WHITE PAPER

THE COMPELLING CASE

FOR

OPEN SOURCE EMBEDDED TOOLS

By

Steven L. Rodgers
James B. Calvin, Jr.

8 December 2002

EXECUTIVE SUMMARY

This paper presents a case that mature, reliable Open Source Software (OSS) embedded development tools, popularly know as GNU Tools, are ready to be widely adopted as a business strategy and a solution to software tools obsolescence, vendor lock-in, and vanishing manufacturing sources as applied to embedded systems in complex, long lifecycle embedded computer systems that are typical of high-end commercial Original Equipment Manufacturer (OEM) and Department of Defense (DoD) systems.

It is now common knowledge that maintenance costs increasingly dominate the overall program lifecycle cost. They can be as much as 60 to 80 percent of the total lifecycle cost. Moreover, software tools obsolescence has long been a major problem and cost driver with these systems, and due to increased systems lifetimes and lifecycle extensions, the problems are only getting worse.

On the hardware systems side, OEMs and the DoD have long recognized problems with using proprietary systems and the DoD has responded with policy guidance promoting 'Open Systems' in acquisition programs as a method of improving weapon systems management and lowering costs. Current DoD policy for Open Systems is expressed in DoD 5000.2R and the DoD Joint Technical Architecture (JTA). The JTA lists the minimum set non-governmental standards needed to maximize interoperability and affordability within DoD. In sum, the JTA is a single unifying technical architecture for integrating all DoD system components.

Software development tools are one of those components, and this paper presents a case that OSS is an evolutionary extension to the Open Systems concept in that an entire software design is exposed to open review, collaboration, and extension rather than being limited to standards for functionality and interfaces. Widespread adoption of OSS as an extension of Open Systems management, acquisition, and development methodology would bring supportability, interoperability, and reuse for large software intensive systems to the next level of maturity.

The key benefits of OSS tools are cost reduction, tools obsolescence abatement, vendor lock-in prevention, and offerings from multiple sources. Because program costs and obsolescence issues are major drivers in long lifecycle projects, the authors believe after reviewing this White Paper, it will become evident to software managers and developers that there is a compelling case for Open Source Embedded Tools.

I. BACKGROUND

Complex systems lifecycle support is complicated by software tools obsolescence: Today, the role of embedded computer systems continues to become more dominant in all electronic systems, devices, and appliances. Nowhere is this truer than in the complex world of software intensive operational systems. Furthermore, it is now common knowledge that maintenance costs increasingly dominate the overall lifecycle cost. They can be as much as 60 to 80 percent of the total lifecycle cost. [Ref. 4] Moreover, software tools obsolescence has long been a major problem with these systems, and due to increased systems lifetimes and life-cycle extensions, the problems are only getting worse. There are many examples of high-end OEM and military systems that within their lifespan have encompassed as many as four generations of computer technology. During this period, software development tools and languages have experienced similar evolutionary changes. An interesting contrast is, that while developers and maintainers of long lifecycle systems are seeking more reliability, maturity, and stability in software support systems, the Commercial Off-The-Shelf (COTS) software technology providers are driving products to ever-shorter lifecycles. This phenomenon is exasperating to managers, developers and maintainers of software intensive systems. Historically, this lifecycle support problem has been demonstrated by the fact that many embedded software systems usually outlive the software tools used to create and maintain them.

Open Systems methodology is recognized as beneficial and mandated solution approach: The DoD has long recognized these problems and has responded with policy guidance promoting 'Open Systems' in acquisition programs as a method of improving weapon systems management and lowering costs. According to Hanratty et al, Open Systems are, "systems that can be supported by the market place, rather than by a single (or limited) set of suppliers due to the unique or proprietary aspects of the design." [Ref. 1] Open Systems and its use in DoD acquisition programs are defined in DoD 5000.2R:

> PMs shall apply the open systems approach as an integrated business and technical strategy upon defining user needs. PMs shall assess the feasibility of using widely supported commercial interface standards in developing systems. The open systems approach shall be an integral part of the overall acquisition strategy to enable rapid acquisition with demonstrated technology, evolutionary and conventional development, interoperability, life-cycle supportability, and incremental system upgradability without major redesign during initial procurement and reprocurement of systems, subsystems, components, spares, and services, and during post-production support.

This concept has primarily described the design approach to development and proliferation of systems components that conform to widely published and accepted standards of interfaces and interoperability. This design approach facilitates multiple sourcing of system components and modules in a manner that allows the system to be updated (technology-inserted during post-production support) with minimal impact to the overall system design. The best example of this concept is the overwhelming evolutionary success of the global Internet infrastructure. Even though the IETF open standards are evolving continuously, multiple vendors track these changes and offer conforming products giving the marketplace choices through competition. Another example of an open system is the hardware interface standard called VME (Versa Module Eurocard), which was created by Motorola in 1985. A standard hardware computer bus interface created a whole new market of competing products for embedded development and system simulation and testing, which reduced the cost of hardware and solved the problems of obsolescence. Today, the VME standard continues to evolve and is one of the most used in high-performance embedded computing.

