Agile, Waterfall, Hardware, Software, and Systems

Richard M. Bixler
netboyz@comcast.net

1 February 2019
Updated 23 October 2021

Summary Agile methodologies effectively address real requirements and development methods, providing an iterative method for developing software that incorporates change and feedback into timely releases. Agile is typically used within a subscription business model, which effectively isolates it from delivery-based corporate business and financial considerations. In contrast, hardware development efforts incur substantial incremental cost, and typically drive a delivery-based business model. Hardware development efforts therefore generally use a variant of Waterfall methodology to provide predictability and control, and to fit the development into that business model.

Hardware development efforts are directly or indirectly dependent on software modules developed concurrently. A method is described to coordinate the Agile and Waterfall methods, to degrees appropriate to specific direct and indirect dependencies, and to connect those efforts to the company's delivery-based and subscription-based business models.

Introduction First, you might review multiple variants of Agile and Waterfall methodologies. A nice example of this, Project Management Methodologies, describes and compares Waterfall, Agile, Scrum, Lean, Kanban, Extreme Programming and others, offering recommendations of applicability for each. There is pressure within companies for all Development Programs to be Agile, and who wouldn't want to be that?

Methodologies are usually chosen by organizations, though, not by program; and programs developing digital computer systems involve many organizations and therefore several methodologies likely co-exist in a single system development effort. Software development is often an Agile variant, and hardware usually is a Waterfall variant or Critical Path methodology. Here are my experience and thoughts about applicability to development of software and of hardware for a platform involving both.

Agile processes are iterative and flexible. Agile is based on short development cycles responsive to input of changing requirements. Bugfixing and design changes may occur in each cycle, resulting in code that can be released as appropriate upon completion of any one development cycle. The net is quick delivery responsive to feedback and changing requirements – software-on-demand. However, Agile processes have more difficulty predicting dependencies and deliverables, development cost, and delivery timing of features to be delivered over time.

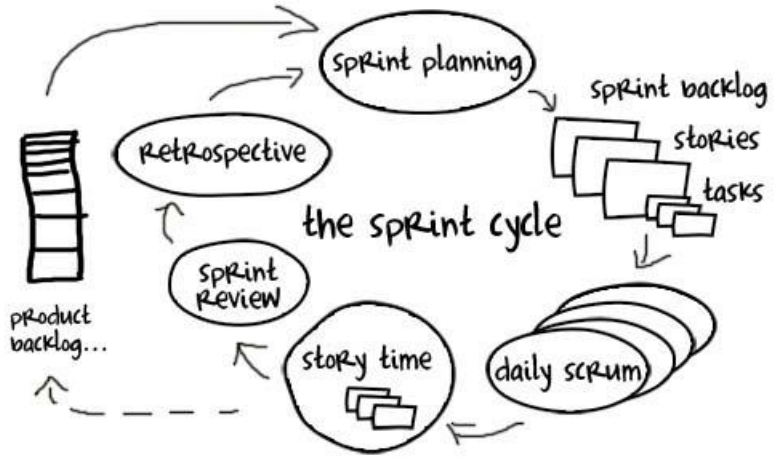


Image Credit: SoftwareTestingStudio.com

Waterfall process on the other hand is a sequential process from requirements analysis building up to delivery of a fully functional product. The process sequences through tasks addressing Analysis and Requirements, Design, Development and Implementation, Verification, Deployment, and Maintenance. The objective of Waterfall process planning and execution is to be prediction-capable of development cost and timing of delivery.

Note use of the term “prediction-capable”. Waterfall methodology is sometimes characterized as “predictable”, as if the result of turning the crank the across the organization to execute an onerous plan, is that the methodology proceeds, automatically leading inexorably to the predicted result.

But nothing could be farther from the truth! No doubt it is true that existence of a plan makes it possible for an organization to work toward meeting its objectives and within its constraints. But that is hardly automatic, it generally requires concerted effort across the organization to accomplish those objectives, even using the plan as the roadmap for doing so.

A plan makes the effort “prediction-capable”, identifying quantitative outcomes for deliverables and dependencies, and costs and revenues, by providing task sequencing and associating those with dates, leading to those outcomes given current best-understanding of conditions. Organizations then can prioritize and execute to attempt to meet those outcomes.

Moreover, the plan will inevitably make small and large updates to incorporate changes and to mitigate errors, reflecting updated understanding of conditions. Such a plan continues to indicate the quantitative expected outcomes; and sometimes the expected outcomes must be changed to reflect feasibility. All this is "prediction-capable", and certainly not automatic "predictability" in the sense of automatically achieved outcomes.

Net: Agile or Waterfall process determination is responsive to characteristics of the hardware and software technologies involved. Cost of delivery of change is a key differentiator of the technologies and methodologies, resulting in differences in feasibility of immediacy of delivery.

A key difference between Agile processes as I have experienced common practice, versus the hybrid-waterfall processes I have experienced and describe in this article, is whether you do the work to be able to identify and take an optimal path from your current position. Envision Google Maps continuously showing the path to your intended destination from your current location. Optimal may include handling lead times, equipment or function availability, timing of expenses or revenue etc. which are perhaps more significant factors for hardware program elements.

Software It is relatively low cost to release an iteration of software with feature changes, new features and bugfixes, and to install that update in customer systems. An update release of software may be downloaded on-site overwriting a prior release, to be installed by customer IT, for essentially no cost. Unlike hardware build processes, software build processes have little effect on software product cost. Incremental software releases may still have issues with certification that affect timeliness. Software revenue is commonly associated with a subscription process, with revenue due at defined intervals. Changing delivery of a feature from one incremental release to another is unlikely to affect subscription revenue, greatly simplifying attention to delivery dates.

A downside to Agile processes is that it is difficult to determine when an Agile cycle will occur that incurs dependency on an external software or hardware deliverable, or to determine when a deliverable will be provided to a using external module. This makes optimal development sequence and dependency material cost timing un-predictable to dependency providers and consumers and can result in incremental cost, or time loss.

