycle	Remarks
1	Component Requirement Analysis –Component requirement analysis is the process of discovering, understanding, documenting, validating and managing the requirements for the Component.	Category A	All the PRIDC project Lots which are part of collection and analysis of user requirements. Based on these functional requirement applications have design.
2	Component Development - Component development is the process of implementing the requirement for a well-functional, high quality component with multiple interface.	Category B	This phase is an internal process happening within the solution provider boundary.
3	Component Certification -Component certification is the process that involves: 1.component outsourcing, 2.component selection, 3.component testing.	Category B	This phase is an internal process happening within the solution provider boundary. Emirates ID may request for this certification if exists.
4	Component Customisation -It is the process that involves 1) modifying the component for the specific requirement;2) doing necessary changes to run the component on special platform;3) upgrading the specific component to get better performance or a higher quality.	Category B	This phase is an internal process happening within the solution provider boundary.
5	System Architecture Design - It is the process of evaluating, selecting and creating software architecture of a component-based system.	Category B	This phase is an internal process happening within the solution provider boundary.
6	System Integration -it is process of assembling components selected into a whole system under the designed system architecture.	Category B	This phase is an internal process happening within the solution provider boundary.
7	System Testing - System testing is the process of evaluating a system to : 1) confirm that system satisfies the specified requirement; 2) identify and correct defects in the system implementation.	Category B and Category C	The solution provider must have their own framework for testing (such as code testing and unit testing) of their product. But as a part of Project Lot in category C (that means sub-systems installation and commissioning – Lot 12, Lot3 testing) ,this task has performed.
8	System Maintenance - System maintenance is the process of providing service and maintenance activities needed to use the software effectively after it has delivered.	No Lot of PRIDC project matched with this phase	This is a one of the major phases missing in the PRIDC system life cycle. This is one of the rigorous drawbacks in PRIDC Project contract.

ISO Standard for System Implementation

In 1987 ISO and IEC(International Electrotechnical Commission) established a joint Technical Committee (JCT1) on Information Technology. In June 1989, the JCT1 initiated the development of ISO/IEC 12207, on software life cycle processes to fill a critical need. The ISO was published August 1,1995.

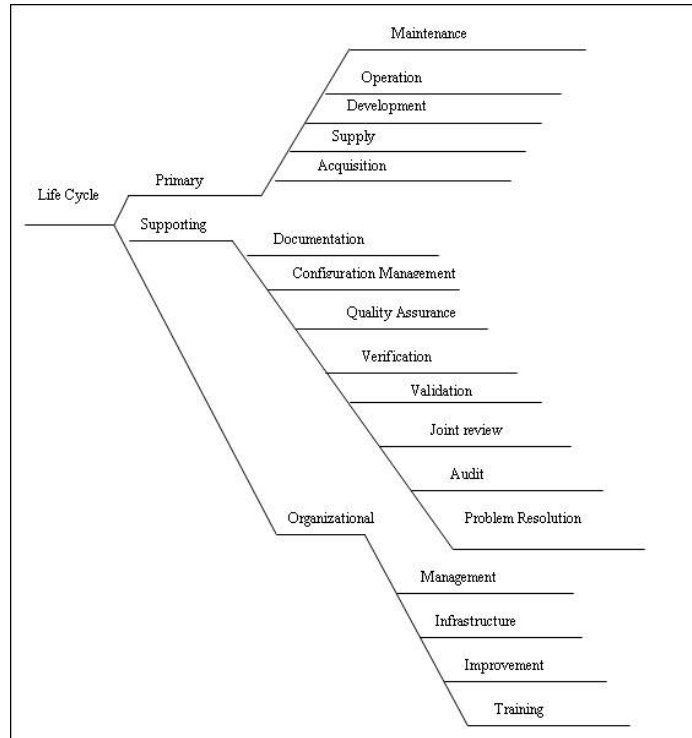


Fig. 9 ISO 12207 standard

A Comparison of PRIDC System with ISO Standard

PRIDC systems Lifecycle is currently based on project implementation phases. The Project implementation is executing in Lot-wise as framed in the contract. A comparative study of PRIDC Systems Life Cycle with ISO 12207 standard can be presented as below:

TABLE II
COMPARISON OF PRIDC SYSTEM WITH ISO 12207 STANDARD

No	ISO 12207	PRIDC System Life cycle
1	Primary life cycle processes	
1.1	Acquisition process	All lots of Category A.
1.2	Supply process	All lots of Category D.
1.3	Development process	All lot of Category B.
1.3.1	Process implementation	All lots of Category C.
1.3.2	System requirement analysis	All lots of Category A.
1.3.3	System architectural design	All lots of Category B.
1.3.4	Software requirement analysis	The solution provider internal process.
1.3.5	Software architectural design	The solution provider internal process.
1.3.6	Software detail design	The solution provider internal process.
1.3.7	Software coding and testing	The solution provider internal process. EIDA had done few of such testing.
1.3.8	Software integration	The solution provider internal process.
1.3.9	Software Qualification testing	The solution provider internal process.
1.3.10	System integration	The solution provider internal process.
1.3.11	Software qualification testing	The solution provider internal process.

1.3.12	Software installation	The solution provider internal process.
1.3.13	Software acceptance Support	Lot 12 and Lot3 testing
4	Operation Process	Need to define.
5	Maintenance Process	Need to define.
6	Supporting life cycle processes	
6.1	Documentation process	Done as a part Project deliverable.
6.2	Configuration management process	Done as a part Project deliverable.
6.3	Quality assurance process	Not done as a part of Project contract.
6.4	Verification process	Can be considered, Lot 3 system compliance test.
6.5	Validation process	Can be considered, Lot 12 system compliance test.
6.6	Joint review process	Can be consider – program management meeting.
6.7	Audit process	Need to perform.
6.8	Problem resolution process	Need to perform.
7	Organizational life cycle processes	
7.1	Management process	EIDA needs to perform.
7.2	Infrastructure process	EIDA needs to perform.
7.3	Improvement process	EIDA needs to perform.
7.4	Training process	Done as a part of Lot 3 – Admin Training.

D. ISO 9126 and PRIDC Mapping

Following is a summary of the evaluation results as per the ISO 9126 quality attributes.

Functionality	the degree of existence of a set of functions that satisfy stakeholder/business implied needs and their properties. Overall, in terms of number of changes requested on the system, as illustrated in Table 1.10, there were more than 213 modification items in the form of 53 change requests (23 major changes) passed to the vendor to implement. This was the first functional test results with the first version of the PRIDC system. This is a significant amount of modifications and it clearly implies that there was a big gap during the system requirement analysis and capturing phase.	
Suitability	Can software perform the tasks required? The degree of presence of a set of functions for specified tasks (fitness for purpose)	checked against specifications & feedback from registration centres.
Accuracy	is the result as expected? The degree of provision of right or agreed results or effects	checked against specifications. Test cases were developed by the test team of the vendor. Besides, there were many other cases that were not tested for accuracy but encountered later after the release of the software.
Interoperability	Can the system interact with another system? the degree to which the software is able to interact with specified systems (i.e. physical devices)	checked against specifications. However, the system was designed to be a closed architecture, as interoperability with future systems was seen to be of big concern.
Security	Does the software prevent unauthorised access? a set of regulations for maintaining a certain level of security; degree to which the software is able to prevent unauthorised access, whether accidental or deliberative, to programs and data (i.e. login functions, encryption of personal data etc).	<p>checked against specifications and in accordance with the Information Security Policy</p> <p>The PRIDC system is a critical system for the country, thus important security features were incorporated into the system to ensure high confidentiality, integrity and authenticity of the data. The security is built around the following main rules:</p> <ul style="list-style-type: none"> • Strong authentication of the operators (each end-user will use both password and fingerprint to logon onto the system), • Network security using Virtual Private Network (VPN) + Demilitarised Zone (DMZ) and Secure Socket Layer (SSL) over Hyper Text Transfer Protocol (HTTP), • Strong physical protection of the Central, Disaster Recovery and Service Points Local Area Networks (LAN) <p>The security scheme was implemented at 4 levels:</p>

