

SDR ARCHITECTURE IMPACTS ON WAVEFORM PORTABILITY AND COST MODELING

Vincent J. Kovarik, Jr. (Harris Corporation, Melbourne, FL, *vkovarik@harris.com*)

Abstract

The ability to port or reuse waveform implementations across multiple Software Defined Radio (SDR) systems has been a goal of both industry and government. However, the realization of this goal remains elusive. Waveform porting remains an imprecise process that is difficult to quantitatively estimate. Although software cost estimation models have been developed, validated, and are widely used, in standard software system cost estimation, no such model exists for porting waveform software. This paper presents some initial thoughts on cost drivers and potential approaches for establishing a cost estimation model for waveform development.

1 Introduction

One of the benefits of a software radio system is the ability to add, update, or enhance functional capabilities of the radio system through software without incurring the cost to change the underlying hardware. This section provides an overview and description of several of the concepts and approaches to software radio construction and the process by which the system may import or reuse applications across radio systems.

Initial assumptions of the Joint Tactical Radio System (JTRS) program were that the waveforms developed to be compliant with the Software Communication Architecture (SCA) would be portable to other SCA-compliant radio systems. However, the SCA was not intended to provide a technical reference architecture for either the underlying radio platform or the waveform implementation. It focused on the

architecture of the management and control of the radio system or what commonly referred to as the Core Framework.

1.1 Waveform Porting and Reuse

Waveform portability has been consistently promoted as a significant benefit of a software radio. Whether it is in the form of reusable components that may be dynamically assembled through interconnection, as in the case of the SCA model [1] or through reuse of software at the source code level. While reuse of a waveform from one software radio system to another without modification was not an explicit goal of the Joint Tactical Radio System (JTRS) program or the Software Communications Architecture (SCA), it has been an implicit goal. However, several years into JTRS development programs, waveform portability remains an unsolved problem.

1.2 Paper Organization

In section 2, a brief overview of software cost modeling is presented. This provides a foundation on which an approach to developing a cost model for developing and porting a waveform will be proposed.

After discussing Software Cost modeling, section 3 presents several areas impacting the effort to port a waveform. Anecdotal observations provided by a waveform port project provide the general framework for extending the cost model.

Section 4 then proposes extensions to the software cost model to incorporate various elements that affect waveform porting. In addition to assessing the composition of a software waveform, the impacts of

unique hardware are discussed. The different physical architectures have a direct impact on the cost and schedule drivers on waveform development and porting.

Finally, section 5 conclusions are presented and some thoughts for continued research in this area are presented.

2 Software Cost Modeling

As software systems were becoming more pervasive and complex in the early 1980's, the need to reduce development costs and times through reuse coupled with the need to better estimate software project development costs drove initial research in the area of software cost modeling. Early work by Boehm[2] and development of the Constructive Cost Model (COCOMO) [3] provided initial empirical models for estimating software development costs.

The breadth and depth of software cost estimation models and tools has continued to evolve and there are a range of tools and methodologies available today. Since the focus of this paper is on the applicability of software cost modeling techniques to waveform porting, basic concepts first developed by the COCOMO model will be used as a foundation.

2.1 Background

The fundamental premise of the cost model is that the effort to develop a software system is driven by the delivered lines of source code (for the purposes of this paper, a *line* of source code is a single statement in a high-level language). The development effort, in person months, is calculated using equations 1 and 2 below.

$$PM = \alpha KDSI^\beta \times \prod_{j=0}^N EM_j \quad (1)$$

$$\beta = 1.01 + 0.01 \sum_{i=0}^N W_i \quad (2)$$

$KDSI$ is the number of delivered source lines (in thousands), α is the factor multiplier for the $KDSI$

parameter. $KDSI$ is modified by the exponent β which is calculated, as shown in equation 2, based on a set of weighting factors that characterize the software, e.g. complexity, required reliability, etc. This is multiplied by EM_j which is an Effort Multiplier that adjusts the calculation based on the level of expertise of the personnel, development tool availability, etc.

2.2 Reuse Cost

The cost of reuse, based on empirical analysis of 2,954 software modules at NASA performed by Selby[4] provided an initial insight into the cost associated with reuse. The results of Selby's analysis are illustrated in figure 1.