OSS is a natural extension of the Open Systems concept: OSS is an evolutionary extension to the Open Systems concept in that an entire software design is exposed to open review, collaboration, and extension rather than being limited to standards for functionality and interfaces. The benefits to interoperability, reliability, and maturity of software systems developed under this model are also well established. [Ref. 2,3] Widespread adoption of OSS as an extension of Open Systems management, acquisition, and development methodology would bring supportability, interoperability, and reuse for large software intensive systems to the next level for both high-end OEM and DoD programs.

OSS development tools (GNU) are becoming widely adopted: The GNU Tools, a free software toolchain created as OSS, have existed in organized form since the mid 1980s, and are now being widely adopted and supported by commercial industry as a standardized software development toolset. The GNU Tools have matured primarily in academic and small commercial cost sensitive environments. Over the past four years, GNU Tools have become appealing to large corporations like Cisco Systems, IBM, Airbus, Lucent, and Lockheed Martin to name a few. Organizations in Europe have created entire programs to promote the evolution of GNU development tools. One example is the European Space Agency (ESA), who created an organization called FRESCO (Free Software for ERC32 Systems Cooperation) to support the GNU Tools for the ERC32 Sparc architecture. ESA's aim is to control the support, evolution, versioning, quality and obsolescence problems normally associated with proprietary tools by providing fully supported OSS alternatives for ERC32 based space borne systems. [Ref. 8]

Though the GNU Tools have been promoted and applied on an increasing basis in commercial systems development, attempts to apply them as a standardized software development system have not been widely successful in high-end OEM and DoD systems for a number of reasons.

1. The historical perception that there is a lack of maturity;
2. Fear, Uncertainty, and Doubt (FUD) cast by proprietary product vendors;
3. Perceived lack of commercialization and support.

II. SYNOPSIS

This paper presents a case that mature, reliable OSS development tools, popularly known as the GNU Tools, are ready to be widely adopted as a business strategy and a solution to software tools obsolescence, vendor lock-in, and vanishing manufacturing sources as applied to embedded systems in software intensive, long lifecycle embedded computer systems that are typical of high-end OEM and DoD systems.

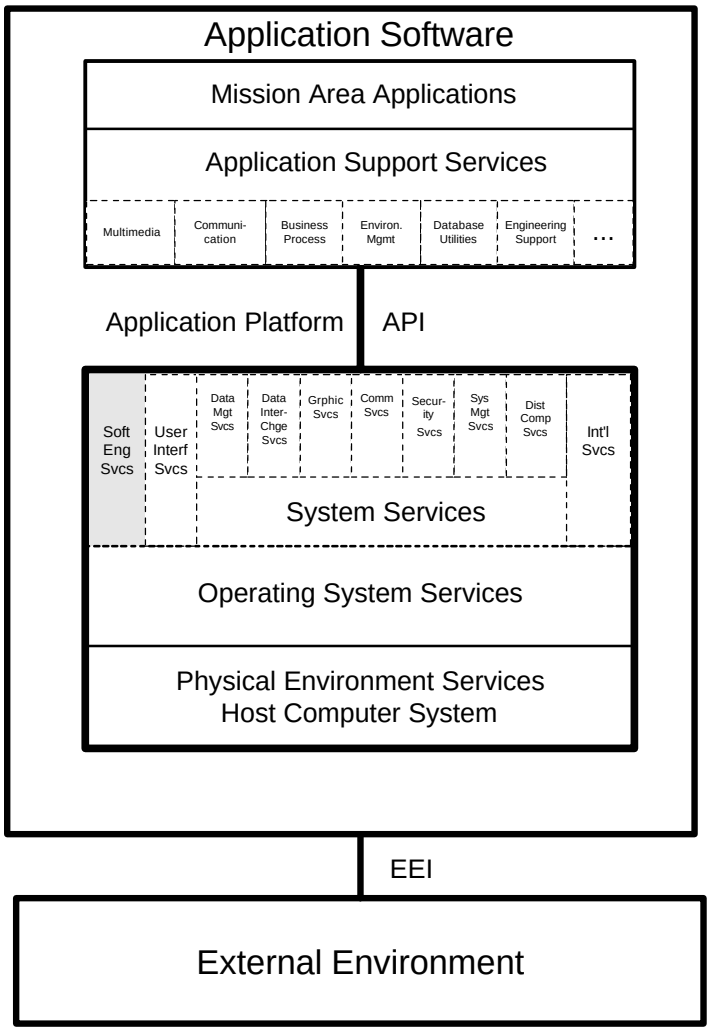
III. STANDARDIZATION

Current DoD policy concerning Open Systems design is expressed in DoD 5000.2R, Mandatory Procedures for Major Defense Acquisition Programs and Major Automated Information System Acquisition Programs. The Open Systems design strategy mandates standards that have been adopted and approved by a DoD sponsored standards organization that created the DoD Joint Technical Architecture (JTA). The JTA lists the minimum set of performance based primarily non-governmental standards needed to maximize interoperability and affordability within DoD and hence is entirely consistent with Acquisition Reform principles and practices. Implementation of the JTA, or the use of applicable JTA mandated standards, is required for all

emerging, or changes to an existing capability that produces, uses, or exchanges information in any form electronically; crosses a functional or DoD Component boundary; and gives the warfighter or DoD decision maker an operational capability. [Ref. 6]

In summary, the JTA is a single unifying technical architecture for integrating DoD system components. It includes a Technical Reference Model (TRM) that describes, from the foundation up a services view of the architecture. It includes physical environment services (host computer system), the operating system services, and the system services such as software engineering services, user interface services, and data management services. Figure 1 illustrates the services view of the TRM. It is the software engineering services portion of this model that is of interest to us (shown in gray background). It provides system developers with the tools that are appropriate to the development and maintenance of applications. Although it has been identified as a major component of the TRM, there are no mandated standards for this services area and a viable approach is needed.

