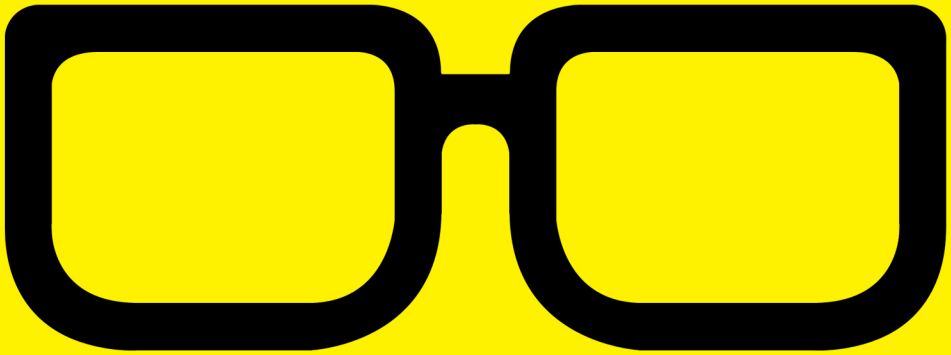
SPONSORED BY

IBM

GEEK GUIDE



Agile Product Development

LINUX™ JOURNAL

Table of Contents

TED SCHMIDT is the Senior Project Manager and Product Owner of Digital Products for a consumer products development company. Ted has worked in Project and Product Management since before the agile movement began in 2001. He has managed project and product delivery for consumer goods, medical devices, electronics and telecommunication manufacturers for more than 20 years. When he is not immersed in product development, Ted writes novels and runs a small graphic design practice at http://floatingOrange.com. Ted has spoken at PMI conferences, and he blogs at http://floatingOrangeDesign.Tumblr.com.

GEEK GUIDES:
Mission-critical information for the most technical people on the planet.

About the Sponsor

IBM

IBM is a globally integrated technology and consulting company headquartered in Armonk, New York. With operations in more than 170 countries, IBM attracts and retains some of the world's most talented people to help solve problems and provide an edge for businesses, governments and non-profits. IBM is focused on five growth initiatives: Cloud, Big Data and Analytics, Mobile, Social Business and Security.

There are more than 9 billion connected devices operating in the world today, generating 2.5 quintillion bytes of new data daily. Cognitive systems help overcome this challenge—learning at scale, reasoning with purpose, and interacting with humans naturally. In March 2015, IBM announced that it intended to invest more than $3 billion to address the needs of organizations that are looking to capitalize on the increasing instrumentation and interconnectedness of the world driven by the Internet of Things (IoT).

Innovation is at the core of IBM's strategy. The company develops and sells software and systems hardware and a broad range of infrastructure, cloud and consulting services. IBMers are working with customers around the world to apply the company's business consulting, technology and R&D expertise to enable systems of engagement that deliver dynamic insights for businesses and governments worldwide.

Agile Product Development

TED SCHMIDT

Introduction

Every day, a new Internet-connected gadget hits the market, promising to make everyone's lives incrementally more convenient. Devices like Fitbit and Sen.se Mother help monitor your health. Trackdot and Bikn let you track your possessions through GPS. Delphi Connect, Nest, SkyBell and LIFX all make it possible to control home appliances from anywhere. By 2020, Gartner forecasts that there will be more than 25 billion of these "things" connected to the Internet. And every day, consumers are demanding more from these things: more features, more interconnectivity, more convenience.

Unfortunately, this makes the life of a systems engineer (SE) much less than convenient. In this new world of an Internet of Things (IoT), traditional, waterfall product development processes fail. They fail to detect defects early. They fail to avoid rework. They fail to provide fulfillment to those using them. Ultimately, waterfall product development processes fail to provide consumer value.

The problem is that the rate of change is so high, and the complexity and integration of the IoT is so great, that waterfall approaches simply don't work anymore. By the time a product is developed using a waterfall approach, it's already obsolete. What's the answer? Agile is the answer.

