A System View of Merging Software and Hardware

Mike McNair
Science Applications International Corporation

No single example is universally applicable in describing the merging of hardware and software into an integrated system. Likewise, no single recipe can be given for making the effort a successful one. In spite of this, there are generalizations that can be identified. The observations presented in this article and the subsequent recommendations are intended to make the effort of merging hardware and software a successful one.

For most of us who have endured a hardware-software integration (HSI) effort, we can all attest to the fact that not only was there technical challenge but programmatic challenge as well. From an engineer's perspective, maybe management just really did not understand what support was needed, that schedules needed to include contingency time, and that there just might have been the added cost of one more analyzer.

There is a flip side, however. From the manager's view, could the tools have not been anticipated, how much schedule allowance is really needed, and maybe more to the point, how could the risk be properly assessed and subsequently mitigated?

It is very easy and somewhat simplistic to think of merging software and hardware as a purely technical issue; instead, it is a mix of technical, programmatic, and personnel interrelationships. In order to understand the interrelationships between these areas, it is important to recognize what merging hardware and software *is* and *what it is not*.

Hardware/software merge activities are not limited to just the following:

- Activities during integration. They really exist throughout the entire development cycle.
- Embedded applications or real-time systems. This applies to being able to move data from a bar code scanner to a database as well as controlling a radar system.
- Device drivers and interrupt handlers. These facilitate the interface but do not form the complete description of the interface.
- Reliance on interface definition. But instead, a successful allocation of functions to satisfy the requirements for a system.

To make it more clear, the following are examples of a hardware/software merge activity:

- Models and algorithms (software) have a physical impact (hardware). Otherwise, software is just an algorithm and the hardware is just a machine
- Engineering disciplines share common ground. This means common milestones, common understandings, and common requirements.
- Demos, prototypes, and products are made possible until the point at which the system is typically depicted as a collection of models, simulations, and emulations.

As a general observation, I have seen many software engineers grow frustrated with trying to trace a software bug only to find it was really a hardware flaw. Likewise, I have seen hardware engineers grow frustrated with a software engineer who does not understand the physical limitations of the device. Sometimes you just cannot make the hardware do everything software can or vice versa.

> *"It is very easy and somewhat simplistic to think of merging software and hardware as a purely technical issue; instead, it is a mix of technical, programmatic, and personnel interrelationships."*

Characterizing a Hardware/Software Project

Regardless of how a project is characterized and modeled, it is usually possible to identify some simple and high-level phases that constitute its life cycle (iterative, spiral, waterfall, etc.). For the purpose of this article, we will assume the presence of these phases since these can usually be found in some form in any life-cycle model: requirements analysis, development, integration, and test/verification.

Each of these phases has a technical and management component that plays a part in the overall effort of merging hardware and software. These technical and managerial activities can be described by a set of activities and characteristics. The following is a look at those activities as they pertain to merging hardware and software.

Requirements Analysis

During the *requirements analysis* phase, it is generally expected that concepts will be pinned down, requirements will be identified, and top-level allocations of functions will be made to subsystems and possibly between hardware and software components. Management is usually busy setting customer expectations and managing the natural influx of ad-hoc requirements. It is important to realize that the customer as well as the product team will be sources of capabilities that are not originally within the product's scope.

Development

Development is almost always planned and implemented with a discipline or specialty focus (product team, hardware/software, logistics/training, etc). Work naturally gets segregated to the groups that make up the development organization. As a result, *stovepiping* is not only common but is considered *normal*. It is typically very difficult to integrate these groups during development whether from a cost and schedule perspective or by an engineering task. Management is usually busy facilitating information flow and keeping the independent disciplines coordinated. It is during this time of a project that emphasis is placed on schedules and milestones; there is little else (aside from status meetings) that a manager has available to keep these groups coordinated.

Integration

As happens with any project, the day

comes when the products of the development effort have to fit together. Whether the pieces have been integrated in an iterative or spiral type approach or done all at once as a part of a waterfall life cycle, there is a real incentive to make the system work. Technically, if all goes well, initial indications are that the software seems to make the hardware *do something*; yet, if problems are encountered, they can be difficult to isolate and fix. From a management perspective, there can be very real cost and schedule impacts – either making up lost schedule when integration goes well or losing schedule when solutions seem difficult and elusive.