Figure 1. TRM Model



As shown in Figure 1 above, the operating system API is the interface between these services and the host platform operating system. The currently mandated operating system API standards

are 1) ISO/IEC-9945-1 (POSIX), which is the Unix operating system, and 2) the Microsoft Win32 API, which is not an open standard but must be listed due to its currently pervasive use.

The ideal software engineering services component of the DoD TRM would consist of a rehostable, retargetable, API compliant software development system supporting both native and cross development and would include the following highly desirable attributes:

1. Include support for all widely-used and standardized compiler languages.
2. Provide all related utilities and productivity tools comprising a complete software development system.
3. Provide JTA compliant and uniform operator interfaces and commands.
4. Possesses uniform assembler language and linker syntax.
5. Demonstrate ease of portability to multiple host platforms.
6. Support a wide range of target CPUs and be retargetable to new CPUs.
7. In its entirety by unrestricted open source software.
8. Be supportable over an indefinite period through both organic and COTS strategies.
9. Have a long-term record of stability, reliability, and supportability.
10. Have a low Total Cost of Ownership (TCO).

A software development system built around a standardized toolset exhibiting the above attributes would become DoD's preferred solution for developers and maintainers of long lifecycle embedded systems who now view selection, sourcing, and sustaining the software development toolset as one of major management challenges of the system lifecycle. Over the past 10 years, the GNU Compiler and related tools have quietly evolved to the extent that these criteria are now being met. Supporting Ada, C, C++, Fortran, Pascal and Java computer languages, the GNU Toolchain is the only universal toolset that can achieve ubiquity. The remainder of this paper elaborates on these attributes and how the GNU toolchain might be incorporated into the DoD TRM and into high-end OEM systems.

IV. OSS DEVELOPMENT TOOLS AS AN OPEN SYSTEM COMPONENT

GNU Tools Compilation System: State and applicability of GNU Tools compilation system to embedded development.

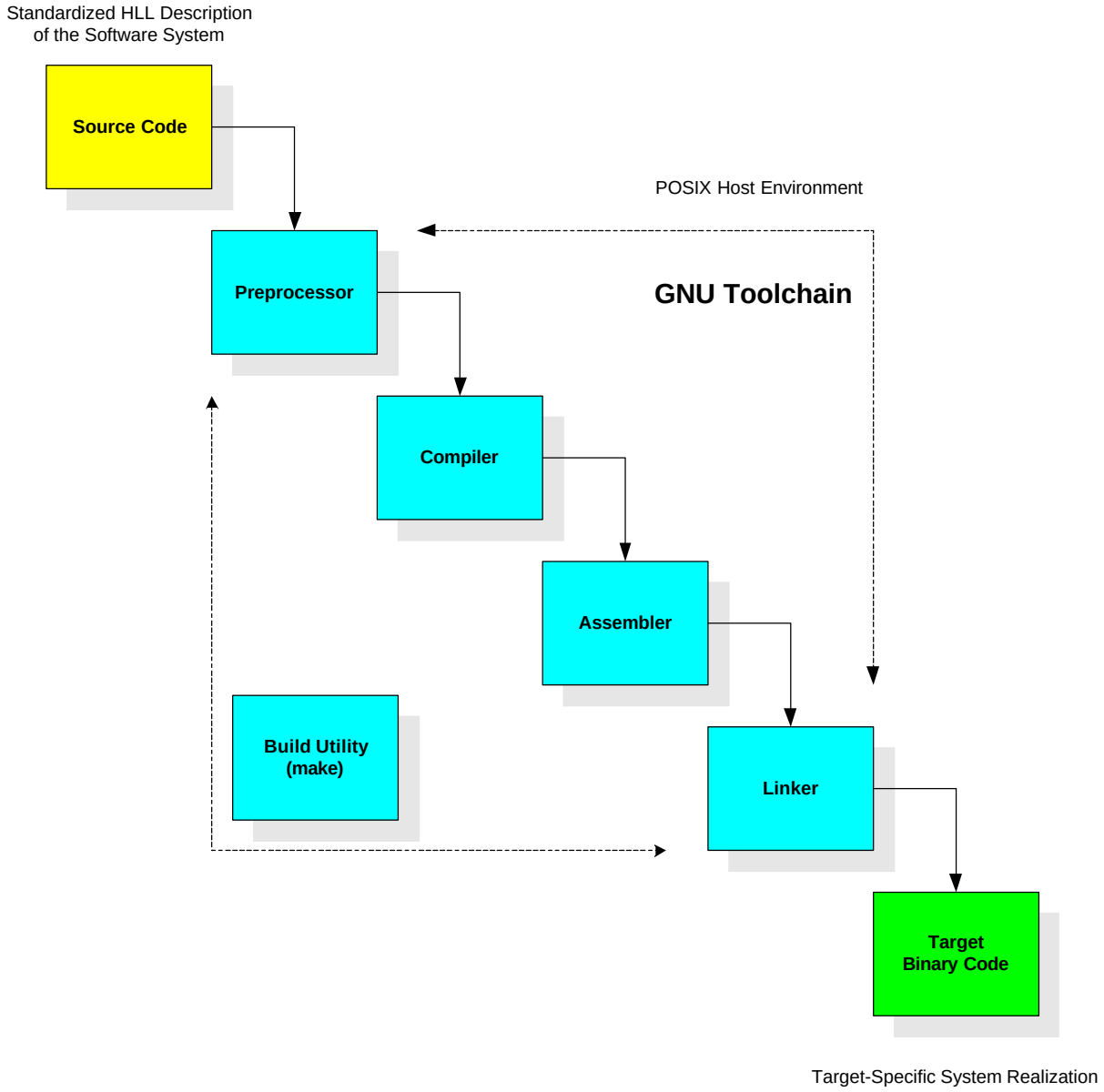
The GNU Tools originated as a successful OSS native compiler for Unix development: What is GNU? According to the Free Software Foundation (FSF), which was started in 1983 by MIT professor Dr. Richard Stallman, GNU stands for "Gnu's Not Unix." It's a recursive acronym. [Ref. 3] The FSF was chartered to support the development of non-proprietary freely distributable software based on collaborative development. One of the organizational goals of the FSF was to develop a free Unix implementation. Dr. Stallman recognized that a good, highly portable C compiler was truly a prerequisite to the success of this effort, so he set out to develop one. As it turns out, it was so well thought out, eloquent, and portable in design that over the years the GNU C Compiler (GCC) has been ported by contributors to almost every commercially viable computer system (and some not) as a native compiler. Attendant to this evolution was the growth of related tools, including assembler, linker, librarian, make utility, and even a debugger. Today the GNU toolchain contains every component that is expected of a professional software development system, and it has reached a high level of reliability and maturity rivaling the best proprietary commercial toolchains.