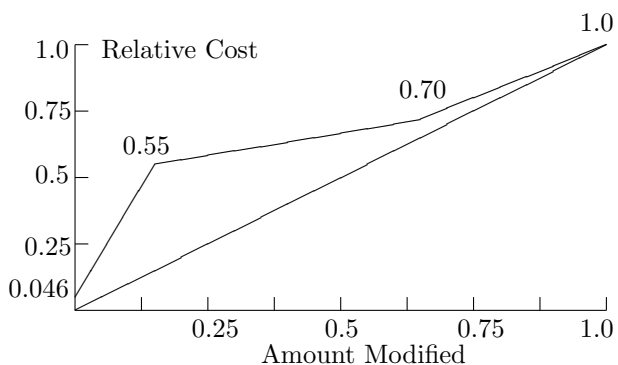


Figure 1: Cost as a Function of Percent Modified

Figure 1 maps the relative cost of development as a percentage of the amount of code modified. If reuse is high, i.e. a lower percentage of code is modified, then the relative cost is lower. The initial assumption is that there is roughly a linear relationship between the amount of code modified and the relative cost. This assumption is illustrated in the figure as the diagonal line.

What Selby found, as shown in the figure, was quite different. First is the fact that if code is reused without modification, there is still a cost associated with the reuse. This is shown by the reuse line segment starting at 0.046 on the relative cost axis. The reason is that, even when code is reused without modification, there is still effort expended to understand the functions, interfaces, and behavior of the code.

Secondly, there is a steep raise in relative cost for roughly the first 12% of the code that is modified. This is also related to the level of effort required to understand the code components and an increase in the complexity due to the number of interfaces that require verification. So, the first 12% of code modified accounted for roughly a 55% relative cost.

At this point, the slope of the cost line decreases significantly. Consequently, there is a relatively small increase in the relative cost between 12% and roughly 60% code modification increasing from a 55% relative cost to 70% relative cost. After this last point, the cost line converges to the end point of the linear assumption.

2.3 Applicability to Software Radio

The above data on reuse cost was collected for high-level programming languages on general purpose processors. There was a mix of embedded, real-time, and general purpose computer platforms. However, much of the underlying computational platforms were generally similar. This is not the case in many software radio platforms. A software radio may have a multitude of processors of different families, such as General Purpose Processor (GPP), Digital Signal Processor (DSP), and Field Programmable Gate Array (FPGA).

Consequently, two key questions arise:

1. How can the reuse data and cost modeling previously discussed be applied to the software radio domain?
2. What changes to the cost model must be made to account for the different processor types and the languages supported on each?

In order to answer these questions, there must be some basis for comparison and analysis. The next section discusses an initial effort for gain some fundamental understanding of the these aspect through a waveform port activity.

3 Waveform Port

Harris has extensive background in software radio development and has been participating in the JTRS program since the initial architectural studies. Harris provides a range of communications systems from manpacks, handhelds, and other small form factor radio systems to satellite terminals and other large-scale communications systems. The fundamental platform architectures between the two groups have evolved very differently due to the resource constraints, i.e. Size, Weight, and Power (SWaP), imposed on the small form factor systems versus the resource-rich environment of the larger terminal systems. In the course of internal discussions, the question was raised on the feasibility of porting a waveform between these significantly different radio systems.

In response to these discussions, an Internal Research and Development (IR&D) project was initiated to port a VHF/UHF Line-of-Sight (VULOS) waveform. The objectives of this effort were to:

1. gain an understanding of the processes and aspects of waveform porting within an SCA environment,
2. gain insight into the cost of waveform porting,
3. understand impacts on the porting process due to the architectural differences between the source and target hardware, and
4. explore the potential application of software costing models to the waveform porting process.

3.1 Hardware Architectures

The VULOS implementation was originally developed for a smaller form-factor radio system that had a GPP, DSP, and FPGA all resident on single board. Direct interconnects between the processors enabled tightly integrated communications between the processors. For example, the software components on the GPP could raise an interrupt to the DSP signaling that data was ready for the DSP and then send the data to the DSP. The target radio system

had higher capacity processing resources, i.e. faster GPP with more memory, faster and larger FPGA, and faster DSP processors. What was significantly different, however, was the physical architecture of the processing resources and interconnections.

