Overview of Software Engineering and Systems Engineering Development Methodology for Embedded System

Azizah Suliman¹ and Nursyazana Nazri²
Department of Systems and Networking, Universiti Tenaga Nasional, Malaysia, azizah@uniten.edu.my, nursyazana@uniten.edu.my

ABSTRACT

As established as it sounds, embedded system however does not have its own dedicated development methodology that developers can adhere to. This is due to the fact that it embodies both hardware and software that makes it different from other software development. Software development has its own methodology, and so does for hardware. This paper considers basic overview of embedded system and discusses several development methodologies in software engineering and systems engineering and how they can be used in designing embedded system.

Keywords: embedded system design, embedded system development methodology, systems engineering

I INTRODUCTION

The evolution of computer generation started back in the year of 1940 where the first generation computers are marked by the use of vacuum tubes and magnetic drums for memory. The technology was short lived due to its unreliability and space consuming. The world has seen how each generation improvised to a new one with a smaller circuitry and more advanced technology. When the Microprocessor Year came in 1970, it changed the computing development all over. The evolutions of subsequent generations were rapid and steep and are adequately addressed by Moore’s Law. The microprocessor is a single chip circuitry that could do all the processing of a full-scale computer. By putting millions of transistors onto one single chip, more calculation and faster speeds could be reached by computers (Hames, 2011). Eventually, it led to the invention and gave rise to personal computer (PC) revolution. This Fourth Generation also saw the development of graphical user interface (GUI), the mouse and other handheld devices. Apart from that, the Microprocessor Year had been the year where researchers started to develop a dedicated computer-based system, called Embedded System.

A common misunderstanding is that computers only exist in PC’s. This is far from true. Computers come in a rich variety of types and sizes, and are often built into modern products to improve and expand the usefulness of the products for the humans that are using them. The keyword here is built-into; or just another word for embedded. Embedded systems, which were formerly associated with military or other exotic environments, are now the norm in everyday applications (Sage & Rouse, 2009). An embedded system is synonym with being pervasive and ubiquitous as its presence can be felt everywhere. It is a dedicated computer-based system for an application(s) or product. It may be either an independent system or a part of a larger system. Constructing a development methodology for designing embedded system can be challenging as one must understand that it is an inter-disciplinary activity – a hybrid of hardware and software. The forthcoming section will discuss several development methodologies available in software engineering and systems engineering and why direct adoption of these is not appropriate for embedded system development.

II EMBEDDED SYSTEM

As stated earlier, the word embedded implies that it lies inside the overall system, hidden from view, forming an integral part of greater whole. Furthermore, an embedded system is a microcontroller-based, software-driven, reliable, real time control system, autonomous, or human- or network-interactive, operating on diverse physical variables in diverse environments, and sold into a competitive and cost-conscious market. A general definition given by IEEE (1992) of embedded system is; it is a computer system that is part of a larger system and performs certain specialized tasks for specific purposes which satisfies some of the requirements of that system. Therefore every system with embedded software with a specific goal can be termed as an embedded system. For easier understanding, we can generally view an embedded system as a domain that combines of both hardware and software, as depicted in Figure 1 below.

Figure 1 Venn Diagram for Embedded System
A. **Embedded System Classification**

Embedded system can be classified into three types and examples of each type are given below (Shibu, 2009):

1. **Small Scale Embedded Systems** – These systems are designed with a single 8- or 16-bit microcontroller; they have little hardware and software complexities and involve board-level design. They may even be battery operated.

2. **Medium Scale Embedded Systems** – These systems are usually designed with a single or few 16- or 32-bit microcontrollers or Digital Signal Processing or Reduced Instruction Set Computers (RISCs). These have both hardware and software complexities.

3. **Sophisticated Embedded Systems** – Sophisticated embedded systems have enormous hardware and software complexities and may need scalable processors or configurable processors and programmable logic arrays. They are used for cutting edge applications that need hardware and software co-design and integration in the final system; however, they are constrained by the processing speeds available in their hardware units.

Since embedded system is a joint of hardware and software, thus the two parts of the system need to be looked at in designing.

### III SYSTEM DEVELOPMENT LIFE CYCLE

Systems development life cycle (SDLC), or sometimes referred to as application/software development life cycle is a term used in software engineering, information systems and systems engineering to describe a process for planning, creating, testing and deploying the system. SDLC aims to produce high quality systems that meet or exceed customer expectations, based on customer requirements, by delivering systems which move through each clearly defined phase, within scheduled time-frames and cost estimates. This section will discuss few common SDLC used in software engineering and systems engineering.