The highly portable design of GNU Tools led to widespread development of GCC-based cross-compilers: Along the evolutionary path, some contributors began to recognize the potential for the GNU compiler as an embedded cross-compilation system. So, today not only is the GNU compiler the most widely used native compiler in the Unix world, but in its embedded configurations supports more target Central Processor Units (CPUs) – different processors -- than any other single compiler toolchain.

The toolchain design is ideally suited for translation of HLL software designs into target code: The GNU toolchain structure is typical, consisting of preprocessor, compiler, assembler, and linker -- all under the control of a Unix-style Make utility. Figure 2 shows how the toolchain sequentially processes the input HLL system description to produce a loadable target system image. The design is atypical in that it was planned to be open and expandable by future contributors to the FSF. The toolchain design is modular, and its table-driven features facilitate porting to new target CPUs and host computers. Although GNU Tools supports Ada, C/C++, Fortran, Pascal, and Java, a recent survey by *Embedded Systems Programming* trade journal, has discovered that about 90 percent of all new embedded systems are coded in the C programming language. Because of the elegant design approach and open source nature of the GNU toolchain, it exhibits the following valuable technical characteristics:

1) The GNU Tools were designed to operate in a POSIX system environment. This characteristic makes the compilation system highly portable across any host platform supporting a POSIX like Application Program Interface (API). This is a requirement for DoD 5000.2R and the JTA. [Ref. 6,7] Existing POSIX compatibility layers also facilitate hosting on Win32 platforms.
2) The modular compiler, assembler, and other components incorporate 'machine descriptions' that facilitate porting of the compiler and tools to new or modified CPUs -- processors.
3) The command-line and language interfaces are completely uniform across host platforms and targets, except for port-specific options, such that little or no training is required as a developer transitions to a new host or new target.
4) The GNU compiler and toolchain backend interfaces are uniform and highly customizable to target specific CPUs, libraries, and memory. This facilitates straightforward customization of target runtime and memory configurations.
5) Because the sources are freely available and unrestricted, the using development team (or designated supporter) can immediately exploit 1 through 4 to meet specific needs both now and henceforth. For example, how often have we been forced to upgrade because COTS vendor 'X' no longer supports and licenses product 'Y' version 'Z'? If we desire to freeze a tool baseline at any point, then we are empowered to do so using the GNU Tools.

Figure 2. GNU Toolchain



GNU Compilation System Benefits

The GNU toolchain is completely open, and it is a development system having no restrictions on use or modification of binaries and sources. This is the only true insurance against software tool obsolescence now hindering most long lifecycle software intensive embedded systems. Because of this, multiple sources for organic and/or commercial sources for support and maintenance can exist. This multi-source support is a primary objective according to current DoD acquisition guidance.