For this factor, the greatest strength of Agile causes an unforced error. This can be avoided by continuously determining and managing critical path of function delivery planned through the sprint sequence (which is how it's done in Waterfall methodology) – and may be seen in a pure Agile context as inaccurate and non-value-added overhead... but even if there's no "right" plan at the moment, there is great value in maintaining coordination through evolution toward correctness. This is especially important in efforts involving hardware, certainly affecting timing of cost and potentially affecting timing of revenue. Here's an article showing how Kanban used with Scrum can effectively accomplish this. A prediction-capable plan may of course prove to become inaccurate as the project progresses. Regarding effects on software elements of the project, predictive planning maintains synchronization among the elements, facilitating schedule optimization even as plans change. If the plan is kept continually up-to-date, inaccuracy will converge to zero. Synchronization will furthermore facilitate changes necessary if structural change is needed for correction. The Conclusion of this article discusses an abbreviated level of synchronization needed to coordinate software development and hardware development efforts within a project involving both.

Nevertheless, excepting critical path issues, incremental feature and product delivery with short cycles are substantially more practical for software product and features, than they are for hardware. Particularly for software modules isolated from dependent modules, Agile processes are more than adequate and provide an advantage for software product that is not available for hardware product.

Software Planning. Following is description of a hybrid process I've used to develop system products, that may incorporate Agile for some software entities, and also aligns with multiple development organizations including hardware and its related processes for phased development and logistics. This process provides enough coordination to support concurrent development of hardware and software including provision of development hardware to all stakeholder organizations. In my experience, 80% of project hardware is provided for use in software development and test. In short, I have found it useful and practical to plan a software development sequence, and to keep it up to date as development progresses through backlog.

This makes it possible to determine needed development sequence, resultant timing of dependencies and deliverables, hardware logistics, and financial consequences.

First, a set of Feature Blocks (FB) is planned, organizing functionality to be implemented across multiple organizations, and coordinating sequence among them. Typically, there is an initial FB that incorporates development of pre-existing functionality which may include, for example, moving existing function into a new environment or onto a new platform. A second FB implements new function intended to expand functionality to use capabilities of the new environment. New infrastructure, transports, or client types etc. might be incorporated. Finally, a third FB contains functionality that can't currently meet the intended completion date and likely forms the core of a post-FCS release. The FBs have sequential ship date targets, but work on them may overlap as permitted by available staff. As the program progresses, function may move among Feature Blocks, via backlogs, as plans change or as a result of problem mitigation.

It's useful to align work among functional areas working in parallel. As illustrated in the following figure, kernel code, firmware and diagnostics, drivers, transport, data path, database, client structure, management, and SQA are often all developed in parallel, with deliverables sequenced from each within Feature Blocks. Development and use of scaffolding, hard-coded, or prior-rev substitutes for preliminary development can be mutually planned, for progress prior to needed replacement by deliverables for preliminary and formal SQA, Compliance, and Certification testing.

It's also useful to coordinate Feature Blocks among disparate development methodologies that may be in use in a single program. This figure shows alignment of development across multiple functional areas and includes alignment with development of a hardware platform.

Also shown, private code branches used for development and bugfix are integrated for SQA on branches indicated as dot releases within each feature block, which ultimately lead to the release candidate for FCS.

FB / Drop	Client / Periph	Envrnmnt	Data Path	Mgmt	Function / App	SQA	HW / Sched
FB0 FB0.1 FB0.2 …	PC Client SW Client App Camera PCIe Client	Op Env PHYs Boot Stability Hard Code	I/O Ctrl Drivers Transport Stability	Function Config Bringup Update CLI	Transport Data	Test Dev P0 Test	**Eval Sys** **P0 Sys** Prelim Qual
FB1 FB1.1 FB1.2 …	Function Discovery	Stability Error Hndlg Stats	Stability Error Hndl Perf	UI BUI Discovery	Stream Qty Stream Types Function	Test P0 Test P1	**P1 Sys** Qual Rdy
FB2 FB2.1 FB2.2 …	Perf Stabillity Function	Bugfix	Bugfix	Stats Bnchmrk Bugfix	Perf Stability Function Platform-Indep Integ Bugfix	Frml SQA Rls Camd Triage Release	**P2 Sys** Frml Qual

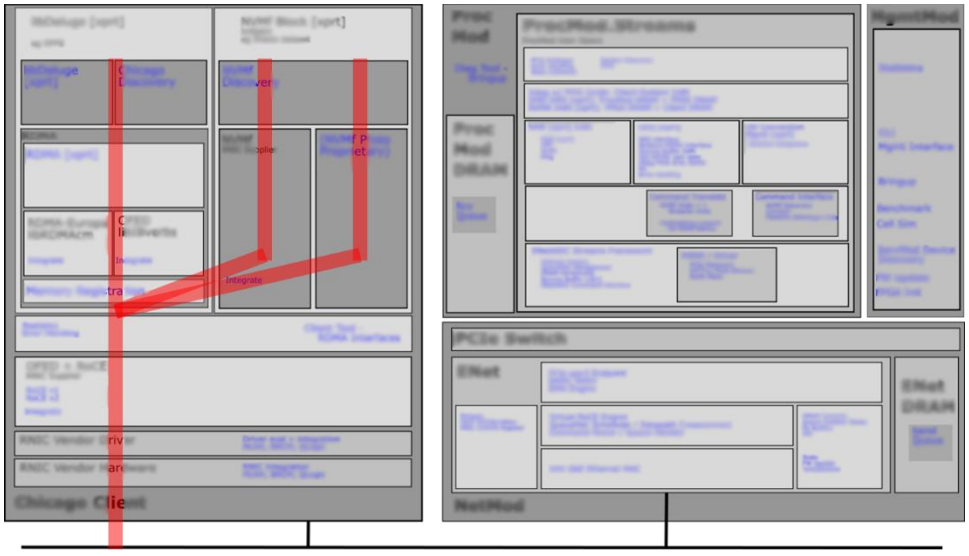
Next, function development can be planned among functional blocks within development entities. A functional block diagram or equivalent method of identifying plannable entities, and relationships among them, is used. Typically, you would see development of function within

functional blocks, followed by integration among the blocks. For example, you could integrate a client entity with a server entity once function is implemented in the client and in the server; and a transport mechanism is created; and sufficient management exists to configure and control each component.

Using this logic, sequential development tasks can be defined, to be implemented in a series of sprints that sequence function development and integration tasks, coordinating timing among the functional areas executing those tasks.