A. **Waterfall**

   Waterfall is a linear framework type and one of its weaknesses is it is difficult to respond to changes. Basic concept of this methodology is the project is divided into sequential phases that consist of initial investigation, requirements definition, system design, coding, testing, implementation and operation and support or sometimes known as maintenance.

   Example of waterfall development stages is as shown in Figure 2.

   ![Figure 2 Waterfall Development Methodology (Pressman, 2010)](image)

   Throughout the life of the project, tight control is maintained through the use of extensive written documentation, as well as through formal reviews and approval/signoff by the user and information technology management occurring at the end of most phases before beginning of the next phase. Changes that occur later in the life cycle are more costly and are thus discouraged. The drawbacks of this method are problems are often not discovered until system testing.

B. **V-model**

   A variation in the representation of the waterfall model is called the V-model. It is like an extension of waterfall model but this model indicates that testing exists in every phase of SDLC life cycle. Example of the V-model is as shown in Figure 3.

   ![Figure 3 V-model Methodology (Pressman, 2010)](image)

   V-model is a variant of the waterfall that emphasizes the verification and validation of the product. Testing of the product is planned in parallel with a corresponding phase of development. This model emphasizes planning for verification and validation of the product in early stages of product development. The development is planned in such a way that each deliverable must be testable. It is an excellent choice for systems requiring high reliability due to the amount of testing involved.
C. Prototyping
Another methodology is prototyping, which is an iterative kind of framework as depicted in Figure 3.

In CMS (2008) stated that prototyping is an attempt to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process. In this methodology, user is involved throughout the process, which increases the likelihood of user acceptance of the final implementation. Small-scale mock-ups of the system are developed following an iterative modification process until the prototype evolves to meet the users’ requirements. Most prototypes are developed with the expectation that they will be discarded, and only few will evolve to working system.

D. Spiral Methodology
Another development methodology is spiral, which is a combination of linear and iterative framework as shown in Figure 5.

Spiral methodology works in four different cycles and there are no established controls for moving from one cycle to another cycle. Without controls, each cycle may generate more work for the next cycle. There is a possibility that the project ends up implemented following a waterfall framework.

E. Systems Engineering
Systems engineering is an interdisciplinary field of engineering that focuses on how to design and manage complex engineering projects over their life cycles, especially the one which involves high-end technology of hardware. This is usually referred as sophisticated embedded systems which are common in avionics, telecommunications, health system, military and industrial production system. Systems engineering ensures that all likely aspects of a project or system are considered, and integrated into a whole. A well-defined systems engineering life-cycle model that is imposed on the system from the beginning can give the maturity derived from lessons learned in using the time-tested model, thus adding an important element of stability to the organizational structure. In their work, (Sage & Rouse, 2009) indicated that systems engineering are concerned with the appropriate definition, development, and deployment phases of the engineered system(s). These aspects compose a set of phases for systems engineering life cycle as depicted in Figure 6.

1. Macro level structure of a system engineering framework. Conceptual illustration of the three primary systems engineering life-cycle phases:

2. Within each life cycle phase, there are a number of steps involved. Formulation, analysis and interpretation are used in each phase before going further to the next one. This is as shown in Figure 7.
3. Eventually, the derivation of the steps will give the final methodology where each of the phases consist more detailed steps as shown in Figure 8.

![Figure 7 Steps involved in each phase](image)

_Sage & Rouse, 2009_

![Figure 8 Systems engineering life cycle](image)

_Sage & Rouse, 2009_

IV DISCUSSION

Methodologies shared in the previous section show that they are made explicitly for software and hardware. While a great deal of research has addressed design methods for software, not as much is known about the joint design of hardware and software. In Wolf (1994), the author stated that embedded computing is unique because it is a hardware-software co-design problem. The hardware and software must be designed together to make sure that the implementation not only functions properly but also meets the extra functionality. Should there be any embedded system design methodologies available, they must be able to support incomplete specifications, design of a single engine to satisfy multiple product specifications, or changes to the specifications during design.

Previously reviewed methodologies have their own shortcomings which make them unsuitable to be directly adopted for an embedded system development. Software developments that depend on machines with the ability to handle millions of transistors may simply not be relevant to designers of systems using 4- and 8-bit microprocessors (Koopman, 1996).