The GNU Tools have been ported to every commercially viable computer system, proving its flexibility and portability. This assures availability in the form dictated by evolutionary forces within the COTS platform marketplace.

Developed within and as part of the Unix software domain, POSIX is the most mature, stable, and standardized computing platform (API) in existence. When host platform portability and interoperability are desired, there are no alternatives, and POSIX compliance is mandated as part of the DoD JTA. Because GNU Tools are POSIX compliant, they are directly compatible with the DoD JTA.

Stability – The GNU Tools have a straight-line maturation path over 15 years long. A marketing department has not driven the GNU Tools requirements. Many market driven products are often 're-invented' with alarming frequency at the sacrifice of maturity, stability, and backwards compatibility. This often creates problems for long lifecycle system maintainers, but is seldom a concern of the product vendor whose competitive vision is always forward-looking.

Because the GNU Toolchain is OSS, it is the most understood and widely supported compilation system. Its freely available sources and design are studied and improved upon by thousands of software engineers worldwide. The GNU Tools are the most peer-reviewed software in the world, which is why there is such high quality in their design and bug-free operation. Both free and commercial sources of support for GNU Tools exist. This situation also promotes the existence of a large pool of subject matter expertise that cannot exist with proprietary software systems.

In spite of this freedom and accessibility, the evolution of the 'officially maintained' sources is tightly controlled, and integrity of the system may be easily guaranteed when required by commercial support organizations. At least a dozen companies worldwide support GNU Tools and other OSS products. Microcross, Red Hat, MontaVista Software, TimeSys, and Viosoft are examples of companies supporting OSS in one fashion or another, and the list of companies supporting OSS is growing.

The unity of design and long-term stability drastically reduces training costs and preserves career investments in toolchain proficiency. Having to re-learn new tools on each new project, or a new, 'whiz-bang' IDE release can be a de-motivator for software engineers who know and desire the tools that they have become accustomed to. The cost of retraining a large software team is usually a major cost impact and sometimes unplanned.

The run-time environment of every embedded system is unique and accordingly, the run-time libraries and target APIs must be tailored to suit project requirements. The open runtime libraries available with GNU Tools and features of the binary utilities and linker all facilitate straightforward modification of the runtime environment to meet user specific needs.

Cross/native duality is a valuable test and validation attribute of GNU Tools that is offered with few, if any proprietary development systems. Because the GNU Tools are normally hosted in native form on the development host platform, source code may be built and tested on the host platform in native mode, as well as on the cross target with the same compilation system, and no changes to the code under test! Moreover, on most cross-compiler versions of the GNU Tools, the environment includes a target CPU simulator that supports yet a third execution mode (targeted simulation) in which cross compiled code may be executed on the development host resident target simulator. These three independent code test and validation facilities provide an unprecedented capability unavailable from proprietary tool vendors.

Heterogeneous Development Environment

Due to the competing needs and requirements of users and organizations, and pervasive nature of network technology as a bridge, many organizations own multi-platform IT environments. Another strength of the GNU Tools for embedded development is that they work well in this situation. The most common example of a heterogeneous environment is a software engineering organization that has historically used Unix platforms for software development, network services, and code management, but now supports desktop automation almost exclusively using Windows on PC systems. In this case the bulk of the organization's computing power resides on the desktop, while code management and configuration control remain centralized in a Unix environment. The GNU Tools and a related open-source POSIX shell environment called Cygwin combine the best of both environments. In this solution environment, desktop users can mount shared sources and run private builds and tests on their desktop using the exact tools and source configuration used on the Unix production and code management platform. This capability, which distributes computing load to the local computer without creating a totally distributed development system, is difficult, if not impossible, to duplicate using non-portable toolsets that are typical of proprietary COTS toolsets.

Integrated and Batch Operation

The GNU Tools and GNU Make comprise the GNU toolchain compilation system. This system originally designed for command shell based operations is also highly integrable with several open source, as well as proprietary, Integrated Development Environments (IDEs). A couple of these environments are particularly well suited because they use GNU Make as the build tool and manage the project by manipulating the makefile. This approach provides the unique ability to execute a project build from within the IDE or from a standalone shell, or better yet from another makefile. These batch operations are often needed to support maintenance and production builds of large systems and are not supported by many COTS IDEs. The vast majority of successful OSS systems and components such as the DARPA TCP/IP protocols and the increasingly popular Linux operating system require the GNU Tools as the standard build toolset.

Multiple Targets

An increasing number of embedded systems are required to be retargeted at some point during their lifecycle (for various reasons). Using the modular GNU cross-tools compiler with its multitude of CPU code emitters and completely uniform front end, the task of retargeting an application can be greatly simplified. Porting even an ANSI compliant source tree to a new and different compiler, assembler, linker, and builder can be complex and time consuming, especially if non-standard compiler extensions are involved. Therefore, the potential for savings of time and money over a system's lifecycle is tremendous. Up-front planning for the use of GNU Tools as the build toolset is all that is required.

