# Agile Hardware

**ESC-4010**

**Matt Liberty**

**Jetperch LLC**

**Boston, September 2012**

As opposed to document-driven, waterfall heavyweight processes, Agile Methodologies are focused on bringing together people to deliver working software while nimbly responding to change. Software teams have applied agile methods for over a decade, and the community has grown. Once the outsider, agile is now helping mainstream companies successfully deliver quality software with greater predictability to happier customers. The Agile Manifesto [Agl12] was signed in 2001 which help align numerous lightweight software processes. In the meantime, the hardware and mechanical engineering worlds have embraced some rapid prototyping techniques, but broad movement into agile has not yet occurred. This paper discusses how principles and practices from the agile processes developed for software could be successfully transferred to hardware.

## What is the problem?

Traditional project management using the waterfall model starts with requirements and then proceeds sequentially through design, implementation, verification and maintenance. Figure 1 shows a typical waterfall schedule for a printed circuit board (PCB) design process.

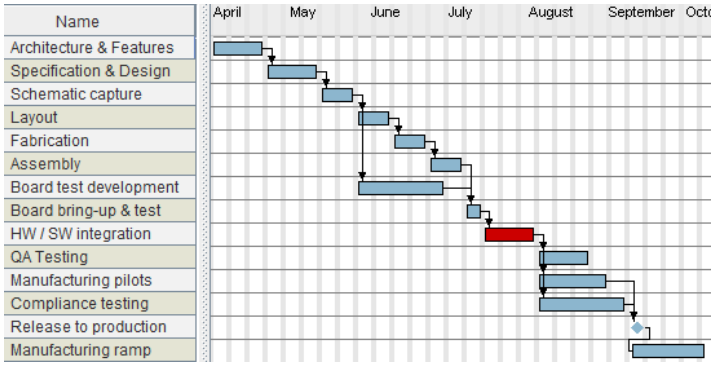


**Figure 1 - Typical waterfall PCB schedule**

In an ideal world, the waterfall process would be the fastest, cheapest way to complete a project. Reality creates several severe issues that can lead to critical projects failures using a waterfall method. The primary problem with the waterfall method is the assumption that all requirements can be captured at the initial phase of the project. Even if the customer could articulate every requirement, language and communication barriers from the customer through the organization to the developers are inevitable due to differences in terminology, domain expertise and background. Real projects also take time, and during that time the customer changes in response to events including competitive products. This perspective change inevitably results in changing requirements which disrupts the waterfall flow. The most fatal engineering flaw in the waterfall model is delayed discovery of problems. The most

incessant issues are not found until hardware/software integration which is most of the way through the design phase of the project. Until this phase is complete, project uncertainty cannot easily be estimated as board respins, rearchitecture or heroic efforts may be required before integration completes.

## What is agile?

The Agile Manifesto [Agl12] was signed in 2001 and helped bring together numerous similar lightweight software development practices that grew in response to the problems with waterfall model development. The Agile Manifesto states:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> - Individuals and interactions over processes and tools
> - Working software over comprehensive documentation
> - Customer collaboration over contract negotiation
> - Responding to change over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more.

The Agile Manifesto also has twelve principles focused around these values. Agile has grown to be a collection of tools that teams customize for their culture, products and objectives. In general agile software teams strive to:

- Always have a working code base
- Incrementally add functionality over time
- Deliver and demonstrate frequently
- Minimize the risk of change through comprehensive unit tests
- Manage features through stories with WHO, WHAT, WHY and conversations.

Unlike the waterfall model which builds sequentially, agile methods tend to have rapid iterative cycles that sequentially build upon themselves. The end of each cycle is a working product, and the definition of working (the feature set) grows with each iteration. Figure 2 highlights the differences between the two processes.

