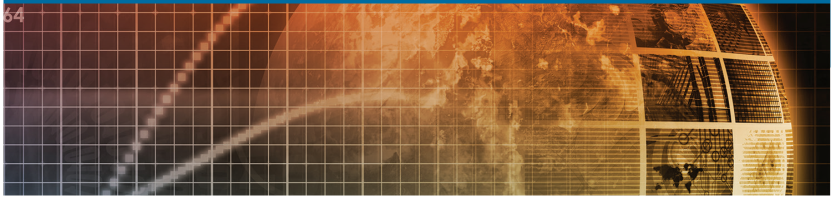
Simulation-Based Embedded Agile Development

Jason Ard, Kristine Davidsen, and Terril Hurst
Raytheon Missile Systems

// When tailored to work in conjunction with agile principles, simulation-centric development can dramatically improve product quality, reduce cost, and rapidly deliver reliable working code. //



AEROSPACE SOFTWARE DEVELOPERS are constantly driven to deliver higher-quality products faster and cheaper. One recent assignment mandated going from a notional paper design to initial flight tests in only 10 months. Previous timelines for similar products had ranged from two to three years. Although we commonly leverage simulation and embedded software frameworks for high-level reuse, there is often no similar preexisting product that can be adapted to meet customer requirements due to the wide diversity of customer needs. At the beginning of a project, there is typically no existing hardware, firmware, nor test equipment. When those resources do become available, they are often insufficient to support the simultaneous needs of all system developers.

In addition to typical software constraints such as available memory, throughput, and performance, the real-time embedded nature of our products routinely imposes requirements such as tight execution timing, concurrency, and conformance to hardware interfaces. There are many ways for a missile test to go awry, so safety and reliability are always high priority. With any new design, our requirements are not fully understood at the outset and are subject to change. Nevertheless, starting very early in our product life cycle, we must demonstrate functional software in parallel with developing the hardware.

To deal with these challenges, we combine a software-in-simulation (SiS) architecture (see Figure 1), with the set of practices collectively known as agile software development. We use simulation not just for periodic performance assessment, but daily, to provide continuous feedback for collaborative design and development.

Software-in-Simulation

Our embedded software consists of many different modules, such as guidance and control algorithms, seeker and signal processing algorithms, messaging and control logic, and hardware device interfaces. In the past, many of our projects developed the flight simulation software and the embedded software separately. This approach frequently resulted in integration problems, and implementation issues were not being discovered until late in the development cycle. We transitioned to agile software development to address such difficulties and identified a need to demonstrate and test the flight software much earlier.

As shown in Figure 1, in SiS, the hardware device drivers and embedded software algorithms in our simulated test and evaluation environment are the same ones that we fly. Figure 2 illustrates how SiS allows us to use simulated hardware components and environments so that

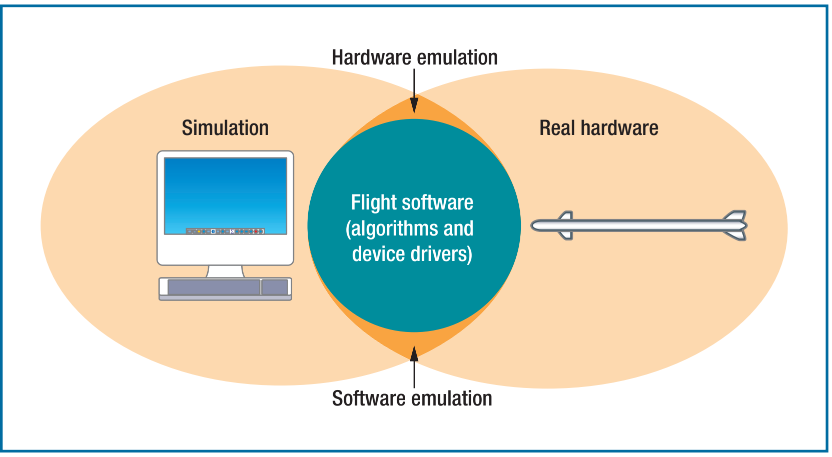


FIGURE 1. Software-in-simulation (SiS) architecture. The hardware device drivers and embedded software algorithms in our simulated test and evaluation environment are the same ones that we fly. SiS allows us to use simulated hardware components and environments so that software and hardware development can be conducted in parallel.

software development and hardware development can be conducted in parallel. The flight software undergoes no modifications before being cross-compiled for execution on the embedded processor.