What comes next is a look at some of the product development challenges posed by the IoT and how taking an agile approach to product development helps better manage those challenges. I discuss agile and the potential benefits offered by taking an agile approach to product development. And finally, I explore some of the processes and methods you can implement, so you actually can realize the benefits of agile as you develop products in this new world of IoT.

Challenges Facing Product Development

Current Challenges of IoT: There was a time when, if the temperature in your home was uncomfortable, you simply walked over to the thermostat on your wall and adjusted it to your liking. Sounds pretty convenient, right? Well, what if you travel for a living or simply don't want to heat or cool your home for the 10–12 hours every day that you are away? The idea of keeping your place comfortable while

The IoT offers great opportunity. But as with any great opportunity, in order to realize the benefits fully, some challenges need to be overcome.

you aren't there may start sounding a little expensive. If you're like me, you probably set your thermostat to use less energy while you're gone and then adjust it when you get home. The problem with this approach is that you have to wait for the temperature to stabilize after you return home—not so convenient.

But, what if your thermostat knew exactly when you were going to walk in the door, the temperature you preferred and how long it would take for the temperature in your home to stabilize? Or better yet, what if you could tell your thermostat when you were going to be working late? All you'd need to do is figure out a way to connect your mobile phone (because it's the one thing you always have with you) to your thermostat. Now, that's convenient, and smart, and it saves energy. That's the Internet of Things.

The IoT offers great opportunity. But as with any great opportunity, in order to realize the benefits fully, some challenges need to be overcome. In the example of the smart thermostat, the most obvious challenges come from complexity and integration.

Say you create a mobile app that tells the thermostat when you're 15 minutes from home, and that you would

like the temperature to be 72° when you arrive. Even though you probably carry only one mobile phone (er, maybe not), not everyone carries the same phone, so the mobile app needs to run on multiple mobile platforms. It needs to work over both Wi-Fi and cell networks. It has to be upgraded and tested every time the mobile OS is upgraded. The thermostat has to be connected to the Internet. It has the same software considerations that the app does. So, you're not just talking about a mechanical device to regulate the temperature in your home; you're talking about a highly complex, tightly integrated system with thousands of lines of code, and multiple mechanical and electrical design considerations.

Now let's say just as you get your heads around all the complexity and integration demands of this smart thermostat system, your customers decide they want their lighting controlled by the system as well. Before you've even finished designing the smart thermostat system, it has changed into a smart home appliance system. Maybe you should integrate the security system too? How about turning the oven on, or recording TV shows on the DVR? How can you actually hope to build anything with all the complexity, integration and change constantly happening?

A Look at Systems Engineering: When thinking about IoT, the need for a holistic, multidisciplinary approach to product development and management seems fairly obvious. That's where systems engineering comes in. Because systems engineering takes a broader perspective of the product, you are much more likely to pick up on potential issues and risks earlier in the process. But even

using a systems approach, product development in the new world of IoT is not without its challenges.

One of the great benefits of taking a systems engineering approach to product development is the creation of requirements specifications that span the entire system. In the case of a product like the smart thermostat example mentioned here, systems requirements specifications (SRS) would include not only how the thermostat should work, but also considerations regarding network, mobile devices, software and so on. If you use a waterfall approach in this smart thermostat system scenario, by the time the requirements got through functional and dependability analyses, the creation of a systems architecture and into the hands of subsystem teams, they likely would be obsolete. Even with management tools like RequisitePro and Rational DOORS, the opportunity for errors and omissions exists.

