Automotive embedded development and testing

Jon Quigley - December 15, 2008

The automotive world, which includes commercial and personal vehicles, developed its own approach to product and process development. Embedded-system developers who work in this environment use a different set of tools and answer to a different set of expectations from those of, for example, developers in the medical or aircraft industries.

The automotive industry uses as its principal framework for development and production APQP (advanced-product-quality planning), which the AIAG (Automotive Industry Action Group) defines. The phases are typically, but not exclusively, conceptualization, product design and development, process design and development, and product and process validation.

The APQP approach is a high-level “waterfall” model. At no point are there any restrictions that would prohibit spiral or agile approaches to software development. The manual does specify a minimum number of deliverable documents and activities. It does not specify a limit to the amount of documentation, although the formality of the PPAP (production-part-approval process) requires a submission—in full form—of at least 18 and often many more documents.

The higher-level automotive-quality standard, ISO (International Organization for Standardization)/TS 16949:2002, is a superset of the ISO 9001 quality-system standard with an automotive spin. It does not explicitly prescribe APQP, but APQP aptly meets the requirements of the standard.

Refining the approach

During the concept phase, the development team defines and plans the development program. This phase of APQP probably has the largest quantity of soft deliverables. Initially, the embedded-development team solicits the voice of the customer in an appropriate format. Some organizations may use quality-function deployment, whereas others may use proprietary approaches to capturing customer desires and needs. In many ways, this phase is the most crucial because incompetence during the opening game will almost certainly result in failure in the end game. In no way should the development team override the voice of the customer with the voice of engineer unless a safety issue is involved; the team can, however, make recommendations.

The APQP framework recommends that the team secure benchmark data. Getting this information from competitors is problematic at best and unethical at worst. However, it may be possible to research the trade journals and similar work at noncompetitors and put together enough information to set performance targets. Additionally, the development team can use internally generated data for the same reasons. Performance targets will allow the team some modicum of reality when estimating
schedules and milestones.

By the end of this phase, the development team will have defined design, quality, and reliability goals; will have developed some idea of the hardware platform; and will have secured formal management support. In short, the team will march into product and process development with a firm footing in reality and a clear understanding of where it is going.

The software-development team should produce a catalog of desirable and undesirable behaviors in the form of a software-requirements specification. The software-specification document, which can be a FAST (function-analysis-system-technique) diagram, is in many ways more important than software-design documentation. The test group and the designers measure the performance of the software against the requirements rather than the design.

APQP organizations tend to develop both products and processes simultaneously or nearly simultaneously. Sometimes, the process design and development starts after the product design and development commences; otherwise, they run in parallel.

The enterprise can manage product design and development using whatever development model it favors. However, the automotive approach has some unique elements. Some organizations prefer a requisite amount of design documentation; however, any well-qualified software engineer recognizes that the code itself is the most up-to-date instantiation of the design. In general, the difficulty with design documentation lies with the divergence between code and documentation almost immediately after the engineers begin writing the actual code.

One of the most powerful tools in the automotive approach is the DFMEA (design-failure-modes-and-effects analysis). Using DFMEA, cross-disciplinary teams of development engineers analyze the functions of the product using a logical approach—for example, FAST—and develop a table that captures failure modes, causes, effects, and qualitative values related to these events. The tool works best when the teams use it to solve design issues. The embedded-software-development team can also use the DFMEA with some forethought: Software becomes complicated exponentially, leading to astronomical requirements for test cases. The DFMEA is a design tool for upfront thought on potential problems, so it is still valid for the software engineers to use.

The embedded-software-development team will also need to consider design verification. The automotive world uses a summary document called a DVP&R (design-verification plan and report) to capture all of the activities that verify whether the design meets or exceeds requirements. The team should derive the activities in the DVP&R from the DFMEA, which has a detection column for this purpose. A minimal design-verification plan should include verification that the software meets customer baseline requirements; stimulation of every input to the firmware using combinatorial approaches, including pairwise testing, three-wise testing, designed experiment arrays, and extreme product-destroying tests; and stochastic testing to present the software with unexpected conditions.

Furthermore, it is common to acquire actual vehicle components, including wire harnesses, to be able to test the software with hardware-in-the-loop simulation. The improved verisimilitude increases confidence that the results are relevant to the final product.

The development team should not ignore the possibility of using one or more software-based simulators to provide the appropriate signals or messages across one or more buses. The team could even use older products with specially developed software to simulate brake controllers, engine-control modules, transmission controls, and other devices in modern vehicles. Other approaches include the use of commercial products, such as The MathWorks’ Matlab/Simulink (Figure 1), National Instruments’ LabView, and other tools that can communicate directly with data buses,
analog-to-digital controllers, digital-to-analog controllers, digital-to-digital controllers, and various serial and parallel ports.

