Developing Innovative Products Across Hardware, Software and Web Platforms

By Jeanne Bradford, John Carter and David Vermette, TCGen, Inc.

1. INTRODUCTION

Who needs this booklet?

Technology products today involve customer experiences that include at least two of these three platforms: *hardware, software* and *the web*. This booklet is for program managers, project managers and team members who develop *multi-platform products: a product that requires two or more of these three platforms.*

The engineering disciplines associated with these three platforms each have their own, very effective methods. This booklet provides practical tools for integrating these varied engineering processes into a framework that drives customer-centric product delivery.

- This booklet is for leaders, at any level, who face the task of managing distinct engineering platforms, while delivering the best possible customer experience, in the most efficient and cost-effective way.
- It is for the hardware, software or web manager who realizes that a successful delivery of multi-platform products cannot rely on multiple, parallel processes that are hastily integrated at the end of the project.
- It is also intended for Program Managers, especially hardware developers, who find working with software and web developers to be a challenge.

What will this booklet do for you?

This booklet gives your organization a framework and specific tools that drive successful, customer-focused, multi-platform product development projects. It first presents some of the challenges of multi-platform development. It then discusses a threefold approach for integrating hardware and software disciplines. Finally, we present five tools that help to integrate multiple platforms into a successful product delivery.

The material in this booklet helps teams:

- Increase the likelihood that they will deliver products that customers truly want.
- Improve the integration of hardware and software to increase speed to market and to enhance the quality of new offerings.
- Work more effectively with their hardware, software or web counterparts to slash waste, create efficiencies and accelerate timelines.

2. THE CHALLENGE OF MULTI-PLATFORM PRODUCT DEVELOPMENT

More Stovepipes Than Ever

Scenario: A technology company is creating a consumer electronics solution that involves hardware, software and web components. Three separate teams, located in different parts of the company's facility, create these three platforms. Each platform has its own development team, its own Program Manager and its own Project Manager. There are no clear and distinct processes for integrating these three elements until the final stages of the project.

Almost w/o exception, technology companies are organized around *functions* and people have a fierce loyalty to their discipline. People live within their own functional world and only occasionally appear at a cross-functional meeting to discuss product delivery.

Technology companies tend to have *cultures that place functional allegiance above cross-functional delivery*.

When three, separate teams execute projects in parallel, they each follow their own methodologies, with very little co-ordination. They inevitably run into problems that could have been avoided had they looked at the product they were creating from the customer's perspective, as an integrated whole rather than as a collection of components. This disconnected, disintegrated approach to product development creates delays, rework, waste and frustration when the teams need to integrate on the back end.

Functional Allegiance Trumps Customer Focus:
Technology companies deliver *a cross-functional product* but they tend to *organize along functional lines*. Companies are preoccupied with internal issues and lose track of their customers. Lack of collaboration causes costly delays, rework, and organizational churn.

Typically, hardware managers become frustrated since their teams have the longest lead times, as well as milestones that require risk builds. Hardware managers often face negative feedback because their teams are perceived to be less responsive to change than their software or web colleagues.

For decades companies have heard about the need to break through the stovepipes and about "flatter" organizations. However, the complexity of today's multi-platform products has *increased* the tendency of organizations to divide into stovepipes, and these fiefdoms jealously defend their turf.

The salient fact is that customers don't care at all about internal structures or internal politics. Teams that want to deliver excellent multi-platform products need to focus on what customers care about the most. The customer-driven approach is for all three teams to contribute to prototypes that resemble the end product that the customer is going to use. *The cross-functional product development process should focus on the customer rather than on mirroring the company's structure.*

Real Differences Between Hardware and Software

Scenario: An exciting new product is in the pipeline. A CEO or SVP comes to have a look at some preliminary designs. She says to the web team, *"This doesn't look sleek enough. We want the product to look thinner and less clunky."* She then delivers very similar feedback to the software team and to the hardware team.

The hardware team and the web team have totally different perspectives on the design implications of this senior management intervention. The hardware team is now sent into a spin as it considers component layout, available technology, suppliers, or whether to move a port from one part of the device to another.

When compared with software or the web, hardware tends to be the least responsive to fast changes. This is natural. It's baked into the very real differences between hardware and software. Unlike their software counterparts, many hardware projects:

- Tend to have longer lead times.
- Rely on components that also have long lead times for manufacturing or delivery.
- Depend on development partners that do things "their own way."
- Include risk buys on materials, often from a single supplier.
- Have components sub-assemblies, and final assemblies created at different locations around the world that require integration.
- Are in the medical products field and require lengthy regulatory compliance cycles.