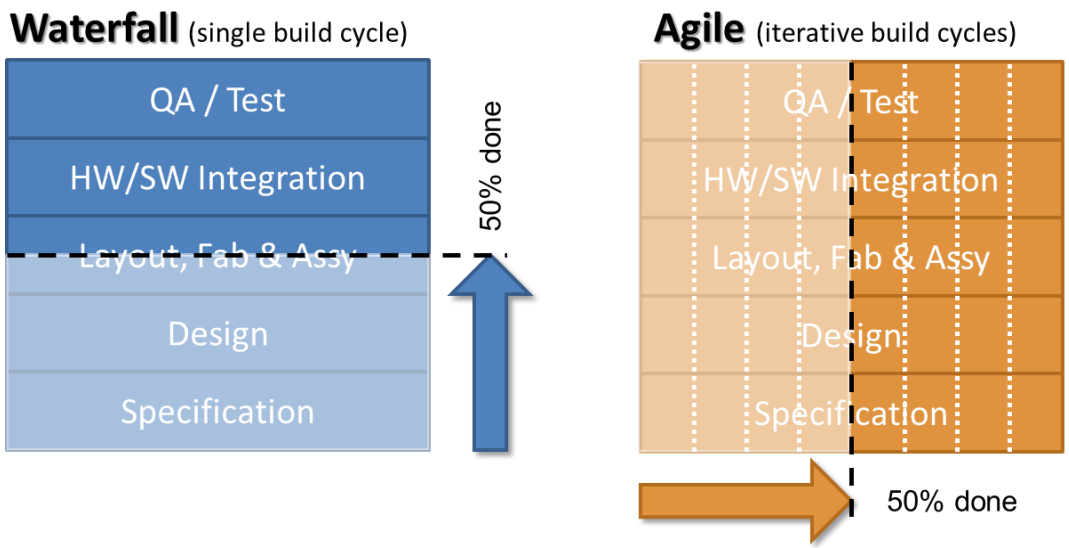


Figure 2 - Waterfall versus Agile

## What is agile hardware?

The Agile Manifesto was targeted directly at software development and software projects. However, many engineering disciplines have aspects similar to software development. Hardware has customers, requirements and deliverables. Hardware projects also suffer from changing requirements and integration challenges. These factors, which originally drove the Agile Manifesto, must certainly be similar between software and hardware.

Hardware teams are more frequently collaborating with software teams that have adopted agile methods. The documentation-heavy processes used by traditional hardware teams no longer meshes with the more dynamic nature of the software teams. As a project team, the goal is to deliver quality product at the lowest cost and risk. Hardware teams often act in isolation to reduce hardware risk but not overall project risk.

Agile methods have the potential to bring teams together more frequently to deliver working hardware, and products, sooner, cheaper and with less risk. Instead of hoping that a single integration goes smoothly, teams can start integration near the start of the project and identify unanticipated problems earlier in the design process. By uncovering issues sooner, teams have more time to react and limit last-minute crises that provoke political infighting.

Although hardware and software share some common ground, the disciplines also have significant differences. The most significant difference is the cost of "building" a product. In software, the cost is limited to the cost of CPU time and possibly some developer time waiting for the build. In hardware, the build time can involve long durations and significant capital. For the purpose of this paper, hardware is

divided into three disciplines including printed circuit board (PCB) design, field programmable gate array (FPGA) design, and application-specific integrated circuit (ASIC) design. PCB build costs are often in terms of weeks and thousands of US Dollars (USD). ASIC build costs are often in terms of months and millions of USD. Both processes involve significant manual steps to build the product. FPGA builds are the exception in that it follows a more software build process, but with longer "compilation" times.

Hardware also suffers from silos of domain expertise. PCB designers often do not do PCB layout. PCB fabricators do not do PCB assembly. Hardware engineers often have technicians. These silos limit the ability to rapidly cycle through the build.

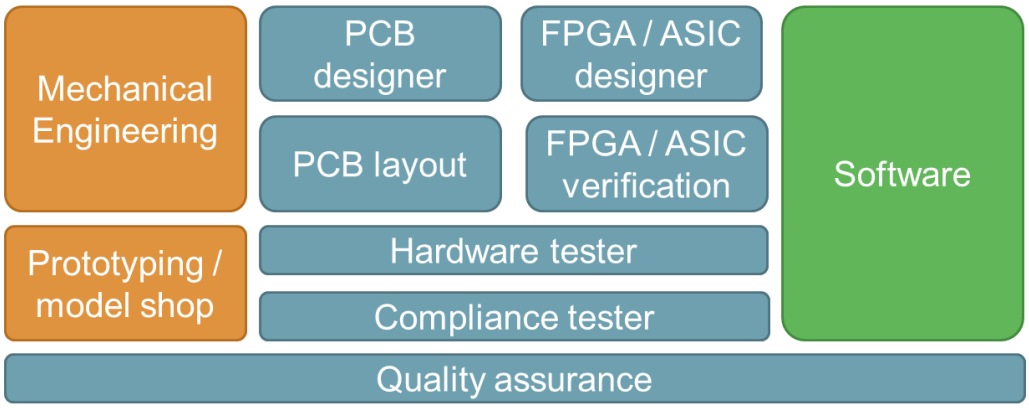


Figure 3 - Differences between hardware, software and mechanical engineering

A key pushback from hardware engineers is that hardware is fundamentally a sequential process. Like building an arch, the bricks must start from the bottom and meet at the keystone at the top. An arch cannot be built from left to right. If hardware is fundamentally like an arch, indivisible with a single, reasonable process, then agile methods could not be applied.

