Being in Control with Agile

Adjusting to ever-changing circumstances while staying in control and remaining compliant

Carlijn Hattink MSc CISA

C.J.M. Hattink MSc CISA
is a Senior Consultant at KPMG LLP in Canada.
carlijnhattink@kpmg.ca

Agile software development processes produce interesting numbers on productivity and quality of software development projects that indicate an increasing value to organizations. Meanwhile, management teams struggle to control this type of approach, as the process deviates significantly from traditional software development methods and as such does not enable the same type of control. This article explores the risks companies employing an agile approach are facing and how these can be addressed, including some suggestions on the auditability of this type of process.

Introduction

> *"While IT projects are becoming larger, on average, large IT projects tend to run 45% over budget, 7% over time, and deliver 56% less value than predicted, with software projects experiencing the highest risk of cost and schedule overruns."* ([Bloc12])

Speed of technology, increased pressure on compliancy with expanding legal and regulatory requirements, as well as fast changing market conditions are driving change in organizational processes. In an effort to minimize the risk in software development and to be able to cope with the changing organizational needs, agile development is chosen over traditional plan-based methods by more and more organizations. Launched in 2001 with the Agile Manifesto, it is now one of the major implementation methods which organizations turn to in order to speed up development and illustrate the value of their projects ([Gart15]). This article will address the major differences between plan-based and agile development methods, the challenges these differences pose in terms of control and auditability as well as some suggestions for addressing these challenges.

Figure 1. **Software Development Process.**

Plan-based vs. Agile Software Development Methods

Plan-based software development has been the predominant software development method for a long time. It moves through four broadly defined phases (see Figure 1), where none of the phases can start before the previous phase has been completed. This can lead to a long trajectory for any project using this approach, implying a number of risks as well. Two of these risks were articulated by ([Cape94]): the insufficient or unclear definitions of requirements for the end product, and changes to the scope or the requirements during the project. Requirements are set long before the actual build and testing of the product takes place, let alone before end users start making use of a product in the production environment.

The increased project duration is no longer in line with the business demands of software development. The market is intense, customer satisfaction is high on the agenda and businesses are subject to a continuously changing environment. It is not sufficient to make use of a structured but rigid process in software development anymore, as the circumstances require a much more dynamic process ([Yu14]).

Agile software development can deliver on the need for a dynamic process. Through multiple iterations a small piece of the software is developed, making it possible to regularly re-evaluate the requirements that the customer sets for the software. When requirements change, this can be included in the development trajectory and also in the software.

Many forms of agile software development exist, of which 'scrum' may be one of the most well-known. Named after a scrum in a rugby match, the scrum methodology of software development is "an iterative, incremental framework for projects and product or application development" ([Suth10]). A recent study by Rally ([Rall13]) included almost 10,000 teams and showed that productivity was doubled when people are dedicated to one team, and teams using full scrum have 3.5 times better quality in terms of the end-product (measured in number of released defects).

Table 1 illustrates the main differences between the two methods.

Many organizations and circumstances require that a mix of these methods is employed. Within larger organizations a full-blown agile approach, relying on small teams, high levels of autonomy and collaboration, may not be the right fit. Some aspects of plan-based methods may be employed in order to ensure a fit with the organization.

	Plan-based Development	Agile Development
Fundamental assumption	Systems are fully specifiable, predictable, and are built through meticulous and extensive planning	High-quality adaptive software is developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change
Management style	Command and control	Leadership and collaboration
Knowledge management	Explicit	Tacit
Communication	Formal	Informal
Desired organizational form/ structure	Mechanistic (bureaucratic with high formalization), aimed at large organizations	Organic (flexible and participative encouraging cooperative social action), aimed at small and medium-sized organizations
Quality control	Heavy planning and strict control. Late, heavy testing	Continuous control of requirements, design and solutions. Continuous testing

Table 1. **Traditional vs. agile development ([Neru05], [Dyba08]).**

Risk Factors in Agile Development

While agile methods have many benefits, they also carry certain risks, many inherent to the agile process (see Table 2).

