Auditing Agile—A Brave New World

“We are running on Agile, so there is nothing to audit” is a refrain auditors hear all too often when attempting to audit clients who use Agile. For a profession rooted in plan-driven methodologies, from validating software development to documenting audit work papers, Agile presents a unique conundrum.¹

THE CASE AGAINST DOCUMENTATION

Conceived by 17 self-professed “organizational anarchists” in a Utah ski resort in 2001, the first two values of the Agile manifesto listed in figure 1 appear to clash with common audit constructs, as internal control design and validation are invariably predicated on process and procedural documentation. Furthermore, Scrum, a popular iterative Agile software development methodology, advocates for self-organizing, cross-functional teams, making audit challenging for auditors who are used to prescribed roles and responsibilities that have clearly demarcated segregation of duties (SoD) to mitigate the risk of wrongdoing or fraud.

To understand the evolution of Agile and Scrum and identify related implications for audit, it helps to go back to the inception of the waterfall model, first proposed in 1970. Even though the waterfall model defines distinct phases for managing the development of large software systems, it nonetheless acknowledges the need for iteration.² Fast forward 30 years and this acknowledgment would have been ideal for proponents of Agile and Scrum, for whom each two-week sprint would culminate in the demonstration of working software. Beginning with a bare bones skeleton and inheriting more features with each successive sprint, the Scrum team seeks to “burn down” the requirements surfaced through the continual grooming of the product backlog.

The waterfall model advocates for significant documentation throughout the development life cycle. Some documentation does help to avoid any miscommunication on what has been agreed upon. Yet, it is the very maintenance of significant documentation during the requirements phase that would, in turn, give rise to more documentation, in the form of change requests seeking authorization for variances to plan. Fundamentally, the Agile manifesto does not so much devalue documentation; rather, it values working software more. Agile focuses on having good enough documentation to initiate and sustain an open dialog among cross-functional team members. The premise behind having good enough, rather than comprehensive, documentation is that, at the start of a project, all that needs to be known is not yet known. A plethora of unanticipated outcomes can arise; for instance, customers can, and often do, change their minds on features, even as the software is being coded (64 percent of features developed never or rarely get used).³ Therefore, having excessive documentation at the start and using it as a benchmark for downstream activities can seem counterintuitive.

Despite a lesser amount of documentation, Agile can actually create greater transparency on uncertainties that may not be otherwise visible during a project’s infancy. According to Jens Østergaard, founder of House of Scrum, the...
risk associated with identified uncertainties tapers off with each successive sprint, whereas with waterfall, the initial, comprehensively documented set of specifications may give a false sense of security, only to be undermined when hidden complexities surface downstream when the project enters the testing and go-live phases (figure 2). From an audit perspective, an auditor who looks for evidence too soon is likely to be reviewing material that is later overwritten as requirements become further clarified.

FROM STORYTELLING TO STORY-TESTING

This is not to discourage auditors from intervening early in the development of software. In fact, an unmined opportunity lies in user stories that are developed to characterize specifications in Agile. Figure 3 depicts a template for user stories.

One of the key ways Agile encourages responding to a change rather than following a plan (see the fourth value listed in figure 1) lies in the active update of remaining user requirements; in Agile, this is known as the grooming of the product backlog of user stories for each sprint. The user stories represent the voice of the customer, so the development team is wise to ensure that they are accommodated to the greatest degree possible.

How does the Scrum team know they have delivered on a user story? Behind every user story is a set of acceptance criteria that helps clarify the specific conditions that need to be met for a story to be delivered. Acceptance criteria identification can also break up bigger stories into smaller, more digestible pieces for developers to consider. User stories typically follow the independent, negotiable, valuable to users or customers, estimable, small and testable (INVEST) set of attributes.

Behavior-driven development (BDD) is a means for discovering and, consequently, testing against what the software ought to do. “A story’s behaviour is simply its acceptance criteria—if the system fulfills all of the acceptance criteria, it is behaving correctly.” Automated testing tools allow the team to describe the acceptance criteria in terms of scenarios; scenarios become automated tests with the addition of step definitions using code. Scenarios take the form illustrated in figure 4.

To illustrate, consider the example of developing a purchase requisition application for expenses over US $1,000 (figure 5).

How can it be confirmed that the Scrum team has delivered on this user story? A scenario to consider is shown in figure 6. Scenarios force stakeholders to clarify just exactly what they need and can help to mitigate the risk of gold plating, which is the addition of features that do not add value. Thus, auditors can get involved early in the software development process not by looking for comprehensive documentation upfront, but rather by taking part in the user story development.
Within each story, there is an opportunity to craft an abuse scenario where the nature of the proposed validation is negative, i.e., covering scenarios that do not follow the plan. **Figure 7** describes the expected application behavior for five abuse scenarios. **Figure 8** maps these scenarios against traditional internal control objectives.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Completeness</th>
<th>Accuracy</th>
<th>Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

It turns out that there are no controls addressing whether or not the proposed spending request is recorded correctly to the right general ledger (GL) account. Therein lies an opportunity for auditors to articulate other roles needed to ensure transaction integrity. By participating in the Scrum team during, rather than after, the sprint, auditors can add a story for finance personnel to provide a second layer of review, as outlined in **figure 9**.

