Quality Attributes for Embedded Systems

Trudy Sherman
Arizona State University
Ira A. Fulton School of Engineering, Department of Computer Science and Engineering
Tempe, Arizona 85281

Abstract- **Software quality attributes (QAs) such as reliability and modifiability have been used to define nonfunctional requirements of software systems for many years. More recently, they have been used as the basis for generating utility trees in the Software Engineering Institute's Architecture Tradeoff Analysis Model (ATAM). Software processes and models, such as the ATAM, are often utilized when developing embedded systems which consist of both software and hardware. In order to determine whether the QA's defined for software are adequate when working with embedded systems, research based on trade studies performed during the development of embedded system architectures were evaluated. The results of the research shows that while many of the embedded system quality attributes map directly to existing software quality attributes, some attributes such as portability take on a modified definition, and others, such as weight, do not normally apply to software systems. This paper presents the quality attributes for embedded systems identified as a result of the trade study evaluation.**

I. INTRODUCTION

This paper presents a set of quality attributes for embedded systems based on the evaluation of eleven embedded system architecture trade studies. In general, embedded systems are systems consisting of hardware and software designed to work together for a specific purpose. These systems exist in virtually every aspect of our lives. They run our home appliances, home entertainment systems, communications devices, medical equipment, automobiles, transportation systems, etc. With such a widespread impact on everyday life, it is important that these systems meet not only the functional requirements of "what" they should be able to do, but also the nonfunctional or "quality" requirements expected of them such as reliability, security, and maintainability. Over the last several years, these nonfunctional requirements have come to be referred to as software quality attributes (QAs).

A brief background on software QAs is provided. Lists of common software QAs are included with definitions for some of the most common ones.

For the purpose of this paper, embedded systems are defined as electronic devices that include one or more microprocessors with a customized user interface. This definition is expanded to discuss the similarities with and differences from software-only systems.

The trade study research is explained. The technique used to evaluate each trade study and record the findings is described.

Finally, the results of the trade study evaluation are presented. Similarities to existing software QAs and the modifications to the definition of existing QAs are discussed. New QAs that are specific to embedded systems are described.

II. SOFTWARE QUALITY ATTRIBUTES

Software QAs have been used by software developers for many years. They are usually first identified in the Software Requirements Specification (SRS) which is written in the requirements phase of development. In addition to the functional requirements specifying "what" a software application must be able to do, many standard templates for the SRS include sections dedicated to nonfunctional requirements (quality attributes) such as reliability, security, maintainability, and usability that specify the characteristics of "how" the system will respond to various types of stimuli. For example, section 3 Specific Requirements, in the SRS template defined by IEEE Standard 830-1998 includes an entire subsection on software system attributes. It gives reliability, availability, security, maintainability and portability as examples of software attributes that should be documented as requirements [1].

In the next stage of software development, most commonly referred to as the Architecture or High Level Design phase, a strategy or software framework is put into place for implementing both the functional requirements and the QAs. Architectural design decisions can have a significant impact on the ability of a software system to fulfill QAs. For example, the security of data within a system is highly dependent both on how the user interface is structured and how the data is partitioned. If passwords and levels of access are not determined and built into the structure of the software at a high level, every function that is performed will have to include its own user validation slowing performance and opening the door to security errors and inconsistent security implementation. If both secure and non-secure data are kept together, it is very difficult to limit access to the secure data. Security checks must be performed by every function that accesses the database again leading to reduced performance and increased opportunities for security breaches.

As a final step in the architecture design phase, more and more software development organizations are evaluating their design decisions using the Software Engineering Institute's Architecture Trade-off Analysis Model (ATAM) [2]. This is a multi-day process that involves representatives of all the major stakeholders. The goal of the ATAM is to understand the consequences of architectural decisions with respect to the quality attribute requirements of the system.

It is important to note that there is not just one standard list of software QAs. Even though most QA lists are based on a standard such IEEE Std 830, the list of QAs used by an organization is added to and deleted from depending on the customer needs and the developer's focus on quality. In addition, the descriptions will also change based on the project needs. Table I shows two lists of software QAs from two similar sources that illustrate how the list of QAs might change as well as how the definitions might vary depending on focus.