In waterfall model, system performance cannot be tested until the system is almost fully done and under-capacity may be difficult to correct. This is not suitable for embedded system because it deals with hardware and must cater for changes. In testing an embedded system, we must not only check the software, but most importantly how does the software interact with the hardware related to it. Waterfall development is also not appropriate for projects that are changing for external reasons, as well as for a real-time system (CMS, 2008). Since embedded system is a real-time system and it normally has to interact with environment that is considered as external reason, it is therefore not suitable to adopt waterfall methodology in developing an embedded system. Same goes to V-model where its weaknesses includes it does not handle iterations or phases and it also does not easily handle dynamic changes in requirements. This is not suitable for embedded systems because sometimes we need to go back to one or two phases earlier due to the changing of requirement to suit the needs of environment. As for prototyping, most prototypes are developed with the expectation that they will be discarded, and only few will evolve to working system as mentioned earlier. This will be costly to be applied to embedded system as developers will have to spend a lot on changing hardware every time they move to next phase. For prototyping in software development, the developer will get response from users as they would be the one who will use the final system. However, for embedded systems, system will have to interact with environment and sometimes have to endure heat, dust and have to stand under difficult environment conditions. On top of that, it can lead to false expectations, where the customer mistakenly believes that the system is “finished” when in fact it is not; the system looks good and has adequate user interfaces, but is not truly functional. This is risky for an embedded system because as mentioned in earlier section, extra functional properties are crucial. The end product of an embedded system normally functions on its own without any human-assistance. So if the product does not respond accordingly, it may harm the end-user (if any), or the environment it has to work for itself.
For spiral methodology, since there is a possibility that the project ends up implemented following a waterfall framework, it is therefore not appropriate to be adopted in designing an embedded system as there is no proper control and it may affect the end product. Due to the nature of embedded system that works under difficult environment, the end product of an embedded system needs to be reliable so it will not jeopardize the end user as well as the environment.

Systems engineering, on the other hand, only cater for sophisticated embedded system which uses high-end hardware. It is only suitable for a complex system, and does not suit for small-scale and medium-scale embedded systems. Nevertheless, there are still elements that we can borrow to suit the need of embedded system other than sophisticated ones.

Embedded system differs from other conventional software due to the fact that it does not only have functional requirements, but also non-functional requirements. In a detailed work of non-functional properties of embedded system, Ramesh (2012) stated that his selection of non-functional properties for embedded systems are based on the ISO/IEC 9126 model which is the international standard for software quality, and are the most appropriate to embedded systems. The quality properties that are taken into consideration for classification in his work are reliability, usability, memory consumption, maintainability and efficiency. Thus, it needs careful design which optimizes different design parameters. While software development aims for flexibility, re-configurability, easy update, etc., hardware development on the other hand aims for speed, power consumption and cost in large volumes (Kuchcinski, 2014). Embedded system design discipline needs to combine these two approaches, because non-functional behavior is a crucial issue when there are real-time constraints imposed by the environment (Stefanov, 2014). To further understand on embedded system architecture, the elements are as below (Feiler, 2009):

Elements of embedded system architecture include all these:
- Software architecture (task & communication)
- Hardware architecture (relevant to embedded software)
- Physical system/environment (relevant to embedded software/hardware)
- Logical interface between software and physical system
- Physical interface between hardware and physical system
- Deployment of software on hardware

Like discussed earlier, embedded system is constrained by its non-functional properties. Non-functional properties are responsible for the proper behavior of concurrent distributed systems, their corresponding threads, synchronization, live-ness and their safety properties (Schmidt, 2003). However, non-functional properties for one particular embedded system might differ from another. For example, reliability property in a small scale embedded system might not be in an equal priority like in a sophisticated embedded system. Each type of embedded system has its own concern of non-functional properties. Therefore, that explains on why embedded system cannot have a generic methodology that can be applied into all classifications and domains. It is now widely recognized that the non-functional properties of a system are at least as important as its somewhat more classical functional properties and that, therefore, they must be considered as early as possible in the development cycle in order to avoid costly failure (Chung, 1999).

The whole research idea is to take very established methodologies in software engineering and systems engineering, considering both are essential in embedded systems, and harmonize it with non-functional properties following ISO/IEC 9126 in order to come out with a model that suits at least for the development of medium-scale embedded system. We propose to hybrid software engineering and systems engineering due to the fact that there are elements we can borrow from these two to build a new model. The model will consider the characteristics at every steps of the development process so that the non-functional properties are taken care of and the end product is not jeopardized. The general idea of the process is as depicted in Figure 9.
V CONCLUSION

The nature of embedded system itself that serves different purposes and needs makes a generic methodology not applicable. A preliminary work has been done to develop the model using selected elements that we can borrow from both software engineering and systems engineering development methodology. The resulting model of the combined methodologies is planned to be used as a case study for a flood detection system and it will be discussed in next publication.

ACKNOWLEDGMENT

The authors wish to thank the Ministry of Higher Education Malaysia for funding this study under Long Term Research Grant Scheme (LRGS/b-u/2012/UUM/TeknologiKomunikasidanInformasi).

REFERENCES