## Agile methods for PCBs

Over the past several years, PCBs and components have become cheaper. For many products, the cost of a fabricated and assembled PCB is now inexpensive compared to engineering labor costs. This change allows rapid board turns and additional boards to be cost effective.

Reference boards have long been used by software teams to start software development before hardware is completed. Reference boards for microprocessors are very common, but many other components from sensors to power regulators have reference boards.

Hardware teams have three primary choices for connected together components: breadboards, perfboards and PCBs.

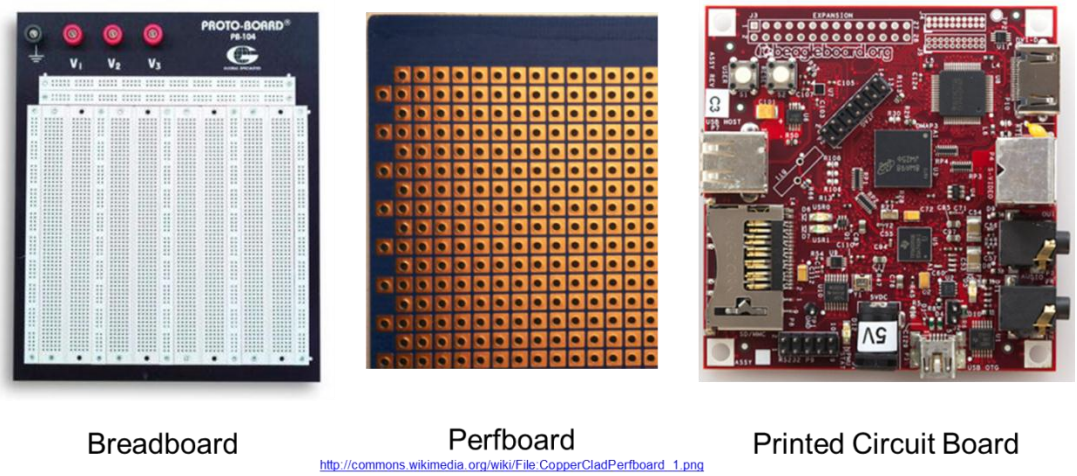


Figure 4 - Hardware connection techniques

Breadboards are quick to assemble but replicating the hardware is challenging. Breadboards also tend to be delicate and their circuits easily broken by a finger errantly brushing a wire. Perfboards are more reliable breadboards, but they are more time consuming to produce. PCBs historically have been difficult to design but highly reliable and easily duplicated. Recent improvements in design tools and costs make PCBs a viable agile tool.

One potential agile approach to PCB design is to connect multiple reference boards together to represent the full product. As the full team would like access to hardware, the prototypes should be easily duplicated. Troubleshooting hardware build problems is wasted time, so the hardware should also be reliable. Simple PCBs that connect two or more reference boards are a great fit. They are often quick to design, fast to build and easy to duplicate. If a reference board does not exist for a component or feature, the team can create their own.

Hardware engineering teams often do not have access to PCB layout software. However, several newer tools offer inexpensive options including EaglePCB ($820 for 6 layer, 10 x 16 cm) and Altium Designer ($4995 unrestricted). The industry has become highly specialized in the PCB build process. Individuals who do PCB design often do not do PCB layout, nor do they have access to the layout tools. For large PCBs, this separation of skills likely benefits the product. For small boards, the handoff unnecessarily burdens the process. Using these newer tools, interconnect boards can be designed in an hour or two, including layout, making the transfer unnecessary. Numerous PCB fabrication and assembly houses specialize in quick-turn products, and 2 layer PCBs can be produced cheaply with a 3 day door-to-door turn time. Overnight is possible at a much higher, but still not prohibitive, cost. Advanced Circuits and Sunstone Circuits (PCBExpress) are two examples of quick-turn PCB fabrication houses. Advanced Assembly and Screaming Circuits handle quick-turn assembly, but the design engineers can often perform hand-assembly in the lab for these interconnect PCBs.