Agile Supports SiS

Agile software development is a set of methodologies emphasizing

- frequent delivery of the most valuable working software,
- close collaboration and frequent communication,
- self-organizing teams that learn as they collaborate, and
- mechanisms to handle changing requirements without crisis.

While simulations containing embedded software need not be developed in an agile manner, Scrum's agile framework helps realize greater benefits from a SiS approach. One Scrum event is the sprint review, in which the development team demonstrates what was accomplished during the sprint. It can be challenging to have something visual to demonstrate with embedded software development as there is often little to "see." We might get only a blinking light or a wiggling fin. Developing software in simulation helps everyone see the missile flying and track trends in performance. When such feedback is used in the sprint review as well as daily collaboration, these collective learning opportunities allow more nimble responses to necessary changes in requirements and design.

Historically, keeping track of the myriad stakeholder concerns during a rapid development cycle has been a significant challenge. We have learned the hard way that integrating large volumes of work developed by independent groups can be time-consuming and painful. Changes to a single component often impact multiple technical concerns, and frequently multiple developers might edit the same components concurrently for different reasons. When such changes are made by isolated subgroups and then merged into the whole in batch form as large integration events, separation of concerns and traceability of change effects can become insurmountable challenges. Practicing continuous integration within each two-week sprint and following the agile principle of close and frequent collaboration between stakeholders helps us develop software without getting stuck on such integration issues. All of the contributors to the embedded software use a common simulation as the tool to develop and test their changes. This helps us avoid the drawbacks of big-bang integration, where large change sets are brought into the software baseline as discrete events. SiS allows all system contributors (simulation and embedded software developers, algorithm developers, performance analysts, hardware developers, and systems testers) to operate in the same, continuous-integration environment. It can get hectic in this shared environment—where there are lots of moving parts—but it is worth it because we are able to integrate smaller pieces more rapidly and thereby have more frequent and focused learning opportunities regarding our system.

For example, we recently integrated the device communication layer into one of our simulations. When we flew this software in the simulation, we immediately saw the missile spin out of control and crash into the ocean. In a short time, the issue was traced to a misprinted acceleration sensor scale factor in the hardware documentation. This resulted in the navigation software being unable to calculate how fast the missile was spinning or which way was up. Fortunately, we caught a potentially catastrophic flight failure early in the development timeline at near-zero cost. After confirming

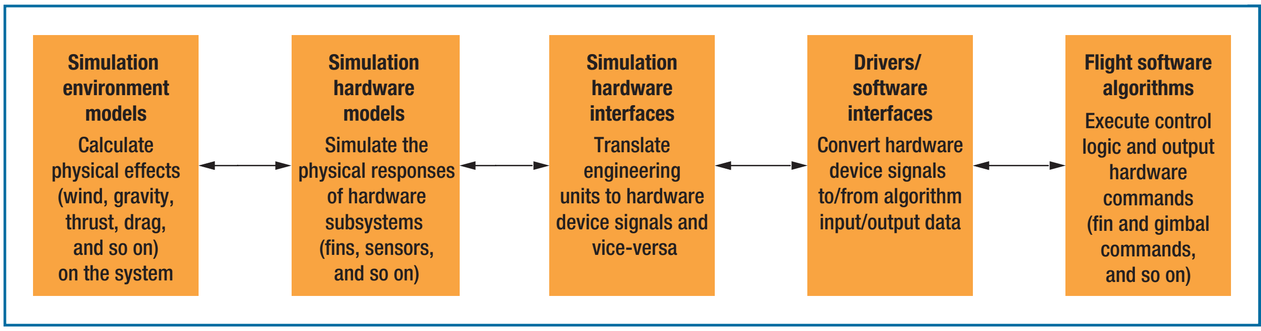

Simulation environment models
Calculate physical effects (wind, gravity, thrust, drag, and so on) on the system
Simulation hardware models
Simulate the physical responses of hardware subsystems (fins, sensors, and so on)
Simulation hardware interfaces
Translate engineering units to hardware device signals and vice-versa
Drivers/ software interfaces
Convert hardware device signals to/from algorithm input/output data
Flight software algorithms
Execute control logic and output hardware commands (fin and gimbal commands, and so on)

FIGURE 2. Layered view of SiS architecture. Environmental models stimulate hardware models, which send data through interface objects to the embedded software. The algorithms in turn issue commands which travel back through the hardware interfaces to control the simulated hardware.

with the hardware supplier that the device documentation was indeed wrong, we modified our embedded software. In less than a day, we were back flying smooth.