Tools Replacement and Source Migration

A major problem affecting maintainers of long lifecycle DoD systems has been software tools obsolescence. Years ago during development, currently available COTS software development tools and platforms were selected. Now years later most of these tools (and their host platforms) have obsolesced out of existence. The maintainers are now faced with the issue of replacing these tools and support platforms. Now wary of these problems, they have an interest in

replacing tools in a forward compatible manner to extend the lifecycle. Moreover, many once viable source languages such as Fortran, Jovial, Pascal, and Ada are now losing market share to C/C++, and COTS compilers are becoming less plentiful and more expensive. Many upgrade programs are considering source-level migration of operational software to C/C++ as a long-term risk mitigation strategy. Others desire to rehost and re-baseline existing code to new tools that can be assured as 'obsolescence-proof' as possible. Of course, cost continues to be a driving factor. GNU Tools can reduce these program risks in three powerful ways:

1. Because of the unrestricted nature of GNU Tools, the end user can never be denied the right to use, proliferate, port, modify or freeze a baseline of the tools to meet specific system needs. If a bug is detected, it can be fixed and the tools updated. No proprietary toolchain offers this flexibility and freedom. In essence this is as close to 'obsolescence-proof' as we can get.
2. The open source community is now maturing alternate language processors as front-ends to the GCC toolchain. Front-ends exist for Fortran, Pascal, Ada, and even native Java. These OSS components may be used to implement a toolchain at reasonable cost, which can be used to re-baseline and support a non-C/C++ software system using GNU Tools.
3. A software toolchain implemented using the GNU C Compiler (GCC) and an appropriate source language processor actually uses a source to C language translator as the front-end. Hence, a rehost of an operational software system to a GCC-based toolchain 'funds' the first stage of the code base migration to the C language. This statement is not intended to trivialize this complex task, but nonetheless demonstrates a large potential cost savings. When organized as an incremental activity (by module or function) additional cost savings and risk reduction benefits come to mind.

When considered together, the combined benefits of reduced cost over proprietary COTs tools, obsolescence abatement, portability and intrinsic capability as a source migration toolset seem to strongly suggest that the GNU Tools compilation system has many benefits to the large, long lifecycle system developer and maintainer that are hard to ignore. While it is true that some of these capabilities are available in the proprietary COTS world, the cost differentials are enormous, and no one can offer the OSS 'obsolescence-proof' guarantee that effectively makes a maintainer the lifetime owner of the toolchain.

Most Often Quoted objections (and Factual responses)

Skeptics often cite the following issues as barriers to successful deployment of OSS development systems. Others are also cited, but have been implicitly addressed in the previous discussion.

GNU Tools, and other open-source / freeware, cannot be high quality software products.

The GNU Tools have been evolving on a straight maturation path since 1983. The requirements have not changed significantly, nor have they been 're-invented' every few years at the whims of marketers. The GNU source maintainers utilize an effective bug reporting, peer review process, and correction methodology that is also proven and mature. The GNU compiler and associated tools have been benchmarked and evaluated by independent studies and organizations, and though recognized not as always the best performer for a given target CPU, it always performs respectably and does a superior job at insuring code correctness. In the authors' experience with code porting, the GNU compiler often identifies coding errors and incorrect constructs that went undetected by 'high-quality' proprietary COTS compilers. Numerous GNU compilation options

allow the compiler to select a desired language 'dialect.' A final testament to the robustness and quality is the simple fact that some of the most robust, tested, and stressed software systems in use today are compiled using the GNU Tools. Examples include several Unix operating systems, Linux, Apache Web Server, and the European Space Agency's embedded spacecraft software systems.

GNU Tools are not supported; developers can claim no single, accountable entity for support.

Although this is true to some extent, it is in actuality the strength that developers seek rather than a weakness. In the traditional sense, the GNU Tools are viewed as un-owned, so no one is accountable. This is now an invalid assumption for several reasons:

(1) The Free Software Foundation (FSF) who legally owns the GNU code base is fully committed to the continuing success of GNU Tools and all free software for that matter. Their charter is devoted to the continuing successful adoption and evolution of the GNU software. As the maintainer of the official GNU source tree, the FSF makes corrections to significant bugs and release improvements on a regular update cycle. This model is the same as for COTS software, insuring that required updates to the product baseline are distributed periodically. With OSS, the updates include sources. Some organizations make the training investment to maintain and build from these controlled source baselines, resulting in an organic tools support posture.

(2) The second and usually even more responsive support mechanism exists in the form of Internet collaboration. Because the GNU software architecture is open, it is studied and widely understood by a large number of software professionals, many who specialize in certain aspects of the design. Most of them are proud to share their knowledge with others, and often respond to enquiries and help requests on list servers and online user forums. Because a developer (or his/her support organization) has the source code, suggested fixes can be easily incorporated by rebuilding the tools from the patched source code. This is in fact what most customers expect to happen (on the part of the vendor) with COTS software, but it rarely does. This concept has been demonstrated when security weaknesses have been discovered in open operating systems like Linux and FreeBSD. In the past, the experts would spring into action and within hours (or at worst case days) fixes would be posted on the Internet for the immediate collective benefit of all users. This level of support is customarily available only to COTs software customers who pay large sums of money for support contracts.

