Accelerating embedded software development via agile techniques
The nine strategies that lead to successful embedded software development
Introduction

The accelerating rate of technological change for electronic products requires rapid market responsiveness to maintain a competitive edge. Hardware-centric product industries have increasingly turned to the flexibility and adaptability of software to enable critical features and functionality that differentiate their products. As a result, embedded software use is growing to the point that it is becoming omnipresent across multiple industries.

The growing importance of software as a differentiator, combined with increasing product complexity, and more frequent release cycles are increasing the cost of embedded software development. Many companies that rely on embedded software are exploring agile software development to cut costs while improving quality and time to market. The challenge for them is that agile software development methodologies differ from traditional hardware product development models, which are typically linear and more rigid in nature. Instead, agile practitioners use a highly adaptive and iterative development cycle based on a collaborative team approach.

Although agile has proved beneficial in mainstream software development scenarios, implementing an agile software development framework within the embedded software environment poses a number of unique challenges. After analyzing leading practices of agile software development leaders, PwC has developed an end-to-end structured approach with nine key strategies to help hardware companies successfully adapt agile practices for their embedded software development. (Figure 1) PwC’s nine strategies, along with key processes and planning activities, can help companies that rely on embedded software reap the value of agile software development.

This article is for product development executives, managers, and others in the many industries that incorporate embedded software. It includes a maturity model that can help organizations identify whether they are leaders, laggards or in the mainstream of embedded software development, as well as guidance about how to improve.

---

**Figure 1: Nine agile development strategies for embedded software**

1. Create a Technical Owner role to complement the Product Owner.
2. Implement hardware milestone-driven release plans.
3. Create a well-defined role for embedded software architecture.
4. Introduce the nomenclature of “System stories”.
5. Incorporate a story dependency-mapping exercise into the release and iteration planning cycle.
6. Institute iterations longer than 2 weeks.
7. Implement a cross-discipline technical leadership team.
8. Implement a cross-functional program core team sponsored by executive leadership.
9. Redesign and implement System Test activities as agile iterations.

Source: PwC
The growth of embedded software

From pacemakers to smart phones to set-top boxes to complex network routers, technology products that use software in an embedded environment are extensive and varied. Embedded software use is a fast growing area, with applications varying by industry application, form factor, and footprint. That growth should continue, as the electronics industry faces an ever-accelerating demand for new chipsets that are faster, smaller, and cheaper and can handle an ever-wider variety of applications.

For example, products such as a smartphone are based on multiple hardware chipsets (such as for baseband and application) requiring numerous external suppliers. These devices can contain up to 50 million lines of embedded software code. (Figure 2)

Developing these complex products requires OEMs to establish broad ecosystems which have grown more complex, with more participants catering to multiple layers of the application stack. At the same time, those ecosystems must work quickly, as OEMs and ODMs accelerate product development to quickly generate new product features and enhancements, while maintaining product quality. Thus, expenditure on software development is becoming a significant portion (often >50% of R&D resource costs) of the overall development expense for embedded product companies, according to PwC estimates.

Given such market pressures and the growing prominence of software, embedded product development organizations are increasingly interested in harnessing the power of agile software development to improve time-to-market, productivity, and quality while maintaining lower costs. However, while agile software development holds great promise, doing so within the typical embedded software environment poses a number of challenges not found in typical application software development.

Background and benefits of agile

Software development has traditionally followed waterfall-style development techniques made popular by manufacturing and construction industries in the 1970s. The waterfall method breaks down software development into discrete steps: requirements, design, coding, and testing, with each step occurring in sequence with no overlap. This linear sequence makes it difficult for development teams to adapt to changing requirements and market needs; moreover, mistakes made in an earlier step are costly to fix in latter steps. Waterfall development also tends to create natural silos, due to the functional nature of the discrete steps, with “throw-over-the-wall” behavior common between product managers, architects, developers and testers.