When organizations mandate that the hardware teams become more agile, without considering the unique challenges of hardware development, then teams tend to become less empowered and project risks increase.

There are real differences between hardware and software. *We do not advocate abandoning the fundamentals of hardware engineering.* Our aim is to make hardware as flexible and as responsive as possible, and to create a framework that enables collaboration between hardware, software and/or web teams.

The differences between hardware and software:
Organizations mandate that the hardware teams become more agile, without considering what is unique to the discipline of hardware engineering. And yet few organizations have a solution to this dilemma. They're generally not sure whose job it is to solve this problem?

No Common Language Across Platforms

Scenario: A hardware team arrives at the Design Validation Test (DVT) milestone. The team requires software and firmware in order to perform its tests and validate the design. How does software communicate with hardware to give them what they need? The term "DVT" appears nowhere on the software team's schedule. It's not a part of that team's lexicon.

The software team is focused only on creating the features that customers want the most. The software requirements for validating hardware's design are toward the end of the team's list of priorities. How does hardware know that software is going to be ready to validate their design when they speak entirely different languages?

The *lack of a common language* encourages stovepipe development and increases the difficulty of product integration. The value for the customer, however, is in a smooth integration of platforms that come together to create an excellent experience. How can teams collaborate to create excellent customer experiences when their vocabularies and timelines do not map to one another?

Applying software methods (like Agile) to hardware, without significant modifications does not work. Different engineering disciplines must remain free to work within their own process, while *a common language* and *set of tools* creates *a bridge* between the worlds of hardware, software and the web.

We Do Not Speak The Same Language:
Hardware, software and Web teams speak different languages. This inhibits coordination and distracts the team from the customer. The solution is not to give up the fundamentals of the different engineering disciplines but to create a bridge between them that enables cross-platform integration and shifts attention to customer needs.

3. A SOLUTION FOR MULTI-PLATFORM DEVELOPMENT

Multi-platform developers need a *set of tools* and a *framework* that enable hardware to maximize its adaptability and speed. These same tools should drive seamless, cross-platform integration.

Multi-platform developers also require *a common language* that will allow teams to integrate and leverage the strengths of the different engineering disciplines into an end-to-end process that focuses on customers.

Companies begin to meet these requirements through a threefold process:

1. Foster a common culture for product development across platforms.
2. Create a mindset shift from a functional focus to a customer focus.
3. Build a bridge between hardware and software processes that becomes a common language across platforms.

---

1. Foster a common culture for product development across platforms.

The *Agile Manifesto* promoted *an attitude* as much as a set of processes. It promoted cultures among software developers characterized by flexibility, trust, team empowerment, and frequent communication between team members. None of these cultural traits are unique to software and all of them are essential for creating customer-centric, multi-platform products.

In a study we conducted with Silicon Valley executives, we found that of the highest rated Agile software practices, three were, at root, cultural: *daily contact with team members, strong team culture,* and *customer ownership*. *Daily standup meetings* are a specific Agile software practice that may or may not apply to hardware. However, the broader purpose of these meetings – fostering a culture of frequent communication and functional integration – is not software-specific. Each of these three Agile attitudes we identified in our research are applicable to any team endeavor.

The following attitudes, cultural traits, and related processes move readily from Agile software to hardware:

- High performance teams
- Self-organized teams
- Senior management trust and team empowerment
- Customer ownership & daily team interaction
- Tolerance around changing requirements

---

2. Create a mindset shift from a functional focus to a customer focus.

As multi-platform development teams begin to have a common culture and a common language, they start to emerge from their internal, functional world, with its concerns, and start to focus on customers. Rather than structuring projects around functions, *they begin* *to create prototypes and interim products that resemble the end product the customer is going to use.*

This approach depends on frequent integration points and a common timeline that binds together the hardware, software and web teams. It also implies an *empowered team* that has considerable control over its resources and that learns and continuously improves.

---

3. Build a bridge between hardware and software processes that becomes a common language across platforms.

While there are significant differences between software and hardware projects, a *customer-centric focus* necessitates a bridge between the two. This bridge forms the starting point for a *common language* between the two groups. The bridge entails creating links between Agile development practices and best practices in hardware development. Once this divide has been bridged, then the separate teams begin to speak a common language.