Program Increment (PI) Planning is a Scaled Agile (SAFe) methodology helpful to structuring this, by identifying dependencies and quantitatively sequencing them into an operational plan. And as SAFe says, “PI planning is essential to SAFe: If you are not doing it, you are not doing SAFe.”

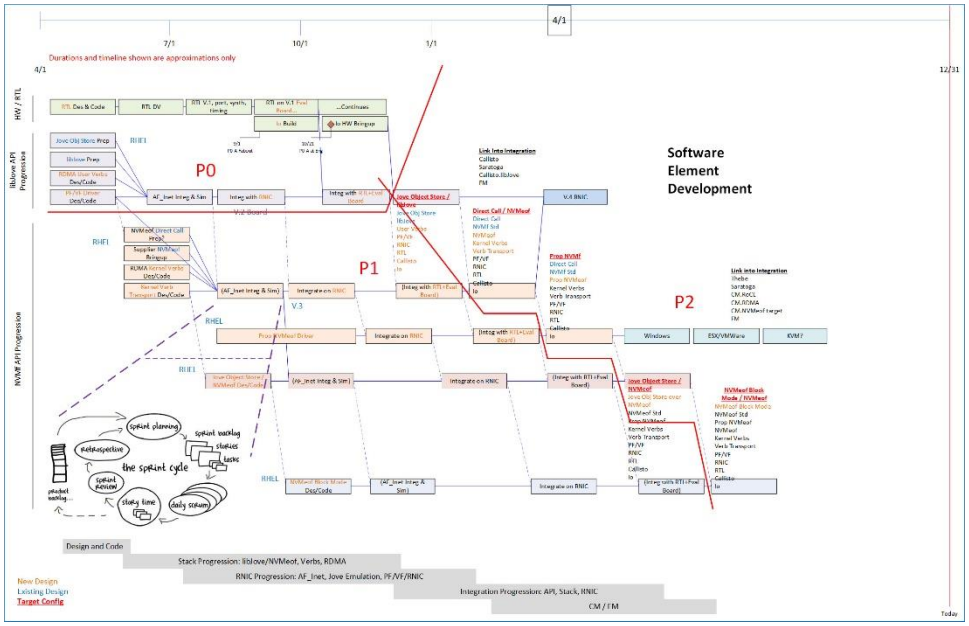
In the figure below, the red lines indicate three data flow paths to be implemented.



As illustrated just below, for each of the three data flow paths to be implemented, a sequence of sprints ports code, incorporates new infrastructure from third parties, implements new functionality, and integrates all that sequentially on a prototype platform and from there onto its target production platform. Below those sequences, following and overlapping, new APIs to be made available are developed and integrated. Finally, the solution is moved across several operating environments and onto several third-party platform components. In the figure, developments of the data flow paths are planned for parallel development, with sequencing among them to build a ladder structure of tasks that converge into shippable product.

The effort is broken into sprints and may use Agile backlogs and scrum structure to control progress. As backlog items are completed, or are pushed to later sprints, mitigations are planned, and the overall plan must be updated. The resulting plan maintains coordination across functional

development areas, and schedule specifics can be planned, tracked and managed. Using that, other development organizations can plan their developments, purchases, integrations, and deliveries.



Dependencies and deliverables control sequencing among sprints. Everybody can see which of their deliverables to prioritize, and when their dependencies are planned for delivery. Coordination with the hardware phased development is shown by the red lines. Placement of those phase lines is not arbitrary – functional sequencing follows the Feature Block structure defined above, and maps, as planned, to the hardware phases.

Hardware Since software logistics are relatively cheap, software methodologies tend to focus on design, integration and test. Hardware development methodology also must consider design, integration and test; but in addition to that, realization logistics to implement or change hardware are expensive, therefore playing a dominant role in hardware development methodology. Several examples follow.

Hardware prototyping cost is a significant part of organizational and company financial processes. The aggregated result of this process is subject to public scrutiny and therefore is carefully managed. Absolute budget can be affected by cost, number and timing of prototypes built for distribution, along with licenses, NRE, contractors etc. all classed as Operating Expense. Capital Expense and Depreciation for the program are also closely identified and managed. Both Operating Expense and Capital Expense are therefore identified at the commitment of a development program, tracked quarterly and budgeted again each fiscal year.

Prototype spending is spiky, tied to builds associated with each Phase of a Waterfall development, with different cash flow characteristics for mechanical, electronic, and purchased elements (servers, switches, displays, cables etc.). An item may be ordered in one quarter and delivered in another. Payment may be due on order, at Net-30 ARO, on delivery, or on supplier completion. The spikes must be managed to smooth expenses across quarters and to avoid spending conflicts among programs.

Personnel Expense for hardware development and software development is treated as a different class of expense, less variable during the course of a development effort. If you acquire expertise just when you need it, it won't be ready due to familiarity with product, and company practice; so both software and hardware efforts must plan and manage personnel. Management of prototyping cost is managed within the hardware effort although prototype costs to provide units for software development are commonly the large majority. The net of this is that plans and budgets for Expense and Capital are tightly managed, drive constraints, and need predictability to development schedule and changes to a hardware plan.

Shipping hardware for production is involved, costly, constrained, and time-intensive, so incremental delivery of hardware function is limited to defined interfaces; and even then, subject to build process and certification. Customer schedules may depend on committed delivery of hardware. Custom product development progress payments may be tied to internal development milestones with contracted schedule. Your organization's business plan factors in these revenues and payments, and change to the delivery date can affect revenue, ultimately affecting company and organizational operation. A changed revenue date may also affect return-on-investment of the product, used as an internal metric for product viability, and as an aggregated external metric on the company's use of capital.

Even an incremental hardware design update usually requires module replacement, since rework is seldom practical: fixes on nets involving high-speed signals can't be wired in; and in any case incorporating change on-site to installed hardware causes more breakage than it fixes. Replacement hardware must be built, involving substantial cost and time. Swap-fix cycles leverage sunk-cost of existing materials but are resource-intensive and much too slow if not local. Material lead-time can be long with critical component lead-times out to 6 months. That

time can be mitigated by anticipating replacement materials into inventory, but cost of the replacement hardware is still incurred along with fab and assembly time. Installation of replacement hardware is a manual process involving shipment and support personnel. Net, updating hardware is a substantial cost taking substantial time.