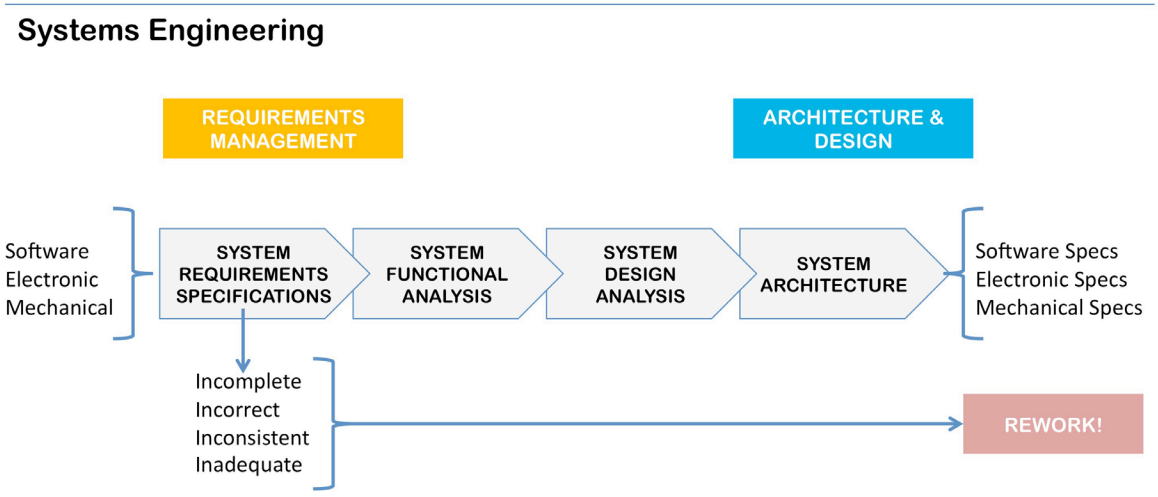


FIGURE 1. The Weakness of a Waterfall Approach

Precisely because of its cross-discipline reach and its attention to detail through multiple levels of analysis, systems engineering absolutely helps you manage the issues of complexity and integration. But applying systems engineering in a waterfall method leaves the challenges presented by the rapid rate of change unaddressed. You can have the best systems engineer in the world, but if you have to wait six or even three months to get specs into the hands of subsystem teams, you've missed the boat and failed to deliver customer value.

Agile Background and Benefits

Agile is a concept that's been around for a while. Until recently, however, it has been limited in its application to the world of software development. I'm not talking about software development here; I'm instead focusing on the ways that agile thinking can be applied to systems engineering and product development, but it's important to understand a few, basic agile concepts.

Agile's main premise is the idea of delivering customer value in small, demonstrable pieces of a larger solution inside short timeframes. It relies heavily on close collaboration between cross-functional development teams and stakeholders to ensure there is a constant feedback loop. This constant feedback loop of small increments of product, combined with an empowered technical team, is where the agility comes in. Think back to that waterfall systems engineering model. If you could manage to get feedback on the compliance and, more important, the veracity of your requirements and engineering data from all the different

Being agile means staying close to the customer, and continually integrating and verifying requirements in small increments.

engineering teams at the beginning of the process, instead of at the end, you could save a lot of expensive rework.

Ultimately, the reason for taking an agile approach is to support rapid delivery of high-quality products that are tightly aligned to customer needs and company goals. Being agile means staying close to the customer, and continually integrating and verifying requirements in small increments.

I've discussed the challenges that IoT presents for product development in the forms of complexity, integration and change. I've also promoted the idea that systems engineering is an incredibly valuable asset in managing the challenges of integration and complexity, and I've argued that using a waterfall approach leaves you vulnerable to the rapid rate of change introduced by the IoT. I've introduced agile and acknowledged some of the benefits offered by taking an agile approach to systems engineering. Now, let's look at applying agile concepts within the context of product development and see just how becoming agile can help you respond to the challenges offered by the world of IoT.

Applying Agile to Systems Engineering

As I mentioned previously, the agile processes and methods that currently exist were developed for software

development. As such, they really can't be applied directly to systems engineering. It's one thing to deliver working software features in two-week increments. It's an entirely different matter to deliver working hardware, much less a complex working system, in two-week increments. Besides, the real output of systems engineering is specifications (software, electronic, mechanical), not the end consumer product. This is where the concepts of iteration, continuous integration and verification in small increments can be applied: to the work products of systems engineering.

In order to apply agile thinking to systems engineering, it is important to focus on the problems you are trying to address—complexity, integration and change—as well as the key agile concepts—stay close to the customer, do only what's valuable, deliver small increments and verify constantly. Combine these ideas with the creation of specifications of systems engineering, and you'll reap the benefits of agile.

