

Regis University ePublications at Regis University

All Regis University Theses

Fall 2012

A Guide to Documenting Software Design for Maximum Software Portability for Software Defined Radios

Joseph Snively
Regis University

Follow this and additional works at: <https://epublications.regis.edu/theses>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Snively, Joseph, "A Guide to Documenting Software Design for Maximum Software Portability for Software Defined Radios" (2012).
All Regis University Theses. 230.
<https://epublications.regis.edu/theses/230>

This Thesis - Open Access is brought to you for free and open access by ePublications at Regis University. It has been accepted for inclusion in All Regis University Theses by an authorized administrator of ePublications at Regis University. For more information, please contact epublications@regis.edu.

Regis University
College for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

Use of the materials available in the Regis University Thesis Collection (“Collection”) is limited and restricted to those users who agree to comply with the following terms of use. Regis University reserves the right to deny access to the Collection to any person who violates these terms of use or who seeks to or does alter, avoid or supersede the functional conditions, restrictions and limitations of the Collection.

The site may be used only for lawful purposes. The user is solely responsible for knowing and adhering to any and all applicable laws, rules, and regulations relating or pertaining to use of the Collection.

All content in this Collection is owned by and subject to the exclusive control of Regis University and the authors of the materials. It is available only for research purposes and may not be used in violation of copyright laws or for unlawful purposes. The materials may not be downloaded in whole or in part without permission of the copyright holder or as otherwise authorized in the “fair use” standards of the U.S. copyright laws and regulations.

A GUIDE TO DOCUMENTING SOFTWARE DESIGN FOR MAXIMUM
SOFTWARE PORTABILITY FOR SOFTWARE DEFINED RADIOS

A THESIS

SUBMITTED ON THE 11TH OF DECEMBER, 2012

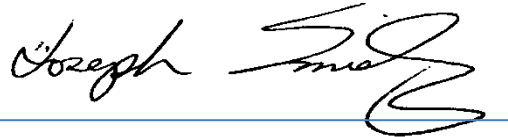
TO THE DEPARTMENT OF COMPUTER SCIENCE

OF THE SCHOOL OF COMPUTER & INFORMATION SCIENCES

OF REGIS UNIVERSITY

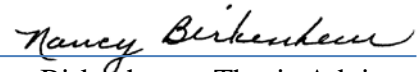
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF MASTER OF SCIENCE IN
SOFTWARE INFORMATION SYSTEMS AND DATABASE TECHNOLOGY

BY

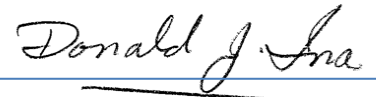


Joseph Snively

APPROVALS



Nancy Birkenheuer, Thesis Advisor



Don Ina, MCT 624B Professor



Abstract

The use of software defined communications systems is growing incredibly fast. The field of software engineering as a discipline has not adequately addressed the subject of software portability which makes large and costly software development efforts less ready to port to future platforms. By understanding the causes of portability problems, they can either be avoided altogether in development or very well documented so that they are easier to overcome in future efforts. Literature, case studies, and surveys are used to collect opinions and information about large software programs where portability is a desirable characteristic in order to best establish the facts and way forward for future research efforts.

Acknowledgements

I wish to acknowledge the efforts of my colleague, Lane Anderson, for helping to thoroughly understand the subject matter and use him as a sounding board for brainstorming sessions. I wish to acknowledge the valuable insights and patience of my unflappable thesis advisor, Nancy Birkenheuer, who repeatedly encouraged me in finding the right topic to discuss, the right turns to take, and the right way of expressing myself. Finally, I wish to acknowledge that part of my motivation for finishing this thesis was to make my dad proud. His excitement about me finishing my Master's Degree was infectious and turned my spirits around whenever I thought about it.

Table of Contents

Abstract	1
Acknowledgements	2
Table of Contents	3
Chapter 1 – Introduction	4
Traditional Radio Development Programs	8
Software Defined Radio Development	8
Distinguishing Between Radio and Radio Application	10
Software Portability	10
Chapter 2 – Review of Literature and Research	12
Domain-Specific Literature	14
Hardware selection.....	15
Application programming interfaces.	20
Modem software.	21
Operating system software.....	21
Modern Software Design Literature	23
Chapter 3 – Methodology	28
Selecting Survey Questions	28
Identifying Survey Candidates.....	29
Survey Bias	30
Survey Delivery and Collection.....	30
IRB Approval.....	31
Chapter 4 –Results	32
Survey Responses	32
Analyzing the Responses	34
Chapter 5 – Conclusions	36
Appendix A - References.....	40
Appendix B - Annotated Bibliography	43
Appendix C – Survey of SDR Platforms	53
Appendix D — Simplified JTRS SDR Architecture	56
JTRS Platforms	56
JTRS Waveforms	58
JTRS Platform Abstraction.....	58
Appendix E - Glossary.....	61
Appendix F – Survey Instrument.....	64

Chapter 1 – Introduction

This thesis is a guide to understanding what documentation is needed to properly capture software dependencies in radio applications running on mobile platforms for the sake of maximizing software portability in Software Defined Radios (SDR). A very good reason that software portability is important is that portable communications devices are becoming increasingly mainstream, and there is a boom in the number of devices that can host software defined radio technology. According to a study in 2011 on the adoption of SDR technologies, “over 93% of the mobile infrastructure market utilizes SDR technology” (Wireless, 2012). The Wireless Innovation Forum, formerly the Software Defined Radio Forum, also stated that “Almost 1 billion software defined radios will be shipped in 2011 for mobile terminal applications. And virtually all tactical radios for military communications utilize SDR technology today” (Wireless, 2012).

Tactical military radios, cellular telephones, and even tablets are representative examples of portable communications devices that are revolutionizing the entire ecosystem of radio hardware and radio software development. Both military and private sector initiatives are taking advantage of the miniaturization of high performance digital hardware in the form of SDR. As the number of devices grows and the hardware evolves, having highly portable software to migrate from platform to platform will yield a high return on investment for developers of radio software. It is the position of this thesis that the key to improving the portability of radio software is capturing the implicit dependencies that software radio applications have on other system software, hardware, and services. Capturing these dependencies and understanding the potential porting targets will yield better design decisions and minimize the amount of rework required to re-host a software radio application on another platform in the future.

Traditional Radios vs. Software Defined Radios

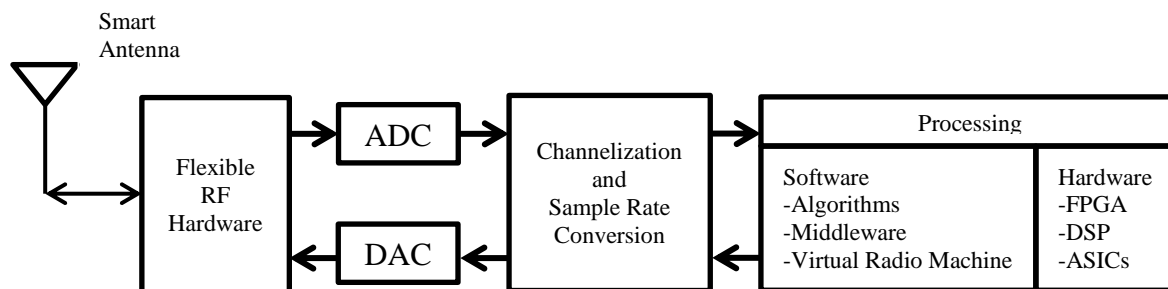
For the purposes of this thesis, a radio is defined as any device that receives and or transmits electromagnetic radiation for communication between two or more nodes in a radio network. In the book, *Software Defined Radios*, Reed (2002) said, “The SDR forum defines the ultimate software radio (USR) as a radio that accepts fully programmable traffic and control information and supports a broad range of frequencies, air-interfaces, and applications software” (p. 2). The Wireless Innovation Forum (2012) similarly stated the definition of a SDR as “Radio in which some or the entire physical layer functions are software defined.” And while these definitions are accurate, they may not be clear to all readers. Thus, a practical distinction must still be made between traditional radios and SDRs. While the exact definition is still controversial, there are some practical areas that are clearly in contrast that will be discussed in the following paragraphs.

A traditional (i.e., analog) radio is comprised of hardware and software that has very limited reconfigurability. The traditional radio works in a particular range of frequencies, with a particular range of performance parameters and protocols. Much of the hardware is analog, and very little of the functionality is software defined. As radio systems have evolved, a common term for the set of functions, capabilities, performance characteristics, and operational configurations is often referred to as a “waveform.” The term “waveform” and “radio application” will be used interchangeably in this thesis. To recap, a traditional radio is designed to implement a single waveform or some subset of a waveform’s requirements.

In comparison, an ideal SDR is comprised of hardware that is much more general purpose and can be reconfigured with software changes to host multiple radio applications, sometimes simultaneously, that potentially have different operational frequencies, protocols, and

networking capabilities. With a SDR, upgrades and bug fixes to a radio application can be made through software updates. In other words, a single SDR is capable of being multiple types of radios by simply loading and running the desired radio application. Reed (2002) provides a simplified architecture of a software defined radio as is seen below.