Beginning from the language of Agile software we can map common terminology and techniques that strengthen both the software and the hardware domains and put them on the same page.

Agile Software Technique	**Hardware Development Counterpart**
User Stories ➡	Boundary Conditions
Burn-down charts ➡	Deliverable Hit Rate Chart
Sprint ➡	Hardware Intervals
Empowered teams ➡	Out of Bounds Process
Sprint retrospectives ➡	Event Timelines

Harmonizing Agile Best Practices with Hardware Development

Below we describe in detail how to apply each of these hardware tools to create a common framework for multi-platform development projects. *Each of these tools may be used in hardware programs alone*, as well as in multi-platform development projects. Used in hardware programs, these tools will help to improve product quality, project speed, and enhance high performance teamwork.

4. TOOLS FOR EFFECTIVE MULTI-PLATFORM DEVELOPMENT

Boundary Conditions

What is the tool?

The Boundary Conditions Diagram is a tool that identifies the critical elements of a project that the project must have in order to ship a successful end product to market. It creates alignment around these priorities among all team members and between members of different teams. When you apply this tool early in the development process, typically at the time of project funding, the Boundary Conditions Diagram creates a lightweight plan of record and helps keep your team focused on the most important aspects of the project.

The Boundary Conditions Diagram helps the team to identify the three or four critical elements of a project typically *features, cost, schedule, and/or quality*. These boundary conditions define the acceptable parameters for a project. The project and management teams agree to a contract based on the agreement that the team will manage to these boundary conditions and that Management will leave to team to its own devices as long as the team remains within the boundaries. These boundary conditions then drive project tradeoff decisions throughout the entire development process.

Why use this tool?

The boundary conditions approach empowers teams to operate freely as long as they remain within the parameters of the project. The tool ensures that teams understand the most important elements of a project. It helps teams gain alignment and creates a common language by defining the elements of the end product. It also provides a clear framework for decision-making.

The tool reduces the number of delays since top management tends to intervene less often. It accelerates innovation by ensuring alignment and a focus on the most critical elements. It provides clear project priorities, across all of the platforms, in that it creates a clear distinction between “must-have” elements and the rest of the project scope.

The Boundary Conditions approach also ensures that the team and management make any necessary project tradeoff decisions in the context of the boundary conditions, which provide a framework for realigning the team if it violates one or more of the conditions.

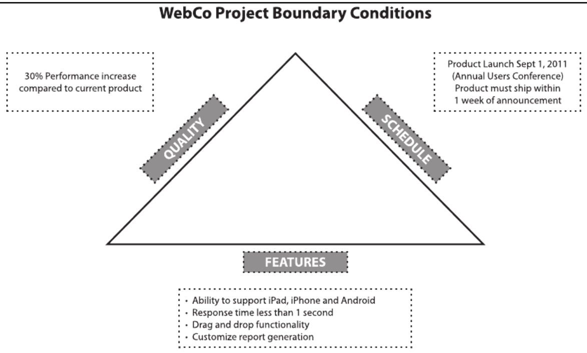
How do we apply the tool?

First identify three or four critical elements of your project. Then assign each element to a side of a triangle or a square (the number of critical elements in your project determine the shape of the diagram). State the boundary for each element and then place specific details to define the boundary conditions in boxes next to each boundary.

Here’s an example of boundary conditions for a hypothetical product. In this case, the team and Management agreed that product quality, features and the schedule were the most important elements. They agreed on the following boundary conditions for the product

- Performance: 30% increase over current product performance.
- Features: Ability to support iPad, iPhone and Android; Ability to generate a custom read-out of the product’s previous maintenance history

- Schedule: The product must launch by September 2016 due to an annual user's conference. The product must be available before this major customer and industry event.


WebCo Project Boundary Conditions
30% Performance increase compared to current product
QUALITY
Product Launch Sept 1, 2011 (Annual Users Conference) Product must ship within 1 week of announcement
SCHEDULE
FEATURES
• Ability to support iPad, iPhone and Android
• Response time less than 1 second
• Drag and drop functionality
• Customize report generation

Example of Boundary Conditions with three parameters; the boundaries are shown on the edges of the diagram.

Once you have created the Boundary Conditions diagram, refer to this drawing in meetings between management and the team throughout the development process. This essential contract between the team and management gives the team the authority to plan and execute the project with minimal intervention, creating a flexibility that enhances multi-platform development.

