Agile system development and the three engineers

26 July 2019 | **Product Engineering** | **Thomas Rahn**
Reading time: 11 minutes

Fast, flexible, and above all on time and on budget – more and more businesses are attempting to apply approaches and methods taken from agile software development to systems and device development. They're streamlining their existing product development processes; they're turning milestones and stage gate reviews into continuous or iterative processes. But building systems in a genuinely agile way takes more than just lifting a few methods from software development and slimming down your processes.

What makes systems development different is that it involves a number of different engineering disciplines, each with their own quirks, constraints and differences in approach, all of which have to work together. There are good reasons for the differences in the system engineering approach taken by these different engineering disciplines, and how you tackle this issue requires careful consideration. These differences are down to essential factors such as the degrees of freedom in the system design process and the constraints typically encountered during development of a finished device. I'm not going to trot out platitudes about the desirability of faster prototyping during hardware development or how to achieve it. We're all familiar with recent advances in PCB manufacturing and 3D printing. Similarly, an iterative approach is something that will be very familiar if you're an electronics engineer or plastics expert and generally needs no big reasoning.

Agile system development needs the right milestones

Achieving a genuinely agile system development process means making more broad-based changes. It's essential to ensure that everyone in the company is on board and to build agile development teams running from product managers through to developers. The change process needs to be proactive in tackling any differences in approach and to establish the best possible mix of methods. The key point for successful agile system development, however, is setting the right milestones. This takes experience – and lots of it – and the ability to think your way into the mindset of the various disciplines involved.

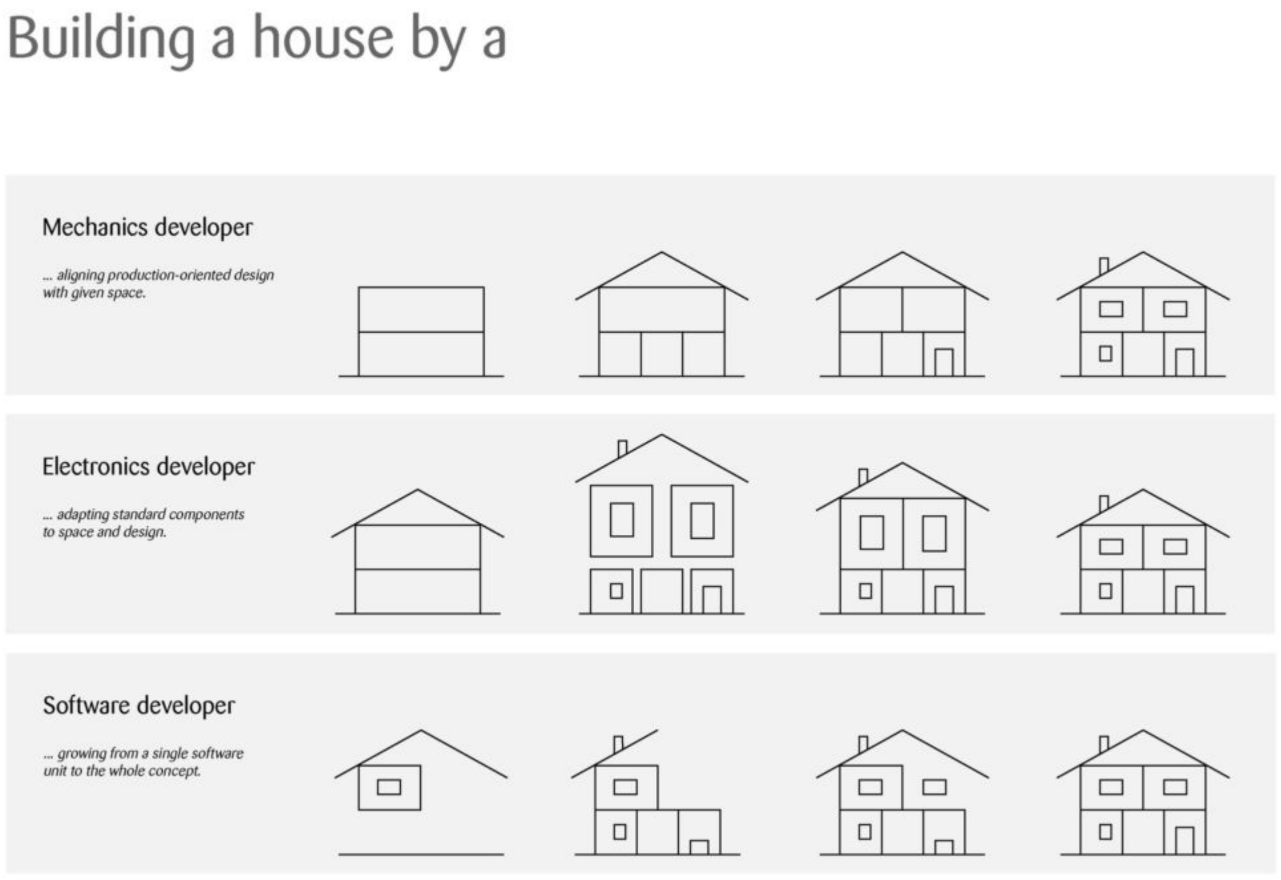
Before looking at a sample project, I'd like to illustrate this with an analogy. What would happen if a programmer, an electronics engineer and a design engineer were tasked with building my dream house, each using their own established approach? I don't want to end up with a minimum viable product (a windowless log cabin with no hot water); I want a nice comfortable home with every modern convenience, where I can grow old in comfort. So, engineers, to work!