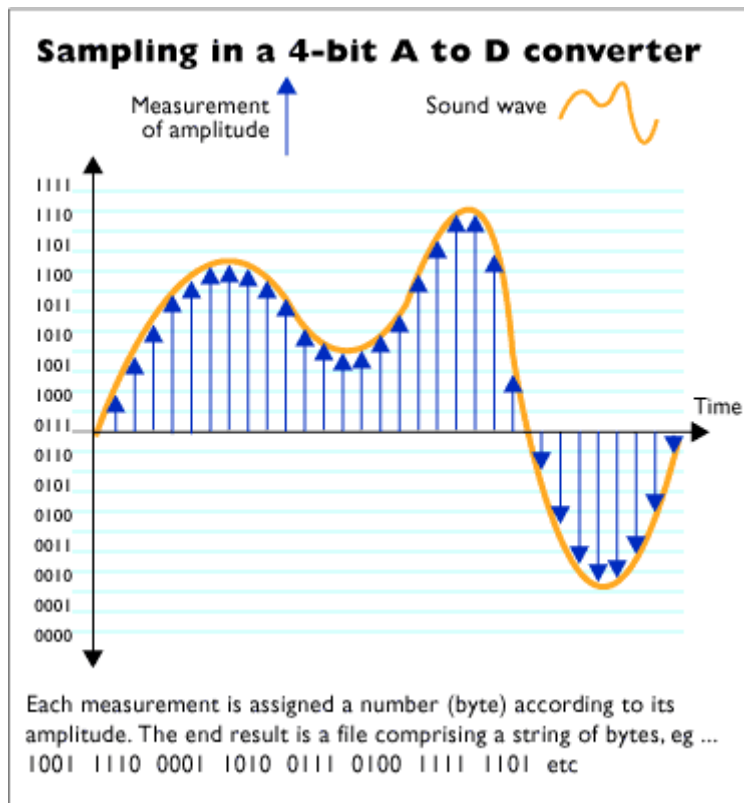
Figure 1: A high-level Software Defined Radio Architecture



The figure above shows a near-ideal software defined radio where nearly every component is configurable. But this is not the case in modern software defined radios which have a narrower range of flexible options, but are still flexible and powerful. The flow of the data is what is of value in this figure. The antenna on the left captures the signal and is sometimes called the “air interface.” The RF hardware performs the filtering that is required to perform various Radio Frequency (RF) protocols such as Orthogonal Frequency Domain Multiplexing (OFDM), Code Division Multiple Access (CDMA, Global System for Mobile Communications (GSM). The Analog-Digital Convert (ADC) is a special purpose piece of hardware that converts the analog signals that have been cleaned up into digital signals that model the analog wave. The website, <http://allthingsembedded.com/wp-content/uploads/2008/12/adc.gif> provided a graphic that depicts what an ADC would do with a continuous wave of energy. The following figure is a simple representation of what it means to sample a continuous (i.e. analog) wave and create a set of digital data points that represents the continuous wave via a technique known as “sampling”.

It is worth noting that this mechanism is present in sound waves as well as electromagnetic waves.

Figure 2: Sampling a continuous wave.



Continuing with the breakdown of Figure 1, once the ADC has churned out a stream of bytes that represent an analog signal, the digital manipulation of signal begins by performing a series of complex digital transformations that extract data, networking information, encryption and decryption services, and more.

Due to the complex nature of SDR technology, even the early decisions regarding hardware, software architecture, system design, programming languages, development environment, application programming interfaces (APIs), and component architecture all have a dramatic impact on the implicit dependencies that the software and hardware will have on one another. These choices can reduce the reuse and portability of the radio application software.

Thus, they must be documented to aid in future efforts. Additionally, coding practices can greatly affect the portability of the source code between compilers and platforms because of how native types are represented, compiler optimization, and other issues. However, it is the quality of documentation regarding these issues that is both a great risk and opportunity. Many of the great technical challenges to porting software can be streamlined with minimal cost and rework with proper documentation. In essence, maximizing code reuse and extending the usability of radio application software can be achieved by properly documenting the design decisions that directly impact software portability.

Traditional Radio Development Programs

Traditionally, the US military developed and purchased radios that were highly reliant on dedicated and proprietary hardware that had little configurability outside the primary function of that hardware, though the radios had interoperability requirements to be functional with other existing radios. This led to very costly and highly repetitive development efforts and interoperability issues still crept in. For example, the military may require that an airplane, a ship, a vehicle and a man-pack radio system all be able to communicate with one another with a standard military protocol such as Single Channel Ground Airborne Radio (SINCGARS). Each radio was its own development effort that duplicated the development of a standard military communication protocol. The hardware was not reusable for another type of communication protocol, and the systems were highly proprietary.

Software Defined Radio Development

Software defined radios are in both commercial and government endeavors. The SDR Forum is a community of experts that are evolving a software defined radio architecture that will one day enable a radio market that resembles the current PC market. The Joint Tactical Radio

System (JTRS) is a congressionally funded program with primary goals of establishing a Software Communications Architecture (SCA) and of acquiring radio platforms and radio applications that are architected towards the use of commodity digital hardware such as General Purpose Processors (GPP), Digital Signal Processors (DSP), and Field Programmable Gate Arrays (FPGA). Appendix D of this thesis contains information about the products being acquired through the JTRS program. Some of the products are radios. Other products are waveforms that will one day run on those radios. Vendors of future radio platforms for the military and commercial entities will employ the SCA as a governing standard in the building of the radio platform and the radio applications. Reed says, “JTRS is a software communications architecture (SCA) specification structured to allow the portability of applications between different SCA implementations, use existing commercial technology to reduce costs, provide an object-oriented framework to reduce the development cycle of new systems, and remain sufficiently open to allow the integration of new commercial frameworks and architectures.”

It is perhaps obvious, but should still be noted here that radio platforms do not use traditional desktop operating systems which are not optimized for these types of functions, and thus there are a variety of embedded operating systems running on hundreds of portable devices. Embedded Real Time Operating Systems (RTOS) such as Green Hills INTEGRITY and WindRiver VxWorks provide secure operating system choices for SDRs. They also provide C Standard Template Library (STL) container classes and implement POSIX profiles that allow radio application developers to be somewhat removed from the low level details of the hardware. Still, applications targeted at these platforms would lack portability and would be tightly coupled to the operating system running on a particular platform.

Middleware such as Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) is being used in SCA compliant platforms and waveforms to define the Application Program Interfaces (APIs) between the radio applications and the radio operating systems which further improves the portability of the C/C++ application code. It should be noted that CORBA is not typically found in the DSP and FPGAs meaning that radio applications with software on those components are still using highly customized code targeted directly at those devices and may therefore be less portable.

Distinguishing Between Radio and Radio Application

The distinction between radio operating environment and radio application software development still needs to be quantified for the reader. A SDR is the physical hardware, interfaces, OS, and middleware required to control the hardware. But, the software that makes up the operating environment such as the OS and middleware performs very little actual waveform functions. A convenient analogy is that Microsoft Windows is not a word processor, nor does it perform those functions. Rather, Microsoft Word is a word processor application and the two are distinctly different. This analogy is also useful to illustrate that different operating systems present different interfaces to the application developer. Thus, Microsoft Word does not run on Linux or Mac OS X unless a special version is created for that purpose.

Software Portability

Software portability is a term that can mean many things depending on the context. There is no root definition in industry practices. In fact, the subject of software portability has not received much dedicated attention as compared to other aspects of software design. For the purposes of this paper, software portability is a qualitative measure of effort required to host a body of software in a different environment than it was originally targeted towards. For instance,

a body of software may be readily portable between platforms that share the same processor architecture and operating system. In contrast, that same body of software may require extensive rework to re-host on a platform that uses the same hardware, but a different operating system. Mooney (1997) stated “A software unit is portable (exhibits portability) across a class of environments to the degree that the cost to transport and adapt it to a new environment in the class is less than the cost of redevelopment.” This is a suitable conceptual definition that is intuitive, practical and meaningful, though not universally understood in this same way by all software developers.

The research and data gathering of this paper will show that there are distinct decision areas that impact the portability of the software. The hardware diversity in the SDR ecosystem presents a significant challenge to application developers even when layers of abstraction are present. Since every SDR is different in form, components, performance, and intention, not all SDRs will even have the same number of ports, processors, antennas, amplifiers, user interface, etc. In modern desktop software development, the target is usually the operating system, and not the hardware elements. This is a challenge for the application developer to overcome. In SDRs, the target is more completely specified down to the component level performance metrics. The software diversity in SDR operating system presents the issue of services being available on one platform, but not another. The rest of this thesis will discuss determining the portability constraints, and which constraints should be captured in documentation.

Chapter 2 – Review of Literature and Research

The literature used to support this thesis paper comes from disciplines related to the design and development of software defined radios. Sources include US government radio programs, academic resources, and software development best practices. Despite the literature coming from a wide variety of sources, most discussions of software design and radio design only address software portability implicitly or as a secondary discussion and instead focus on traditional software design documentation constructs. Furthermore, the best discussions in software portability were targeted towards the development of applications for more common PC platforms or special purpose military projects for which there is very little publicly available information. As Mooney (1997) stated, “Portability is recognized as a desirable attribute for the vast majority of software products. Yet the literature on portability techniques is sparse and largely anecdotal, and portability is typically achieved by ad hoc methods.”

Overview of Literature Resources

This review of available literature will show that the disciplines and techniques of documenting and designing specifically for software portability are inadequately discussed and not widely accepted at this time, especially for embedded platforms. This lack of quality literature and guidance is a primary motivator for this research. In short, this literature review establishes the need for documenting software and hardware dependencies to improve portability. The literature reviewed also helps to establish specific areas that would benefit most from documentation. The literature is organized by topic. The topics reviewed are the impact that hardware, APIs, middleware, operating systems and modem software affect the radio application space which ultimately impacts portability by creating dependencies on specific choices.

Broadly speaking, the SDR ecosystem is comprised of platform integrators, hardware vendors, operating system vendors, and application software vendors. Platform integrators combine radio hardware with embedded operating systems and the necessary interfaces for application development. Platforms and operating systems are often special purpose or proprietary. One of the challenges for writing portable software is that potential platforms have to use similar standards, otherwise the software requires rework. There are similarities to the situation when application developers for PCs were developing software before PC hardware and PC operating systems shared a group of known standards. Application developers went to great lengths to try and guess which PCs were going to sell well that year and try to develop software for those PC platforms. The PC market has since seen a commoditization of hardware components. Standards exist for virtually every component, connection, and connector. The result is that vendors of PC parts that wish to be competitive must meet these standards. Developers of software applications are reasonably assured of certain platform APIs and hardware capabilities which opens the application market for software innovations. Until the SDR market has these supporting standards, developers of SDR platforms will continue to have proprietary hardware, proprietary interfaces, proprietary connectors, and non-standard interfaces for developers to work around.