The PO Is the New SE: In product development organizations, not only are the systems engineers responsible for managing requirements across multiple technical disciplines and integrations, but they also are expected to plan the development program, manage risks and understand emerging trends in multiple technologies. Often, they will partner with a product manager (PM) when it comes to interfacing with the stakeholders and understanding consumer sentiments. For the most part, however, the SE maintains a highly technical focus, which is a luxury that simply is untenable in the ever-changing

world of IoT.

In comparison to the SE, the product owner (PO) in agile is an amalgamation of the SE and PM roles. Like the SE, the PO is responsible for managing requirements. In agile, requirements often are referred to as "features" and maintained in what's called a "product backlog". The product backlog is a constantly evolving list of requirements that are sorted and resorted according to business and customer value. Compare this to a system requirements specification in a waterfall world, which is completed, approved and handed to the next stage in the process. By the time the SRS is fully consumed, it's probably full of errors and things that simply aren't needed anymore. This results in costly rework (known as "technical debt" in agile) and lost value.

FIGURE 2. How the SE Role Changes with Agile

If the SE is going to maintain a product backlog that constantly is changing based on business value and consumer sentiment, it means the SE now has to be closer to stakeholders and consumers. What used to be the exclusive realm of the product manager now belongs to the SE in the world of agile. Brace yourselves, engineers. You're going to have to work hand in hand with marketing, sales and even embrace social media. The bottom line is the SE now has to understand the technology as well as the people who use it.

Focusing on Value by Managing Requirements: Agile thinking focuses on delivering value to the customer, as well as to the business. In a systems engineering world, this means an end to documentation for documentation's sake. Of course, the amount of documentation created for a product will depend on the product itself and the industry and its associated regulation. But the point is to reduce waste—do only that which is necessary—meaning that it adds actual value.

Focusing on value also means valuing the SE's time. Time spent by an SE generating documentation solely for compliance is not time well spent, especially when the SE is now expected to spend more time getting closer to the business and the customer. Leveraging concepts like requirements traceability means looking at ways to auto-generate non-value-added (but required) documentation, like fault matrices and other summary analysis documents, when they are required.

It's absolutely necessary to have requirements traceability across the systems engineering process. My favorite

Using a tool like IBM Rational DOORS, an SE can create traceability, using drag-and-drop functionality as requirements are created or changed, and see the links between stakeholder requirements and system requirements, as well as potential gaps.

illustration of this point comes from the *Apollo 13* mission, when it was discovered only during a critical emergency that the air filter in the command module was a different shape from the one in the lunar lander. Fortunately, some heroic engineering saved the day, but the point is that traceability of requirements solves that kind of problem before it ever occurs.

Using traceability, systems engineers can see the ripple effect of changing a requirement. They also can confirm that all requirements actually are implemented by using coverage analysis. Finally, they can see whether the various elements across multiple work products are in sync with one another. Traceability, per se, is not necessarily agile. Incremental traceability is agile. Using a tool like IBM Rational DOORS, an SE can create traceability, using drag-and-drop functionality as requirements are created or changed, and see the links between stakeholder requirements and system requirements, as well as potential gaps.