Knowing that most projects run into challenges along the development path, we have also created a complimentary process, the Out-of-Bounds Check. When the team crosses the boundary conditions, they are required to execute the Out-of-Bounds Check, which helps get the team back on track (see page 23).

Deliverable Hit Rate

What is the tool?

This chart monitors the progress of completed tasks against a target over time. Best applied to complex programs with a large number of tasks, the tool is a high-level graphical representation that indicates whether the rate of task completion is on track for delivering the program on time.

The Deliverable Hit Rate Chart is a high-level snapshot that indicates the true status of a large program. The chart allows a manager to step above the details and get an accurate read on whether or not the team is moving at the right speed. The Deliverable Hit-Rate Chart shows *actual work accomplished* as compared with *how the project was planned.*

Why use this tool?

The tool is easy to construct and provides an early warning when the team is not executing the target number of tasks. It then drives corrective action to get the team back on track. It is an elegant solution for communicating a large amount of data in an easy-to-interpret graphical representation.

How do we apply the tool?

Construct the Deliverable Hit Rate Chart by dividing the total number of tasks required to complete the program by the program's duration, typically measured in months. At the end of each month, map the number of completed tasks against the target for that time period. Then, update the line chart to present the actual deliverable hit rate.

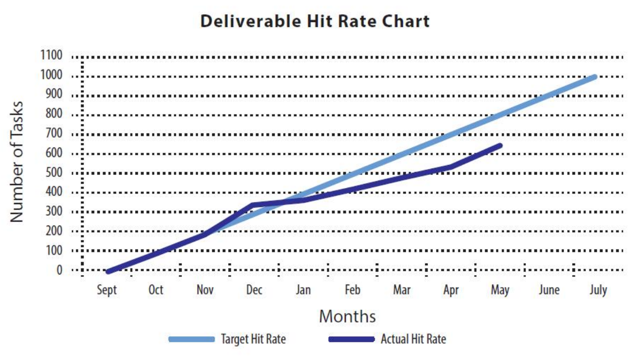


Chart shows the actual vs. target rate of completed deliverables over time.

- Vertical axis defines the total number of tasks tracked in the project.
- Horizontal axis is a function of time as defined by the project schedule.
- Shows the contrast between the targeted and the actual rates of completed tasks.

To apply this tool to *multi-platform projects* take the following steps:

- Identify the key tasks that should be accomplished during a Hardware Interval (see page 20)
    - Can features be implemented?
    - Can features/specs be validated?
    - Can tasks be performed?
    - Is there a customized metric that will gauge progress?

- The nature of this list of tasks varies from interval to interval.
    - **The front end** is weighted toward *product definition.*
    - **The middle of a project** is weighted toward *accomplishing tasks.*
    - **The back end** is weighted toward *validation.*

- Create a target curve over the hardware interval
    - Don't become overly concerned about perfection.
    - Assume that the events can be distributed evenly, unless you have clear knowledge otherwise.

Hardware intervals

What is the tool?

Hardware Intervals are short execution cycles, based on meaningful deliverables, that occur within the broader product development timeline. The end point for each of these cycles should be linked to project integration points, where hardware, software, and firmware come together to create a quick prototype of the product. The aim is to create a series of rapid prototypes that come as close as possible to the product that a customer might eventually purchase.

With Hardware Intervals, the team divides the project into the smallest possible increments, as long as these increments represent genuine integration points or milestones. The intention is to divide hardware development into iterative cycles with short feedback loops.

Why use this tool?

Hardware intervals help to break down stovepipes and create customer focus. They accomplish this by integrating hardware and software more frequently, over the course of the development cycle.

Hardware Intervals:

- Prevent costly mistakes caused by infrequent hardware/software integration points.
- Encourage rapid learning that places the focus on customers.
- Encourage cross-functional coordination and break down stovepipes.
- Encourage hardware and software to speak a common language that eases integration while it mitigates avoidable communication issues.

How do we apply the tool?

In a six to eight week period, a software team might have three or four two-week Sprints while the hardware team has a single Interval. The aim is to align the end of each Hardware Interval with a software (or firmware) integration point. Each hardware interval ends at a clear integration point, with a definable milestone that represents a clear product development goal. In most cases, hardware teams will need to increase the number of integration points in order to align the hardware and software timelines.