The high-level flow of the literature research is to discuss domain-specific literature on hardware choices, software architecture, performance characteristics, and portability guidelines. After the domain-specific literature is reviewed, the popular and widely accepted software abstraction and documentation constructs are discussed to illustrate to the reader of this thesis that the challenges of portability are not directly addressed by conventional software design

constructs. These two complimentary discussions will adequately set the stage for the survey research and suggested portability documentation in the later sections of this document.

Domain-Specific Literature

At a high level, the domain-specific literature agrees that software portability is a high goal for radio application software, and that there has not been a distinct effort to establish the criteria to quantify and capture the relevant details that occur during software design that relate to portability constraints. In other words, the literature available today does not discuss how and where to document portability constraints for maximum return on investment. For example, the paper, *Bringing Portability to the Software Development Process*, convincingly discusses that software portability is a desirable quality across many different software domains, but software engineering texts still focus on ideas such as software reuse or simple implementation strategies instead of systematically discussing how to incorporate portability characteristics into the software. Mooney (1997) stated that the keys to improving portability, at the unit level, are to “control interfaces, isolate dependencies, and think portable” (p. 3). This is a valid opinion that is supported throughout his paper, however, the remainder of the literature research done for this thesis shows that interface control is only way to manage dependencies that impair portability. Mooney (1997) stops short of recommending how to actually document and measure portability. Instead he gives guidance that any part of the software which is “system dependent” be documented (p. 7). Kovarik (2006) stated the following:

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 (p. 1).

The remainder of the literature survey discusses the decision making processes, the constraints to portability, relevant software architectures, the use of standard APIs, and the underlying need for documentation. This thesis is the extension of that knowledge, which is to identify the most important aspects of software design, architecture, and implementation that improves the portability profile of a given radio application software. In support of this thesis, the literature review is organized to give the reader an understanding of radio hardware architecture, radio software architecture, and radio software development and the decisions that affect portability. The sub-sections that follow discuss the major themes of developing SDRs: Hardware selection, application programming interfaces, Modem software, application layer software, operating system software, and middleware.

Hardware selection.

Radio hardware and services in the SDR community is very diverse. This diversity is at the root of so many problems with writing portable software. Appendix C contains a survey of several known software defined radios. Some of the radios have common purposes and functions which would imply that software reuse across those platforms is desirable. Other platforms are used strictly for scientific research and have fewer hardware concerns. This survey is just a sample of all radios. The military uses many radios, but information about those radios in the public is scarce for obvious reasons of national security. The academic radios, while having a great deal of openness about their architecture and performance, are at the other end of the spectrum having very cheap components that are easily reconfigurable. The purpose of sampling the SDR market is simply to understand where SDRs are today and where the opportunities for portability might be now and in the future.

One conclusion from the survey is that military systems use a security architecture that employs cryptography. It is not known if all, or only some, military radios employ cryptographic systems for encrypting data, but none of the commercial systems did at all. The difference in security architectures is definitely an area of concern for portability. Depending on how the hardware of the radios is configured, and what policies the operating system is enforcing, the waveform components would be architected and distributed differently across the computational elements.

Also, the performance, capacity, and interfaces of the hardware elements that make a software defined radio are vastly different between platforms. The reasons for this are myriad. Size, weight, and power constraints force hardware vendors to make customized components that perform specific tasks. The developer of a software application must take this into account when developing for the original target platform. Any non-standard interfaces in the hardware will likely create a dependency between the hardware and the piece of the software application that is configuring it. The platforms have different processor architectures, different motherboards, different RF hardware, and different power consumption from each other. These factors make it very difficult for application developers to write software that can control the various pieces of hardware. The High Performance Software Defined Radio group is attempting to modularize and standardize the individual components of an SDR for amateurs and enthusiasts. The selection of the hardware that goes into a radio is influenced by a number of factors including system functional requirements, component level performance requirements, operational requirements, environmental compliance, reliability, and more. This complexity cannot be avoided if a system is to be robust and meaningful to the customer. Reed (2002) said “Determining the digital hardware composition of a software radio is a key design step in its creation. The hardware

design is, of course, much more complex for a software radio than a conventional radio because of the software radio's additional capability.”

It was quite difficult to find resources that described how to best approach the separation of software components that interface hardware and implement application functionality. Most hardware vendors provide drivers with their product that are intended to be installed on to the host operating system of the radio.

From the point of view of an application developer, the literature and radio platform survey done, makes it appear that the application should likely expect to have a significant portion of the software dedicated to individual platforms that can be tailored while the rest of the application interfaces the hardware through abstractions and internal interfaces wherever possible. This way, if the application moves to a platform with dramatically different hardware, the pieces of the application that configure the hardware at a low level can be switched out.

In the JTRS family of radios, five platforms were identified for initial development. These five platforms ranged from handheld units, to vehicular mounted, to airborne vehicles, to shipboards. They ranged from ones of watts to thousands of watts in transmitter power. Some had as many as eight processors for multi-channel use. This variety in platforms greatly influenced the operating environment software as well as how waveforms would be implemented.

Kovarik (2006) studied the architecture impacts on waveform portability using the Very High Frequency (VHF) – Ultra High Frequency (UHF) Line of Sight (VULOS) waveform as a case study noting that portability remained a problem several years into development because of hardware choices. Specifically, the VULOS software which includes GPP, DSP, and FPGA software was developed for a small form-factor radio system that had all three computation

elements (CEs) built onto the same card. When VULOS was ported to another platform that had higher performance hardware, the new platform was built with the GPP on a Single Board Computer (SBC) that connected to a DSP which was on a digital modem processing card. This specific choice of architecture forced the GPP and DSP to operate in a manner that was not originally intended. If the GPP needed to raise an interrupt on the DSP, the GPP must first signal the SBC which would then signal the modem card which would then signal the DSP. This forced a significant rework in the VULOS project because these architectural dependencies were not well documented (p. 3)

Reed (2002) stated that there are four interrelated and conflicting issues that define the digital hardware composition: flexibility, modularity, scalability, and performance. “Flexibility is the ability to handle a variety of air-interfaces and protocols, even if they have yet to be defined. This means that the software radio must handle different data rates, which implies that the overall system clock rate must be adjustable.” For the purposes of this paper “air-interface” refers directly to the part of radio system that receives and transmits the radio waves and then amplifies the signal for consumption. This would normally include antennas and amplifiers. Reed (2002) also said “Modularity of radio subsystems allows easy replacement or upgrading of subsystems to take advantage of new technology. An important aspect of hardware modularity is its ramifications to software development.”

Reed (2002) says “Scalability is related to modularity. Scalability allows the radio to be enhanced to improve capability such as increasing the number of channels that a base station could handle. Again, flexible and fast I/O between modules is an important aspect that enables scalability.” Reed (2002) also said, “Performance is the last issue. Obviously this is closely tied to the other three issues. Performance may be quantified by power consumption, relative cost,

and computational capability metrics. Each of these performance metrics must be traded for each another in the overall design. Defining comprehensive fundamental performance metrics is a challenging aspect in itself.”

In summary, while the available literature on hardware limitations to portability were scarce, they did implicitly and explicitly state that device interfaces and device performance characteristics are key factors in application development. The application developer should document these in great detail so as to be able to best target the application at new platforms. The application developer must document the physical performance characteristics, constraints, and operational scenarios of the hardware to properly design the piece of the application that interfaces with the hardware and hardware drivers so that the application is maximally portable. During development of the application, any software constructs that arise as workarounds for hardware limitations and constraints should be explicitly documented and abstracted for the sake of understanding what parts of the application should change and which parts should not change if the application is moved to a platform that has different underlying hardware. Any calculations of software size and performance that are dependent on hardware metrics should be expressly documented as well so that when the application is moved, the calculations can be more readily tweaked for the next platform. The Waveform Portability Guidelines (2009) state several recommendations with respect to the impact hardware has on portability by calling out the use of Device Interface Abstractions that hide the details of the communication path between the application and the hardware. The Waveform Portability Guidelines (2009) is an authoritative resource since it was used in the acquisition and development of the JTRS radio platforms and waveforms.

Application programming interfaces.

An Application Programming Interface (API) is a public or private interface that allows the programmers of an application to use the resource of existing application. For example, Apple Inc. has published an API for programmers that want to create “apps” for the iPhone products. APIs are generally an improvement to portability as long as they are well documented and tested. The Software Communications Architecture (SCA) was used extensively in the JTRS family of radios to create a set of common interfaces which each platform would implement and each waveform would use instead of using native resources of the operating system itself or the hardware drivers. Fogarty (2004) says that cost is a major driver of development and that it should be done only once, so maximize portability. Fogarty (2004) recommends that since specialized hardware creates dependencies in the DSP, FPGA, high-speed busses, Intellectual Property (IP), and even things like multi-function displays or web browsers that application developer should use standard APIs, use hardware abstractions, and even limit the exposure to non-standard APIs. However, APIs are not a silver bullet that always simplifies software development. Rather, it shifts the effort to another specific area. Bulat (2006) showed that the Extremely High Frequency (EHF) Lite Software Defined Radio program faced significant problems integrating the SCA APIs into the modem software (p. 3). The inclusion of a new API that hadn't existed in prior versions of the EHF software required architecting new adaptors and wrappers and a new testing strategy along with constant verification and validation activities to see if the software model had changed due to using middleware services provided by the SCA API.

Modem software.

Modem software is subject to many portability risks due to how closely it operates on hardware.

The Waveform Portability Guidelines (2009) states what aspects of modem software development should be clearly avoided or documented to maximize portability of FPGA modem software. Following is a summary of those recommendations from page 22.

- Verify the maximum FPGA resource utilization at peak operational levels and sure that there is a 25% surplus of logic gates remaining.
- Abstract or eliminate the use of off-chip resources including memory, the ADC, and interfaces to the DSP and GPP.
- Do not use the primitive data types of the vendor hardware development system. If they are used, document them thoroughly.
- Retain simulations for comparison and contrast on future platforms.