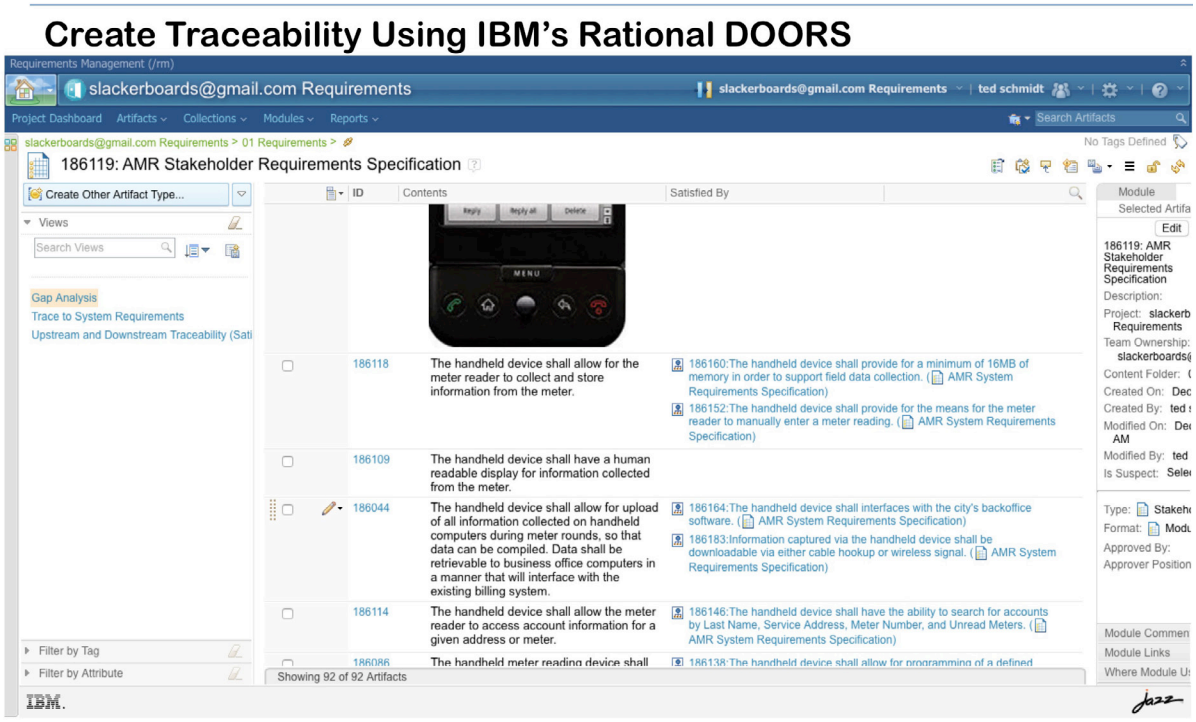


FIGURE 3. IBM Rational DOORS

Consider also the time spent managing changes. Requirements traceability provides systems engineers with the ability to make changes to a product backlog and have that change ripple through not only all the required, compliance documentation, but all the downstream architectural specifications as well. The overhead associated with change management and audit in a waterfall approach is reason enough to think about being agile.

Think back to the topics mentioned earlier in this ebook. The biggest challenge offered by the IoT is its rapid rate of change. The biggest limitation of waterfall is the inability to absorb change as it occurs. And, the greatest advantage of agile (if you couldn't guess it by its name) is precisely its ability to

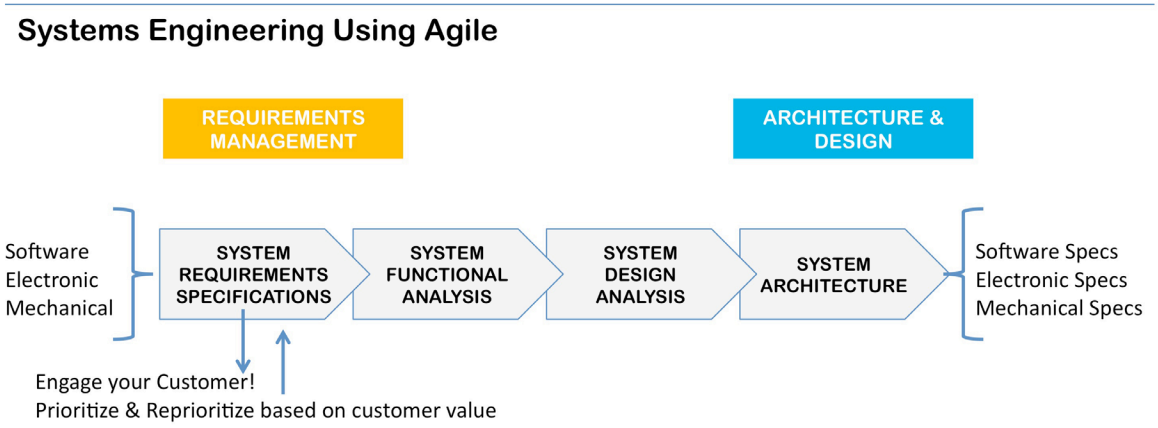


FIGURE 4. Managing Requirements by Grooming the Backlog

shift and adapt to change—changing requirements, changing priorities and changing consumer tastes. If you aren't managing your requirements effectively, you aren't agile.