Figure 2: Lines of code* in device

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pacemaker</td>
<td>250K</td>
</tr>
<tr>
<td>Set top box</td>
<td>1.5M</td>
</tr>
<tr>
<td>Smart TV</td>
<td>3M</td>
</tr>
<tr>
<td>Cellular base station</td>
<td>4M</td>
</tr>
<tr>
<td>Advanced video games console</td>
<td>5M</td>
</tr>
<tr>
<td>Commercial aircraft</td>
<td>18M</td>
</tr>
<tr>
<td>Fly-by-wire system</td>
<td></td>
</tr>
<tr>
<td>Luxury automobile</td>
<td>20M</td>
</tr>
<tr>
<td>Network equipment operating system</td>
<td>20M</td>
</tr>
<tr>
<td>Smart phone</td>
<td>50M</td>
</tr>
<tr>
<td>and many, many more...</td>
<td></td>
</tr>
</tbody>
</table>

* These are high level estimates for product category
Agile principles transform traditional software development capabilities to deliver value

“Agile” refers to a set of adaptive software development methodologies that share common characteristics: working software, responsiveness to change and collaborative processes.

Agile methodologies are completely different—they are adaptive and iterative. Key characteristics of agile development are as follows:

- Intentional blending of define-build-test cycles and continuous integration of software (Figure 3)
- Frequent releases of fully tested features, ready for use by the customer
- Highly collaborative work teams comprising both business people and developers
- Reliance on face-to-face communication as the dominant form of interaction
- High value placed on simplicity

Agile also offers additional benefits for effective software development, such as early customer engagement, early test engagement, focused engineering, enhanced collaboration and empowerment, early program risk visibility, and consistency. The aggregate quantitative impact of these benefits appears across a number of vital industry performance metrics (Figure 4)

---

**Figure 3: Governance and teams**

**Figure 4: Agile development impact on key benchmarks**

<table>
<thead>
<tr>
<th>Measure</th>
<th>Benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality improvement</td>
<td>31% 50%</td>
</tr>
<tr>
<td>Productivity improvement</td>
<td>14% 97%</td>
</tr>
<tr>
<td>Time to market improvement</td>
<td>18% 91%</td>
</tr>
<tr>
<td>Cost reduction</td>
<td>7% 29%</td>
</tr>
</tbody>
</table>

Clearing roadblocks to embedded software agile implementation

PwC has analyzed the leading practices of agile leaders, as well as challenges facing laggards, and created a set of recommendations and strategies to enable agile development of embedded software. Collectively the characterizations of leaders, laggards and mainstream implementation of agile development highlight relative levels of maturity, so the table below can also be considered a maturity model. (Figure 5)

Developers of embedded software should embrace a transformative approach that encompasses the entire product development cycle and provides a holistic solution for end-to-end agile development. Therefore, these recommendations and strategies affect both agile methods and agile team structures and behaviors.

Figure 5: Agile embedded software development maturity model

<table>
<thead>
<tr>
<th></th>
<th>Laggards</th>
<th>Mainstream</th>
<th>Leaders</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Iteration planning and execution</strong></td>
<td>• Iteration backlogs do not exist</td>
<td>• Backlogs exist, but prioritization is siloed</td>
<td>• A leading iteration planning and execution framework exists to cover backlog management, estimation, agile ceremonies, testing, and support roles/responsibilities</td>
</tr>
<tr>
<td></td>
<td>• Estimation of effort is informal</td>
<td>• Accuracy of estimates vary by project and individual</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Agile ceremonies (e.g., daily stand-ups, iteration reviews, retrospectives) are not followed</td>
<td>• Agile ceremonies are loosely followed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Unit testing is informal</td>
<td>• Unit testing processes exists, but practices are ad-hoc</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Agile support roles and responsibilities are not defined</td>
<td>• Agile support roles are fulfilled by developers</td>
<td></td>
</tr>
</tbody>
</table>

| **Requirements and backlog management** | • User stories are not used                                                | • Teams define and develop user stories, but lack dependencies and additional aggregation levels | • A formal requirements hierarchy is defined and documented, linking user-stories, themes, and epics |
|                                          | • Requirements are inconsistently documented and lack testability and traceability | • Requirements are documented but testability is lacking                   | • Requirements are measurable, and test cases are developed and maintained for all requirements |
|                                          | • Change requests are not formally captured and tracked                    | • Formal functional change control board exists                            | • Cross-functional change control board exists                            |

| **Continuous integration**              | • No continuous integration of code due to lack of clear branching strategy | • Branching strategy exists, but rules are inconsistently defined and practiced | • Continuous integration from a single source code repository using automated builds |
|                                          | • Lack of automated build-level testing                                   | • Builds include smoke tests                                              | • Static analysis and successful unit test suite execution are requirements of a successful check-in |

