Embedded Software

Kevin Thompson, Ph.D. www.kevinthompsonphd.com

The worlds of hardware development and software development merge when companies develop integrated products that contain both hardware and software. We see this in high-technology products ranging from personal fitness-trackers to rockets.

While some of the software that is associated with such devices can be external, much resides in the device itself. Broadly speaking, this internal software can itself be application oriented or provide the low-level interfaces between the internal software and the physical device. The latter is usually referred to as embedded software, or firmware.

Development of firmware poses challenges that go beyond those of application software. The engineers must address the fine details of the hardware and the specific chip sets and device drivers required for the device. Debugging may require logic analyzers, multimeters, oscilloscopes, spectrum analyzers, and other equipment as much as it does setting breakpoints and observing variables in the code.

Another difference between internal software development and application development is the use of Real-Time Operating Systems (RTOS). An RTOS differs from an OS like Windows or Linux in that it guarantees a response time to requests. This is a critical characteristic for environments where milliseconds of response time spell the difference between success and failure. Examples include automotive safety products, sensors, drilling equipment, avionics systems, and radiation-monitoring equipment.

The tight integration of firmware with hardware requires careful management of their concurrent development. Fortunately, Agile techniques and frameworks (such as Scrum) make the management of this concurrent development easier than classic project-management approaches, such as the Waterfall process. The emphasis on effective team definition in Scrum leads to an effective solution for concurrent hardware-firmware development, as shown below.

One approach to developing and testing firmware is to have a firmware team that is responsible for developing and testing firmware, and a hardware team that is responsible for developing the circuitry that will host and execute the firmware. At some agreed-upon milestone, the prototype circuitry is made available to the firmware team so they can develop and test their firmware. The engineers on the firmware team develop and test the firmware and likely discover some problems with the circuitry in the process. They file a defect ticket describing the problem, so that the hardware team can follow up to rectify it.

The problem with this model is that it introduces delays. There is a delay between the initial circuitry development and the handoff to the firmware team. There is another delay between the defect ticketing until the hardware team can address the problem. If the problem requires multiple back-and-forth handoffs, delays accumulate and become a problem for the overall schedule.

It is better to have firmware engineers on the same Scrum team as mechanical and electrical engineers. In this model, one of the firmware engineers begins work with the actual circuitry as soon as it is able to

host and execute the firmware. Any defects discovered in the circuitry can then be handed back to the hardware engineer on the spot to be addressed immediately. This tight feedback loop gets the hardware-firmware combination developed without any handoff delays.

This approach provides for maximum collaboration, and earliest discovery of integration errors. The latter is critically important, because late discovery of integration errors is a large source of cost and schedule risk in product development.

This scenario is an example of a more general concept, which is that interactions that are frequent between individuals should be addressed by putting both individuals in the same team, with the understanding that they will work together to complete their mutual work items as quickly as possible. This is especially true if their work is likely to involve some back-and-forth handoffs (i.e., their interactions are tightly coupled).

In contrast, interactions that are infrequent can be handled by individuals who are on different teams. Handoff delays and some back-and-forth interactions may still occur, but they will be infrequent, and easier to tolerate.