Figure 5 shows two examples of small interconnect boards. The total time spent by the hardware engineer to design, send the boards out for fabrication, order parts and hand-assemble was 1.5 hours

for the board on the left and 2.5 hours for the board on the right. The total cost for quantity 5 of each board with a three day turn was about $200 not including the engineering time.

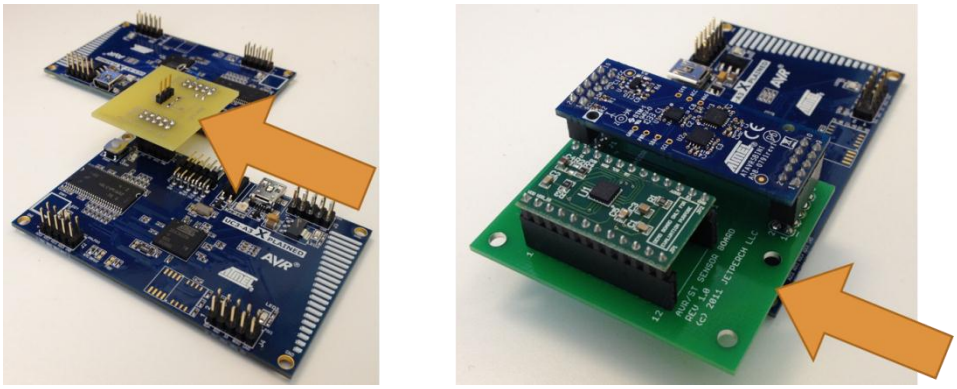

Figure 5 - Example interconnect PCBs

Both of these examples are relatively small interconnect PCBs. Interconnect PCBs are possible for significantly larger interfaces at reasonable expense.

When off-the-shelf reference designs are not available, the team can rapidly prototype their own reference board. The goal of reference designs should be to fit into an iteration cycle. Hardware engineers typically have a hard time not overdesigning since hardware has historically been expensive to prototype and errors have been heavily penalized by many managers. Ideally, the reference design schematic and layout can be used directly in the next iteration. Since the objective is to iteratively improve a product, future iterations such accumulate the features of all past iterations.

The result of each iteration should be a working product. Working is often defined by a set of passing tests on the actual hardware and a demonstration. The demonstration should be a fully integrated hardware and software product. Beyond forcing frequent integration, the goal of the demonstration is to communicate current progress and solicit feedback. The demonstration should be suitable for the entire team including customers, project managers, marketing and executives.

With hardware designs, sometimes it is more practical to tackle a parallel vertical feature slice in an iteration cycle. This approach can be useful to make progress on a different feature, but such parallel slices should be limited to a single iteration. The potential risk for divergent vertical paths is a waterfall-style integration at a later date.

The real justification for any project methodology comes down to the ability to reliably deliver product on time and on budget. Convincing the team and management of this benefit can be challenging. One such financially-based risk argument is shown in Figure 6.

| Item | Quantity | Units | Cost | Units |
|---|---|---|---|---|
| Annual engineer cost (fully loaded) | | | $175,000 | USD/year |
| Cost per day per engineer | 200 | days/year | $ 875 | USD/day |
| Team cost | 6 | engineers | $ 5,250 | USD/day |
| HW/SW Integration (50% confidence) | 10 | days | $ 52,500 | USD |
| HW/SW Integration (95% confidence) | 30 | days | $157,500 | USD |
| Cost for 50% to 95% confidence | 20 | days | $105,000 | USD |
| Last minute respin duration | 15 | days | $ 78,750 | USD |
| Estimated risk cost | | | $183,750 | |

Figure 6 - Agile hardware cost argument

What if we could potentially reduce risk by 70% for only 30% of the estimated risk cost? An argument framed like this can be compelling and translate from engineering across to project management and product marketing.

## Agile methods for FPGAs

Of the three hardware types, FPGAs follow a process closest to software development. Unlike software development, the builds take longer, the toolchain has less flexibility and the definition of "working" involves more aspects including timing constraints, placement constraints and power. Also unlike software, the testbenches (unit tests) are often designed at a very low level, and the full emulation or simulation of the final design is very slow compared to the final product.