Test/Verification

Finally, the test and verification team has an opportunity to determine what the integrated whole actually is able to do. The engineering team is reminded of the requirements, and an outside team is able to assess how effective the effort to date has been at providing the product as it is specified. As the results are unveiled, the functions become capabilities and the problems become limitations. Management is again working with the customer to reset expectations and to define the measures of success. If success can be properly defined, the product is released and put into use.

The Big Picture

Along the way and with each program phase, software and hardware have been merged. If we review the development phases with this in mind, we begin to understand what role merging hardware and software has in the overall process.

In the *requirements analysis* phase, allocations are identified between hardware and software. The processors for the software to run on are assessed and selected. The interfaces to the different devices are specified, which in turn results in design decisions for the software. As algorithms are developed and models are created, hardware selections are made to achieve necessary performance. During the *development* phase, these allocations are used to allow detailed work to progress. Interfaces are identified, defined, and refined. The effort is managed in terms of the work packages that result from these allocations. During the *integration* phase, these same interfaces and allocations are used to identify hierarchical levels of testing with an eye toward increasing levels of integration with each successive level. Finally the product undergoes test and verification. Internal interfaces in essence *go away* and the external interfaces characterize the system. Internal interfaces are seams of the product that either strengthen the set of capabilities or cause stress points where the system can be broken.

Merging Hardware and Software Successfully

From the experiences of all of this, there are some valuable principles that can be identified. With each new project, the hope is for the disciplines to work together better and to act in a more integrated fashion. There is a push for resources and risks to be identified and assessed through lessons learned. It is hoped that the development environment and test tools can be standardized to increase familiarity and make the whole engineering effort run more smoothly and enhance the creativity of the engineering staff. While there are many ways to document these principles (certainly the following list is not comprehensive), these principles and observations can make the merging of hardware and software more productive.

"While it is good to take ownership of the assigned task, it must be remembered that functions as well as defects are a characteristic of the product and not the person."

- **Observation One:** *Interfaces are viewed differently by different groups.* Systems engineering uses interfaces as a means of identifying allocation boundaries and defining allocated component interactions (not behavior). Development engineering uses interfaces as a means of determining *what is mine* and *what is yours* for the purpose of bounding the solution space. The integrators use interfaces as the means of identifying components and how they *bolt* together. Here, interfaces are not just boundaries; interfaces are how expected functionality is scoped. The test and verification team relates to interfaces depending on where they are: Internal interfaces are the seams where the product is most likely to break, whereas external interfaces create customer perceptions and expectations.
- **Observation Two:** *Software and hardware groups use interfaces differently.* These two groups see each other in a very unique way, but it all boils down to one basic premise: Software executes on hardware and hardware is exercised by software. This principle is exemplified where hardware and software merge. The software team sees interfaces in terms of messages, queues, services, etc. Interfaces, including where hardware needs to be controlled, are a part of an abstract model that is depicted in code and data models. The hardware team sees interfaces in terms of wiring, connectors, drivers, etc. Interfaces are a means of achieving connectivity with other components in the system. The usual pitfall is that neither group focuses on system-level functions – both groups tend to lose a system view and the role they play in the overall system capability. It is not sufficient to know the interface; it must be understood that the interface is a piece of the whole system.
- **Observation Three:** *People personalize the components.* When hardware and software do not merge easily, it is easy to fall into the traps of *I said/you said, the interface spec is wrong, we are dealing with bad requirements*, and other finger-pointing type phrases. Engineers discuss components in terms of *I will send you the message* or *with this signal, I will enable an actuator*. Personalities become an inherent part of the system. In short, people take ownership of parts of the system and personally associate with it. While it is good to take ownership of the assigned task, it must be remembered that functions as well as defects are a characteristic of the product and not the person.
- **Observation Four:** *Hardware/software allocation is becoming more a difference in performance than functionality*. As a project moves through requirements analysis and into design, functional requirements are allocated to some combination of hardware and software for implementation. Before the introduction of Application Specific Integrated Circuits, Floating Point Gate Arrays, microcontrollers, and other special purpose processor targets, the allocation approach was simple. Now, instead of just relying on general purpose processors that execute application specific code, algorithms can be

embedded on the processor itself, thus bringing in the added option of application specific hardware. Identifying the proper allocation of a system function can now be through software or application-specific hardware. The initial factor in making this allocation decision usually rests with satisfaction of the system performance requirements.