(3) The third support option for OSS support is COTS, which now exists. If an organization desires to delegate accountability for support, there are companies that now have the resources and knowledge-base to provide contract support, regular updates, and accountability for these services. These services may be purchased pre-paid, subscription, or a la Carte depending on the vendor's business model and the agreement negotiated. At any rate this option transforms the OSS into a COTS product whose only difference is the vendor's inability to deny the user access to the source code. The vendor is protected because his value-adds are in packaging, technical support, customization, porting, training, and integrating updates. In perspective, what becomes clear is that the GNU Tools offer a wider spectrum of support options and flexibility than any competing proprietary COTS product. At one end of the spectrum, limited budget users may utilize the GNU Tools using do-it-yourself build and test collaborative support and at no-cash investment. At the high

end, an organization may purchase the GNU Tools as a commercially supported product with custom support and accountability. Between the extremes lie a multitude of options and combinations. When the added benefits of right-to-source and obsolescence proofing are considered, all alternatives to the GNU Tools appear weak.

Ready-to-run products are not available.

Most of the freely available OSS tools distributions are available in source form only and must be built from source code, requiring significant time and mastered skills. Moreover, hardware drivers, debugger integration, and other issues still remain. Even at best, this can consume weeks of work and days of run-time testing. This negative aspect of OSS tools has been recognized, and OSS tool vendors have taken advantage of the opportunity to offer solutions that solve this problem. Some vendors now offer the GNU toolchain configured for cross-development in ready-to-run packages for the most popular host platforms. These offerings provide a quick-start approach, so a developer can bring up an entire development system hardware baseline in a matter of hours or days. This capability, though historically not attributed to OSS development tools, is now available and rivals the high-cost, turnkey development platform products once only available at high cost from the proprietary COTS tool vendors.

A fully integrated development environment is needed.

The GNU Tools are a shell driven compilation system, as are most other compiler toolsets. There are numerous Integrated Development Environments (IDEs) available, both open source and proprietary, which have been directly integrated with the GNU Tools. Some of the vendors provide software development kits that are pre-integrated with the GNU Tools. A strength of this integration approach is that the toolchain is well partitioned and may be used and maintained independent of the IDE. A valuable benefit of those IDEs that construct GNU makefiles is the ability to execute software builds from a non-interactive shell environment. This is often necessary with large software intensive systems consisting of multiple projects for production, test, and maintenance. An IDE based development system using GNU Tools is available in both supported OSS tools and COTS proprietary forms.

V. BUSINESS CASE FOR OSS TOOLS

Acquisition costs of the GNU Tools are by far lower than competing proprietary tool vendors. Acquisition costs can be as little as free (if the developer is willing to invest significant time to build, test, and install the tools) to as low as $500 for a limited support, shrink-wrapped COTS product, and as high as $2,000 for a fully supported product. This is compared to the average proprietary tools vendor who charges $3,000 to $75,000 for a comparable functioning product. Moreover, the proprietary COTS tools vendor locks the user into a product that may have an uncertain future in regards to long-term support and configuration control; in addition, no source code will be available to counter the obsolescence threat or fix defects.

Although multiple sources was once a problem, OSS COTS tools are now available through multiple sources, including the FSF and several commercial vendors. On the other hand, proprietary software products are essentially single sourced, unless designed against strict interoperability standards.

Supportability over the long-term (10-40 years) is of major concern for OEMs and the DoD on modern embedded computer systems. Even the slightest modification to a system has to have at least a five-year return on investment. The proprietary COTS tools vendor is typically market driven due to economics, and long-term supportability has not been high on the priority list as evidenced by the obsolescence problems confronting the OEMs and DoD today. The best assurance possible to ensuring long-term supportability and preventing tools obsolescence is using OSS tools.

Security is always an issue in software intensive systems. When the developer/maintainer is vendor-locked by proprietary systems, there are few, if any, alternatives to solving a security issue (bug). If the developer/maintainer uses the GNU Tools, then by nature of having the source code under control is assurance that security bugs can be resolved quickly and inexpensively even without the OSS tool vendor's support. Whereas if the developer/maintainer is totally dependent on vendor XYZ and this vendor no longer exists or decides against making fixes, the security problem will become virtually unsolvable – or one that will be costly and time consuming.

Scalability is an attribute desired in all software intensive embedded systems, and it is a key feature of the GNU Tools. From small, deeply embedded applications to large-scale systems-on-chip solutions, the GNU Tools are widely accepted and used. For example, on the low-end the GNU Tools are used to build 16-bit microcontroller applications. Typical applications on the low end include control systems, medical devices, computer peripherals, etc.. On the high end, the GNU Tools are used to build 32/64-bit applications. Examples include the Unix and Linux operating systems, Internet routers and switches, telecommunication soft switches, aerospace navigation systems, electronic warfare systems, and transportation control systems.

Last, GNU Tools' popularity or brand reputation is in the top five. Red Hat (formerly Cygnus Solutions) boasts over 200,000 users of their popular version of GNU Tools as a cross-development platform. [Ref. 5] On the native development side, nearly every Unix and all Linux OS distributions contain GNU Tools. Without question, GNU Tools are the most popular POSIX compliant compiler toolset in the world.