		<p>1) Application level, 2) Network level, 3) System level, 4) Physical level.</p> <p>The security features carried out at each of the above levels included a wide range of advanced international security standards and measures: X509 V3 certificates, X500 directory, LDAP V2 and V3, DES, 3xDES, RC2, RC4, AES ciphering algorithms (used by CISCO VPN), RSA (PKCS#1) signature algorithms, MD2, MD5, SHA1, Diffie-Hellman and RSA key exchange algorithms, pkcs#12, pkcs#7, pkcs#10, IPsec, IKE. TOO MUCH SECURITY!</p>
Compliance	the degree to which the software adheres to application-related standards or conventions or regulations in laws and similar prescriptions	checked against specifications
Reliability	the capability of the software to maintain its level of performance under stated conditions for a stated period of time (This is assessed based on the number of failures encountered per release)	
Maturity	Have most of the faults in the software been eliminated over time? the frequency of failure by faults in the software	If looked at the number of sub versions released of the PRIDC system (i.e., ver 1.5, ver 1.6 and ver 3.0) as depicted in Table 1.7, the evolution of these versions was unplanned (i.e., previously not specified) versions of the system, which signifies that the immaturity of the system in terms of business requirements and needs. At the time of carrying out this evaluation, the software was still seen to require further modifications before the system can be finally accepted.
Fault tolerance	is the software capable of handling errors? the ability to maintain a specified level of performance in cases of software faults or of infringement of its specified interface; is the property that enables a system to continue operating properly in the event of the failure of some of its components.	Although the system had a centralised architecture, its architecture allowed the different systems to continue operation in the cases of failure of the central system through replication and redundant systems.
Recoverability	Can the software resume working and restore lost data after failure? the capability of software to re-establish its level of performance and recover the data directly affected in case of a failure	Databases were continuously replicated on the Disaster Recovery site. The system insured that no more than one hour of work would be lost following a database crash/failure. However, in case of a major disaster that would lead to the loss of the operational capacity of the main data centre, the PRIDC system was planned to be restarted within 24 hours.
Usability	the effort needed for the use by a stated or implied set of users.	
Understandability	does the user comprehend how to use the system easily? evaluates the attributes of software that bear on the users' effort for recognizing the underlain concept of the software. This effort could be decreased by the existence of demonstrations	Usability testing uncovered many difficulties, such as operators having difficulty understanding system interface, business logic and processes. With the lack of on-line help function, the GUI interface of the system did not seem to follow any clear standard, as operators started guessing what different buttons may mean. For example, two registration centre's operators deleted the files of all registered applicants on one day when they pressed the button 'Abort' to cancel an operation, where the system was performing the action of 'Delete' with the 'Abort' button. In general, Interface functions (e.g., menus, controls) were no easy to understand.
Learnability	can the user learn to use the system easily? evaluates the attributes of software that bear on the users' the user's effort for learning how to use the software	User documentation and help were not complete at the time of carrying out this evaluation. The system was not easy to learn as users had to repeat the training sessions many times as the cases of data entry errors was raising when post-audit procedures for data quality check were implemented.
Operability	can the user use the system without much effort? evaluates the attributes of software that bear on the users' effort for operation and operation control (e.g. function keys, mouse support, shortcuts e.t.c.)	The interface actions and elements were sometimes found to be inconsistent - error messages were not clear and led to more confusion and resulted in operators guessing and attempts to rectify problems which in turn led to deeper problems as the system was not designed to handle user play-around cases (i.e., to handle unexceptional errors). Some important functions such as deletion was being performed with prompt to confirmation.