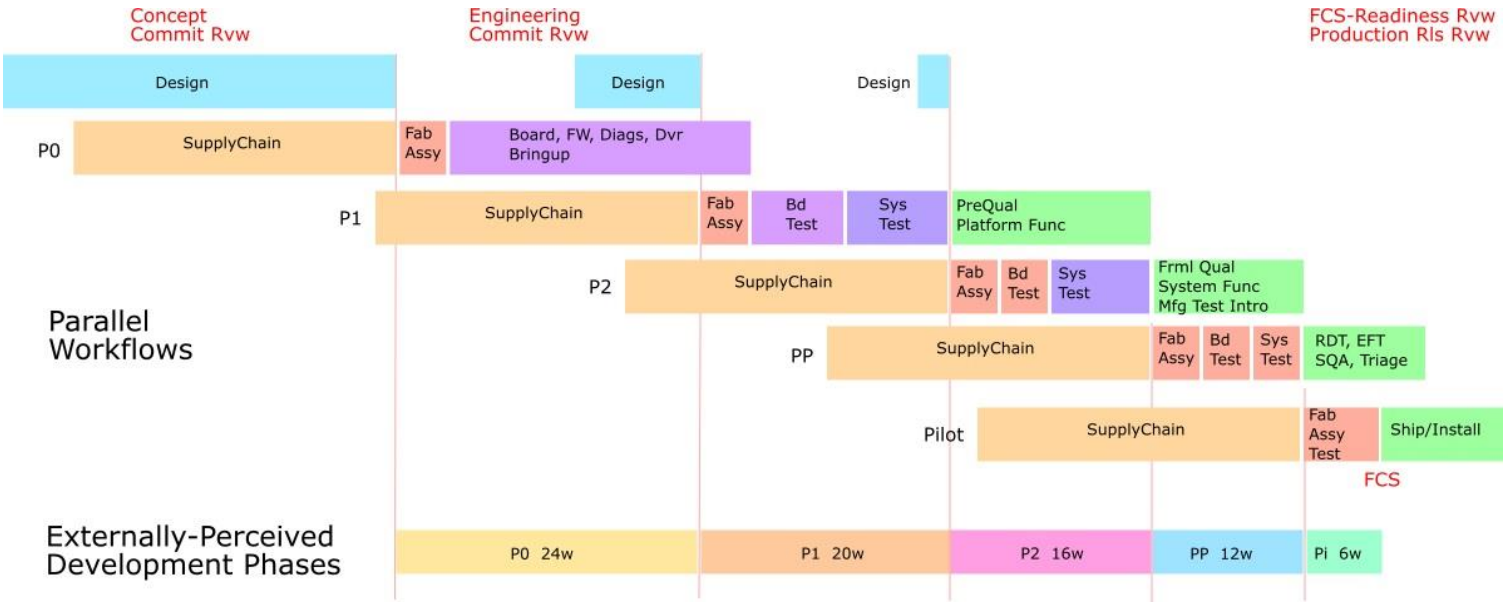
Revenue timing differs between software and hardware. Customer payment for hardware products is tied to delivery. As above, installed hardware cannot readily be updated, so must be complete on delivery. Not only can we not ship hardware with known bugs to be fixed, in fact shipped hardware must provide for anticipated upgrades to be added to installed systems. Product margin testing must be verified as supporting manufacturing yield. Subsequent product enhancements can be accomplished on interfaces provided, as well as updates to firmware and software. There are also certification processes for production hardware, some of which are legal requirements, that affect delivery. Hardware is susceptible to environmental factors so those must be solid for customer delivery. Government agencies require compliance on electromagnetic emissions. Safety standards are accident prevention, legal requirements, and legal due diligence. Requirements differ among countries. Product build processes and marking among partners must be established and may affect taxes based build location. By way of analogy, Software certification can cause similar inflexibility to otherwise-quick software releases.

Requirements for visibility and predictability of cost and timing for hardware Operating Expense, Capital Expense and revenue, driven by prototype builds, multiple organizations, lead times, and revenue, biases hardware development methodology strongly toward prediction-capable Waterfall methods.

The figure just below shows a Waterfall process as it may be visible to external observers. One development "Phase" follows another (P0, P1, P2 etc.), with specific purpose for each Phase so it appears that reversion is inflexible to accommodate change to requirements. (Each figure below shows a larger view if clicked.)

But there's more going on than meets the eye from such a diagram. Commonly even participants in a program pretty much see effort in their own and directly related tasks. Most can say various principles of the program: builds are needed for this or that qual; Manufacturing Ops organization with CM or JDM partner acquires material and builds prototypes; firmware, diags, and drivers are needed for some of the builds. But few see, and fewer quantify, the extent of interactions and parallel operations and dependencies that drive the program overall. Almost nobody sees the whole program. So here is a view by the program manager, of a set of processes spanning the duration of a platform program effort, and spanning hardware, software, supply chain, operations, and partners.

Even this figure represents only the core portion of a system development program: development of boards and directly-associated software. In real life, Program Management must provide plan breadth and detail addressing many other program elements: Mechanical system infrastructure; firmware modules; development, SQA and release of platform software, application environment services, and applications; builds of integrated systems; system validation processes for Qual, Compliance, RDT, EFT; Field support and training. Also, the platform development depicted here must contain further detail on build processes. Multiple program elements may need separate individual but coordinated schedule plans. Frequently builds are split to provide a small initial build to evaluate build issues followed by a larger build to provision overall build deployment to higher-level development efforts.



The visible summarized Phases described in the earlier figure are shown at the bottom of the figure just above, to show how detail of the tasks maps to visibility. Here, each Phase is seen to start when its fab steps begin, and continues into key testing defining the purpose of the Phase. Completion of gating tasks enables start of the next Phase, while key testing continues to completion in the current Phase.

The figure above shows, to scale, actual work timing for program phases directed toward specific types of testing shown to complete each phase. The **tasks for multiple phases are executed in parallel** to handle material lead time (leftmost task in each phase). There is sequencing among the phases: it doesn't make sense to build P1 units for distribution until P0 boards have been brought up and debugged. It doesn't make sense to build P2 units for broader deployment until platform software is functional, and until hardware from P2 is ready to be used for formal Qual steps. Such phase sequence constraints usually align for both hardware and software development elements. The sequencing logic is common-sense, and usually agreed by the Program team.