Operating system software.

The host operating system on the radio significantly impacts the development of the application software due to the fact that it is the primary source of hardware control for an application and a major factor in the security concept for the entire radio. For software defined radios, there are several “real-time” operating systems (RTOS) that offer multiple levels of security, deterministic operation, and native support for many embedded types of processors and processor boards.

Green Hills (2012) stated on the company’s website that Green Hills INTEGRITY RTOS is the only solution that offers these memory protection mechanisms. According to Wikipedia (2012) there are 119 known RTOS’s available on the market today. It is not reasonable for an application developer to assume that the original target platform will always be the one chosen

and that the services of that operating system will remain the same. It is for these reasons that the dependencies and operational scenarios, or use cases, of the interactions between the operating system and the application must be known and well documented.

For further information on the above points, the reader is encouraged to read the book *Software Radio: A Modern Approach to Radio Engineering* that was written by Jeffery Reed. It was originally published in 2002 at a time when software radios were much less common and less evolved than they are today, so it was found that some information is outdated, but nonetheless informative. This book is still relevant to an individual looking to learn more about software radios due to the fact that the author chose to describe an ideal radio, using abstractions, and not a particular implementation. Additionally, many of the terms and diagrams in the book are still in use today which are evidence to its relevance. With regards to this thesis, the book is influential and supportive. Particularly the importance of software radios, the degree to which software plays a role, the degree to which software will be reused across many radio platforms, and the challenges that influence the division of requirements between hardware and software.

In the preface, the author refers to software radios as a way to “future proof” radio technology from becoming obsolete, by having many radios having the same capabilities all upgradable through flexible and scalable software. The body of the book conveys detailed information about the individual physical elements that make up a software radio, and what software can be used to drive them. In other words, it discusses the individual elements that have to be present and the challenges with integrating them physically as well as logically with software. Chapters 2-8 discuss the challenges in radio hardware selection, and inadvertently make the case for understanding portability constraints as it pertains to processing hardware. The

book describes the vast differences in hardware choices for software radios and how much harder the system engineering phase is due to the additional disciplines required to properly implement a radio in software. There are many decisions that influence the type of hardware that could be chosen for a radio. Without going into excessive details, the choice of air-interfaces, transmitter architectures, multirate signals, analog-to-digital (ADC) converters, and microprocessors each uniquely dictate how software will be a part of that radio system. If the designer of a radio application does not understand what radio platforms will potentially host the application, a lot of rework and re-engineering will be necessary to make that application operational on multiple platforms.

The other virtues of this book are that it introduces the reader to the terminology, motivations, and challenges behind software radios. The reader is not required to understand every word, either. The topics are discussed relatively independent of each other. A reader curious about DSP architectures in chapter 7 will not have to read all of chapters 1-6. The book concludes with a case study on the Joint Tactical Radio System (JTRS) which is valuable to the reader of this thesis since JTRS is a major motivation for writing this thesis guide. However, this book cannot solely support the views of this thesis. The reader looking for a deep discussion of software portability will be disappointed. The author focuses almost entirely on the part of the radio that is considered the modem. And while modem software is demonstrated to be highly susceptible to portability constraints, the author does not extend this to general purpose processor (GPP) code in the application layer.

Modern Software Design Literature

Many readers of this thesis will be very experienced with the concepts of software programming and are aware of the many forms of documentation that relate to software

development such as Unified Modeling Language (UML) diagrams. Very early in the education of a programmer the ideas of abstraction are applied to model various static and dynamic aspects of software. The remainder of this section on Modern Software Design Literature is simply to show that given the portability constraints already discussed, there is no standard way of addressing portability concerns and that some should be developed. Mooney (1997) noted that “The principal task of portable documentation is to identify and separate system-dependent and independent portions into distinct sections or separate documents. For a new implementation, ideally, only the limited system-dependent documentation needs to be redeveloped.”

Note that this discussion of UML diagrams below is not an exhaustive list. Many use the same stereotypes and visual elements. It is conceivable and probable that from the models discussed below, a new and unique type of modeling construct could be devised to specifically capture the portability details mentioned in the literature.

- Structural UML diagrams
 - Class diagram: The class diagram is good at communicating the abstract static architecture diagram of the internal structure which encompasses key Object Oriented (OO) concepts such as inheritance (Agile, 2012). These static relationship diagrams do not directly capture details related to hardware dependencies, external interfaces, and system architecture constraints, but the class diagram is supportive of porting efforts if it is done well.
 - Component diagram: The component diagram is a high level architectural diagram that is very abstracted from the implementation (Agile, 2012). For this reason, it is a supportive design artifact, but not ideal for capturing the key issues that impair portability.

- Composite structure diagram: The composite structure diagram is a tool that is used to capture the static structure of collaborations between the elements of a class. This is a valuable design tool and would be helpful to the understanding of how CORBA objects are used within the classes to achieve transparent interfacing to other components in the system.
- Deployment diagram: A deployment diagram is another high level abstraction of the system where each node may be software, hardware, or both (Agile 2012). The interfaces between the nodes are generically specified and this model is simply too highly abstracted to capture the specification detail necessary to truly improve portability.
- Object diagram: An object diagram is closely related to the class diagram and is considered to be an instance of the class diagram in that it depicts a static structural relationship between class instances at some point in time (Agile, 2012). This particular diagram could be used in to understand certain uses of multithreading, singleton patterns, and other issues that relate to the ability for a porting team to understand the design and implementation of a particular system, this particular diagram captures no detail that relate to the portability of the software.
- Package diagram: The package diagram is a construct that is useful for capturing classes that belong together and what their relationships are (Agile, 2012). And while this diagram is useful for understanding the structure of a package and the relatedness of a set of classes, it does not capture any details that would improve portability.

- Behavioral UML diagrams
 - Activity diagram: The activity diagram is used to capture use cases of the business process (Agile, 2012). This is potentially a candidate for being able to capture some types of portability issues by being explicit in the use cases where components in the waveform must interact with components in the operating system or use the CORBA framework to reach out to the hardware. Though, at this time the basic notation would require rework to match the portability concerns that would include timing issues and underlying assumptions.
 - Interaction overview diagram: An interaction diagram can be used to capture control flow (Agile, 2012). It is similar to the activity diagram. No specific details relating to portability constraints are captured here.
 - Sequence diagram: The sequence diagram is used to model the logic of a scenario in a use case. According to Agile (2012), this is typically discarded after initial use. However, like the activity diagram, this construct could be adapted to model when external interfaces impose requirements on the waveform components that are timing related.
 - State diagram: A state diagram is a very useful modeling tool for capturing the states a class can be in and the transitions used to get between these states (Agile, 2012). The state diagrams model what goes on inside a single class and because of this, they are not necessarily capturing details that relate to the portability implications discussed in the previous paragraphs.

- Timing diagram: Timing diagrams are often used in embedded software design where they can be used to explore the time dependent behaviors of a system (Agile, 2012). Timing diagrams, if targeted at the proper modules that are used to interface the GPP to the DSP, could potentially be used or adapted for portability constraints.
- Use case diagram: A use case diagram is primarily used to capture major usage requirements which are essential to the user. The use case diagram is very high level and abstracted from the implementation details and has no utility in documenting portability details.

From the information given above, it is reasonable to believe that given the large number of existing software design modeling tools and constructs, that a focused effort to extend those models to the domain of portability could yield exciting results. But, as it stands today, these models are not equipped to visually represent the interface constraints or the device-level dependencies that these abstract models attempt to stay above.

Chapter 3 – Methodology

In order to properly consider the various dimensions and aspects of software portability and how documentation is a key factor, a vigorous search was done through channels such as the Association for Computer Machinery (ACM) Digital Library, Google Scholar, Google, and in trade books on the subject of software design for references to software portability and software design documentation. The Joint Tactical Radio System (JTRS) program, a prime example of the need for software portability, had created a publicly available document titled, Waveform Portability Guidelines, that represented a significant contribution to the discussion of portability through all resources surveyed. These resources each took a narrow view of software portability from a subset of all issues pertaining to software portability. While there were few disagreements between the resources, there was very little consistently repeated advice on best practices at improving software portability.

In order to bring a consistent voice to the subject, a set of topics that are collectively seen as vital to the discussion of portability for radio application software was assembled so that a survey could be conducted among individuals that have current and related experience to software programming in radio systems.

Selecting Survey Questions

Condensing the reviewed literature provided the important topics to be discussed. Though there are potentially other subjects worthy of discussion, the questions for this survey would be the related to the most repeated topics in the current literature available at the time of this research. The survey would be as follows, and in no particular order.

- The importance of documenting the software architecture diagrams (control flow, data flow, etc) for the GPP, DSP, and FPGA.

- The importance of documenting the DSP chip architecture (floating point, integer, etc).
- The importance of documenting the FPGA interfaces to the DSP.
- The importance of documenting the Analog to Digital Converter (ADC) or Digital to Analog (DAC) performance characteristics.
- The importance of documenting the implicit and explicit dependencies between the radio application software and specific computational hardware performance characteristics.
- The importance of documenting the implicit and explicit dependencies between the radio application software and third-party or external software.
- The importance of documenting the implicit and explicit dependencies between the radio application software and the developer's build environment.
- The importance of documenting the implicit and explicit dependencies between the radio application GPP software and the operating environment such as middleware or the operating system software.
- The importance of documenting the interfaces between the radio application GPP software and hardware resources. For example, audio devices, displays, DSP and FPGA.

Identifying Survey Candidates

The nature of this research topic is such that not just anyone could provide meaningful perspective. To have a meaningful survey, candidates must have education and experience in software engineering. Ideally each candidate would have worked in a software defined radio program before. Candidates that the author identified as having valuable insights included authors of technical reference books, one university professor who teaches the subject of Software Portability, and local engineers that have had direct experience with software defined radios. If the respondents understand the survey and answer honestly, the survey should be an

excellent tool for creating a consistent voice in the subject being surveyed. The candidates selected known as a population of convenience since they were not randomly selected from a large population. Instead, they were chosen because of their known associations and abilities.