TABLE I
EXAMPLES OF SOFTWARE QUALITY ATTRIBUTE LISTS

Software Quality Attribute	QA List 1 [3]	QA List 2 [4]
Accuracy	- Not Included -	Accuracy of the computed result
Availability	Proportion of time the system is up and running.	Deadlock, synchronization, and data consistency
Conceptual integrity	Underlying theme that unifies the design of the system at all levels.	- Not Included -
Functionality	Ability of the system to do the work for which it was intended.	- Not Included -
Modifiability	Ability to make changes to a system quickly and cost effectively.	Impact of an expected change
Performance	Responsiveness of the system - the time required to respond to stimuli or the number of events.	How long things take.
Portability	Ability of the system to run under different computing environments.	- Not Included -
Reliability	Ability of the system to keep operating over time.	- Not Included -
Security	Ability to resist unauthorized attempts at usage and denial of service.	Authentication and integrity concerns
Subsetability	Ability to support the production of a subset of the system.	- Not Included -
Usability	- Not Included -	Information presented to the user, data reuse, operations such as cut-and-paste and undo
Variability	How well the architecture can be expanded or modified to produce new architectures that differ in specific, preplanned ways.	- Not Included -

III. EMBEDDED SYSTEMS

Philip Koopman of Carnegie Mellon University [5] defines an embedded system as:

"A computer purchased as part of some other piece of equipment

- typically dedicated software (may be user customized)
- often replaces previously electromechanical components
- often no "real" keyboard
- often limited display or no general-purpose display device"

For these reasons embedded systems are also known as "hidden" computer systems. These systems are built into a wide variety of products such as audio/visual equipment, appliances, vehicle engines, elevators, and medical instruments. It is common for these systems to appear to be very simple when they are actually quite complex.

Because embedded systems consist of both hardware and software, they have many of the characteristics of software-only systems. Every one of the QAs listed in Table 1 might apply to an embedded system.

Even though embedded systems have many characteristics in common with software-only systems, there are a number of characteristics which set them very much apart and can pose tough challenges throughout the development process. Characteristics that are common to embedded systems include multi-tasking, real-time response to external events, and event handling without human intervention [6]. There are also a number of constraints that are commonly imposed on embedded systems. These include small size and low weight, low power consumption, harsh environments (heat, vibration, corrosion, etc.), safety-critical operation, and sensitivity to cost [5].

The issues which face the embedded system developer start at the onset of requirements gathering and functional definition just as they do in software-only systems. And, like the software-only systems, in addition to very specific functional requirements, the embedded system is also faced with nonfunctional requirements (NFRs). Even though they are not usually referred to as QAs, these NFRs are the QAs of embedded systems. Embedded system NFRs deal with issues such as timing, memory requirements, and cost. The architecture and implementation choices must adequately address the NFRs as well as the functional requirements or the system will fail. A few specific examples of embedded system NFRs found in literature are defined in Table II [6].

TABLE II
EXAMPLES OF EMBEDDED SYSTEM NONFUNCTIONAL REQUIREMENTS DEFINITIONS

Embedded System Quality Attribute	Definition
Debugability	Failures in an embedded system must be diagnosable (and repairable) even though the human interface is minimal or non-existent
Memory Space	Embedded systems have a limited fixed amount of memory. The usage of memory must be monitored constantly throughout development and is usually a driving force in design and implementation choices.
Reliability	The embedded system is not typically allowed to crash. The software must function (or recover) without human intervention.
Response	The embedded system must meet all required response times even if it is in the middle of doing other tasks (such as handling data efficiently). Failure to do so may have catastrophic consequences.
Testability	The embedded system must be shown to be able to properly respond to all unexpected and simultaneous events, usually without human intervention, even though not all potential failures are known.
Throughput	The embedded system should not be a bottleneck on the machine-to-machine interfaces. Data handling must be extremely efficient.

IV. RESEARCH

Eleven architectural trade studies for several different embedded systems developed by three different organizations were evaluated. Although the systems being developed were similar in nature, there was enough diversity in their functionality to assume that they were representative of embedded systems in general.

The trade studies were evaluated to see what characteristics in terms of QAs were being looked at when the developers were making architectural design decisions. Each trade study was given a numbered identifier and entered into an Excel spreadsheet. As each trade study was reviewed electronically, every reference to an attribute of concern was highlighted and annotated. Attributes that were mentioned but were not used as part of the trade study decision criteria were not highlighted.

After all attributes in a trade study had been highlighted, the data was entered into the spreadsheet. Data was entered in two tables. The first table tracked the total counts for each attribute in each trade study. The second table tracked only whether a specific QA was referred to or not in each trade study. Both tables provided totals for the QAs referenced.

Table III summarizes the research findings. The first column lists the QAs by name. The second column shows how many total times each QA was referenced. The third column indicates how many trade studies referenced a particular QA at least once. The rows that are highlighted are attributes that were referenced by five or more trade studies.