VI. OSS TOOLS SUMMARY

In summary, the authors have explained from a technical perspective, what the GNU toolchain is, its potential role as a pervasive (and possibly) standardized embedded development platform, and why it makes logical sense to adopt their use in long lifecycle systems. Numerous merits and benefits have been cited, most notably cost reduction and tools obsolescence abatement. There are a couple of admitted risks. Since there is no single entity that owns the OSS, it requires the end user to accept and take ownership. In the overall analysis, however, the benefits appear clearly overwhelming. Wide spread adoption of the GNU Tools can have a coalescing effect on the entire science of embedded software engineering, without the attendant commercial domination of a single vendor and the disruption of continual product reinvention. This standardization would bring commonality to training, creating a sustaining labor force of embedded developers with skills right out of college. This will bring the cost of development down, providing lower cost goods and services that use embedded computer technologies. Colleges and universities are already adopting the GNU Tools for both the benefit of the industry demand and the student. Last and most important, GNU Tools can solve the problems of vendor lock-in, vanishing manufacturing resources, and tools obsolescence.

REFERENCES

1. *Open Systems and the Systems Engineering Process*, Michael Hanratty, Robert H. Lightsey, & Arvid G. Larson, Acquisition Review Quarterly Published by the Defense Systems Management College,1999, http://www.dsmc.dsm.mil/pubs/arq/99arq/hanratty.pdf
2. *A Business Case Study of Open Source Software*, Carolyn A. Kenwood, Mitre Corporation Publication MP01B0000048, 2000 http://www.mitre.org/support/papers/tech_papers_01/kenwood_software/
3. *Open Source Initiative (open case for business)*, 1999-2002, http://www.opensource.org/advocacy
4. *Guidelines for Successful Acquisition and Management of Software Intensive Systems*, Department of the Air Force Software Technology Support Center, Feb 1995 – 2001, http://www.stsc.hill.af.mil/
5. *Embedded.com Buyer's Guide*, 2000, http://www.embedded.com
6. *DoD Joint Technical Architecture (JTA)*, Version 4.0, 2002, http://www.disa.mil
7. *DoD 5000.2R*, Mandatory Procedures for Major Defense Acquisition Programs (MDAPs) and Major Automated Information System (MAIS) Acquisition Programs, 2002, http://www.acq.osd.mil/ar/library.htm
8. *FRee Software for ERC32 Systems COoperation (FRESCO)*, 2002, http://www.estec.esa.nl/wmwww/EME/compilers/Fresco/

LIST OF ACRONYMS

API	Application Programming Interface
COTS	Commercial Off-The-Shelf
CMM	Capability Maturity Model
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DOD	Department of Defense
ESA	European Space Agency
FSF	Free software Foundation
GCC	GNU C Compiler
GNU	GNU's Not Unix
HLL	High Level Language
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IT	Information Technology
JTA	Joint Technical Architecture
OEM	Original Equipment Manufacturer
OSS	Open Source Software
PM	Program Manager
POSIX	Portable Operating System Interface for UniX

ABOUT THE AUTHORS

Steven L. Rodgers

Mr. Steve Rodgers has over 30 years of engineering experience in multi-disciplined environments having hardware and software development projects as complementary activities. Mr. Rodgers co-founded Microcross, Inc., now growing fast and diverse in supporting embedded systems to developers in 35 countries. Since 1999, Mr. Rodgers has been leading the Microcross development team in creating commercially supported GNU X-Tools™ as a product line for commercial business enterprise. Previously, Mr. Rodgers served as chief engineer at the TRW Aerospace and Defense Electronics Division in Georgia. Mr. Rodgers has authored patents on GPS technologies and has experience managing and developing aerospace communication systems with emphasis on wireless, mobile telecommunications, embedded microcomputers, Electronic Warfare (EW) systems, and digital/RF hardware design. Mr. Rodgers has been embedding computers since 1979 (pre-PC era) and is most noted for work as a pioneer in using the PC (circa 1985) as an embedded computer in aerospace and DoD / Government applications. Mr. Rodgers is a native of Macon, Georgia and holds a BEE from the Georgia Institute of Technology and several industry certifications in voice/data communications and network engineering.

James B. Calvin, Jr.

Mr. James Calvin has over 20 years of technical leadership experience in various embedded system development projects. Currently, Mr. Calvin is championing the cause of free and open source software tools as a solution to tools obsolesence in long lifecycle, software intensive support programs that are typical of military and aerospace systems. Mr. Calvin co-founded Microcross, Inc., which is becoming known in the embedded community as the 'Open Source Embedded Tools Company.' Since starting Microcross, Mr. Calvin has managed the GNU Cross-Tools product development activities, including software test engineering, application software development, user documentation creation, and sales and marketing. Mr. Calvin's previous embedded experience includes project leadership over developing complex embedded system test stations that are used by the USAF to support aircraft operational flight program modifications, system integration and flight testing. Mr. Calvin is a native of Jackson, California and holds a BEE from the Georgia Institute of Technology, an MSEE from the Air Force Institutue of Technology, and an MSA from Central Michigan University.