In summary, the decisions that went into selecting the survey questions, identifying potential respondents, selecting a delivery vehicle, and making the survey anonymous were impactful on the outcome and findings. The relevance of the survey is predicated on the notion that the majority of respondents would have recent experience working in a software radio development program whether it is in the development of an application or the operating system. Due to the lack of a consistent and thorough discussion available through existing literature on the topic of portability, the next best resource was to go directly to the people who would have professional experiences in that field. Thus, current employees of local businesses and government institutions known for research in communications technology were sampled through personal associations and networking opportunities.

Survey Bias

Several of the candidates work for defense contractors and government organizations. In order to maximize the size of the audience this survey could go to, the candidates were asked not to identify themselves. Instead, the candidates were asked to identify what their professional educational experiences included. Since socio-economic factors were not gathered, the possibility of a survey bias is present, though it is not well understood what that bias would be.

Survey Delivery and Collection

Appendix F has the survey instrument used in this thesis for the reader's reference. The issue of how to deliver the survey proved to be impactful as well. Initially, the idea was to deliver the survey in a Microsoft Word document format via email that included instructions and

background rationale. This method would have potentially worked, though respondents would be forced to email back the results which would have removed their privacy. The survey could be tailored infinitely to provide the best possible survey that would be the easiest to analyze statistically. Instead, a website known as Survey Monkey was chosen as a viable, though not perfect, method of delivery. Ultimately, the decision to use Survey Monkey came down to two factors: the responses are entirely anonymous, and the survey was very quick which improves participation rates.

IRB Approval

The Institutional Review Board (IRB) reviewed and approved this survey and the population to which it was targeted. Due to the nature of the questions being asked of the audience and their status as being not at risk, the survey was deemed as “Exempt” and was much less formal and less controlled than many other research projects that target human subjects. The author of this thesis was trained and certified through the courses taken at the Collaborative Institutional Training Initiative (CITI) website found at www.citiprogram.org/

Chapter 4 –Results

Each candidate was sent an email with the introduction, instructions, and web link to www.surveymonkey.com which hosted the survey. Candidates clicked on the link and immediately responded to the survey questions. After each survey was completed, Survey Monkey would tally the results. After 3 days, the vast majority of candidates had responded, though several more responded before the 10-day window expired. In all 23 individual candidates were emailed directly. The survey was also posted on Facebook’s website and sent to a group of local programmers that do application development for Google Android operating systems. As of Sept 27, 2012, 10 candidates had responded by completing the entire survey.

Survey Responses

Table 1: Survey Responses

Question	Higher Priority	Lower Priority
Rate the importance of documenting the software architecture diagrams (control flow, data flow, etc) for the GPP, DSP, and FPGA	100%	0%
Rate the importance of documenting the DSP chip architecture (floating point, integer, etc)	60%	40%
Rate the importance of documenting the FPGA interfaces to the DSP	80%	20%
Rate the importance of documenting the Analog to Digital Converter (ADC) or Digital to Analog (DAC) performance characteristics	50%	50%
Rate the importance of documenting the implicit and explicit dependencies between the radio application software and specific computational hardware performance characteristics	80%	20%

Rate the importance of documenting the implicit and explicit dependencies between the radio application software and third-party or external software	80%	20%
Rate the importance of documenting the implicit and explicit dependencies between the radio application software and the developer's build environment.	70%	30%
Rate the importance of documenting the implicit and explicit dependencies between the radio application GPP software and the operating environment such as middleware or the operating system software.	80%	20%
Rate the importance of documenting the interfaces between the radio application GPP software and hardware resources. For example, audio devices, displays, DSP and FPGA.	100%	0%

Table 2: Survey Candidate Experiences

Candidates Experiences	Percentage of Candidates Self-Reporting as Having Relevant Experience
C/C++ Programming	90%
FPGA Programming	30%
Software Defined Radio Programming	40%
Assembly Language Programming	40%

Software Design	90%
Embedded Systems Programming	50%
DSP Programming	20%
Requirements Design	50%
Master's Degree	30%
Doctoral Degree	0%

Analyzing the Responses

The goal of performing a survey in conjunction with a literature review for this thesis was to find a consistent voice in the absence of good publicly available data regarding portability constraints and methods of documenting them. Implicit to this goal is that the survey results would be analyzed for statistically significant responses whether they were to be in disagreement or in agreement. The hope is that there is indeed something in the conscious thoughts of the respondents that is common to all of them and that prioritizing the future research would be very easy given the responses of this survey. The choice to use Survey Monkey unintentionally reduced the meaningfulness of the survey. The original intent was to allow the candidates to review a list of issues relating to portability and have them rank them in order of importance. What happened instead was that the candidates were not asked to rank, instead just to state if the topics were important or not. The survey mechanism did not allow for a relative ranking. Thus, there is less variation in the data which reduces the amount of statistical significance of the responses. It was anticipated that only one or several topics would be consistently ranked high.

Instead, the majority of topics were ranked over 70%. Still, two topics were unanimously considered as highly important to the documentation for portability.

- Documenting the software architecture diagrams (control flow, data flow, etc) for the GPP, DSP, and FPGA
- Documenting the interfaces between the radio application GPP software and hardware resources.

The self-reported skills gave a much wider range of results for us to determine if the population surveyed is a valid population and perhaps what their biases might be. 90% of all respondents stated that they had professional experience with C/C++ programming. This is a dominant programming language in the field of software defined radios and elsewhere for its general purpose robustness. 90% of all respondents also stated that they also had experience in professional experience in software design. Only 40% claimed experience programming directly for a software defined radio, and only 20% claimed experience programming the DSP, and still another 30% claiming experience programming the FPGA. These figures seem to imply that the DSP programmers and the FPGA programmers are likely the 40% that programmed directly for a SDR since those types of devices are very particular to the field of SDRs. And if it is true that 90% of all respondents have experience in C/C++, then the FPGA, DSP, and SDR programmers are all likely C/C++ programmers as well.

Two questions that perhaps could have been asked differently were the questions regarding experience with SDRs and experience with embedded systems. Since many SDRs are by definition an embedded system, the same people likely responded to both and thus one question was wasted. We cannot be for sure since Survey Monkey only aggregates responses and the correlations between experience and responses cannot be drawn directly.

It is the opinion of this author that the responses of the survey and the literature review are adequate for establishing some truths about the future of documentation for software portability. Another opinion of this author is that in the future, surveys should select participants that have embedded systems programming experience, FPGA systems experience, and SDR experience. If only these characteristics are targeted, the other qualities such as C/C++ programming, assembly programming, and requirements design will come automatically.

Chapter 5 – Lessons Learned, Future Research, and Conclusions

The outcome of this research was very close to what was targeted. The literature reviewed provided enough information to lay a foundation of what is known about software portability for software defined radios. The literature reviewed showed that the topic is not widely discussed in common terms in the same way that other software development principles are being discussed. Thus, one outcome had been achieved which was to validate that this research on software portability was necessary and relevant. Another important outcome was that there was enough information in literature to conduct a survey which would elevate the topics which were most relevant to the subject of portability for future research. The literature review should stand as a useful beginning to any further research done on this topic. A minor, yet notable, outcome was that the survey responses indirectly supported the literature reviews which validates that the literature itself is valid. Considering that it is unlikely the survey respondents were familiar with all the literature reviewed for this research and still agreed in large part with one another provides a measure of significance that the correct survey population was targeted. If the population was improperly selected, it is likely that the results would be less unanimous. According to the survey, only 40% of respondents had worked directly on programming a software defined radio, and it would be optimistic to assume that even all of them had direct knowledge of all the literature reviewed.

Finally, it is the conclusion of this author that the topic of portability should be studied in greater detail in the areas which were unanimously agreed upon by literature and survey candidates. Further research could be targeted at the mechanics of improving the documentation of portability constraints. A study could be done to use UML-styled diagrams to document portability constraints, then test them on various applications or use them as survey questions.

Another aspect of study would be to create “metrics of portability” to try and quantify and measure the portability of an application for a platform. This research could also be redone in whole with a different survey methodology and population to confirm these results.

Lessons Learned

Performing a survey provided the experience to learn several valuable lessons to be taken forward in future research efforts. After reviewing the survey responses, one lesson learned is that the survey being re-done in a different format would allow for a deeper understanding of the relevant discussions in software portability. If survey respondents had been able to respond to the questions differently (i.e., rank the choices), then future research would be already more targeted at the correct areas. Also, as Chapter 4 indicated, future surveys could put a premium on targeting a population of individuals with embedded programming experience, and software defined radio experience as these individuals have a great tendency to have the other underlying skill sets in C/C++, FPGA programming, and assembly language. These are very critical areas of expertise since it is in these skill areas where developers have to work the hardest to integrate the software and hardware.

Future Research

It is the opinion of this author that the stage is very well set for follow on research that was not within the scope of this thesis to truly revolutionize the area of documentation for portability. The literature review and survey sufficiently scoped the research down from a very broad set of portability constraints to a few important constraints as well as a set of standard UML diagrams that could be leveraged to model the software behaviors and hardware constraints that lead to portability problems.

One specific documentation strategy identified by the survey was that software architecture diagrams (control flow, data flow, etc) for the GPP, DSP, and FPGA should be captured as a standard practice for portability. But learning to apply this modeling to the components of the software which are impacted most will be an important area of exploration. Another specific documentation strategy identified by the survey was to document the interfaces between the radio application GPP software and hardware resources. For example, audio devices, displays, DSP and FPGA. Learning to organize the behavioral and structural relationships between the GPP software and hardware resources into visual diagrams is an important area of exploration.

The subsequent stages of research would be to prove or test the proposed diagrams in a real world porting effort and keep track of the issues that arise when porting a piece of software from one platform to the other.