For example, where the source hardware enabled the GPP to raise an interrupt to the DSP directly, the target hardware had a Single Board Computer (SBC) hosting the GPP and the DSP resided on the digital modem processing card. Both cards were in a VME backplane. In order for the GPP to raise an interrupt to the DSP, it had to make a call to a that raised the signal to the modem card over the VME bus. The signal received by the modem card which then raises the interrupt line to the DSP. Thus a significant difference in the control interaction between the GPP and DSP existed between the source and target hardware..

3.2 Architectural Drivers

As the waveform port progressed, a additional impacts were encountered. Each impact could be categorized into one of three architectural categories, Processor, Control, and Data Input/Output (I/O).

Conceptually, these architectural aspects can be thought of as a three dimensional space with each aspect representing one of the orthogonal dimensions. The relative cost of a waveform port can be qualitatively represented as the magnitude of the vector between the source and target implementations. Each of these dimensions are discussed below.

3.2.1 Processor Impacts

Differences in processor architectures impacts the effort to port the baseline code. This includes differences in compiler technologies, tool availability, processor models, operating systems, etc. For example, porting signal processing code from a fixed-point DSP to a floating-point DSP would be, at first glance, relatively simple, i.e. the difference between the source and target points on the Processor dimensions is small. However, the impact due to data representation differences between the fixed-point and floating-point could have significant ripple effects through

other components. For example, if the DSP is forwarding fixed-point sample data to an FPGA, the change in representation would result in additional code modifications on the DSP or on the FPGA to ensure that the data is formatted and interpreted correctly.

In the case where a component previously implemented on a FPGA is to be ported to a GPP, there is a much greater 'distance' between the source and target points. This is due to several factors including significant differences between the source language, VHDL, and the target language, C/C++. In addition, the design of the component will significantly change. The VHDL implementation is inherently a parallel state machine. However, the GPP is single instruction stack machine. This transition often requires significant re-design to successfully execute on the GPP.

3.2.2 Control Impacts

Control of the waveform refers to the timing, synchronization, and coordination between the waveform components. Differences in physical architecture can have a significant impact on the timing and synchronization between the components. This aspect is highly dependent on the physical architecture of the underlying hardware. Components that are co-located within a single processor are typically integrated on a new platform with minimal impact to the control protocol between the components.

Problems arises when two components with control interdependencies are placed on different processors resulting in a different set of control interconnections or paths between the two components.

3.2.3 Input/Output Impacts

The third architectural aspect is the Input/Output (I/O) transport between waveform components. This refers to the end-to-end signal processing chain. Some of the factors related to I/O that impact the effort to develop or port a waveform, include data formatting, throughput, data path width and rate, and establishing connections between components. Data formatting is concerned with the format of the data

as it is marshaled over an interconnection path. This is particularly important when marshalling data between dissimilar processors. The bandwidth of an I/O path may impact the level of modification to compensate for differences in throughput.

4 Waveform Cost Modeling

The fundamental problem with attempting to apply a software cost model to waveform development is that the traditional cost models are based solely on traditional or high-level programming languages on general purpose processors. Although waveform implementations may be limited to a single processor and/or language, typically the waveform is a hybrid implementation and trends more towards VHDL implementations for high-performance waveforms.

Consequently, any cost estimation model must account for and support the differences in the languages as part of the total implementation. Equation 3 extends the effort calculation first presented in equation 1 by representing the effort as a total of each language and incorporating effort factors for each.

$$PM_{Hybrid} = \alpha_g K DSI^\beta \times \prod_{j=0}^N EM_j \quad (3)$$

$$+ \alpha_d K DSI^\beta \times \prod_{j=0}^N EM_j$$

$$+ \alpha_v K DSI^\beta \times \prod_{j=0}^N EM_j$$

This equation extends the effort estimate by providing separate effort elements for each of the development languages and multipliers, α_g , α_d , and α_v , for GPP, DSP, and FPGA code respectively.