If the software team is fortunate, it may be able to test its work with prototype designs using ICEs (in-circuit-emulators), which allow for hardware, software, and interaction assessment during verification. The use of prototypes is highly desirable but can be expensive. Simulation will allow for some level of testing before a prototype becomes available.

IV&V (independent verification and validation) brings a level of honesty to the testing process. It is not so much that the development is dishonest but rather that the developers understand too well how their software works. An IV&V group develops its own understanding of customer requirements and builds its own test-specification documents. Alternatively, the enterprise may prefer to have a third-party organization provide this function.

The automotive-product-design process requires design reviews; however, the manuscript does not prescribe the frequency of these reviews. Experience suggests that relatively frequent, short reviews add more value and reduce risk better than do infrequent, long reviews. The individual enterprise may have a launch process that specifies the exact number of formal reviews. The software team can always implement more product reviews than the launch process specifies.

Because the embedded software will reside on the product as firmware, the development team must consider how to implement its software on the product and, further, how it will coordinate software activities with the manufacturing process.

During this quasiparallel phase, the team decides how to program the microcontroller and how and when to release the software to production. It also coordinates with the automated-test-equipment group to make sure the expected and unexpected behavior of the product is well-understood. The software-development team can also make life easier for the manufacturing people by adding, for example, built-in self-testing, which simplifies life for the automated-test-equipment group; self-calibration features if the product has gauges or some other device that provides visual indications of a measurement; and boot-loading capability, so the team can “reflash” the product if the hardware has flash memory.

Product validation is not equal to design verification, although, in practice, the test suites are often similar. The purpose of product-validation testing is to assess whether the product meets engineering standards the teams set during the product-and-process-development phases. Because validation normally occurs during the end game, the developer should no longer be performing design verification.

Process validation is decidedly different from product validation. The expectation is that you use the final process, the production-ready materials, and the launch software release and that a PCP (process-control-plan) automotive document describes all other activities. The PCP details how the manufacturer will fabricate the product—from measurement values to reaction plans to process capability, an index of quality.

**Corrective action**

The APQP shows corrective action occurring throughout the development and launch, which makes sense; issues requiring formal correction occur from the start of the project, up to launch, and sometimes afterward. In the automotive world, developers frequently document corrective action using the 8D (eight-disciplines) model.
The 8D model is a rational approach that provides steps such as emergency action, containment action, and irreversible corrective action. An emergency action involves stopping the manufacturing line if, for example, the verification team or the software team determines that the software has a safety issue. A containment action occurs when you can detect the problem and sort out the bad units. For example, when you have captured a marginal hardware design and weak firmware, some of the components may still be good enough to ship if you properly test the unit. Irreversible corrective action occurs after deep probing to find what is usually called root cause. Eliminating the root cause eliminates the problem. Sometimes, containment is the best you can do. Each action has a verification component to ensure compliance.

The embedded-development team can use all of the 8D components as a means of formally documenting corrections. The key idea with this approach is the elimination of the problem's cause.

The feedback component of APQP involves the assessment of customers’ reactions to the product, measurement of issues that arise, product and process changes, and lessons learned during development and launch. The APQP process allows learning to occur naturally, and capturing errors and potential errors becomes a way of building a culture of prevention.

Other components of the feedback system are risk assessment and mitigation. Experience suggests that compressed schedules will lead to software errors and that customer dissatisfaction will follow. One approach that incorporates feedback as well as risk management is the TEMP (test-an-evaluation-master-plan) technique, which the US Department of Defense employs. In this approach, the customer and developers agree ahead of time to deliver a sequence of software packages. Each new software package is a superset of the previous package, and each package is fully functional. This method helps restrict the errors to the new software and simplifies the testing activity, because the test function will need to run a regression suite only on the previous software package to verify that no detrimental interactions occurred during development. If this approach sounds to you like a precursor to agile-software development, you’re right.

**Relevance elsewhere**

The PCP approach is suitable for any sequence of steps in a process and provides a framework for documenting requirements, particularly those involving measurement. It is similar in concept to the HACCP (hazard-analysis-and-critical-control-point) process-flow documentation that the food industry uses. You can also use the PCP to document a software-development process. The development team would then apply a PFMEA (process-failure-modes-and-effect analysis) to assess the need for added controls on the process.

Both design and process FMEA tools aim to eliminate product and process problems before they occur. Prevention is generally cheaper than detection and correction, not to mention the wasted time that prevention eliminates.

The reviews, the production-part-approval process, and all the other automotive documents help the development teams assess their work so they can release a product they can be proud of. In general, some process is better than no process. The APQP framework provides a general process without being heavily prescriptive when it comes to software development—allowing automotive companies the flexibility to evolve systems that suit their needs and still fitting within the overall model. Use of
the APQP model for development also has the benefit of providing a common vocabulary and similar metrics and expectations from one enterprise to another.