**Figure 9**—Finance Review of Purchase Requisitions

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Purchase Requisition</th>
<th>Finance Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Correct Selection of GL Account</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Incorrect Selection of GL Account</td>
<td></td>
</tr>
</tbody>
</table>

Source: Chong Ee. Reprinted with permission.
Had the development approach been waterfall, auditors would have been satisfied obtaining sign offs of the initial requirement to have the enterprise buyers correctly code their proposed purchase to the right general ledger accounts, even though, in reality, this process is not executable. In addition to participating in user story development, auditors need to also adapt the manner in which they perform audits.

A specification should do something. Thus, when auditing Agile, instead of expecting a three-ring binder of written specifications, a more appropriate approach may be for auditors to request the system log of executable specifications. The following questions can help the auditor gain insight on specifications:

- Where are the automated test results?
- What happens when they fail?
- What is the test coverage?
- How often are automated tests updated?

The good news is that Agile and Scrum artifacts are not vastly different from the traditional audit artifacts that auditors rely on to evidence a system of internal controls. User stories mirror user narratives, and acceptance criteria mirror application controls that ensure the accuracy, completeness and validity of the transactions processed. Auditors who consider a more complete picture of varied roles and scenarios (illustrated through user stories and acceptance criteria) make a valuable contribution to the Scrum team.

Auditors can also add value by challenging the prevailing mind-set that tests at the end of a project are the only way to produce the required quality. It is widely accepted that the later a bug is detected, the more costly it is to fix. In the same manner, if acceptance criteria of stories are developed as a part of BDD, developers can employ test-driven development (TDD) to write tests before writing code. Studies on the adoption of TDD in three Agile teams at Microsoft and one at IBM reveal a 40 and 90 percent decrease, respectively, in the prerelease defect density of products. Further, when combined with a continuous integration (CI) server that triggers testing every time new code changes are checked in, testing becomes part of, or a constant accompaniment to, coding, as opposed to a separate downstream test phase. Consequently, the earlier identification of unit and functionality errors reduces the cost to fix them later and frees up time for testers to add value through exploratory and end-to-end transaction testing.

ROLES THAT OVERLAP

The remaining Agile manifesto value addressed in this article emphasizes customer collaboration over contract negotiation (see the third value in figure 1). With its emphasis on distinct phases and handoffs, the waterfall model can be likened to a relay race, while Scrum is derived from a restarting play where players huddle with their heads down to attempt to gain possession of the ball. When applied to software development, the Scrum approach focuses on having the distinct phases overlap through collaboration.

What implications does this have for audit? A key precept behind the emergence of design thinking as a means to solving problems is the emphasis on collaboration to attain sustainable product design. By playing a key role in the Scrum team by considering abuse scenarios in user stories, auditors engage actively in the collaboration.

Another useful adjustment auditors can make is to expand their scope of auditees. As mentioned earlier, the product owner, insofar as he/she drives the product backlog of user stories, is a key resource in making sure that compliance and security needs are met. Auditors can make a valuable contribution by ensuring broad representation of stakeholders on the Scrum team. The composition of the Scrum team plays a key role in aligning expectations for the project among all those involved.

However, just because the Scrum team is cross-functional does not mean all development, test/staging and production environments are accessible to anyone on the team. The audit objective behind SoD—restricting access by environment or configuration so that no one single individual has the ability to circumvent or hack the prevailing system of internal controls—is not mutually exclusive from the development objective of maintaining code integrity, which is making sure that valid code changes are not inadvertently overwritten or diluted by competing ones that have not been tested or do not integrate well with existing interfaces.

Learn more about, discuss and collaborate on audit tools and techniques in the Knowledge Center.

www.isaca.org/topic-audit-tools-and-techniques
With the emphasis on customer collaboration over contract negotiation, it can be easy to be misled into thinking that Agile favors process over outcome. Interviews conducted with teams from three organizations, one using waterfall, another using Agile and a third using a hybrid of both, revealed that the waterfall organization saw a greater emphasis on product or outcome with preventive controls, whereas the Agile organization leaned toward process with detective and corrective controls. With Agile’s fundamental focus on outcome—working software—one can make a counterargument that in Scrum, control practices such as TDD, CI, and automated unit and acceptance testing are really focused on outcome, whereas the waterfall model, with its focus on formalized signoffs, is really born out of an emphasis on process. Figure 10 illustrates the different types of process- and outcome-based controls in Agile and Scrum.