Appendix A - References

- Agile Modeling. (2012). Artifacts for Agile Modeling: The UML and Beyond. Retrieved from <http://www.agilemodeling.com/essays/modelingTechniques.htm>
- Bulat, N., Dyer, Scott., Zhang, Y. (2006). *Strategies and insights into SCA-compliant waveform application development* [PDF document]. Retrieved from http://www.mitre.org/work/tech_papers/tech_papers_06/06_0825/06_0825.pdf
- Defense Acquisition University. (2012). Acquisition life cycle. Retrieved from <https://dap.dau.mil/acquimedia/Pages/ArticleDetails.aspx?aid=9c591ad6-8f69-49dd-a61d-4096e7b3086c>
- Defense Acquisition University. (2012). Integrated life cycle chart. Retrieved from <https://ilc.dau.mil/>
- Defense Information Systems Agency. (1994). Software design description [PDF document]. Retrieved from https://assist.daps.dla.mil/quicksearch/basic_profile.cfm?ident_number=205915
- Ettus Radios. (2012). USRP Radios. Retrieved from <http://www.ettus.com/>
- Fogarty, G. (2004) *Standardizing on a set of radio set APIs to ensure waveform portability* [PDF document]. Retrieved from http://www.omg.org/news/meetings/workshops/SBC_2004_Manual/03-4_Fogarty.pdf.
- General Dynamics. (2012). Digital Modular Radio. Retrieved from <http://www.gdc4s.com/dmr>
- Green Hills. (2012). INTEGRITY Real-Time Operating System. Retrieved from <http://www.ghs.com/products/rtos/integrity.html>.
- Harris Radios HMS. (2012). Handheld Manpack Small Form Fit. Retrieved from <http://www.gdc4s.com/jtrshms?taxonomyCat=396>

- Harris Radios AN-PRC-152. (2012). Type-1 Handheld Multiband Radio. Retrieved from <http://rf.harris.com/capabilities/tactical-radios-networking/an-prc-152/>
- Kovarik, V. J. (2006). SDR architecture impacts on waveform portability and cost modeling [PDF document]. Retrieved from <http://data.memberclicks.com/site/sdf/sdr06-4.6-3.pdf>
- List of real-time operating systems. (2012). Retrieved from http://en.wikipedia.org/wiki/List_of_real-time_operating_systems
- Mooney, J. (1997). Bringing Portability to the Software Process. Retrieved from <http://www.cs.wvu.edu/~jdmooney/classes/cs533/notes/refs/tr-sproc.html>
- Network Enterprise Domain. (2009). Network Enterprise Domain (NED) Test & Evaluation (T&E) Waveform Portability Guidelines [PDF document]. Retrieved from http://www.public.navy.mil/jpeojtrs/sca/Documents/20091228_1.2.1_NEDTE_PORT_GUIDE.pdf
- Object Management Group. (n.d.) CORBA. Retrieved from <http://www.corba.org>
- Ossie. (2012). SCA-Based Open Source Software Defined Radio. Retrieved from <http://ossie.wireless.vt.edu/>
- Pucker, L. (2007). Component-based development of radio systems and subsystems: Are we there yet? [PDF document]. Retrieved from www.spectrumsignal.com/publications/Component-based_Radio_Systems_Are_We_There_Yet.pdf.
- Reed, J. (2002). Software radio: A modern approach to radio engineering. Upper Saddle, NJ. Prentice Hall.
- Rodriguez, A. (2005). Achieve true waveform portability in SDRs [PDF document]. Retrieved from http://www.eetindia.co.in/ARTICLES/2005NOV/B/2005NOV01_RFD_TA.pdf

Thales. (2012). <http://www.thalescomminc.com/media/Thales%20ANPRC-154%20Rifleman%20Radio-v1.pdf>

Wind River. (2012). VxWorks RTOS That powers more than 1 billion embedded systems around the globe. Retrieved from <http://www.windriver.com/products/vxworks/>.

Wireless Innovation Forum. (2012) SDR Rate of Adoption. . Retrieved November 27, 2012.
http://www.wirelessinnovation.org/sdr_rate_of_adoption

Wireless Innovation Forum. (2011). What is Software Defined Radio [PDF document]. Retrieved November 29, 2012.

<http://www.wirelessinnovation.org/assets/documents/SoftwareDefinedRadio.pdf>

Wolf, W. (2006). High-performance embedded computing. San Francisco, CA. Morgan Kaufmann.

Appendix B - Annotated Bibliography

Agile Modeling. (2012). Artifacts for Agile Modeling: The UML and Beyond. Retrieved from

<http://www.agilemodeling.com/essays/modelingTechniques.htm>

The Agile Modeling page provides a quick description and comparison of UML artifacts that are commonly used when developing business models. This page describes the UML artifacts by giving the reader the common names, the common applications, misapplications, and more. The information is neatly organized into a table that also gives the reader valuable references to go further in his/her studies. A criticism of this resource is that it does not provide any examples, however that was not the intent of the authors. A particularly valuable insight that this reference provides to the public was a rating of how likely that a particular construct would be kept as valuable for future efforts. Developers should keep this thought in mind when choosing constructs that will have continuing value. This reference is valuable to the thesis because it is shown how the common diagrams are not finely tuned to handle the challenges of portability as they are currently understood.

Bulat, N., Dyer, Scott., Zhang, Y. (2006). Strategies and insights into SCA-compliant waveform application development [PDF document]. Retrieved from

http://www.mitre.org/work/tech_papers/tech_papers_06/06_0825/06_0825.pdf

Strategies and Insights into SCA-Compliant Waveforms is a resource for the developer of an SCA-Compliant waveform that uses CORBA as middleware interface between radio components and the platform. EHF Lite is used as a case study for the writers who port the waveform between two very different hardware platforms. The exercise offered insights into the types of complexities and dependencies that exist even when “standard

APIs” exist to minimize porting efforts. This resource validates the notion that APIs are not a solution to porting problems and that great care must be taken especially when a waveform has multiple computational elements such as a GPP, DSP, and FPGA. This resource is a valuable and unique case study that has not peer in the research done for this thesis. And while the conclusions of the paper are not necessarily solutions, the data in it correlates well to the observations in other unrelated literature regarding the challenges of portability and the role of the SCA.

Defense Acquisition University. (2012). Acquisition life cycle. Retrieved from

<https://dap.dau.mil/acquipedia/Pages/ArticleDetails.aspx?aid=9c591ad6-8f69-49dd-a61d-4096e7b3086c>

The DAU Acquisition lifecycle prescribes a Systems Engineering Process (SEP). The SEP uses discrete steps to structure requirements and functionality of a system in a logical fashion. The SEP assumes a “top down” development process that begins with Requirements Development. This technical process, while valid, is currently absent of the discussion of portability or how the impact of multiple targets imposes new technical challenges. This particular resource is simply evidence of the need for processes that explicitly account for the future of the software. Outside of this research, this SEP is in fact the Department of Defense guideline for technical development of most systems including software defined radios.

Defense Information Systems Agency. (1994). Software design description [PDF document].

Retrieved from

https://assist.daps.dla.mil/quicksearch/basic_profile.cfm?ident_number=205915

The Software Design Description (SDD) is a template for vendors must use when documenting the design of a software based system such as the JTRS waveforms and radios. This document is generated as a part of the Systems Engineering Process (SEP). The template, while valid, leaves the vendor with very few specific criteria for how to express portability constraints. Thus, vendors may provide a wide variety of descriptions which are not necessarily fitting to the situation. This resource is further indication that neither policy documents nor detailed descriptions of design documents in military systems approach portability deliberately even when the goal of the system is to be portable. This is a lacking that should be addressed.

Ettus Radios. (2012). USRP Radios. Retrieved from <http://www.ettus.com/>

Ettus Research provides a number of solutions for researchers and practitioners of low-cost Software Defined Radios (SDRs). The primary product offered is the Universal Software Radio Peripheral (USRP) that allows researchers to develop software on a standard PC that commands the radio through an attached USB port. The family of radios Ettus offers to the public are simple radios that offer one type of computational element, a general purpose processor, one Ethernet port, and no embedded operating system or cryptography. Developers on the Ettus radio do not program to a specific set of standards that are applicable to industry, so while the general level of exposure to the field of software defined radios is improving, standards for porting and managing embedded systems is not improving at that same rate. The Ettus radio is a widely used experimental radio and is valid as well as valuable for this type of research.

Fogarty, G. (2004) Standardizing on a set of radio set APIs to ensure waveform portability [PDF document]. Retrieved from

http://www.omg.org/news/meetings/workshops/SBC_2004_Manual/03-4_Fogarty.pdf

Fogarty briefly describes that the current state of the SCA and the future focus of the SCA may vary dramatically and that currently it is inadequate for truly improving portability because the wide variety of SCA interfaces that are available to all waveforms and platforms are not guaranteed to be in perfect sync. In other words, the waveform may be developed with certain SCA APIs to access the hardware of the original target platform, but the next target platform may not implement all of the same SCA APIs or have the same radio services. API standardization and finding the “best” SCA APIs is going to be a slow progression for developers. This resource has a particular importance to it because Fogarty is a Boeing engineer that was directly involved in the early work done with Boeing on the Army’s JTRS radios. The subject matter fits well with the other resources and while it does add the unique insight that not all radios will develop all possible radio services, the paper does not conflict with other points of view found in the literature used for this thesis. For these reasons, the resource seems applicable and a worthy inclusion.

General Dynamics. (2012). Digital Modular Radio. Retrieved from <http://www.gdc4s.com/dmr>

This resource is a data sheet for the Digital Modular Radio (DMR) that briefly discusses the publicly available information about its capabilities and components. This radio is a software defined radio that was developed before the JTRS program and before the SCA APIs were developed. As such, the waveforms that operated on the early versions of the DMR did not port easily to the future JTRS platforms. This resource is very valuable as it

comes directly from the vendor of the DMR and was useful in gathering specific information about existing SDRs for this thesis.

Green Hills. (2012). INTEGRITY Real-Time Operating System. Retrieved from

<http://www.ghs.com/products/rtos/integrity.html>

This resource is a data sheet for the Green Hills INTEGRITY operating system that discusses its publicly known capabilities and features that are unique to this operating system. This is a commonly used operating system in embedded real time systems. The unique features offered by this operating system were researched as this implies that any waveform which makes use of those features will then be tightly coupled to that operating system regardless of whether or not standard SCA APIs are used to access those features.