Attractiveness	does the interface look good? evaluates how attractive is the interface to the user?	the system design and screen layout and colour was not so appealing
Efficiency	Have functions been optimised for speed? Have repeatedly used blocks of code been formed into sub-routines?	
Time Behaviour	how quickly does the system respond? evaluates the time it takes for an operation to complete; software's response and processing times and throughput rates in performing its function	<p>To be checked against specification. However, the full testing of this characteristic was not possible at the time of carrying out this study since the daily enrolment throughput was around 1200 people a day, subsequently the same figures for the card production.</p> <p>From a database capacity view point, the PRIDC system was dimensioned to manage records of 5 Million persons. Whereas the throughput of the system was as follows:</p> <ul style="list-style-type: none"> • allows for up to 7,000 enrolments per day. • able to produce up to 7,000 ID Cards per day. • The Biometric Sub-System is able to perform up to 7,000 person identification (TP/TP) searches per day. <p>The processing operations was designed as follows:</p> <ul style="list-style-type: none"> • New enrolment: 20 minutes • Card collection: 3.5 minutes • Card Renewal: 8 minutes • PR Functions: 8.5 minutes • Civil investigation: 11 minutes • Biometric subsystem: Within 22 hours
Resource Utilisation	does the system utilise resources efficiently? is the process of making code as efficient as possible; the amount of resources and the duration of such use in performing the software's function	This task was not possible at the time of carrying out the evaluation, since the source code was still not handed over to the client.
Maintainability	the effort needed to make specified modifications	
Analysability	can faults be easily diagnosed? the effort needed for diagnosis of inefficiencies or causes of failure or for identification of parts to be modified	During system installation and with the release of the software (also during business operations), undocumented defects and deficiencies were discovered by the users of the software. Those encountered faults were very difficult to analyse and diagnose even by the vendor technical team and encountered software inefficiencies usually took long time to fix, as problems were usually passed to the development team in France for investigation and response.
Changeability	can the software be easily modified? Changeability is the effort needed for modification, removal or for environmental change	<p>The system architecture was so complex, as the word 'change to the system' meant a nightmare to the vendor. The vendor always tried to avoid changes all the time with the justification: 'the system in the current form, allows you to enrol the population and produce ID cards for them'. The client concern was that the software in its current version opens doors for many errors from user entry errors to incomplete business functions that were not captured during the phase of requirement specifications.</p> <p>it is worth also to mention that changes to the system when agreed was taking so long to implement. For example, adding a field to the system (job title) took a work of 1 month to implement with an amazing amount bill.</p>
Stability	can the software continue functioning if changes are made? the risk of unexpected effects of modifications	as mentioned above, the system complex architecture implied that a change in one place would almost affect many parts of the system. A change in one part of the system, would normally cause unexpected effects as a result of the modification.

Testability	can the software be tested easily? the effort needed for validating the modified software.	in general, system (business) processes and functions were tested against specifications. However from a technical perspective, the complex architecture of the system made it impossible to test many areas of the software. The vendor was pushing for the system to be accepted from a functional perspective (including the network setup).
Portability	A set of attributes that bear on the ability of software to be transferred from one environment to another	
Adaptability	can the software be moved to another environments? the software's opportunity for adaptation to different environments(e.g. other hardware/OS platforms	The software was designed and coded to operate within a unique environment of databases, operating systems and hardware. Most of the hardware used proprietary APIs' (Programming Applications Interface) to interface with the system. This automatically locked the system to only use the specified set of hardware but not otherwise.
Installability	can the software be installed easily? the effort needed to install the software in a specified environment	Though installation files and guides were available, the software architecture was not clear at all. All attempts made by the technical members failed in this regard. Despite the several requests, the vendor felt that the system should not be installed other than the vendor himself.
Co-existence	does the software comply with portability standards? Conformance is the degree to which the software adheres to standards or conventions related to portability	the system did not comply with any portability standards other than the vendor's own.
Replaceability	does the software easily replace other software? the opportunity and effort of using the software in the place of specified older software.	The PRIDC software was expected to take over the current population register database maintained part of the immigration system in the Ministry of Interior. However, this was a long-term objective. The software needed to go under several revolutions, before it can achieve this objective.

V. REFLECTION

“Some problems are so complex that you have to be highly intelligent and well-informed just to be undecided about them.”