Tasks that can start without awaiting completion of a prior phase are not prevented from starting and in practice usually some risk is taken to start somewhat early to execute as much in parallel as possible, to minimize unnecessary wait times (Lean!). If Agile is a mindset not a specific methodology, one can see that many Agile principles are employed: Incremental development, continuous delivery to customers (many are internal), working system each iteration, customer engagement, plus major efforts on dependency management plus critical path management.

In the figure just below, as in the figure above, horizontal bars show task sequences implementing an "Integration Cycle" in each Phase. These are long-duration sequences that build, integrate, and test Elements, function, and processes that implement the product of the Program. Each Integration Cycle includes a Prototype Cycle to build, and a Qual Cycle to validate processes and product configurations that it implements.

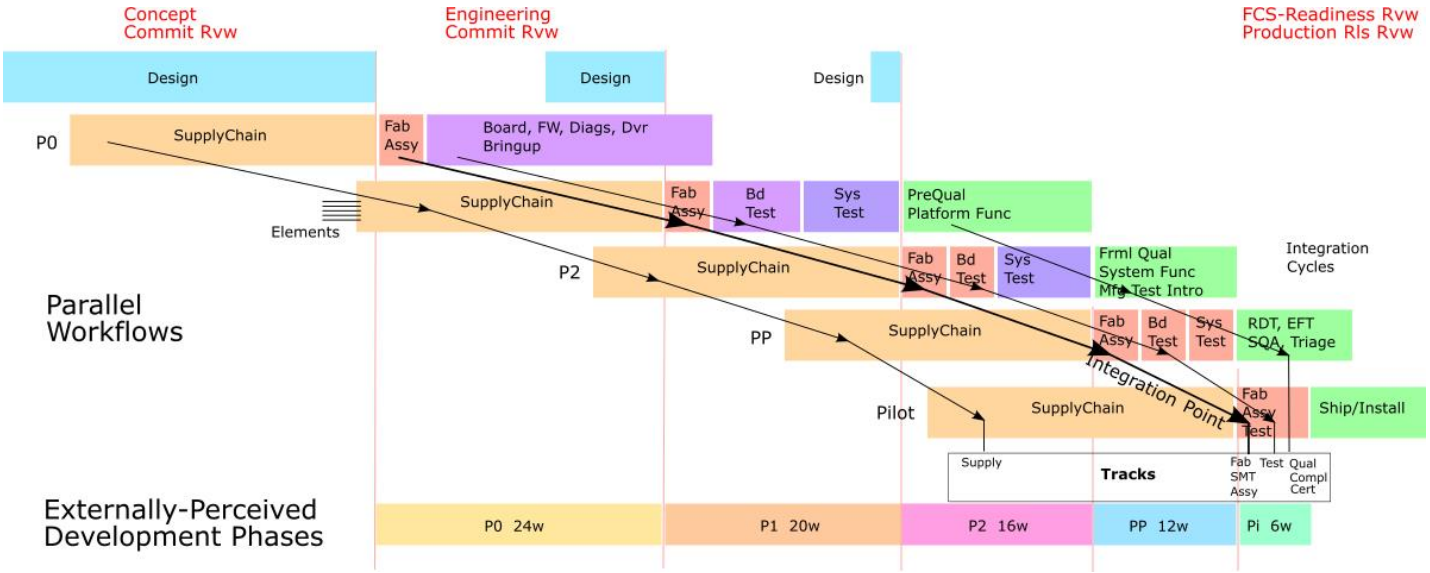
Development Tracks cut across these Integration Cycles using each Cycle to implement progression of intended Track content. Content starts as initial prototypes in the first Cycle and progresses toward fully functional and configurable production customer shipment in late Cycles. Each Track defines its function to be realized in each Integration Cycle, aligned in agreement with all other Tracks as of a defined "Integration Point".

Within each Integration Cycle an Integration Point is defined to quantify date of alignment or synchronization of functionality for the Integration Cycle. This is usually the start of the SMT solder step for hardware elements, which commits critical acquired material to the Integration Cycle without possibility of repudiation. This point sets the date within each Integration Cycle by which precedent tasks and functionality to commit to SMT must be complete, and which drives the task sequence following for which their predecessor tasks and functionality must be complete.

The figure below illustrates Development Tracks likely to be included in a system development Program:

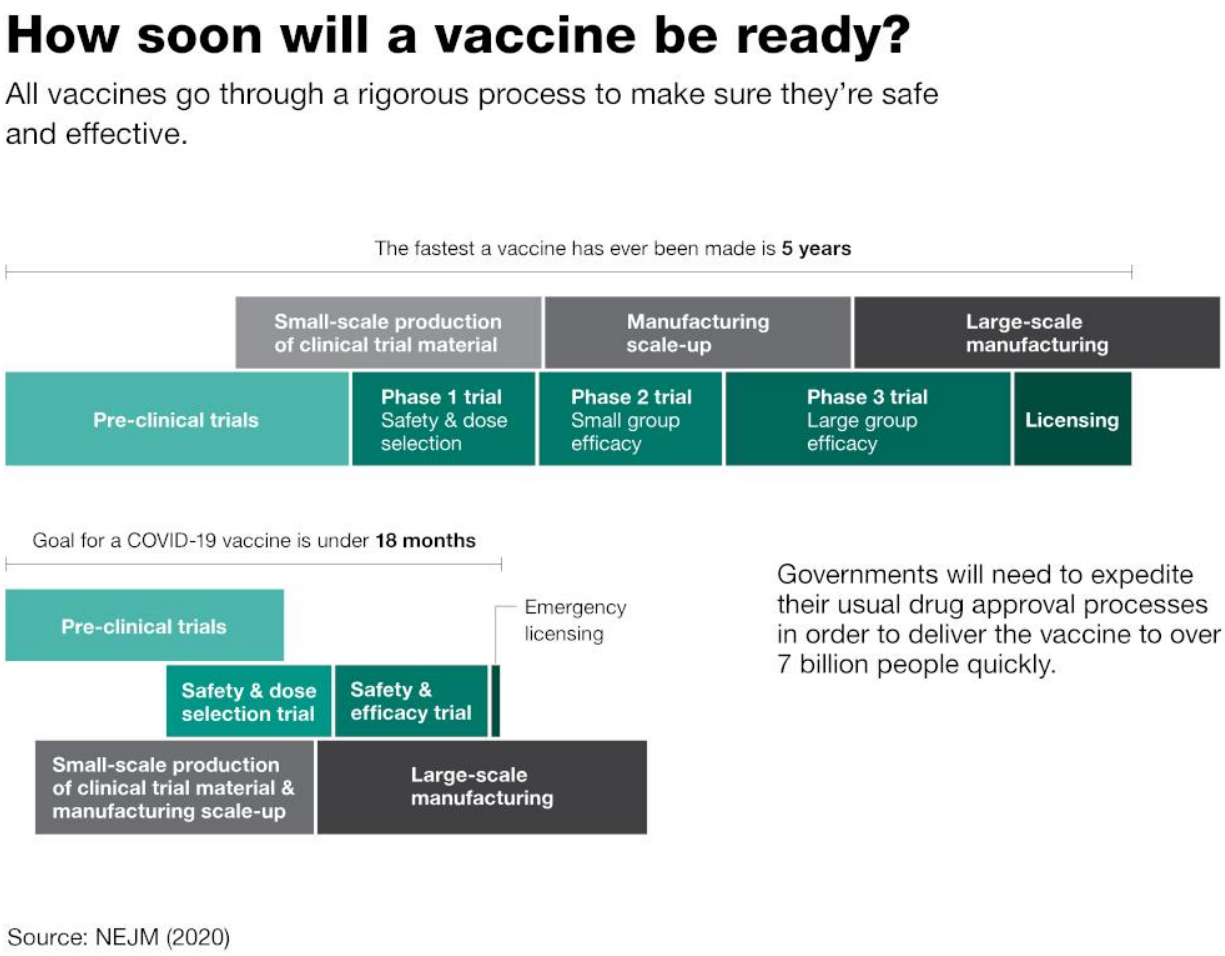
- **Supply** (Material acquisition or fab, initially local distribution; then procurement by local CM then production CM volume purchase then volume production and production supply agreements).
- **CM (Contract Manufacturer)** Protos initially local fab and CM with manual test, later to production fab and CM with test automation and assembly construction).
- **Function** alignment among product software and hardware elements as shown in the table earlier in this doc.

- **Qual/Cert Test** (**Engineering** bringup, PVT/Corner, pre-Qual, **Formal Qual** (environmental, ESD), **Compliance** (SQA, Environment, Safety), **Certification** (Security, Privacy).



Such a task plan, together with resulting schedule, material plan and spending plan, represents "the best-known development path at this point in time" for the project. Such a plan would be updated continuously to reflect reality as it is perceived. The dates are not set arbitrarily - they are driven by the task plan and used to continuously coordinate tasks among groups to optimally sequence dependencies and deliverables. It would be expected that the project team would make conscientious effort to mitigate changes to avoid affecting the critical path, and to avoid impact to committed dates. Most tasks are not on the critical path so that's not always hard. And when there is an unescapable time tradeoff, critical path time reduction is often more precious than money focused on mitigation.

Parallelism and attention to Critical Path apply to scheduling of most endeavors. This article is based on projects in the experience of the author; but here is an example from Bill Gates' blog of schedule optimization using parallelism, in the field of immunology:



Credit: www.gatesnotes.com

Agile users often describe Waterfall methods as requiring substantial documentation and up-front design, and fixed sequential development sequence; resulting in inflexibility to respond to changing requirements. Many hardware/Waterfall program constraints have been described above, and they certainly affect flexibility. Some organizations may impose inflexibility in a top-down attempt to drive schedule accountability. But that inflexibility argument seems commonly based on the summary description of Waterfall implementation, trivialized and unlikely to be used to develop digital hardware relevant to software related to it. Comparison is made between the best way to develop software versus the worst way to develop hardware. A more powerful analysis is to examine the best way to develop software, with the best way to develop hardware.

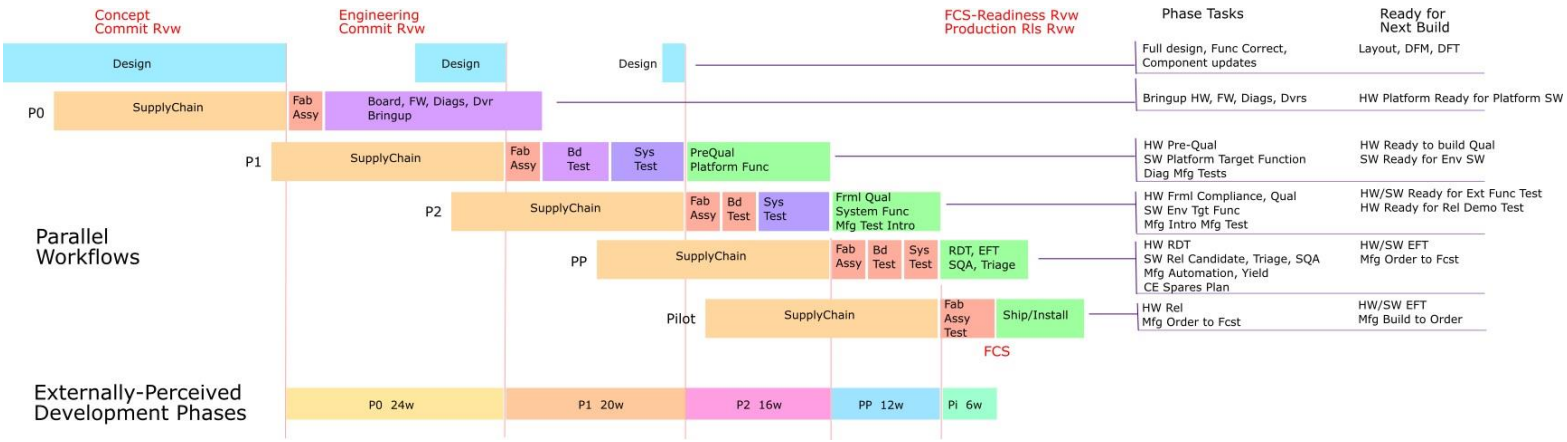
Despite constraints inherent to hardware, there is change within a system program every day requiring flexibility, and those changes are possible but require planning to mitigate the

shortcoming. Inflexibility can be mitigated by using imagination and addressing problems. It's not that fewer things change or go wrong using Waterfall methods because it's more cookie-cutter. Actually, in addition to hardware product design issues and changes, additional factors including logistics, supplier and process issues all introduce more that can go wrong. Instead, ways have developed to manage those issues within the project. Most constraints can be mitigated to a fair degree until formal testing begins that is tied to a specific design revision (compliance, RDT etc.).