Tools like IBM Rational DOORS/DOORS Next Generation provide the requirements traceability that enables agile by providing centralized control of specifications and visualizing the linkages and dependencies between requirements. Bruce Douglass also explores this topic in depth in his new book, *Agile Systems Engineering* (https://www.elsevier.com/books/agile-systems-engineering/douglass/978-0-12-802120-0).

Working in Increments: I've touched on the idea of small, incremental delivery several times. In systems engineering, where the output is specifications, this works on several levels. Not only do you build specifications documents incrementally from requirements, but you also strive to group requirements into testable use cases incrementally. From a systems perspective, traceability

becomes very valuable if you want to derive requirements for each sub-team based on a system-wide use case. You also want to develop the system architecture incrementally based on those use cases, rather than making broad architectural assumptions that are riddled with errors. Finally, you want to verify those incremental steps along the way.

The way it works is quite simple. The SE begins by constructing a use case, adding requirements to it and verifying each increment. This is repeated until all requirements are added, and then repeated for all use cases. In a traditional sense, a use case could contain other use cases. In an agile sense, you begin as small as possible, in what's called a "nano-cycle", and then begin adding to it after validation.

Continuous Verification, Simulation and Testing: Remember, one of the problems with the waterfall approach is that requirements aren't validated until months after they are written in textual form. One of the benefits you get from agile is the ability to respond to change earlier by receiving validation earlier and more often. For an SE, that translates into the need to take a use case and model the entire system for just that case. Then, based on the results of the verification, either correct any errors or deficiencies, or incrementally add to the model.

In a waterfall world, requirements are defined, the system is designed, engineers build it and QA validates it. In an agile world, using approaches like test-driven development (TDD), testing is moved to the front of the process to improve quality and reduce rework. In small increments, requirements-based test cases are created. As those test cases are executed and fail, specifications are changed to address only the failures.

The idea is that efficiency comes in addressing only the limitations (failures) of the system. This echoes the idea I discussed earlier about focusing on delivering the most value.

In a typical use case, the actor performs an action, and the system reacts. In the case of systems engineering, use cases are based on high-fidelity models, with specific states, transitions and system actions. By using automated tools, like IBM Rhapsody Test Conductor, systems engineers can create and update highly detailed use cases from new requirements and execute them multiple times a day in nano-cycles.