SiS Supports Agile

Prototyping can be difficult to implement in early development, when the physical assets needed to construct a prototype system do not yet exist. SiS facilitates earlier development of functioning embedded software by providing digitally modeled components. This allows us to write the device drivers and algorithms and to observe their impact on the system before having the actual devices.

Simulation provides virtual hardware for developing and testing our software. As hardware becomes available, the corresponding hardware models are validated by comparing measured system and subsystem responses to simulated responses to assure that the simulation models accurately represent the hardware. Furthermore, the virtual hardware provided by the simulation extends the testing environment throughout the project's lifespan. In some cases, this simulated virtual hardware is absolutely necessary due to cost and safety issues. It would be problematic if we had to fire a real rocket booster during each launch sequence in integration and test.

A central tenet of agile development is to learn as we go and to incorporate that learning in the next iteration. Using SiS, small increments of software are developed and the results of the changes are observed and analyzed, helping us manage complexity and write better software. Areas of the system that have not yet been developed can be stubbed out with simplistic simulation components and then replaced and updated iteratively. We can shorten our integration cycles to as little as a few minutes using the simulation. SiS also facilitates concurrent hardware development and integration because we can swap hardware components such as a field-programmable gate array (FPGA) or embedded processor onto a known working software configuration and test the full end-to-end effects on the system. SiS allows us to integrate continuously, constantly exercising the embedded flight software, giving us higher confidence and more accurate system performance predictions.

Simulation performance predictions aid in achieving the agile goal of responding to changes in requirements quickly. In one case, while developing a system with a motor that utilized a new type of fuel, our supplier discovered—one year into the project—that it was unable to create a usable form of the new fuel. Without a usable fuel, there could be no product, and the whole project was in jeopardy. The design had to change. We considered several alternate forms of fuel from multiple suppliers. Using the simulation, we showed the impact that each new fuel would have on system performance. These timely results provided our technical leadership with the data needed to rapidly adapt the design and continue working toward production.

Regardless of the procedure used to develop a new software-intensive system, the process can be decomposed into the phases of requirements development, design, code, and test (see Figure 3). Simulation feedback is used throughout the process, adding new and earlier paths between design, development, and test. This helps close the loop between the iterative phases, which is particularly important in agile development because all phases must occur—to some degree—during each two-week sprint.

SiS also provides a great way to

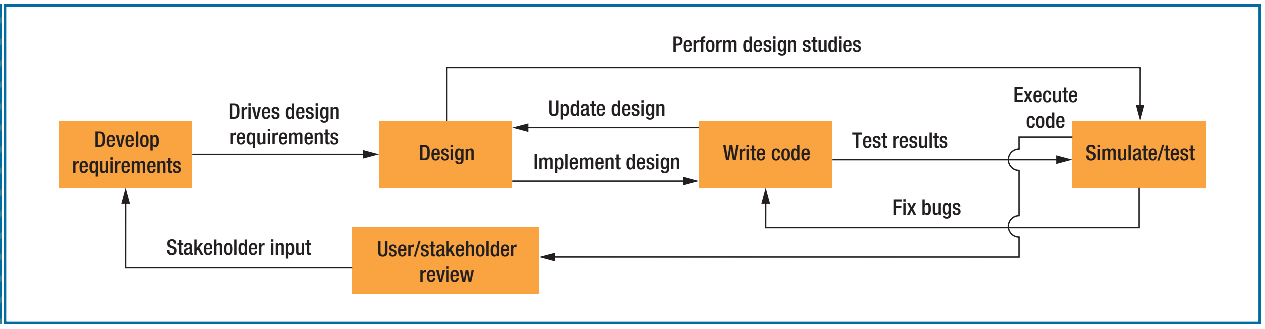


FIGURE 3. Requirements and design are updated from simulation prototyping. Simulation feedback is used throughout the process, adding new and earlier paths between design, development, and test, which helps close the loop between the iterative phases.

achieve the agile objective of frequently demonstrating product updates to users and other stakeholders. Details regarding hardware-driver code maturity, internal message-passing operations, or processor-loading optimization are often too low-level to be demonstrated independently. However, exercising these embedded capabilities in a system-level simulation provides a more tangible feel to the product. Through simulation, users can see how the embedded software and algorithms are flying the missile toward its target.