The programmer
The comment about hot water certainly didn’t escape our programmer, so he’s got straight down to designing the bathroom. It’s going to be a temple of relaxation, a true home spa experience. Naturally he needs a bit more space (cf. a faster CPU, more memory, more bandwidth, etc.), even though he hasn’t actually been given a specification yet. He’s also given plenty of thought to the electricity and water connections. For his self-selected sub-task, he produces a bunch of mock-ups, ensuring that virtual water flows from the virtual taps and that the plugholes do what plugholes should. At the end of the first phase of construction, the build passes extensive unit testing and the first release, in a modular design, is all set for the next sprint.

The electronics engineer
He’s quick to come up with some ideas for what will be needed for the shell and installations, and some thoughts on a floor plan. He soon works out what he needs by way of underfloor heating, windows, etc. (in a real development project these would be components and sub-assemblies). To clarify some of the more complicated points, he gets hold of some materials, flicks through catalogues and makes enquiries with a few specialist suppliers. At the end of the first phase of construction, there are various raw prototypes for features like shower fittings, water distribution systems, meters and stopcocks – all with added connectivity and automatic control systems. It’s all a bit clunky and oversized, with lots of different bits of pipework bolted together, but hot water does come out of the hot water tap.

The design engineer
He is concerned to note that the site is quite small and doesn’t have much space for a crane or a cement truck. He’s also wondering how it’s all going to fit together and how he can fit it around a flight of stairs which also needs to allow for potentially installing a stair lift, should one be required at a later date. Once he has identified and prioritised all of the more finickety functions, he starts with the basic concept, makes some sketches and considers different variants. He filters out the less critical details and starts by checking how the various rooms and installations might fit around a stair lift (in real life, this would mean choosing functional principles and thinking about the system layout). At the end of the first phase of construction, scaffolding has been set up, mirroring the outline of the house, and a series of ropes have been stretched across the space to show the layout of the interior. There’s also a 3D model showing a number of additional rooms. Detailed plans have been drawn up for the stair lift, but a ladder will do as a stand in for now.

Traditionally, the first phase of construction would involve digging trenches for the foundations and pouring these and the concrete base (hopefully without forgetting any of the connections for the utilities) – a very different picture from our fictitious scenario above. Each of the three engineers can show me concrete results, which I can look at, try out and evaluate – forming a good basis for agreeing the next step. All are confident that they've made good progress and do not anticipate any major problems.


Building a house by a
Mechanics developer
... aligning production-oriented design with given space.
Electronics developer
... adapting standard components to space and design.
Software developer
... growing from a single software unit to the whole concept.

So why does viewing the phase 1 results leave me with a bad feeling in the pit of my stomach? Well, it could be because I still have a number of unanswered questions. Will the electronics engineer's individual components actually fit together when it comes to the crunch? Will the rooms still be big enough once wall thicknesses and wall cavities have been taken into account? Have all of the pipes and utility connections been laid? How will the bathroom module be hoisted up to the first floor? And of course there are still a host of other questions which still need to be clarified.