It is true that cost of logistics plays a dominant role in development of hardware, causing tighter limits to flexibility in products using Waterfall. And it is also true that cycle times for Waterfall hardware projects are longer than for Agile processes because of the logistics inherent to the hardware technology using the Waterfall process. The Waterfall process is usually used for hardware development because it implements requirements driven by the technology and by corporate business practices described earlier.

The methodology and tools described on this website involving spreadsheet, database, Gantt, and visualization have been used by the author in several projects to commit and meet function, schedule and budget in hardware and software while satisfying all constraints and reporting required by the businesses. Visibility, coordination of tasks, and attention to critical path and sequence are powerful tools for the Program Manager.

The next figure describes logic for execution of each "Phase", and criteria for principal gates to the subsequent phase. Notice that each phase starts earlier than and continues past its next-phase gate logic on the subsequent phase. Tasks and gates are shown here schematically: they would be detailed and committed by the program team.



Coordination of Agile and Waterfall - Methodologies tie to characteristics of hardware and software development efforts, so a system development effort usually uses both Agile and Waterfall methods concurrently. How can they be coordinated?

Chip or FPGA Verilog design, front-end and verification can operate within simulation tools, and can be implemented using Agile processes. Chip back-end layout, physical implementation, chiptest, validation and integration usually needs to follow the Waterfall methodology of the physical platform for the reasons stated above.

Firmware, diagnostics, drivers, and hardware management make functional entities implemented by hardware (interfaces, memory spaces, processor cores, shaders, DMA, Streams, switches, management etc.), available to the software environment by providing interfaces for function and for management. That software, tied to hardware, usually needs to be highly responsive to the Waterfall development process of the associated platform hardware.

Above that, software including kernel, transport, system management et al, provides virtualized platform functionality that characteristically isolates higher-level software function and services from hardware specifics. That virtualization software and software using it can usually use Agile processes, albeit observing the need for software function at key distribution dates: Early prototype, Internal distribution, External distribution, and Customer distribution (EFT and FCS) - essentially, a sync point of defined functionality required for start of each Phase. Comments at right in the figure indicate Next-Phase start-readiness criteria; detailed criteria would be determined and committed by the Program team.

Prior to each sync point, it is common that workarounds will be used to enable continued progress until dependencies are resolved. Workarounds might include prior revision hardware; software released on prior system; infrastructure stub; external hardware satisfying function but maybe not full bandwidth; etc.

Formal testing in each Phase must be executed with all dependencies resolved that are required for that testing. Deliverables at each Phase sync point must anticipate requirements that must be completed prior to starting the next Phase. These sync points are the key dates to hold - meeting the dates of these sync releases is key to delivery of a complex system.

The start-date controlled in a Next-Phase is one that commits material with key values (spent-dollars, or lead-time elapsed) to take advantage of that value in a specific Next-Phase task, in a way that is not reversible without losing the key values. A typical example: once chips are soldered to boards, neither the chips nor the boards can be recovered if there is a mistake, so cost and time are at risk, necessitating close management of such a commit.

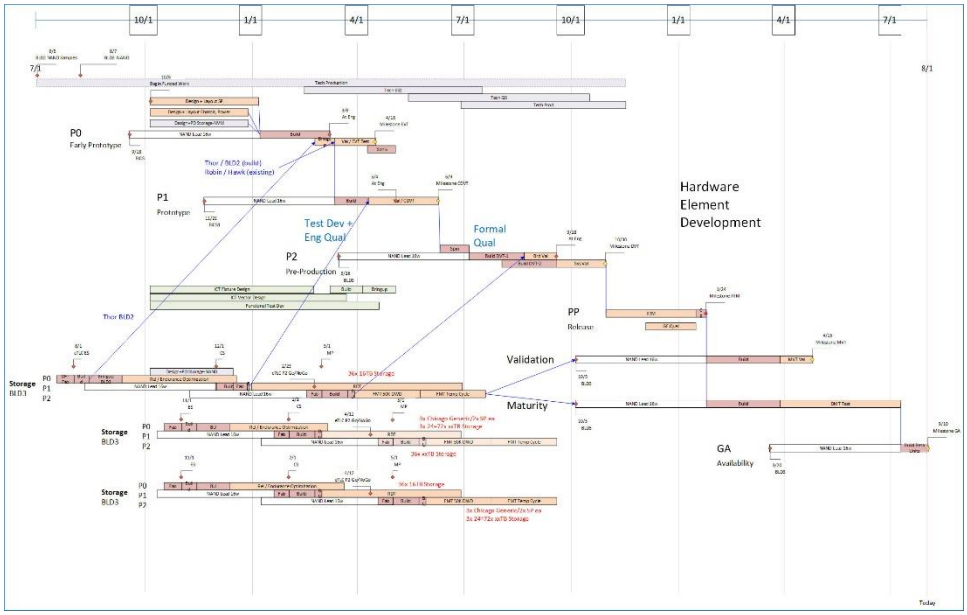
Conclusion Technology costs, business-model logistical constraints, and closeness of dependency-deliverable relationships determine suitability of Agile or Waterfall methodology for each Element developed in a System program.

Each Element developed must examine its Dependencies on, and Deliverables to, Elements developed external to itself. Closer ties to date and sequence of dependencies and deliverables should bias to Waterfall for both consumer and provider Elements. Looser or virtualized ties permit the flexibilities of Agile methodology of both dependency consumer and deliverable provider Elements.

Coordination of hardware (almost always Waterfall) and software (Agile above a virtualization level, Waterfall below) can be accomplished by running program Phases in parallel, with specified and limited constraints from one Phase onto its Next-Phase. The constraints specify functionality required to begin tasks critical in the Next-Phase, in order for the Next-Phase to satisfy its purpose.