Test-Driven Development with IBM's Rational Test Director

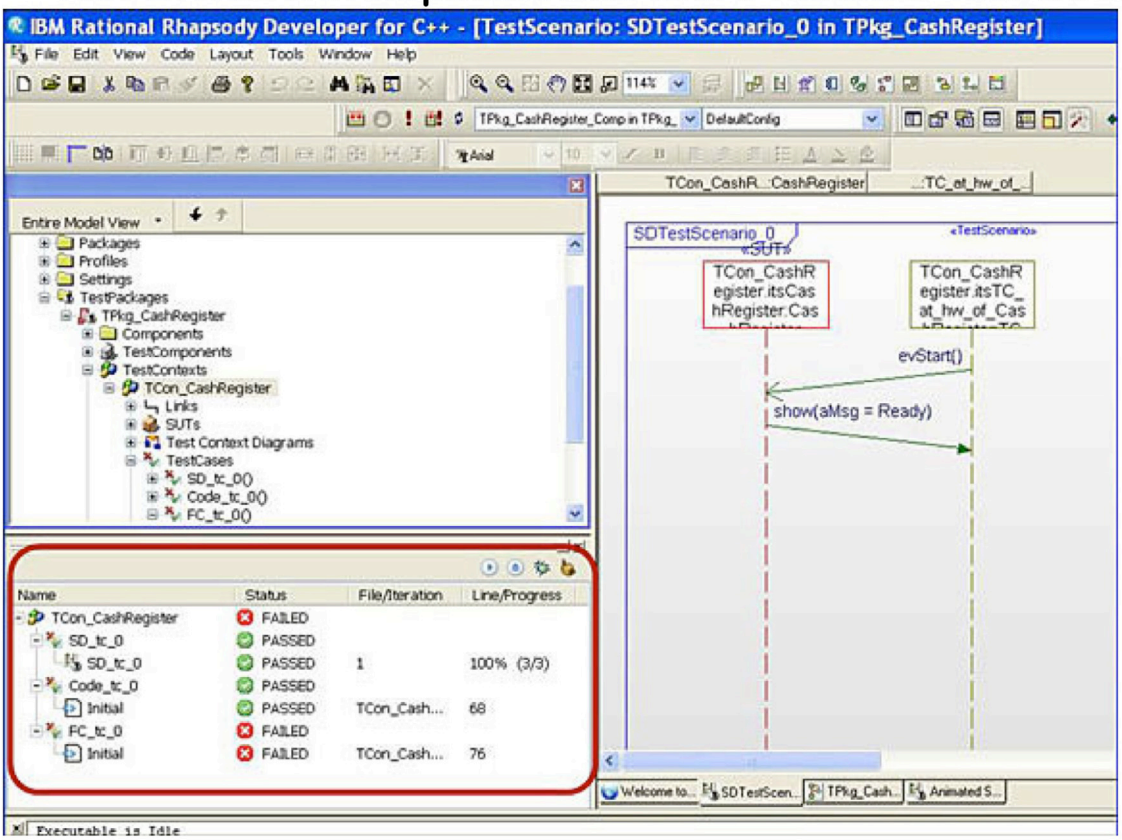


FIGURE 5. IBM Rational Test Director

Using an approach like TDD not only improves quality and reduces rework, it also often reveals the future, because creating small incremental "fixes" to failed tests typically causes other parts of the system to fail as well, thereby revealing future needs before customers even know they exist. This pulls dependability analysis into the systems engineering workflow.

Because it's difficult, at best, to predict future consumer tastes accurately (take, for example, The Segway, Google Glass or Microsoft Zune), the next best thing is to verify that you are moving in the right direction through the creation of executable specifications to simulate system behaviors. Although it's not the same as releasing a working system to customers, it does support the ideas of iterative verification and ensuring quality of your work early and often—certainly it's more agile than waiting several months to find out if you're building a quality product like you would using a waterfall approach.

Modeling: I've already acknowledged that product development is inherently more complex than software development because of all the additional considerations and integrations that have to work. There is simply no way to build an entire system in small, working increments. So the way to solve for this in systems engineering is through modeling.

In modeling, you take requirements, again in the form of use cases, and break them into engineering data. In other words, you model the behavior of the entire system, using diverse engineering data (integration, architecture,

component behaviors and so on) according to specific use cases. It's not a new concept. What makes it agile is that it's done, again, in small increments, and those increments are validated before additional detail is added.

Typically, a good model will describe what it's trying to test, including the levels of precision and accuracy required. It will describe who owns the model and is responsible for validating it. Additionally, it will describe what is included in the model, why and what diagrams are used to support its purpose. It's important to remember that the model is not the diagramming used in support; rather, the model is all the data that goes into