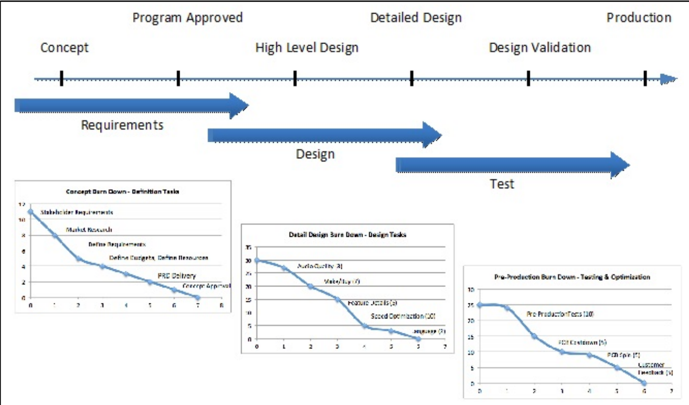

Program Approved
Detailed Design
Production
Concept
High Level Design
Design Validation
Requirements
Design
Test
Concept Burn Down - Definition Tasks
Detail Design Burn Down - Design Tasks
Pre-Production Burn Down - Testing & Optimization

Diagram showing how Burn Down charts may be used to plan Hardware Intervals.

In most cases, the Hardware Intervals will correspond roughly to Engineering Validation Test (EVT), Design Validation Test (DVT) and Production Validation Test **(**PVT), however, each of these larger phases are subdivided to correspond with milestones or integration points. For example, the first Interval may include all activities up to prototype build. At this point, the hardware team integrates, as far as possible, with the software and firmware team in order to create a prototype. Only then, when the team has learned as much as possible from the prototyping phase, will the team move on to the Intervals related to DVT and then PVT.

D. Out of Bounds Map

What is the tool?

In the Boundary Conditions process described above, the team and management negotiate parameters at the time of project approval around such factors as *schedule, features,* and *quality*. They then create a contract between the team and management that allows the team to move forward with minimal guidance provided that the project stays within the boundary conditions.

If the project crosses those boundaries, the *Out Of Bounds Map* is a mechanism that corrects the team's course and realigns it around a new plan of record. The tool provides a mechanism to quickly conduct a root cause analysis, evaluate alternatives, and recommend remedies.

Why use this tool?

The Out-of-Bounds (OOB) process creates a common language and a mechanism to quickly align project and management teams when a project faces a major change. It minimizes confusion within the team *and between the hardware team and the software and web teams* by establishing a single, agreed-upon communication path. It also diminishes wasted time since your organization does not need to create a process each time a deviation occurs. It helps your team realign projects within hours or days rather than weeks.

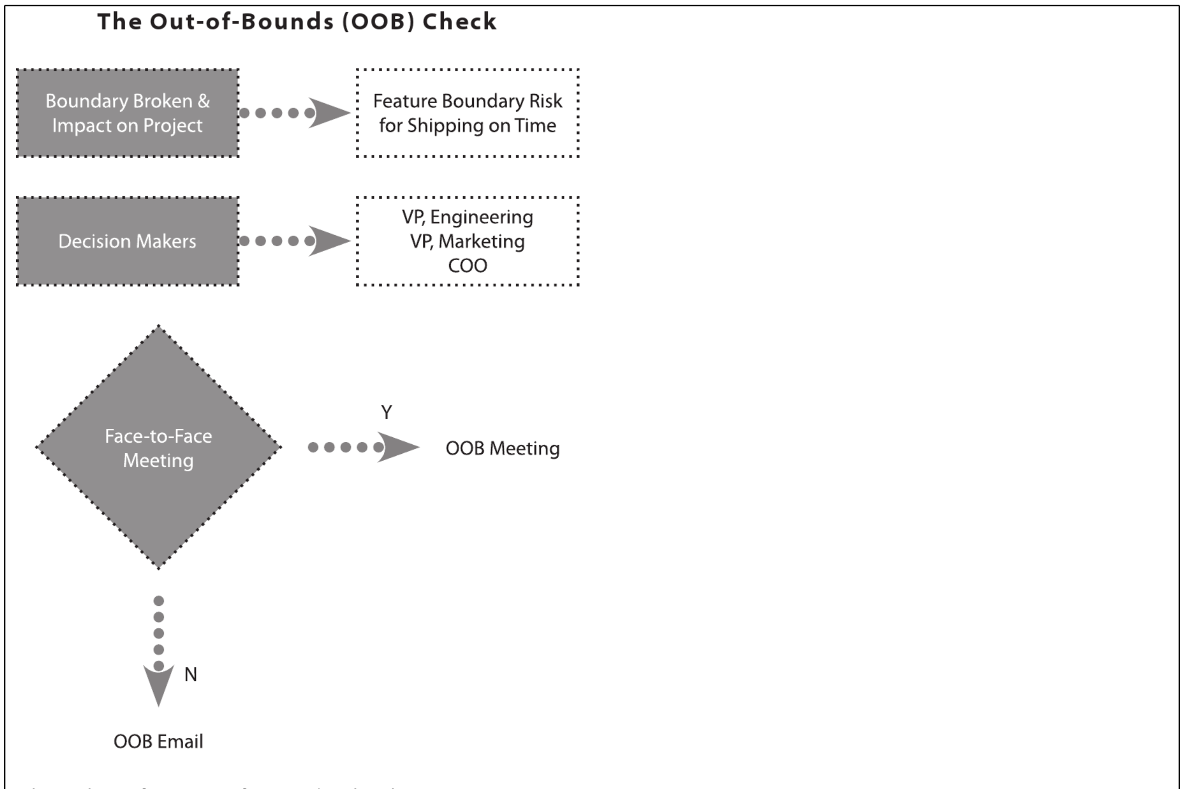
This process empowers the team to move forward with minimal guidance after management establishes the boundary conditions. The team becomes more engaged because of the greater trust that management places in them.