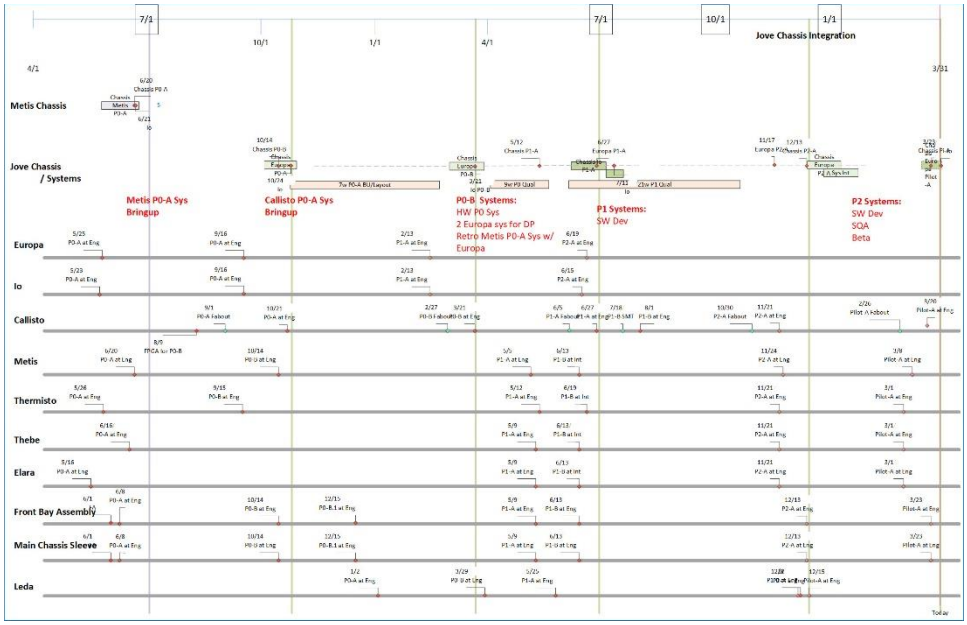
Postscript When a program is constructed in this way, what is the overall structure of the development program? To plan and manage such a program structure, the Power Operational Intelligence methodology and tools (Power OpI for short) were developed to manage complex efforts. This methodology integrates MS Project schedule plans with Excel logistics plans. Integration is done using database JOINs in SQL Server, Power Query, or MS Access, which provide great leverage to handle the complexity, and from which it can be reported. Several reporting tools can be used, but often focus on Excel or Power BI Pivot Tables. Reporting is also available using figures in MS Visio, some examples of which are shown below. Power OpI plans are fully integrated with referential integrity among the file components and can be analyzed and displayed at desired points of focus and with desired levels of detail. This is how plans are created and kept rapidly up to date through change and progress. The following images are from such a plan.

First, here is an image showing a fuller picture of development and integration of a hardware subsystem that is part of an overall system effort. This image shows a main path plus an additional board included in the subsystem, with phased hardware development and test in each phase. Dependence on external technology is included as well.



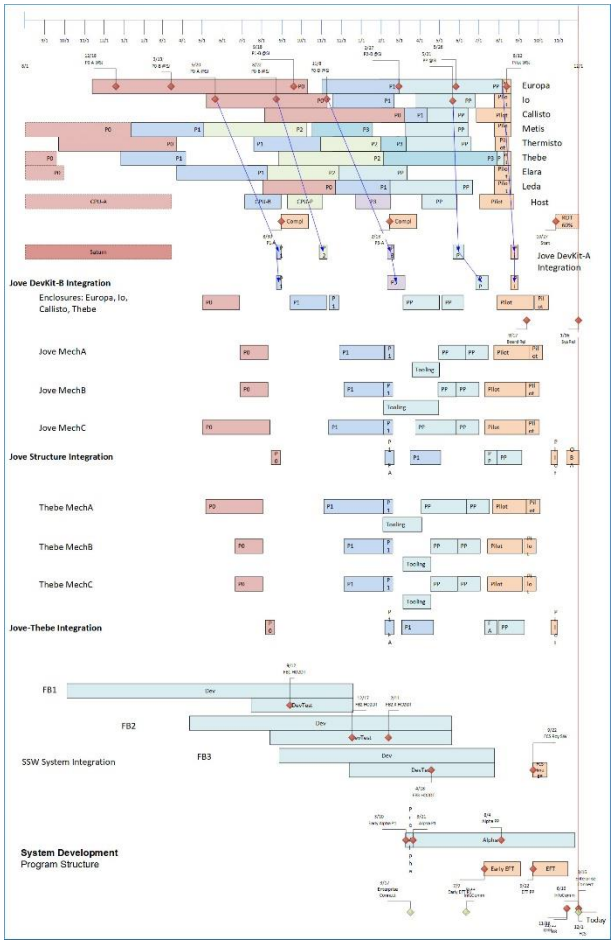
A system program generally consists of multiple such sub-assemblies. The following figure shows development of hardware board-level sub-assemblies (horizontal rows at bottom part of figure), timed for integration into system sub-assembly enclosures and then into full system assemblies in the top part of the figure. It is these integration points that are defined for Feature

Blocks of the system software described earlier, and the image shows alignment of Feature Blocks with the hardware Phases.



Finally, the full system effort can be shown in perspective and to scale in the following figure. The board-level sub-assemblies at top followed below by system sub-assembly and then by system mechanical structure development.

Following that is shown development of the three Feature Block structures described earlier, included in the integration steps, and used in system-level test. At the bottom of the image is shown deployment into customer environments first for test and leading to customer production use of systems produced.


Europa
Io
Callisto
Metis
Thermisto
Thebe
Elara
Leda
Host
Jove DevKit-A Integration
Jove DevKit-8 Integration
Enclosures: Europa, Io, Callisto, Thebe
Jove MechA
Jove MechB
Jove MechC
Jove Structure Integration
Thebe MechA
Thebe MechB
Thebe MechC
Jove-Thebe Integration
FB1
FB2
FB3
SSW System Integration
System Development
Program Structure
Alpha
Early EFT
EFT
Today