TABLE IIII

EMBEDDED SYSTEM QAS RESULTS

Embedded System QA	# of Total Times Referenced	# of Trade Studies In
Accuracy	4	2
Availability	1	1
Bandwidth	1	1
Commonality/Compatibility	13	7
Complexity - CPU, System, Peripherals	25	5
Code Size	3	1
Cost	10	5
CPU Time Used	2	2
Debug-ability	3	2
Durability	1	1
Ease of Integration	2	2
Ease of Use/Usability	8	4
Expansion Capability	6	1
Features/Functionality	17	6
Maintenance/Logistical Burden	2	2
Maturity	1	1
Memory Usage	7	3
Performance	26	6
Physical Size	16	6
Power Consumption	30	10
Processor Choice	3	3
Reliability	17	6
Safety	3	2
Schedule	1	1
Security	2	2
Speed/Timing	7	3
Standards Compliance/ Ease of Certification	3	2
System Resource Usage	3	2
Versatility/Flexibility	2	2
Weight	22	8

V. RESULTS

Thirty embedded system QAs were referenced in the trade studies. Of these, three were easily identified as software-only QAs. They are code size, CPU time used, and memory usage. However, even though they can be considered software-only attributes, they still apply to embedded systems since embedded systems consist of both hardware and software.

Seven of the attributes apply only to systems that include hardware. They are durability, maintenance/logistical burden, physical size (length, width, height, and footprint), power consumption, safety, and weight. These are considered new QAs from the perspective of software-only systems.

The remaining attributes are often seen as software QAs. However, their usage in the trade studies was expanded to include hardware and firmware (programmable logic) stimuli and responses. The fact that so many of the software QAs correlated so well to those in the trade studies suggests that the quality focus of embedded systems architecture designs might have a good deal in common with the quality focus of software architecture designs. Table IV summarizes these findings.

Two correlations were found when evaluating the results. First, as the highlighted rows in Table III show, the QAs that were referenced by the most trade studies were also the ones that were referenced the highest total number of times. This was an expected correlation.

The second correlation was that the QAs that were referenced the most often (the highlighted rows) were either new QAs or existing software QAs that had taken on a new or expanded definition. None of them were software-only QAs. This is a reasonable correlation, but it was not expected. It is possible that the software-only portion of the embedded systems being evaluated performed their own independent architecture trade studies.

A final finding of interest was that there were six of the QAs that were referenced by only one trade study with a total number of times referenced of three or fewer times. They are availability, bandwidth, code size, durability, maturity, and schedule.

A possible explanation for why these QAs were referenced so few times is that some of them are actually a subset or specific instance of another QA. For example, availability, durability, and possibly maturity are all ways of looking at reliability. Bandwidth is a specific measure of performance. Code size is a direct contributor to physical size and has an indirect effect on power consumption. The more code there is, the more memory is required which eventually can lead to more power being consumed.

An explanation as to why schedule showed up so few times is that it is not really a QA but rather a project management issue. This is also true of cost even though it was referenced a total of 10 times in 5 trade studies.

VI. SUMMARY

Based on the findings of the trade study evaluations, it is clear that there are QAs that are specific to embedded systems only and not to software systems. However, the majority of the QAs apply equally well to both software-only and to embedded systems. In many cases the definition of the QA might have been modified, but that is already being done with software-only QAs, so it does not appear to be an issue.

These findings open the door to safely assume that embedded systems and software-only systems have many characteristics in common. As a result, it is possible to use software processes and models such as the ATAM to evaluate embedded systems and their architecture design decisions in a more rigorous way with a focus on quality.

TABLE IV
EMBEDDED SYSTEM QAS RESULTS

Embedded System QA	SW QA	Modified Definition	New
Accuracy		X	
Availability		X	
Bandwidth		X	
Commonality/Compatibility		X	
Complexity - CPU, System, Peripherals		X	
Code Size	X		
Cost		X	
CPU Time Used	X		
Debug-ability		X	
Durability			X
Ease of Integration		X	
Ease of Use/Usability		X	
Expansion Capability		X	
Features/Functionality		X	
Maintenance/Logistical Burden			X
Maturity		X	
Memory Usage	X		
Performance		X	
Physical Size			X
Power Consumption			X
Processor Choice			X
Reliability		X	
Safety			X
Schedule		X	
Security		X	
Speed/Timing		X	
Standards Compliance/ Ease of Certification		X	
System Resource Usage		X	
Versatility/Flexibility		X	
Weight			X

REFERENCES

[1] IEEE Software Engineering Standards Committee Std. 830-1998, IEEE Recommended Practice for Software Requirements Specifications, IEEE, New York, 1998.

[2] R. Kazman, M. Klein, and P. Clements, "ATAM: Method for architecture evaluation," *Technical Report* CMU/SEI-2000-TR-004, 2000.

[3] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies,* Reading, Massachusetts, Addison Wesley Professional, 2001.

[4] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, et al, *Documenting Software Architectures: Views and Beyond,* Reading, Massachusetts, Addison Wesley Professional, 2002.

[5] P. Koopman, "Embedded Systems In the Real World", Course Slides, Carnegie Mellon University, January 14, 1999.

[6] D. Simon, "An Embedded Software Primer", Reading, Massachusetts, Addison-Wesley, 1999.

[7] David E. Simon, "An Embedded Software Primer", Addison-Wesley, Reading, Massachusetts, 1999.