The key for agile FPGA development is to separate the smallest integral component and develop just that component. For an FPGA, this component implements a single vertical feature. As FPGA designs vary significantly, describing this component is challenging. The goal is to section off a part of the design that is both testable and demonstrable. For an SoC, a first iteration may involve creating the processor and the associated communication path to the outside world. For a networking FPGA, it may be implementing loopback functionality.

Using agile methods does not mean that engineering teams skip architecture, but agile does mean to only document and fully architect what is needed now. Architecture thoughts should be captured for future engineer use, but detailed documents add less value that working hardware. Planning for the future helps produce higher quality designs, but implementations should only follow what is currently needed.

Each iteration should result in a working product. As with PCBs, working is often defined by a suite of passing tests and a demonstration. To encourage and facilitate change, tests should ideally be written so that they can be easily modified and extended. Since robust verification, timing and compilation are not required, they can be written at a much higher level than synthesizable VHDL or Verilog.

FPGAs are particularly well-suited for test driven development (TDD), another aspect commonly found in agile software projects. In TDD, engineers develop a test and then implement the minimum functionality required to make the test pass. The engineer then implements a new test and iteratively

builds both the tests and the implementation together. The TDD process results in thoroughly tested code that can more safely be modified, extended or refactored later in the project. A strong argument for TDD can be found at [Gre11].

## Agile methods for ASICs

The author of this paper has no direct experience with ASIC development. Unlike FPGAs, the build of an ASIC is extremely expensive and time consuming. On first glance, this cost/benefit appears to be at odds with an agile design flow. However, much of the ASIC design process can parallel the FPGA development process and commercial-off-the-shelf FPGA boards can be used to run the full ASIC design. All demonstrations can be done on this platform, albeit at a slower clock rate than the final ASIC. The ASIC development process can still be agile, but the final build and manufacturing process is likely not agile.

## Releasing agile hardware and other complications

At some point, hardware will transition from development to production. For most organizations and products, this transition incurs additional costs and shifts responsibility between teams. The costs can include compliance testing, regulatory approval, tooling and manufacturing test development. Most companies intentionally have a release gate process that is intentionally not agile in nature. The gates offer business units the opportunity to assess risk and make sound financial cost/benefit decisions. Despite not being agile, the development process can continue using agile methods. When sufficient benefit exists, the product can be released through the gate process. Using agile planning tools and the experience of many iterations, experienced agile teams and project managers can more accurately predict these target dates relative to their waterfall counterparts [Pie11].

Like any process, agile methods have some real-world complications. In hardware, complications include:

- Long-lead parts
- Limited part quantities / on allocation
- High-speed interconnects between reference modules
- Very high pin count or pin geometries near current assembly process limits
- Large number of layers (> 6)
- High component cost relative to NRE

No single item on this list prohibits the project from using agile methods, but the techniques discussed above will need to be adjusted to account for these complications. For example, long-lead parts for prototypes must be ordered during previous iterations and manufacturing quantities must be committed before the release product is fully known. Although this situation sounds bad for agile, the reality is no different from waterfall projects that slip schedule. With agile methods, the burndown progress chart is a more accurate predicator than a waterfall schedule developed at project initiation.

## Conclusion

Agile methods offer the potential to help hardware meet the increasing demands for tighter schedules and increasing complexity. Agile teams break down the us versus them mentality and work together across disciplines to reduce overall project risk. The major contributor to this friction is last-minute integration problems. By integrating early and often, this friction can transform into collaboration.

The cost balance between thinking and acting has long favored thinking for hardware development. With newer techniques, tools and cost reductions, the balance is tipping in favor of acting sooner. Agile methods offer one possibility for moving into action, failing sooner if needed and adapting. Agile for hardware is still a new and growing topic, but agile hardware methods hold the potential for teams to develop quality products with less risk, lower cost, better feedback and more satisfied customers.

## References


[Agl12] Manifesto for Agile Software Development, Retrieved 24 Feb 2012.

[Gre11] The Careful Way is the Fast Way, James Grenning, Retrieved 27 Feb 2012.

[Pie11] Velocity and Release Planning, John Piekos, Retrieved 27 Feb 2012.

Top 10 questions when using agile on hardware projects, Larry Macceronem, Retrieved 20 Feb 2012
Agile in a Nutshell, Jonathan Rassmusson, Retrieved 20 Feb 2012
www.AgileSoC.com, Neil Johnson, Retrieved 20 Feb 2012