<p>| <strong>Tools and infrastructure</strong>            | • No tools deployed for management of defects, test cases, configuration, build, backlog, portfolio, and resource | • Management tools are deployed, but information transfer is loose         | • Tools are available and integrated wherever possible to enable automated and seamless transfer of information |
|                                          | • No tools for code analysis and test automation                          | • Basic code analysis tools exist, but usage is inconsistent; Automation is only for critical features |                                                                        |</p>
<table>
<thead>
<tr>
<th><strong>Tools and infrastructure</strong></th>
<th><strong>Laggards</strong></th>
<th><strong>Mainstream</strong></th>
<th><strong>Leaders</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• No tools deployed for management of defects, test cases, configuration, build, backlog, portfolio, and resource</td>
<td>• Management tools are deployed, but information transfer is loose</td>
<td>• Tools are available and integrated wherever possible to enable automated and seamless transfer of information</td>
</tr>
<tr>
<td></td>
<td>• No tools for code analysis and test automation</td>
<td>• Basic code analysis tools exist, but usage is inconsistent; Automation is only for critical features</td>
<td></td>
</tr>
<tr>
<td><strong>Release roadmap planning</strong></td>
<td>• Value proposition and business case for the roadmap is not consistently developed</td>
<td>• Value proposition of the roadmap is defined and documented. A qualitative business case for the release is developed and documented</td>
<td>• Value proposition of roadmap is defined and documented. Business case includes quantitative cost-benefit analysis</td>
</tr>
<tr>
<td></td>
<td>• Roadmap planning process is ad-hoc; Release strategy is non-standard</td>
<td>• Roadmap planning and release cadence are defined, but not consistently followed</td>
<td>• Release roadmap planning and cadence is thoroughly followed with documented governance and roles/responsibilities for all release types</td>
</tr>
<tr>
<td><strong>Release management</strong></td>
<td>• Release methodology is not distinguished from development methods</td>
<td>• Release methodology is defined and guides preparation of code for release and distribution</td>
<td>• Release governance linked to cross-functional product and release lifecycle process</td>
</tr>
<tr>
<td></td>
<td>• Release scope/criteria/planning is loosely defined</td>
<td>• Release scope/criteria/planning adopted, but ignored under pressure</td>
<td>• Release scope/criteria/planning prioritized, rationalized and consistently applied</td>
</tr>
<tr>
<td><strong>Architecture and design</strong></td>
<td>• Architecture and design are largely by-products of coding. Interfaces are not formally defined nor managed</td>
<td>• Component-based system architectures coordinate the interaction and design of components. A System Architect role is formally defined but inconsistently fulfilled across programs and change control is inconsistently executed</td>
<td>• Cross-product coordination exists for architecture development and management. System architect from a central architecture group has formal responsibility to support the design lead for each program and ensure cross-product alignment. Interfaces have clear and testable contract</td>
</tr>
<tr>
<td><strong>Global R&amp;D planning and orchestration</strong></td>
<td>• No formal mechanisms around talent acquisition, skill development, retention, site strategy, offshoring, and teaming</td>
<td>• Semi-formal mechanisms exist with non-uniform execution around talent acquisition, skill development, retention, site strategy, offshoring, and teaming</td>
<td>• Formal mechanisms, metrics, and uniform execution for talent acquisition, skill development, retention, site strategy, offshoring, and teaming</td>
</tr>
<tr>
<td><strong>Portfolio management</strong></td>
<td>• No formal decision processes in place</td>
<td>• Decisions are focused on resource allocation rather than portfolio balance and strategic fit</td>
<td>• Decision process is fully cross-functional with active involvement of business management at regular cadence</td>
</tr>
<tr>
<td></td>
<td>• No clear roles and responsibilities defined</td>
<td>• Roles are not clearly defined but a true attempt is made at cross-functional portfolio management</td>
<td>• Roles and responsibilities are clearly defined with specific responsibilities for individual parties throughout the portfolio review cycle</td>
</tr>
</tbody>
</table>

Source: PwC
Successful adoption of agile practices requires a careful adaptation of methods and patterns that are aligned with business objectives and organizational maturity, and supported by a continuous focus on organizational change management that addresses barriers and resistance to change. Using such an approach, PwC has helped clients transform their embedded software development practices to become more agile. (Figure 6)

**Figure 6: A structured approach to agile development transformations**

### Overall transformation approach

#### Top-down

- Work with Executive Leadership to define business objectives and articulate the case for change