To get to the point: somewhere along the line, we've forgotten to do the systems engineering. Each discipline has interpreted the task at hand from its own perspective and based on its own previous experience; each has approached it with a different focus.

How are these three parts going to come together to make my dream house?
From their point of view, each of our three specialists has tackled the key issues, found a good solution and given proper thought to the interfaces with the other two. Despite this, at the end of the first phase of construction it's not at all clear to me, as the client, how these three parts are going to come together to make my dream house.

So why did they all take such different approaches? The key reason is the degrees of freedom, constraints and project risks typically encountered in these three different disciplines:

- In electronics development, unless we're talking about a bespoke ASIC, with all the risks and work that this involves, we are generally able to use standard components. In this case, the early project phases are about showing that we can take care of the most high-risk functions using the components selected. Standard, low-risk components can be added later, and the shape and layout can be adapted as required.
- Limiting factors for the solution space in design engineering include layout space, materials/manufacturing processes and the anticipated production run size. Consequently, layout and space constraints are considered at an early stage, as is ensuring that the design is actually manufacturable.
- In software, we can make good use of virtual interfaces. It's often easier to start by implementing defined software units using mock objects than to start implementing all of the various software units simultaneously.

Given the differences in emphasis even in the early stages of a project, it's clear that successful agile system development is going to need shared milestones. These milestones or integration points should answer an outstanding technical question or implement a system feature. Integration points also need to include a proper definition of the level of maturity of deliverables (quality criteria). We need to remain focused on reaching this alignment on goals throughout the project.

A real world example of agile development at Zühlke

So what do these agile system development milestones look like in practice? Let's look at an example taken from our day-to-day project work here at Zühlke. We were asked to develop a new device. From the client's perspective, the key function and most demanding, highest-risk feature was the sensor systems. We needed to demonstrate early in the project that the required sensor functionality was fully attainable under real world usage conditions. The project team agreed on this proof of function for the first integration point. All of the engineering disciplines involved in developing the device had their specific contribution to that goal. To ensure we could get to this integration point quickly whilst still producing meaningful results, it was important to agree the scope and quality (in terms of the maturity of the implementation) for this integration point. This reflects the two core values of agile development: timeline and client benefits.

The agreed objective for this integration step was that “it should be possible to capture raw sensor data”. This meant prioritising some requirements and planning workarounds for others.

- We needed to be able to record analogue signals as accurately as possible and with a sensor hardware setup as close as possible to the final product. This meant keeping paths to the amplifier short and using an analog circuit similar to the final production circuit even in the raw prototype. Conversely there was no need to design the power supply unit, we could just use a high quality lab unit. An eval board for the chosen microprocessor was sufficient for downstream data recording, with no need for a fully developed PCB.
- Mechanical engineering had to come up with the design concept for attaching the sensor electronics, providing reliable contacting as well as fitting into a tiny space. At this stage, we were still free to select some off-the-shelf motor drive not caring about its size or power consumption. In addition, as long as defective parts could be replaced quickly as required, it was not a problem if the components were only good for a few hundred readings.
- The control software just needed to take readings. It was fine if it only supported some fixed parameters and not a fully configurable setup. At a later stage in the project, we could put together the PC-based software for configurationparameters and thresholds.
- One other software requirement was to capture and analyse data in real time. But for the first step, we were satisfied with storing the raw data in RAM and subsequent analysis by a downstream process.
- For operator interaction, we resorted to a simple command-line interface. So initially we used as quick and dirty remote console with no GUI – the GUI being developed at a later date.
  Of course, we could equally have taken the approach of properly implementing most of the requirements right from the off instead of putting them on the backlog. That, however, would have meant planning the integration point for a much later date and until then, we would have accumulated a lot of additional project risks.
  In our experience of agile development, we have found that it is worth having frequent system-level integration points, testing against system requirements and with system processes as often as possible. At this level of abstraction, the objectives and scope of integration points can be defined early on without requiring a great deal of up-front specification work. Following these best practices, we achieve project progress at a clearly predictable rate and are able to react with flexibility to new findings, insights and changes arising in the course of the project.

We have fleshed out the methodologies and best practices, live by them in our daily project work and regularly tell our story in a number of blog posts and at various conferences. We will be doing so again at our forthcoming seminar on agile system development for medical devices.

Further interesting links:
[Agile System Development Part 2 – Continuous System Integration]

[Link to seminar (German)]