How do we apply the tool?

When a project team detects that an OOB condition may occur, the program manager gathers information to determine if the team can resolve the issue and still remain within the boundary condition. If the team cannot, the program manager crafts an OOB communication and sends it to key decision makers.

This communication indicates:

- Which project boundary the team will break or has broken.
- The root cause for the broken boundary.
- Alternatives to resolve the issue (with supporting schedule and/or cost-impact data).
- The remedy the project team recommends.


The Out-of-Bounds (OOB) Check
Boundary Broken & Impact on Project
Feature Boundary Risk for Shipping on Time
Decision Makers
VP, Engineering
VP, Marketing
COO
Face-to-Face Meeting
Y
OOB Meeting
N
OOB Email

Flow chart for Out Of Bounds Check.

The program manager can deliver this communication via email or in a scheduled meeting. Key decision makers respond by granting approval or by suggesting a modified approach.

To accelerate realignment, it is best to empower teams to communicate the recommendation and move forward unless they receive directions otherwise. It should be the intent of both the project team and key decision makers to complete this process quickly, i.e. within hours or days.

In the example diagram above:

- Time flows from the top to the bottom of the diagram.
- Steps are shown in gray
- Boxes to the right of each step provide details.
- The shaded diamond represents a decision point.

In this example, a face-to-face meeting with key decision makers was not practical due to international travel schedules. The program manager provided the above information in an email to the decision makers and copied the core project team in order to accelerate the process. Management approved the two recommended solutions sent by email.

The Boundary Conditions and Out-Of-Bounds process are critically important for multi-platform development. They focus on the conditions that will lead to a successful product launch. That means they focus on something a customer would care about, rather than on internal processes. Most importantly, they create a common framework and a common language that hardware, software and the web team use to communicate and coordinate. The result is faster decision making and better, customer-focused products.

Event Timeline Retrospectives

What is the tool?

Project post mortems are an excellent tool for learning from mistakes and implementing process and decision-making improvements. Fast cycles of learning and process improvement are essential in multi-platform development products due to the complexity of these types of projects and the number of inputs required to develop the finished product.

The Event Timeline is a simple tool that allows teams to get their project post-mortem started on the right track. It allows teams to come together at the end of a project to collect and prioritize information that will serve as a credible basis for the post-mortem process. It helps the team to conduct a focused, fact-based investigation by identifying the key *planned and unplanned* events that drive a project's performance. While typically conducted at the end of a project, this tool may be used in the middle of a project, as a tool for re-evaluating and resetting a project.

Why use this tool?