#### Build from middle

- Form cross-functional working group of thought leaders and champions to adapt practices and capability changes

#### Drive bottom-up change

- Selectively pilot new processes and capabilities. Refine designs based on pilot feedback and deploy enterprise-wide

### PwC-Client engagement approach

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 3</th>
<th>Task 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline and plan</strong></td>
<td><strong>Design</strong></td>
<td><strong>Pilot and refine</strong></td>
<td><strong>Scale and sustain</strong></td>
</tr>
<tr>
<td>Baseline current hardware and software development practices and identify key organizational challenges</td>
<td>Develop details of agile practices in collaboration with a cross-functional design team using PwC’s agile framework</td>
<td>Pilot agile practices and refine design based on results</td>
<td>Institutionalize agile and develop plans for wide roll out</td>
</tr>
<tr>
<td>Define overall transformation plan</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Nine vital strategies

Once a structured transformative plan is in place, PwC recommends the following nine strategies to resolve the challenges of supporting embedded software development via agile techniques.

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Strategy</th>
</tr>
</thead>
</table>
| **Challenge 1: Nature of the product is deeply technical and challenging for Product Owners.** The Agile Product Owner, typically from the product management discipline, brings in the customer perspective to build out prioritized work content for the agile teams. However, the complex nature of embedded products makes it difficult for the typical Product Owner to effectively guide agile teams. | **PwC Strategy 1:** Create a Technical Owner role to complement the Product Owner. The Technical Owner will bring deeper technical expertise to the following responsibilities:  
  - Collaborate with Product Owner for overall backlog prioritization based on technical dependencies and complexity  
  - Support the team’s effort to simplify or reconfigure user stories so they can fit within an agile iteration cycle while maintaining key technical considerations  
  - Collaborate with other Technical Owners on identification of technical dependencies  
  - Collaborate with the Product Owner for acceptable technical exceptions to “Done” criteria for a given iteration  
  - Enable key technical decisions within the team to achieve iteration plan |
Challenge 2: Hardware development follows waterfall techniques, while software is agile. With hardware development typically following traditional waterfall style development, development organizations struggle to determine the right organization and process approach to adopt agile. (Figure 7)

PwC Strategy 2: Implement hardware milestone-driven release plans. Define software releases that are aligned to particular hardware development milestones, such as simulator readiness, or the availability of early proof of concept hardware. Another useful method is to organize the teams and categorize code into hardware-dependent and hardware-independent tracks.

Figure 7: Typical hardware development process

Challenge 3: Products are highly complex to architect. Traditional agile development emphasizes light up-front design and limited importance for the role of an architect, in favor of rapid Design-Code-Test iterations. However, developing embedded software for these products is complicated because of multiple chipsets from multiple suppliers, the multi-layered nature of software, and stringent performance, scalability and availability requirements. There is a need for upfront system architecture with expertise and decision-making capabilities to properly weigh hardware vs. software trade-offs as well as platform vs. application type architecture decisions.

PwC Strategy 3: Create a well-defined role for embedded software architecture by defining a mechanism for upfront end-to-end system architecture, system engineering decisions, hardware/software and application/platform partitioning. Define the role of an architect that incorporates the following core activities as well as development support:

Core architecture activities:
- Requirements analysis for right choice of architecture (hardware vs. software, platform vs. application)
- End-to-end architecture description across hardware-software and platform-application
- Documentation of key elements of architecture (e.g., system level APIs)
- System level performance and scalability analysis

Development support activities:
- Participation in epic and user story (requirements) definition
- Prioritization and resolution of architectural issues across teams
- Guidance on corporate technical standards and systems
Challenge 4: Work is more system-focused than end-user focused. In traditional agile development, work content is defined in the form of user stories, written in the “As a user.....” format. Most embedded software, especially products aimed at enterprises, lack the concept of an end-user, making it difficult to write user-focused stories. Also, early stages of embedded systems development involve low-level platform type work requiring “systems” thinking.

PwC Strategy 4: Introduce the nomenclature of “System stories”. Encourage stories to be written as “As a system, it will ...” or “The system should...” from the point of view of the system under development. This easily replaces the traditional “As a user...” syntax that traditional agile developers are used to.

Challenge 5: Functional interdependencies need upfront identification. Agile teams for embedded software development have both intra-and inter-team work dependencies. For example, an application team can start work on its stories only after the lower-level platform team builds the foundation. Also, software teams have dependencies on hardware to enable functionality.