Harris Radios HMS. (2012). Handheld Manpack Small Form Fit. Retrieved from

<http://www.gdc4s.com/jtrshms?taxonomyCat=396>

This resource discusses the publicly known features and capabilities of the Harris HMS radio. This radio is one of the JTRS platforms. This resource comes directly from the developer and vendor of the Harris HMS radio and is valuable as a resource to glean what capabilities and components are in the HMS for comparison to other radios such as the DMR and PRC-152.

Harris Radios AN-PRC-152. (2012). Type-1 Handheld Multiband Radio. Retrieved from

<http://rf.harris.com/capabilities/tactical-radios-networking/an-prc-152>

This resource discusses the publicly known features and capabilities of the Type-1 Handheld multiband radio. This radio has a hardware based encryption system built in and is a JTRS radio. This resource comes directly from the vendor of the Harris PRC-152

and is a valuable resource for comparing its components and features to other software defined radios such as the DMR and HMS.

Kovarik, V. J. (2006). SDR architecture impacts on waveform portability and cost modeling

[PDF document]. Retrieved from <http://data.memberclicks.com/site/sdf/sdr06-4.6-3.pdf>

This important paper is a direct discussion of many of the important topics of this thesis.

The paper provides discussion that the original assumption by JTRS that the SCA compliant waveforms would port easily to many SCA compliant radios is not as simple as it sounded for a variety of reasons. Ultimately, the underlying nature of radio hardware is so variable from one platform to the next that it is impossible to be certain when developing a waveform that the underlying hardware will have similar behaviors even if it has similar APIs. This paper is from an SDR technical conference and Kovarik is an engineer for Harris which is a major vendor of software defined radio products for the military.

List of real-time operating systems. (2012). Retrieved from

http://en.wikipedia.org/wiki/List_of_real-time_operating_systems

This resource is simply to illustrate the vast number of real-time operating systems that may potentially be used in the development of a portable communication device. It is not reasonable to assume that any set of APIs would be adequate for overcoming the differences in platform functionality and behavior. Dependencies on the operating systems should be expressed deliberately.

Mooney, J. (1997). Bringing Portability to the Software Process. Retrieved from

<http://www.cs.wvu.edu/~jdmooney/classes/cs533/notes/refs/tr-sproc.html>

Unlike other resources used in this document, this is not focused on real-time operating systems, software defined radios, or APIs. It is a general discussion on portability that states the importance for portability to be considered explicitly during development and not to confuse the subject of portability with software reuse or to view it as simply an implementation detail. Mooney states that a study was performed which examined a wide range of software design methods. None of them discussed portability directly. His paper serves as an expert's opinion that not only is portability misunderstood or poorly understood, but that it is not well studied or appreciated in modern software engineering circles. Mooney provides a framework for thinking about how to include portability in the development lifecycle by focusing on controlling interfaces, isolating dependencies, and constantly assessing the portability of the software. This thesis is in harmony with Mooney's views and it is recommended by this author that the reader thoroughly understand Mooney's arguments. This resource has no peer in any of the literature surveyed for making specific recommendations to improve portability through documentation, design, and discussion.

Object Management Group. (n.d.) CORBA. Retrieved from <http://www.corba.org>

The Object Management Group (OMG) is the parent organization that specifies the Common Object Broker Request Architecture (CORBA) specification which has been widely adopted in industry and the military. This resource is authoritative, and offers excellent tutorials for a soft introduction to the concepts behind middleware and the CORBA specification.

Ossie. (2012). SCA-Based Open Source Software Defined Radio. Retrieved from <http://ossie.wireless.vt.edu>

The Ossie radio is a unique open-source software defined radio platform used by schools and researchers to further the study of SDRs. It is widely used, can be used on the Ettus radio, and is a worthy inclusion for this thesis.

Pucker, L. (2007). Component-based development of radio systems and subsystems: Are we there yet? [PDF document]. Retrieved from

www.spectrumsignal.com/publications/Component-based_Radio_Systems_Are_We_There_Yet.pdf

This particular resource is unique to this thesis in that it reviews some current trends in the area of bringing portable application components to Digital Signal Processors (DSPs) in SDRs. Pucker states that there has been significant progress in private industry to use four technologies in concert to remove certain portability barriers. Pucker states that a common modeling language for DSP components, standard functional blocks, common hardware abstraction, and standard application framework will address many of the dependencies that inhibit reuse. Pucker's statements seem central to the idea of software reuse in the DSP and this is the only resource that suggests a common modeling language is a solution to one of the portability problems. No other resource identified design modeling as an area that is a challenge to portability and reuse. Pucker's other assertions are more in line with what the rest of the literature tends to agree on. SCA frameworks, hardware abstraction, and commonly reusable blocks of software will improve portability and reuse.

Reed, J. (2002). Software radio: A modern approach to radio engineering. Upper Saddle, NJ. Prentice Hall.

This resource is unique to this thesis in that it was the only book found that directly discussed the design challenges of software defined radios. Reed goes into great detail about the entire design process, design goals, and design artifacts of a software defined radio. His book stands alone and peerless in describing the hardware choices that impact portability.

Rodriguez, A. (2005). Achieve true waveform portability in SDRs [PDF document]. Retrieved from http://www.eetindia.co.in/ARTICLES/2005NOV/B/2005NOV01_RFD_TA.pdf

Thales. (2012). <http://www.thalescomminc.com/media/Thales%20ANPRC-154%20Rifleman%20Radio-v1.pdf>

This resource is the vendor's webpage for the Rifleman radio that is a software defined radio in the JTRS family of radios. It is a valuable resource in the comparison of existing radios such as DMR, HMS, and the PRC-152.

Wind River. (2012). VxWorks RTOS That powers more than 1 billion embedded systems around the globe. Retrieved from <http://www.windriver.com/products/vxworks/>

The VxWorks real time operating system is a competitor to the Green Hills INTEGRITY operating system that is used on software defined radios in private and military applications.

Wireless Innovation Forum. (2012) SDR Rate of Adoption. . Retrieved November 27, 2012. http://www.wirelessinnovation.org/sdr_rate_of_adoption

The Wireless Innovation Forum is an active body of SDR experts and enthusiasts that are advocates for SDR technology. This body was previously known as the SDR Forum. They organize events, webinars, tutorials, and challenges that keep the body of interested parties well connected. This is a valuable resource for any enthusiast.

Wireless Innovation Forum. (2011). What is Software Defined Radio [PDF document].

Retrieved November 29, 2012.

<http://www.wirelessinnovation.org/assets/documents/SoftwareDefinedRadio.pdf>

The Wireless Innovation Forum is an active body of SDR experts and enthusiasts that are advocates for SDR technology. This body was previously known as the SDR Forum.

They organize events, webinars, tutorials, and challenges that keep the body of interested parties well connected. This is a valuable resource for any enthusiast.

Wolf, W. (2006). High-performance embedded computing. San Francisco, CA. Morgan Kaufmann.

Kauffman presents a detailed analysis of the hardware architecture choices that must be made for embedded systems. This book is not intended to illuminate issues regarding software defined radios, and because of that reason it is not heavily referenced. But, due to its thorough treatment of architectural issues such as memory management, chip architecture it must be included in this study.

Appendix C – Survey of SDR Platforms

This survey is a compilation of SDRs that are in development for commercial, scientific, and military use. This survey is specifically inclusive of SDRs that are available today. Specifically excluded from this survey are platforms that have been cancelled or are no longer available, such as the JTRS Ground Mobile Radio which was cancelled prior to delivery in October 2011. The intent of this survey is to aggregate and organize important architectural details about modern SDRs. The data gathered in this survey shows clearly that application developers have a very wide range of target platforms which have different form factors, operating systems, hardware resources, and APIs.

Platform	Open Source	OS	Form	SCA	Uses CORBA	Provides HMI	Provides Dev Kit	Frequency Range	MHAL	Application
Handheld Manpack Small Form-fit (HMS)	Partial	Green Hills Integrity	Handheld	Yes	Yes	Yes	No	225- 450MHz, 1250- 1390MHz, 1750- 1850MHz	Yes	Military
Digital Modular Radio (DMR)	Partial	Green Hills Integrity	Shipboard, Rack- mounted	No	Yes	Yes	No	2MHz- 2GHz	Yes	Military
AN-PRC-152	Partial	Unknown	Handheld	Yes	Yes	Yes	No	30- 512MHz	Yes	Military
Ettus Radio	Yes	None	Small desktop package	No	No	No	Yes	Varies by model	No	Scientific

GNU Radio	Yes	None	None, Software only	No	No	No	Yes	Not Specified	No	Scientific
4DSP	No	None	Small desktop package	No	No	No	Yes	Varies by model	No	Scientific
Ossie	Yes	None	None, Software Only	Yes	Yes	No	Yes	Not Specified	No	Scientific

Table 3: Survey of Existing Software Defined Radios

Appendix D — Simplified JTRS SDR Architecture

The Joint Tactical Radio System (JTRS) is a Department of Defense program that is approached as a joint-project between all branches of the military to produce a family of radios that are able to host multiple waveforms of the same type for each service. This ultimately saves on cost if the waveforms are developed once and run on many platforms in the JTRS family of radios.

JTRS Platforms

Originally, the JTRS program office identified five distinct platforms, also known as form factors, which would be fielded for the Army, Navy, Air Force, and Marines. These five platforms were termed “Clusters”.