Laurence J. Peter

Many researchers and practitioners argue that measurement is an essential issue in project and process management and improvement from the logic that it is not possible to control what is not understood and it is not possible to scientifically understand what is not measured [4]. Using measurement practices may well increase the rate of project success to a higher level, statistically [30]. In the real world, however, this may be argued to be a valid issue at the organisation level not at the individual project level. This is to say that projects usually have very short term strategies with very tight deadlines and tend to be the result of an opportunistic behaviour; where applying such measurement strategies may not be seen to add value, bearing in mind the time and cost associated with such measurement analysis activities.

As the UAE national ID system will become the most critical system in the country as the main central hub for population identity cross checking and service eligibility (i.e., online with 24/7 availability requirement), it becomes important that the software goes under a thorough quality check. Taking into consideration the CBS nature of the system, some components were viewed to be more critical to go under a through quality checks as a failure in different software components may lead to everything from public frustration to complete chaos when the card becomes ‘the means’ to accessing services.

The evaluation study carried out here attempted to provide a short but thorough overview of the PRIDC system, and measure the system quality against ISO 9126 standard. Many limitations had been encountered that will help greatly the project team to address before the final acceptance of the system from the vendor.

From the evaluation, the system was found to have been developed as a component-based software system, but most importantly was observed to be a closed system. This closed architecture—although it was promised to work as prescribed in the specification documents— was viewed to likely cause the following major drawbacks in the short and long run:

1. the system supported only a few hardware vendors, as this was seen to result in the system losing certain amount of autonomy and promoting it to acquire additional dependencies when integrating COTS components;
2. system evolution was not a simple plug-and-play approach. Replacing one component was more typically to have rippling affects throughout the system, especially where many of the components in the system were black box components; and

3. the system architecture forced the client to return again and again to the original vendor for additional functionality or capacity.

The closed architecture with the different proprietary platforms it incorporated were altogether more likely to slowdown the pace of organisational business and process excellence as changes to the system would be expensive and extremely difficult to maintain. The literature has not been kind to the closed system architectures as research show that such systems have proven to be too slow and too expensive to meet the rapidly changing market needs, as it restricts the level of quality that can be achieved [30],[31],[32],[33]. However, some vendors and service providers strongly advocate standardised systems via closed architectures. Their argument is that such architectures are so necessary in their system standards efforts and that the openness of the component-based approach leads to a chaos of choices and integration headaches, and that such architectures to address the ‘security’ needs.

Moreover, over the long-term life of a system, additional challenges may well arise, including inserting of COTS components that correspond to new functionality and "consolidation engineering" wherein several components may be replaced by one "integrated" component. Following are further reflections on the major ISO 9126 quality attributes:

A. Functionality

The functionality factors were mainly checked against the system specification documents. However, it was discovered on the release of the first version of the software that many business functions were not covered in the specifications, resulting in the need for subsequent releases to address and fill the operational gaps. However, the evaluated software version in this study was not at an acceptable state, as it required additional enhancements to cover some of the additional business functions and rectify identified deficiencies, errors and bugs. It is also worth to mention that the overemphasis of security requirements during the specification phase contributed exponentially to the existing high complexity of the overall system, and its interoperability with other sub-systems.

The fundamental problem concerning software development is defined as to try to understand the customer’s sometimes unspoken needs and requirements and translate these into a tangible software solution. The literature shows that one of the principle causes of information system failure is when the designed system fails to capture the business requirements or improve the organisational performance. Researchers argue that such failures were because many organisations tend to use rule-of-thumb and rely on previous experiences [35]. The vendor adopted the waterfall system development approach when it came to user requirements analysis and system implementation. The vendor was reluctant to make any modification to the developed system, and was presenting high cost impact on each change to it even if it was a change to modify labels of text fields on user screens. The

common response of the vendor was that ‘the system is developed according to the agreed specification, and any deviation from that is probably to have a cost impact.’

This attitude of the vendor opened doors for long discussion meetings and arguments around this area, and slowed down the progress of the project, as changes got parked for long periods as some got buried and lost into the huge project documents and long meeting minutes. However, system functionality is a temporary matter that can be resolved once attended to. The most critical items that needed to be addressed along with the functionality concerns were the areas discussed next.