<table>
<thead>
<tr>
<th>Figure 10—Agile Outcome vs. Process Controls</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outcome Controls</strong></td>
</tr>
<tr>
<td>Percent of test coverage</td>
</tr>
<tr>
<td>Number of failed builds</td>
</tr>
<tr>
<td>Number of failed unit tests</td>
</tr>
<tr>
<td>Number of failed acceptance tests</td>
</tr>
</tbody>
</table>

Source: Chong Ee. Reprinted with permission.

As for whether Agile has more detective or corrective controls than waterfall, there are shades of gray when it comes to labeling a control as preventive or detective. Because the project is seen from production, the testing phase from the waterfall model is preventive, i.e., it reduces the likelihood of coding errors from arising in production by uncovering them in a test environment. From the perspective of development, however, testing is detective as it uncovers errors in coding from the development phase. Likewise, TDD in Scrum and Extreme Programming (XP), by forcing developers to fail the test before the code is written, detects the error even as it prevents it from arising ultimately in production. The same argument can extend to automated unit and acceptance testing: while they detect errors in development and staging environments, they really prevent them from arising in production. To characterize Agile controls as more detective than preventive is to miss an opportunity to dive deeper into what truly goes on.

CONTROL READINESS IS A FUNCTION

How control-ready an enterprise is depends not just on the practices adopted (whether that is waterfall or Agile in software development), but also on the environment, which is unique to each enterprise. Whether management employs a more top-down or bottom-up approach in controlling software development can make a difference in whether waterfall, Agile or a combination of both is ultimately embraced. In effect, auditors can audit the Agile team on how well it does Agile.

For far too long, auditors have been relying on validating the operating effectiveness of processes in the hope that a carefully controlled process will yield a positive outcome, e.g., verifying evidence of sign-offs. The problem with process controls is that while they may be necessary for fostering a positive outcome, they are by no means sufficient. Consider the user story of submitting purchase requisitions for approval outlined in figure 9.

When auditors sample invoices, they may find that all have been matched with approved requisitions. They are able to find a 100 percent two-way match with no exceptions. It would appear that all purchases made have been approved beforehand. Yet they cannot help but notice from the system audit timestamps that for a particular group of buyers, requisitions are almost always created shortly before invoices are applied—sometimes a matter of mere minutes. It turns out, for this group of buyers, because proposed spend is highly volatile and hard to predict upfront, requisitions are created only upon receipt of invoices, rather than before receipt of service.

These illustrative audit findings cast doubt upon requisition approval as a means to control spend precisely because it is not so much performed before services are rendered as it is after receipt of services—and often as a means of generating evidence for audit. Auditors must be proactive to ensure that audits remain effective safeguards against errors or fraud, not ritualized practices of audit for audit’s sake. Agile, with its emphasis on working software, focuses on outcome and, thus, provides auditors the opportunity to adapt their approach accordingly. A widely circulated observation among Agile and Scrum circles is that the Standish Group’s study of software projects conducted between 2002 and 2010 revealed that Agile is three times more likely to succeed than waterfall. Yet, auditors point out that the same Standish Group study reported that both Agile and waterfall projects shared a 50/50 chance of being challenged.
As lightweight frameworks, Agile and Scrum are not intended to be comprehensive. They do not address risk management, product strategy and other areas that comprise the slew of activities to enable and sustain product launch, continual enhancement and maintenance. When auditing Agile, it is important for auditors to realize that enterprises employing Scrum or Agile are not running on empty—there are indeed artifacts and ceremonies from a process perspective and metrics to track test coverage and automated test results from an outcome perspective. Perhaps, more important, auditors are best served by seeing that Agile, like its waterfall predecessor, is not a silver bullet to resolving the age-old struggle of bridging compliance and security with software.

ENDNOTES

1 This article uses several Agile terms that are critical to understanding. A sprint is a set period of time during which specific work has to be completed and made ready for review. A burndown chart is a graphical representation of work left to do vs. time. A product backlog is a prioritized features list containing short descriptions of all functionality desired in the product.

2 Royce, W.; Managing the Development of Large Software Systems, Technical Papers of Western Electronic Show and Convention (WesCon), 25–28 August 1970, Los Angeles, California, USA

3 Johnson, J.; Standish Group Study reported at XP2002
6 Ibid.
8 Nagappan, N.; E. Maximilien; T. Bhat; L. Williams; “Realizing Quality Improvement Through Test Driven Development: Results and Experiences of Four Industrial Teams,” Empirical Software Engineering, vol. 13, iss. 3, June 2008
12 The CHAOS Manifesto, The Standish Group, 2012