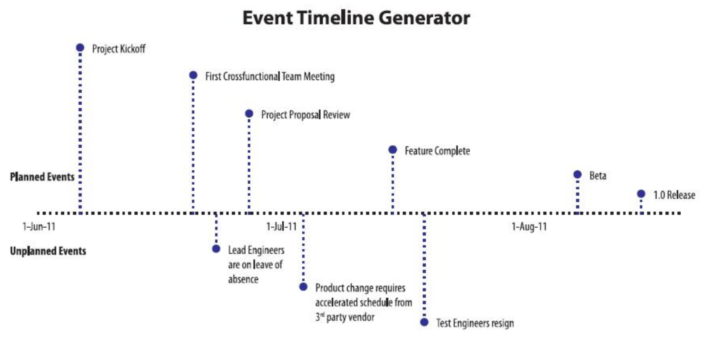
The Event Timeline tool is a fast and easy way to create and communicate a project history that takes into account the realities of what actually occurs in projects. It flags those *unplanned events* that cause schedule delays and increase costs. It also provides a holistic view of project execution.

Most importantly, this tool is a tool for organizational learning. This is an essential skill as product development programs become more complex. Finally, the tool supports a fact-based, objective analysis of the shortcomings in project execution and supports incremental process improvement.

This tool allows you to drive a root cause analysis and understand deeply the causes and effects of unexpected changes. This means that managers can make better and faster decisions, to avoid the impact of these changes or to help get the team back on track.

How do we apply the tool?

The project manager completes the Event Timeline Generator with input from the cross-functional team. The Event Timeline template displays events as a function of time. *Planned events*, for example the major milestones in the product development process, appear above the timeline. During the execution of the project, as they occur, unplanned events are entered below the timeline.



The Event Timeline Generator maps both planned and unplanned events.

Updating the timeline with unplanned events is important because such events have a material impact on the speed and cost of delivering a project.
For instance:

- What would be the impact if your lead architect resigned during the concept phase of the project?
- What if you reassign key team members for two weeks to resolve a burning customer issue?
- What if an early *risk buy* of materials leaves your team with yields too low to ramp up your project?

The Event Timeline Generator allows you to record unplanned events as they occur and to make the best decisions to mitigate the risks such events entail.

Also, when you capture unplanned events, you create an effective tool for conducting mid-course reviews.

Teams should conduct retrospectives for every program and the Event Timeline tool is just one important part of these retrospectives. Create a consistent, formal process for each project that has the following attributes:

- Fact based, and data driven
- Reviewed at every interval.
- Involving cross-functional team members.
- The team should own the process.

Other elements of a retrospective process, in addition to Event Timelines, may include:

- *Root Cause Diagram*, a tool for identifying ultimate causes of complex phenomena
- *Affinity Diagram*, a tool for analyzing and synthesizing language data.

5. CONCLUSION

In 2013, we interviewed executives in about 20 companies, mostly Silicon Valley technology firms with experience in Agile software development. Our research found that those aspects of Agile that were most effective were *more cultural than procedural* and that many of Agile's attitudes could be modified and applied to any team endeavor.

This is evident in the Agile Manifesto itself. The Manifesto's authors declare that they 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.

Despite the word "software" in the second statement above, in principle, none of these values speak to software development alone. They also apply to multi-platform programs, or any program that wishes to reap the benefits of close collaboration with customers, and the ability to adapt to changes in requirements or other market conditions.

Organizations can begin to reap these benefits, across hardware, software and web platforms by:

- Embracing a culture of team empowerment and trust.
- Communicating frequently with customers and with each other.
- Accepting the inevitability of change.

As a result, hardware and other development programs might begin to produce the type of results we observed in our research with Silicon Valley executives, such as:

- Reducing time to market by a factor of four.
- Cutting internationalization time from 11 months to 1 month.
- Shaving 40-50 percent out of the product development cycle.

The tools and concepts we describe above are not about applying Agile to Hardware. They are about a cultural change from a *"freeze the specs and don't look back"* mentality to a *"develop fast, and then learn and improve"* mindset. The approach we describe in this booklet is about a cultural shift that looks in part to Agile software for help in creating that shift.

Multi-platform programs are too complex and the challenges of integration are too weighty for a "freeze the spec" approach. A better approach for multi-platform projects is to create a common language and a common timeline that hardware, software and the Web team will align around. The tools and approaches presented in this booklet create this common language by:

- Defining the basic boundaries for a project.
- By empowering teams to work unhindered within these boundaries.

- By creating intervals that require cycles of iteration and learning, with frequent integration points.
- That allow teams to learn all they can from projects and then to apply that learning to the future.

Further Information

For more information, please visit our website at www.tcgen.com or write to one of our Principals, Jeanne Bradford (jbradford@tcgen.com) or John Carter (jcarter@tcgen.com).