B. Reliability

Software reliability is the probability that a software system will not cause the failure of the system for a specified time under specified conditions. The different tests carried out during the deliverables acceptance relied on systematic software testing strategies, techniques, and process, and software inspection and review against specifications. However, and during this study, it was found very useful to incorporate less systematic testing approaches to explore the ability of the system to perform under adverse conditions.

C. Usability

The software seemed to have many usability concerns as system users struggled to understand system processes and functions, as minimal user documentation were available that also did not cover the areas users needed most. Extensive training was required to educate the users on the system, as much effort was required from the registration centre supervisors to support the users. The system was required to go through a major review to evaluate its usability. It also needed to be enhanced to follow a standard GUI methodology overall.

D. Efficiency

System processes and functions were checked against the time indicated in the specifications from a functional perspective. Nonetheless, code review was not possible because the source code was not handed over to the client at the time of carrying out this evaluation. Overall, the technical team had concerns about the capability of the system to provide acceptable performance in terms of speed and resource usage.

E. Maintainability

The complex architecture of the system made the analysis and diagnoses of discovered system faults and their maintenance so difficult where problems were usually passed to the development team in another country for investigation and preparation of bug-fix patches. Besides, the complex architecture acted also as a huge barrier to making urgent changes to the system as it required long analysis to evaluate the impact on the different components of the software, associated with an unrealistic cost impact of implementing such changes claimed by the vendor.

F. Portability

The system had many proprietary API's to interface with the different components of the system, locking the system to use a specified set of hardware but not otherwise. Installation files and guides did not enable the reinstallation of the system. Overall, the system was observed not to comply with any portability standards other than the vendor's own, which can be carried out only by the vendor himself. The vendor was asked to add APIs to the system to allow the plug-in of new components to the system both data and hardware wise.

VI. CONCLUSION

“You don't drown by falling in the water; you drown by staying there.”

Edwin Louis Cole

As widely quoted in the literature, the application of software metrics has proven to be an effective technique for improving the quality of software and the productivity of the development process i.e the use of a software metrics program will provide assistance to assessing, monitoring and identifying improvement actions for achieving quality goals (see for example: [3],[4],[5],[6],[8],[9],[12],[14],[29],[35],[36],[37]. In this study, the author attempted to use the ISO 9126 quality model to evaluate the PRIDC system; mainly from a product quality angle. See also Fig. 10.

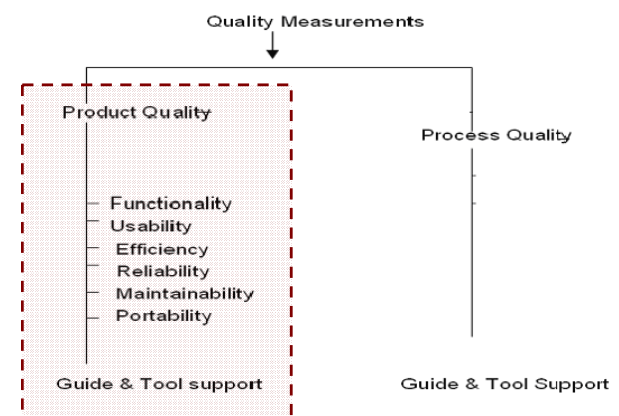


Fig. 10 software quality metrics framework - Source: [37]

The study presented in this paper contributed to a great extent in spotting some of the system deficiencies that were addressed prior to the final acceptance and handover of the system. It was also the author's observation that the project team, with the workload and responsibilities put on them, seemed to be overloaded and to have a scattered vision of how things be done and achieved. Everybody wanted the project to conclude as quickly as possible as everybody seemed also to be confident of the work produced by the vendor. The use of quality framework showed in this study can be a very

useful and supportive methodological approach for going about software quality assessment. ISO 9126 framework can act as a comprehensive analytical tool as it can move beyond superficial evaluation to achieve a more thorough view of the system's strengths and weaknesses than can be provided by less systematic approaches.