PwC Strategy 5: Incorporate a story dependency-mapping exercise into the release and iteration planning cycle. Once hardware-based intra- and inter-team dependencies have been identified, they should be documented for a given story in the Agile Lifecycle Management (ALM) tool. Inter-connected stories, including dummy stories for hardware dependency, should be linked, and a communications process should be set up to identify dependencies between teams and prioritize implementation.

Challenge 6: Embedded agile teams often struggle to deliver meaningful work within typical 2 week agile iterations. Longer iteration lengths are typically required to offset the:

• High technical complexity inherent in embedded products and the ease with which the product design can be changed and tested
• Tightly coupled nature of hardware and software product architectures, and the challenges this poses for build and test automating
• Longer test and debug times necessary to isolate and triage between hardware vs. software issues.

PwC Strategy 6: Institute iterations longer than 2 weeks to accommodate the complexities and constraints of embedded development. At the same time, work towards the decoupling of the product architecture to improve the ability to more easily change, build, and test specific architectural layers and/or product components.
Challenge 7: Multiple agile teams need to be coordinated effectively. Support roles such as Scrum Master and Product Owner focus on one agile team. However, embedded product organizations have multiple agile teams. (Figure 8)

PwC Strategy 7: Implement a cross-discipline technical leadership team that manages across multiple agile teams, consisting of Scrum Masters, Product and Technical Owners, and Test and Architect Council members.

Figure 8: Adaptation—Cross-discipline leadership teams (an example)

<table>
<thead>
<tr>
<th>Support Team</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scrum of Scrums (SMs)</td>
<td>• Maintain a coordinated agile team schedule for planning and execution</td>
</tr>
<tr>
<td></td>
<td>• Resolve cross scrum-team roadblock issues impacting execution: Tactical items such as builds, equipment, short-term debug support, and specific defects requiring resolution</td>
</tr>
<tr>
<td>Product Owners (POs)</td>
<td>• Overall program priority setting related to product requirements and customer interactions</td>
</tr>
<tr>
<td></td>
<td>• Joint responsibility with TOs and Architects to define content (Release and Iteration Planning)</td>
</tr>
<tr>
<td></td>
<td>• Joint responsibility for prioritization of content across scrum teams</td>
</tr>
<tr>
<td>Technical Owners (TOs)</td>
<td>• Functional and detailed design planning, content integration, and cross-team issue resolution</td>
</tr>
<tr>
<td></td>
<td>• Joint responsibility with POs and Architects to define content (Release and Iteration Planning)</td>
</tr>
<tr>
<td></td>
<td>• Joint responsibility for prioritization of content across scrum teams</td>
</tr>
<tr>
<td>Test Council (TLs)</td>
<td>• Program level system test strategy, planning and coordination</td>
</tr>
<tr>
<td></td>
<td>• Definition of cross-scrum team test user stories</td>
</tr>
<tr>
<td>Architect Council</td>
<td>• System and program level architecture definition, planning and issue resolution</td>
</tr>
<tr>
<td></td>
<td>• Joint responsibility with POs and TOs to define content (Release and Iteration Planning)</td>
</tr>
</tbody>
</table>
**Challenge 8:** Strong, balanced governance structure needed for overall execution and market delivery. Given its cross-functional nature, agile implementation directly challenges traditional functional governance structures such as testing and program management. However, the complex nature of embedded software development requires effective governance to oversee overall execution and delivery.

**PwC Strategy 8:** Implement a cross-functional program core team sponsored by executive leadership and responsible for the execution and delivery of the product development team. The program core team provides two-way communication and direction to both agile teams and other program teams.

**Challenge 9:** Most embedded environments require the concept of System Test. An additional layer of testing, known as System Test, typically follows the development cycle in embedded software development environments. In addition to black box functional testing in a customer-like infrastructure environment, System Test focuses on performance, availability and scalability verification of the embedded product. With agile delivering frequent, working increments of the product, the role of System Test needs to be integrated into the agile framework.

**PwC Strategy 9:** Redesign and implement System Test activities as agile iterations, with small teams of testers executing functional, performance, scalability and other system-level testing. (Figure 9)

---

**Figure 9: Adaptation—Illustrative agile iterations of system testing**

System test activities designed as agile iterations with small teams of testers executing functional, performance, scalability and other system-level testing.