Letting Simulation out of the Box

When initially exploring system designs that advance current technology, understanding of requirements can be notional at best. As we begin implementing these requirements, we discover unforeseen design constraints. Agile development frameworks such as Scrum expect and plan for change to accommodate the inherently iterative nature of requirements development. Our use of Scrum facilitates the maturation of our requirements with minimal to no crisis because it occurs during sprint planning.

Through continuous prototyping, developing, and testing, the agile SiS methodology allows us to update our requirements iteratively, feeding our Scrum product backlog between successive sprints. Simulation provides a means for exercising the skeletal system to monitor and assess system properties and performance as the system matures. Model fidelity is enhanced iteratively as needed, providing increasing levels of detail and accuracy over time.

Simulation development is a microcosm of the overall system development process shown in Figure 3. Strict discipline is required to establish and maintain the simulation's credibility to suit the evolving needs of system developers. System and subsystem requirements must be stated in a way that enables verifying that they are correctly implemented in the simulation. The more stringent the simulation requirements, the more effort and expense are required to develop sufficient fidelity within the simulation. Reference data from real-world tests must be obtained in order to evaluate whether the simulation is sufficiently accurate for its intended use, which continuously evolves. Simulation accreditation efforts can incur additional time and expense. In SiS, simulation changes should be accompanied by ongoing regression testing to assess the health of the simulation and to detect any change in the predicted system performance.

While considerable cost savings can be obtained through simulation, creating and maintaining a reliable simulation environment is not free. Before building a simulation, the savings in development cost and risk must be weighed against the costs of simulation development. In cases where hardware is readily available and well suited for the required testing, the costs of simulation development might not be warranted. This might occur when developing software for an existing, easily accessible hardware asset, such as a cell phone.

The special constraints of the avionics domain have led us to use simulations in nontraditional ways. Unlike field-testing cell phone or automobile software, when we field-test the embedded processor on a missile, we don't get it back. By using SiS to perform virtual flight tests, we avoid the prohibitive cost of flights and preserve our hardware assets. Scrum theory is founded on making decisions based on known experiences. We justify procuring expensive and long-lead-time hardware components using knowledge acquired from simulation to determine whether one system design performs better than another. As hardware components become available, we use them in tests to validate—not replace—virtual simulated components. We update the simulation with measured data from hardware, and we continue to use virtual hardware everywhere we can, freeing up hardware assets to be used by others. By the time the hardware components of the system start to resemble a missile, the simulation has matured to the point of being instrumental in preparing for flight tests.

In your business, you will inevitably be asked to deliver a software-intensive product in less time, with higher quality, and with greater reliability. Here is how you can get there:

- Build your reusable resources using architectural patterns that will help create a system that is easy to modify, scale, and loosely couple the embedded and simulation software. This will allow you to build up the system iteratively.
- Use agile practices, such as the Scrum framework, which will allow you to adapt to changing requirements that result from the inevitable learning that occurs when designers observe working software.
- For the engineers who are developing the simulation, software, tests, and even hardware, establish a shared development environment featuring a common repository, simulation, and test suite.
- Think of simulation not as a destination or side activity but as an integral part of your development process, to be utilized continuously throughout the product development life cycle.

Using these techniques, we have been able to meet aggressive deadlines, demonstrate value to our customers, and satisfy our flight test objectives. With simulation-based agile software development, you too can achieve success developing your embedded software.

ABOUT THE AUTHORS

JASON ARD is a simulation lead and software architect in the Performance Simulations Department at Raytheon Missile Systems. He is a certified ScrumMaster and SEI Software Architecture Professional. Since joining Raytheon in 2007, Ard has developed integrated flight simulations, computer-in-the-loop simulations, image-processing algorithms, and software architectures on numerous programs. He received BS degrees in applied physics and in computer science from Brigham Young University. Contact him at Jason_M_Ard@raytheon.com.

KRISTINE DAVIDSEN is a senior principal software engineer at Raytheon Missile Systems. In her 14-year career, she has worked in test, distributed systems, and embedded software development. Davidsen received a BS in mathematics from the University of Arizona. She is a certified ScrumMaster, a member of the Society of Women Engineers, and a member of IEEE. Contact her at kldavidsen@raytheon.com.

TERRIL N. HURST is an Engineering Fellow in Raytheon Missile System's Modeling, Simulation, and Analysis Center in Tucson. His specialties include design and analysis of simulation experiments and simulation verification and validation. Prior to joining Raytheon in 2005, Hurst worked for 27 years at Hewlett-Packard Laboratories on data storage technologies and systems. He received a PhD in mechanical engineering from Brigham Young University. Contact him at terril.hurst@raytheon.com.