When implementing big projects such as National ID schemes, project management and technical teams should use quality models for evaluating the overall architecture prior to the final acceptance of the system. As such and if used as a guide in an early stage of the project it can arguably provide a basis for informed and rational decision making and have the potential to increase the project success rate.

From a technical view point, the ISO software quality metrics may also be extended throughout the phases of software development life cycle. The framework is designed to address the wide range of quality characteristics for the software products and processes enabling better description of software quality aspects and its importance.

ACKNOWLEDGMENT

The author would like to thank Mr. Naorem Nilkumar for his contribution and the technical input that improved the overall work presented in this paper.

REFERENCES

- [1] A.M. Al-Khouri, "UAE National ID Programme Case Study," *International Journal Of Social Sciences*, vol. 1, no. 2, pp.62-69, 2007.
- [2] E. Folmer & J. Bosch (2006) "A Pattern Framework for Software Quality Assessment and Tradeoff Analysis," *International Journal of Software Engineering and Knowledge Engineering*, 2006 [Online]. Available: <http://www.eelke.com/research/literature/SQTRF.pdf>.
- [3] S.N. Bhatti, "Why Quality? ISO 9126 Software Quality Metrics (Functionality) Support by UML," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 2, 2005.
- [4] E.J. Garrity & G.L. Sanders, "Introduction to Information Systems Success Measurement," in E.J. Garrity & G.L. Sanders (editors) *Information System Success Measurement*. Idea Group Publishing, pp.1-11, 1998.
- [5] R.B. Grady, "Practical results from measuring software quality," *Communications of the ACM*, vol. 36, no. 11, pp.63-68, 1993.
- [6] S. Hastie (2002) "Software Quality: the missing X-Factor," Wellington, New Zealand: Software Education [Online]. Available: http://softed.com/Resources/WhitePapers/SoftQual_XF-actor.aspx.
- [7] S.L. Pfleeger, *Software Engineering Theory & Practice*. Upper Saddle River, New Jersey: Prentice Hall, 2001.
- [8] R.A. Martin & L.H. Shafer (1996) "Providing a Framework for Effective Software Quality Assessment - Making a Science of Risk Assessment," 6th Annual International symposium of International council on Systems Engineering (INCOSE), *Systems Engineering: Practices and Tools*, Bedford, Massachusetts [Online]. Available: http://www.mitre.org/work/tech_transfer/pdf/risk_assessment.pdf.
- [9] B.W. Boehm, J.R. Brown, H. Kaspar, M. Lipow, G.J. MacLeod, G.J. & M.J. Merritt, "Characteristics of Software Quality." *TRW Software Series - TRW-SS-73-09*, December, 1973.
- [10] J.A. McCall, P.K. Richards & G.F. Walters, "Factors in Software Quality," volumes I, II, and III, US. Rome Air Development Center Reports NTIS AD/A-049 014, NTIS AD/A-049 015 and NTIS AD/A-049 016, U. S. Department of Commerce, 1977.
- [11] M.F. Bertoa, J.M. Troya & A. Vallecillo, "Measuring the Usability of Software Components," *Journal of Systems and Software*, Vol. 79, No. 3, pp. 427-439, 2006.
- [12] S. Valenti, A. Cucchiarelli, & M. Panti, "Computer Based Assessment Systems Evaluation via the ISO9126 Quality Model," *Journal of Information Technology Education*, vol. 1, no. 3, pp. 157-175, 2002.
- [13] R. Black (2003) "Quality Risk Analysis," USA: Rex Black Consulting Services [Online] Available: <http://www.rexblackconsulting.com/publications/Quality%20Risk%20Analysis1.pdf>.
- [14] G.G. Schulmeyer & J.I. Mcmanus, "The Handbook of Software Quality Assurance" (3rd edition). Upper Saddle River, New Jersey: Prentice Hall, 1999.
- [15] A. Abran, Al-Qutaish, E. Rafa, J.M. Desharnais, & N. Habra, "An Information Model for Software Quality Measurement with ISO Standards," in SWEDC-REK, *International Conference on Software Development*, Reykjavik, Islande, University of Iceland, pp. 104-116, 2005.