- **Observation Five:** *Not everyone is adept at merging hardware and software.* Every member of the project team brings experience and training to apply to the problems they are assigned to solve. These differences in experience and training generally result in some people having a knack for requirements analysis over design, or test over integration. It is especially important to recognize this when selecting personnel to conduct the integration effort. When software and hardware meet, the integrator cannot necessarily be someone who just happens to have some spare time. Integration is not usually viewed as the *fun* part of a project. In fact, people are usually recruited from an engineering group or integration is assigned to the test team. Neither of these provides the independence and immersion in the effort that is required.

  If the effort cannot be staffed with someone focused on the effort of making hardware and software meet and work together, the risks increase, and as the risks are realized the schedules stretch, the engineering teams lose their effectiveness and the costs seem to skyrocket. Someone must be responsible for managing an engineer's desire to get results and a manager's need for trade-offs and options. That person has to understand that there is a decision and not a solution.
- **Observation Six:** *A properly specified and defined interface is not the* end all *to successful integration.* Integration cannot rest completely on the definition of the interfaces. The interfaces were identified in the first place through an act of allocating capabilities to systems and components – both hardware and software. If the allocation is wrong or ineffective, no amount of interface definition will help the system. Interfaces, when they are identified, are expressed in terms of software interfaces (possibly documented in Interface Requirement Specifications and Interface Description Documents) and hardware interfaces (similarly documented in Interface Control Drawings). What is missing is the role they play in overall system functionality and the identified system components. In the final product, internal interfaces existed to provide interconnections. If they cannot provide this between properly allocated systems and components, integration (including merging hardware and software) will be very painful to accomplish.
- **Observation Seven:** *Vendor support is beneficial.* There are times when you just need to find an expert who can help you build your system. Usually at a time when schedules, budgets, and patience have all been stretched thin, calling in help is not at the top of the to-do list. The usual fears abound of unplanned cost, uncertain return, lack of domain knowledge, etc., but these are combined with the realization that money is running low, the delivery date is quickly approaching, and there are still significant problems to solve. Admittedly, calling in a consultant or some kind of niche expert brings its own set of risks.

  The better solution is to use these experts along the way, whether from your tool vendor or as a consultant. These subject matter experts (SMEs) tend to have very specific exposure to problems and solutions that the more general development is likely not to have. Allowing an SME to provide a third-party assessment of technical approach can be a great source of implementation alternatives as well as a source of lessons learned. Keep in mind that a vendor has probably seen many more applications of their products than either you or anyone else on your team. Make use of their expertise through training, phone calls, e-mail, Web support, and even on-site support when necessary.

"Every member of the project team brings experience and training to apply to the problems they are assigned to solve."

Closing Thoughts

The observation that stands out the strongest and the lesson that is relearned the most is that hardware/software projects are tough and as such demand careful and realistic planning and execution. While every system and application is different, one thing is clear: Merging hardware and software crosses teams and disciplines. In order to be successful, all members of the project team must acknowledge their part in working as an integrated team that will deliver an integrated product.◆

Background Reading

1. Walker, Royce. Software Project Management: A Unified Framework. Addison-Wesley, 1998.
2. CMMI Product Team. "Capability Maturity Model Integration Version 1.1 Continuous Representation." Pittsburgh, PA: Carnegie Mellon University, Mar. 2002.
3. Project Management Institute. "A Guide to the Project Management Body of Knowledge (PMBOK Guide)." 3rd ed. Newton Square, PA: Project Management Institute, Oct. 2004 <www.pmi.org>.

Additional Reading

1. SoftwareProjects.org <https://www.softwareprojects.org>.
2. Software Program Managers Network <http://www.spmn.com>.

About the Author

Mike McNair is a senior systems engineer for Science Applications International Corporation where he is a part of the chief engineer team for unmanned ground vehicles on contract to the U.S. Army. His background includes more than 20 years of experience as a programmer, technical lead, and software program manager on projects ranging in size, complexity, and targets for a variety of customers. McNair has also served on several process improvement (including Software Engineering Process Group lead) and training initiatives.

8303 N Mopac EXPY
STE B-450
Austin, TX 78759
Phone: (512) 366-7834
Fax: (512) 366-7860
E-mail: mcnairmk@saic.com