FIT=Feature Integration Test
These strategies are designed to help address commonly found challenges in embedded software development. However, some embedded software development teams may also need to consider additional strategies.

For example, many embedded products, especially consumer electronics, use over-the-air (OTA) upgrades after the product is released to enable new enhancements and critical defect fixes. In such cases, product organizations often establish dedicated small feature, or sustenance, agile teams to handle the post-release product backlog.

Moreover, many embedded product developers do not entirely build their software in-house, so the agile development process needs to be further adapted. Organizations that buy and integrate code from third-party software vendors or communities that follow their own development/release processes, or simply outsource some software development to third-party companies, may confront development environments that may not necessarily follow agile techniques.

In these cases, the sourcing process needs to be adapted to facilitate agility. Upfront service-level agreements with suppliers on software release content, scope, and cadence, need to be established. Once the agreements are in place, a milestone-based hybrid approach, similar to the one used to adapt to waterfall-style hardware development, can be adopted. Another option, particularly useful in offshore staff-augmentation scenarios, is to get the supplier to move towards agile style development.

Agile embedded software development case study

Looking to facilitate early customer adoption by providing incremental functionality for its next-generation platform at periodic intervals leading to the final release date, a networking equipment manufacturer asked PwC to help support the transition from waterfall to agile software development practices. With the goal of institutionalizing agile practices and accelerating time to customer adoption, PwC helped the client implement a structured agile transformation framework that allowed the company to implement a three-phase transition:

Phase I: Mobilize and Construct. The team created a baseline map of the existing environment and product architecture. They then defined the methodology, team structures, tools, practices, and metrics necessary for an agile-based operating model, and an implementation plan that encompassed pilot and ramp-up phases.

Phase II: Operationalize. The client gradually instituted the agile program by launching the pilot program for four agile teams and developing a plan to scale-up for the remaining 9 teams. It streamlined the move to the new operating model by utilizing PwC organizational training and coaching to implement incremental change management and secure buy-in from all levels of organization.

Phase III: Sustain. Achieved steady-state program performance targets and removed external supports. Program is positioned to deliver to market at desired pace.

Conclusion

The benefits of agile software development are real, especially for companies looking to accelerate the product development cycle, a common refrain in many industries that use embedded software. With careful adaptation of agile design and development practices, software teams can overcome the inherent constraints of embedded software development and successfully accelerate new product development.
PwC can help
For more information about how PwC can help your organization improve its embedded software development processes by adopting agile techniques, please contact one of our practice leaders.

Aritomo Shinozaki
Principal, Product Innovation & Development
Advisory Services
408 817 5870
ari.shinozaki@us.pwc.com

Walter Sun
Principal, Product Innovation & Development
Advisory Services
408 817 5949
walter.c.sun@us.pwc.com

Let’s talk
Please reach out to any of our technology leaders to discuss this or other challenges. We’re here to help.

Tom Archer
US Technology Industry Leader
408 817 3836
thomas.archer@us.pwc.com

Cory Starr
US Technology Assurance Leader
408 817 1215
cory.j.starr@us.pwc.com

Kayvan Shahabi
US Technology Advisory Leader
408 817 5724
kayvan.shahabi@us.pwc.com

Diane Baylor
US Technology Tax Leader
408 817 5005
diane.baylor@us.pwc.com

Acknowledgements
The following PwC professionals contributed their experience, knowledge and expertise to produce this paper.

Bill Noel
Director, Product Innovation & Development
Advisory Services
617 530 7184
bill.noel@us.pwc.com

Shankar Subramaniam
Manager, Product Innovation & Development
Advisory Services
312 298 5217
shankar.p.subramaniam@us.pwc.com

About PwC’s Technology Institute
The Technology Institute is PwC’s global research network that studies the business of technology and the technology of business with the purpose of creating thought leadership that offers both fact-based analysis and experience-based perspectives. Technology Institute insights and viewpoints originate from active collaboration between our professionals across the globe and their first-hand experiences working in and with the technology industry. For more information please contact Tom Archer, Technology Industry Leader.

About PwC’s Product Innovation and Development Practice
PwC’s Product Innovation and Development advisory services help clients develop new innovative businesses, products and services to achieve market-leading revenue and profitability growth. We also help our clients build lasting internal capabilities, ranging from Innovation Incubators to global R&D centres to PLM systems, that enable sustained, consistent results.