- [16] K.-K. Lau, (editor) 'Component-based Software Development: Case Studies,' World Scientific (Series on Component-Based Software Development), vol. 1, 2004.
- [17] F.P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, vol. 20, no. 4, pp. 10-9, 1987.
- [18] A.W. Brown, "Preface: Foundations for Component-Based Software Engineering," *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute*. Los Alamitos, CA: IEEE Computer Society Press, pp. vii-x, 1996.
- [19] A. Brown & K. Wallnau "Engineering of Component-Based Systems," *Proceedings of the Second International IEEE Conference on Engineering of Complex Computer Systems*, Montreal, Canada, 1996.
- [20] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. New York, NY.: Addison- Wesley, 1997.
- [21] X. Cai, M.R. Lyu & K. Wong(2000) "Component-Based Software Engineering: Technologies, Development Frameworks and Quality Assurance Schemes," in *Proceedings APSEC 2000, Seventh Asia-Pacific Software Engineering Conference*, Singapore, December 2000, pp372-379 [Online]. Available: http://www.cse.cuhk.edu.hk/~lyu/paper_pdf/apsec.pdf.
- [22] A.W. Brown & K.C. Wallnau, "The Current State of CBSE," *IEEE Software*, vol. 155, pp. 37-46, 1998.
- [23] M. Kirtland, *Designing Component-Based Applications*. Redmond, Washington: Microsoft Press, 1999.
- [24] G.T. Heineman & W.T. Councill (editors) *Component Based Software Engineering: Putting the Pieces Together*. Boston, MA: Addison-Wesley, 2001.
- [25] G.T. Leavnes & M. Sitaraman, *Foundations of Component-Based Systems*. New York: Cambridge University Press, 2000.
- [26] R. Richardson, "Components Battling Component," *Byte*, vol. 22, no. 11, 1997.
- [27] R. Veryard, *The Component-Based Business: Plug and Play*. London: Springer-Verla, 2001.
- [28] G. Pour, "Component-Based Software Development Approach: New Opportunities and Challenges," in *Proceedings Technology of Object-Oriented Languages, TOOLS 26*, pp. 375-383, 1998.
- [29] N.S. Godbole, *Software Quality Assurance: Principles and Practice*. Oxford, UK: Alpha Science International, 2004.
- [30] L. Bass, P. Clements & R. Kazman, *Software Architecture in Practice*. Reading MA.: Addison Wesley, 1998.
- [31] J. Bosch, *Design and use of Software Architectures: Adopting and evolving a product line approach*. Harlow: Pearson Education (Addison-Wesley and ACM Press), 2000.
- [32] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, & M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*. New York: John Wiley and Son Ltd, 1996.
- [33] M. Shaw, and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. New Jersey: Prentice Hall, 1996.
- [34] P.B. Crosby, *Quality Is Free: The Art of Making Quality Certain*. New York: McGraw-Hill, 1979.
- [35] R.G. Dromey, "A model for software product quality," *IEEE Transactions on Software Engineering*, vol. 21, no. 2, pp. 146-162, 1995.
- [36] J.T. McCabe, (1976) "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. SE2, no. 4, pp. 308-320, 1976.
- [37] K.H. Möller & D.J. Paulish, *Software Metrics*. London: Chapman & Hall Computing, 1993.



Ali M. Al-Khouri, has been involved in the UAE national ID card project since its early developments as a member of the technical executive steering committee, and he was later appointed as the head of the technical committee when the Emirates Identity Authority (a federal government organisation formed to oversee the management and implementation of the national ID card system rollout in the United Arab Emirates) was established. He received his Bachelor's and Master's degrees in Business IT Management with

honors and distinction from Manchester and Lancaster Universities in the UK, and is currently doing his doctorate degree in the field of engineering management and advanced technologies. His research interests are in leadership & management, e-government and the applications of advanced technologies in large contexts. Email: alkhouri@emiratesid.ae