However, it does not address cost impacts due to control and I/O interdependencies discussed previously. Adjusting the level of effort to account for control and I/O dependencies is dependent on the *number* and *complexity* of the interdependencies that must be resolved. Equations 4 and 5 below provide an initial proposition for calculating this factor.

$$PM_{Total} = \delta PM_{Hybrid}^\kappa \quad (4)$$

PM_{Hybrid} is the overall effort value from equation 3, κ is the multiplier calculated based on the extent of the interdependencies, and δ represents an overall multiplier for the interdependency impacts.

$$\kappa = 1.0 + \mu \sum_{i=0}^N I_i(x, y) \quad (5)$$

κ is calculated as a sum of the set of individual dependencies where $I_i(x, y)$ represents an impact factor due to a single instance of dependency between two modules hosted on different processors, x and y , N is the total number of pair wise dependencies, and μ is a multiplier to adjust the sum of the dependency factors.

Given the above equations as an initial proposition for estimating the effort associated with porting a hybrid waveform implementation, some definition of the dependency factors, $I_i(x, y)$, must be developed.

Pair	Processor	Control	I/O
GPP-GPP	L	L-M	L-M
DSP-DSP	L	L-M	L-M
DSP-GPP	L-M	L-M	M-H
FPGA-FPGA	M	M-H	M-H
FPGA-DSP	M-H	M-H	M-H
FPGA-GPP	M-H	M-H	M-H

Table 1: Qualitative Pair Wise Dependency Factors

Table 1 provide an initial set of qualitative assessments of the impact due to dependency factors porting from a source processor to a target processor. The factors are simply identified as Low (L), Medium (M), or High(H) at this point in time. The Pair column identifies the source and target processors. The simplifying assumption is that porting across processors will always occur between like processors, e.g. DSP-DSP, or between a higher performance processor to a lower performance processor, e.g. FPGA-GPP. This table will be developed for each type of dependency impact, Processor, Control, and I/O.

5 Conclusions

Understanding and estimating the effort required to develop and port a waveform has proved to be a difficult problem. This has been largely due to the hybrid nature of software radio waveform implementations with the most significant impacts driven by the underlying hardware architecture.

Based on our waveform porting experience, several key considerations and impact factors have been identified. These key aspects are presented below.

- **Porting Cost Function** - Waveform porting and reuse is a cost function. The magnitude of the cost is driven by the architectural factors described in the previous paragraphs. The cost function is non-linear and, even for reuse without modification, is a non-zero cost. Factors that can reduce the effort are the same as those in traditional software estimation, domain expertise, experienced personnel, comprehensive tools, etc.
- **Logical Hardware Modeling** - A logical model of the target hardware enables the ability to represent hardware architecture constraints and more accurately identify impacts to the waveform development. In porting scenarios, the degree of dissimilarity between the source and target hardware architectures results in a corresponding delta in the effort required to perform the waveform port.
- **Separation of Functions** - General software design tenets and practices can be applied to VHDL to help reduce the magnitude the porting effort by limiting the modifications to a subset of the total VHDL modules. Separating control and transport modules from functional transformations can have a positive impact in reducing the porting effort.

An initial approach to extend software cost modeling techniques for software radio waveform development has been proposed. The extensions address factors for incorporating impacts on control and data I/O due to specific hardware architectures. The factors proposed are based on the qualitative assessment based on the waveform port.

The ability to quantitatively estimate development costs for hybrid applications has potential benefits to other embedded development domains for example avionics, automotive, consumer electronics.

Further research is require to collect empirical data that can refine the initial approaches to cost models for hybrid development. This will begin to identify specific values for effort factors, α_g , α_d , and α_f , values for the dependency factors in Table 1, and the pair wise impact factor, δ , .

As a better understanding of the relationships between the underlying architecture differences in a hybrid system and the resultant impacts is gained, refinements to the proposed cost model extensions in equations 3, 4, and 5 will be required as well.

References

- [1] Joint Tactical Radio System (JTRS) Office, *Software Communications Architecture Specification*, JTRS-5000SCA, Version 2.2.2, May, 2006.
- [2] Boehm, B., *Software Engineering Economics*, Prentice Hall, 1981.
- [3] Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R. and Selby, R., *Cost Models for Future Software Life Cycle Processes: CO-COMO 2.0*, Special Volume of Software Process and Product Measurement, J.D. Arthur and S. M. Henry Eds., J.C. Baltzer AG, Science Publishers, Amsterdam, The Netherlands, 1995.
- [4] Selby, R., *Empirically Analyzing Software Reuse in a Production Environment*, In *Software Reuse: Emerging Technology*, W.Tracz (Ed.), IEEE Computer Society Press, 1988, pp. 176-189.