Model in Small Increments Using IBM's Rational Rhapsody

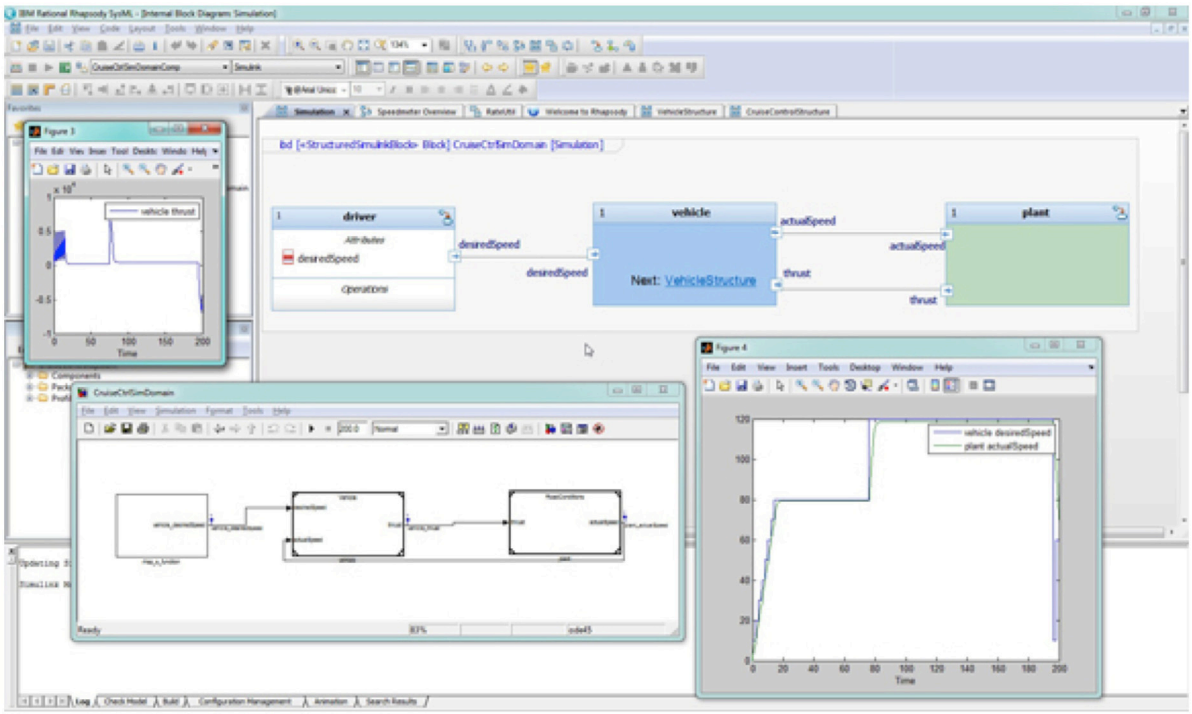


FIGURE 6. IBM Rational Rhapsody 8.0

Tools like IBM's Rational Rhapsody help cross-functional teams work together to produce and validate models using SysML and UML much earlier in the process.

describing the model itself.

Using modeling languages, such as Unified Modeling Language (UML) and Systems Modeling Language (SysML), an appropriate level of detail can be achieved in engineering data to allow this type of incremental modeling of use cases. With this approach, defects are found earlier in the process, during modeling and not later, when rework is hugely expensive. If the same modeling language is used by systems engineers and subsystem engineers, misinterpretation of downstream specifications is greatly reduced, and rework is again avoided. Finally, because modeling is represented visually, it improves overall understanding by adding another dimension of interpretation.

Tools like IBM's Rational Rhapsody help cross-functional teams work together to produce and validate models using SysML and UML much earlier in the process. Again, the idea of modeling complex systems is not really new or agile, per se. What makes it agile is the application of the tool. Validating small increments and then adding on to those validated pieces is where the application becomes agile.

Conclusion

The Internet of Things is a complex, tightly integrated place where change occurs at a dizzying pace. It's easy to see that using a waterfall approach to developing products for the IoT isn't a competitive approach. Change happens too fast. What's needed is a more agile approach, an approach that focuses on delivering what the customer values despite all the complexity, integration and change.

The role of systems engineers is going to evolve to be more engaged with stakeholders and the end consumer. Although it may not be practical to implement agile methods wholesale, you can adopt specific agile approaches and tools to systems engineering and expect several benefits. You can expect to identify and remedy defects earlier, when they are least costly to correct. You can reduce misinterpretation of requirements specifications and focus on delivering value. Through iterative requirements management, you will understand the impact of change and improve traceability across the system and all documentation. Test-driven development and modeling allow systems engineers to verify engineering data precisely in the work products from different engineering teams.

Tools like IBM's Rational DOORS and Rational Rhapsody provide the agile, collaborative environment necessary for engineers to identify errors using traceability and hi-fi models early in the process by simulating system behaviors based on real engineering data and use cases. For a deeper examination of the application of agile concepts to systems engineering, read *Agile Systems Engineering* by Bruce Douglass.■