- Cluster 1: A joint-service project lead by the US Army to “established to provide the warfighter with a multi-channel software programmable, hardware-configurable digital radio networking system. Cluster 1 was to support requirements from the Army Aviation Rotary Wing, Air Force Tactical Control Party (TACP), and Army and USMC Ground Vehicular platforms.”

http://www.globalsecurity.org/military/systems/ground/jtrs_cluster1.htm

- Cluster 2: A joint-service project lead by the Special Operations Command (SOCOM) “the interim JTRS handheld solution known as the Multiband Intra-Team Handheld Radio (MBITR). This program was subsequently renamed Joint Tactical Radio System Enhanced MBITR (JTRS JEM). Cluster 5 (subsequently JTRS Handheld, Manpack, Small Form Fit or JTRS HMS) was to be the final JTRS handheld solution. JTRS JEM was being developed as an Engineering Change Proposal to the MBITR program. It would provide a nominal JTRS compliant capability to the joint warfighter beginning

with a production decision during third quarter FY05”.

(http://www.globalsecurity.org/military/systems/ground/jtrs_cluster2.htm)

- Cluster 3: A joint-service project lead by the US Navy. “The JTRS program was developing software-defined radios that would interoperate with existing radios and also increase communications and networking capabilities. A Joint Program Executive Office provided a central acquisition authority and balanced acquisition actions across the services. Program/product offices were developing radio hardware and software for users with similar requirements.”

(http://www.globalsecurity.org/military/systems/ship/systems/jtrs_cluster3.htm). This project was restructured to be combined with Cluster 4 due to similarities in their requirements.

- Cluster 4: A joint-service project lead by the US Air Force. “Design a family of software-reprogrammable, multi-band/multi-mode airborne radios meeting user's needs as defined in the JTRS ORD, as well as platform-specific needs.”

(<http://www.globalsecurity.org/military/systems/aircraft/systems/jtrs.htm>). This cluster was joined with Cluster 3 due to similarities in their requirements.

- Cluster 5: A joint-service project lead by the US Army now called the Handheld Manpack Small Form Fit (HMS). “Cluster 5 was initially to oversee acquisition development and production of JTRS handheld and manpack units and forms suitable for embedment into platforms requiring a Small Form Fit (SFF) radio. Cluster 5 will include several variants of the Small Form Fit radio and two versions of the handheld radio - a single channel model and a two-channel model (with the objective of producing a three-channel version). The manpack radio will have two configurable channels (with an

objective of four configurable channels).”

(http://www.globalsecurity.org/military/systems/ground/jtrs_cluster5.htm)

JTRS Waveforms

At the same time as procuring these five platforms, the JTRS program office also began development of multiple waveforms that would be hosted on these platforms. The waveforms originally under procurement for JTRS were

- Soldier Radio Waveform (SRW)
- Single Channel Ground Air Radio System (SINCGARS)
- HAVE QUICK II
- Mobile User Objective System (MUOS)
- Enhanced Position Location Reporting System (EPLRS)
- Wideband Networking Waveform (WNW)
- Link-4A
- Link-11B
- Link-16
- Link-22
- Very High Frequency (VHF)-Amplitude Modulation (AM)
- High Frequency (HF)
- Very High Frequency / Ultra High Frequency Land Mobile Radio

JTRS Platform Abstraction

The following diagram taken from the Network Enterprise Domain (NED) Test & Evaluation (T&E) Waveform Portability Guidelines (Network, 2009) describes a generic abstraction of hardware, software, standards, and logical connections inside a JTR Set. In

general, JTRS waveforms have components running on each of the computational elements of the platform which could include multiple General Purpose Processors (GPP), Digital Signal Processors (DSP), and Field Programmable Gate Arrays (FPGA).

- Each GPP could be an Intel or PowerPC architecture integrated circuit that hosts the operating system (OS) and the source code that implements the JTRS waveform Human Machine Interface (HMI), the functional logic of the waveform program which processes voice and data for the application.
- Each DSP has specific signal processing algorithms that are particular to the waveform. Its interfaces to the FPGA and GPP are of particular interest to portability since they are potentially going to be influenced by proprietary standards.
- The SCA Device Abstraction is a software interface between the high level waveform application components and the hardware resources of the radio such as the microphone, Ethernet ports, etc.
- The Modem Hardware Abstraction Layer (MHAL) API is the set of software interfaces between the SCA Device and the OS that abstract the modem hardware resources. Logically, the MHAL API is implemented by the platform which is the host for the modem hardware. The MHAL API is an attempt to provide standardized interfacing between the high level waveform components and the modem without the application developer having to be responsible for the low level implementation details of the platform's modem.
- The FPGA is a physical device that also has waveform components on it that implement the specific Analog-to-Digital Converters (ADC) and Digital-to-Analog Converters (DAC).

For further readings dedicated to the discussion of the JTRS program, objectives, performance hurdles, success, and transition to the Joint Tactical Networking Center (JTNC) please refer to the JTNC homepage at <http://jtnc.mil/Pages/Home.aspx>.

Appendix E - Glossary

Application Programming Interface	An Application Programming Interface (API) is a specified set of interfaces used by software modules to communicate with each other. An API typically fully specifies class definitions and function signatures which would include an argument list and return types.
Application Specific Integrated Circuit	An Application Specific Integrated Circuit (ASIC) is an integrated circuit that has been designed to efficiently perform a narrow range of functions. An ASIC is generally expensive to initially design and is then cheap to make in bulk. An ASIC is powerful and fast for a few applications, but inflexible to change.
Digital Signal Processor	A Digital Signal Processor (DSP) is a specialized microprocessor with additional hardware that makes it appropriate for handling the high-speed parallel computations needed for processing signals.
Field Programmable Gate Array	A Field Programmable Gate Array (FPGA) is a configurable or programmable logic that can be programmed and re-programmed to be a microprocessor such as an ASIC, DSP, or GPP. An FPGA has no default architecture or instruction set. It is programmed by the user to be the type of processor that the user needs.
General Purpose Processor	A General Purpose Processor (GPP) is any kind of RISC or CISC processor that is not an ASIC or special purpose chip such as a DSP. Modern GPPs are characterized by high clock frequencies, variable length instructions, multiple layers of on-chip cache, and multi-core architectures.
Joint Tactical	Joint Tactical Radio System (JTRS) is a Department of Defense (DoD)

Radio System	initiative that was intended to design and procure a family of networkable and advanced wireless systems that supported all nodes of the tactical network from fast moving airplanes to mobile infantry to shore installations. JTRS has been officially cancelled and renamed the Joint Tactical Networking Center (JTNC).
Modem Hardware Abstraction Layer	The open specification of standard software interfaces in modems to facilitate software reuse and portability.
Middleware	Software that serves as a translator and communication channel between other pieces of software. The connections can be over a network and connect software running on different platforms. Common Object Request Broker Architecture (CORBA) software is an example of middleware that connects applications to the operating system and services.
Common Object Request Broker Architecture	Common Object Request Broker Architecture is a vendor neutral middleware specification that allows vendors of middleware to design and develop standard middleware software for connecting applications residing on a network.
Operating Environment	Operating Environment (OE) refers to the total collection of software resources provided by the platform including the OS, any middleware, any APIs, any utilities, etc.
Operating System	Operating System (OS) refers to a vendor specific computer operating system which may be embedded, real-time, or not. An operating system provides the device drivers, security mechanisms, user interfaces, and hardware abstractions for a hardware platform. Examples of operating

	systems include Green Hills INTEGRITY, Windriver VxWorks, Microsoft Windows, Red Hat Linux, etc.
Platform	For the purposes of this paper, a platform is synonymous with the hardware and operating environment software of a radio.
Software Communications Architecture	The Software Communications Architecture (SCA) is an open framework that is designed to standardize some of the common interactions in an portable, modern communications device. If widely adopted and adhered to, the cost of application development is reduced on an enterprise level such as for JTRS.
Radio	For the purposes of this paper, a radio is any device that is capable of transmitting and/or receiving Radio Frequency energy for the sake of connecting two or more nodes in a radio network.
Software Defined Radio	For the purposes of this paper, a Software Defined Radio (SDR) is any radio that is substantially configured and programmed through software and powered by modern digital hardware such as GPPs, DSPs, and FPGAs.
Waveform Application	For the purposes of this paper, the term “waveform”, “radio application”, and “waveform application”, will refer to the software on a radio that defines the communication modes, functions, and protocols of the radio. The waveform is an application that runs on top of the OE which provides the hardware resources, hardware abstractions, and security mechanisms for the device.

Appendix F – Survey Instrument

This appendix serves the reader by displaying the survey that was offered to a limited number of engineers in a variety of organizations both governmental and private who have operated on software defined radio systems. Though, this was a public survey and absolutely anyone with access to this link was able to take the test. This survey was anonymous and was conducted at www.surveymonkey.com. If the candidate chose to participate in the survey, he/she needed only navigate a web-connected browser to Survey Monkey where the candidate would be offered 10 questions. Nine questions related to the prioritization of known portability constraints. The 10th question asked for non-identifiable information from the respondent about his/her technical experience.

This survey instrument and selected population received Institutional Review Board (IRB) approval as well as the approval of the advisor before any of the population was sampled.

Portability Questions

- Rate the importance of documenting the software architecture diagrams (control flow, data flow, etc) for the GPP, DSP, and FPGA.
- Rate the importance of documenting the DSP chip architecture (floating point, integer, etc)
- Rate the importance of documenting the FPGA interfaces to the DSP
- Rate the importance of documenting the Analog to Digital Converter (ADC) or Digital to Analog (DAC) performance characteristics
- Rate the importance of documenting the implicit and explicit dependencies between the radio application software and specific computational hardware performance characteristics

- Rate the importance of documenting the implicit and explicit dependencies between the radio application software and third-party or external software
- Rate the importance of documenting the implicit and explicit dependencies between the radio application software and the developer's build environment.
- Rate the importance of documenting the implicit and explicit dependencies between the radio application GPP software and the operating environment such as middleware or the operating system software.
- Rate the importance of documenting the interfaces between the radio application GPP software and hardware resources. For example, audio devices, displays, DSP and FPGA.

Candidates Experiences

- C/C++ Programming
- FPGA Programming
- Software Defined Radio Programming
- Assembly Language Programming
- Software Design
- Embedded Systems Programming
- DSP Programming
- Requirements Design
- Master's Degree
- Doctoral Degree