Feature of Agile	Challenges in Risk and Control
Team autonomy	• Lack of direct control • Heavy reliance on team members' skills and knowledge • Lack of strategic direction
Speed	• Decrease in quality • Lack of testing efforts • Minimal communication
Flexibility	• Changing requirements • Dynamic project management
Short-term focus	• Tactical decision-making • Business & IT misalignment
Minimal documentation	• System maintenance issues • Lack of monitoring opportunities
Small teams	• Lack of independence • Increase in risk-appetite • Underestimating complexity

Table 2. **Features of agile and the challenges they pose.**

Team Autonomy

To enable speed and flexibility, the teams in an agile environment work with a high degree of team autonomy. Without highly skilled resources, teams may spiral out of control and lose sight of business objectives. There are four capabilities teams need to possess in order to make an agile project a success: technical (technical expertise), behavioral (interpersonal relationship skills), business (understanding organization context), and infrastructure capability (providing firm-wide IT infrastructure services) ([Goh13]). Agile requires resources to be fully committed to the team. Specifically, large projects often encounter issues trying to include and maintain the best resources in agile teams, as they often cannot be spared from their day-to-day activities.

Speed

Decisions need to be made quickly and by a small group of people, who may or may not have the right information to make a decision at that point in time. The quality of the process may be undermined by a lack of time for proper documentation and the quality of the product may be diminished by a lack of required testing efforts.

Control is hard to exercise when an entire development cycle can take only three weeks and then simply starts all over again. Deviations from plans are not uncommon, flexibility is key and collaboration crucial. Under intense

time pressure the focus on communication becomes minimal, and collaboration may actually fail.

Flexibility

With agile software development, the requirements for the end product are less clearly defined at the beginning of the project and are likely to change. The benefits are obvious, as organizations are more flexible and better able to respond to rapidly changing market conditions and customer demands. However, it does mean that the end product will almost by definition deviate from what was proposed at the beginning of the project. Intensive customer involvement is required to steer the project in the right direction.

Short-term Focus

As teams work within very short timeframes and have the autonomy to make their own decisions, they tend to focus on tactical decision-making and run the risk of neglecting or even steering away from long-term (strategic) objectives ([Drur12]). As customers are crucial in agile development and are encouraged or even required to be part of the agile team, the business value on an operational level should be guaranteed. The business value to the organization at large, on a strategic level, may not be evident when the product is delivered.

Minimal Documentation

While it is a myth that agile makes all documentation requirements obsolete, it is fair to state that documentation is minimized as much as possible to ensure that the administrative burden is low and efficiency is high. The aim should be to minimize documentation to a level where security and quality can still be guaranteed and control can be exercised. Also, continuity of agile processes needs to be ensured, even if one or more resources need to be replaced. Knowledge management should be optimized and learning enabled, so that knowledge is no longer tacit, residing within the teams.

Without the proper documentation of source code and system functionality it can be a challenge to maintain it in a proper manner, even if the product was delivered meeting high quality and security standards.

Small Teams

Small teams with high levels of autonomy are prone to overestimate their capabilities and underestimate complexities. Tushman and O'Reilly ([Tush96]) found that a high level of team autonomy is also very likely to lead to a higher risk appetite and more experimenting.

From a more practical point of view, it is very hard to have independent testers when there is a small team and a developer is forced to test his or her own work.

Addressing Risks and Exercising Control

When it comes to addressing these risks and exercising control, the challenges in auditability of these projects are also highlighted. As agile is still relatively new and relies on autonomy of a small group of team members, controlling the process can be a challenge for management. There are certain aspects that can be included in an internal review on the effectiveness of the agile process.

There are different forms of control that can be exercised: behavioral control (e.g. management monitoring), output control (e.g. defined output requirements), clan control (e.g. social norms within teams), and self-control (e.g. individual empowerment) ([Kirs96]). While self-control would not be a logical choice in an environment where team collaboration and autonomy are crucial, a combination of the other types of control seems appropriate.

Output Control

Agile development aims to implement user requirements, and makes that the lead indicator in the process. This means that planning is not on a task-level, but on a feature-level ([Rubi11]).

To be able to exercise control over the agile process, there should be less focus on cost and schedule estimates, but more on the features that are created, the requirements that are defined during the process, and the accomplishment of those requirements.

While documentation is minimal, there are still a number of deliverables (or artefacts) in the agile process that can be reviewed. The *product vision statement* paints the first complete picture of the product to-be. The *overall design* can also be included in the vision statement. Without detailing the technical features extensively, the components

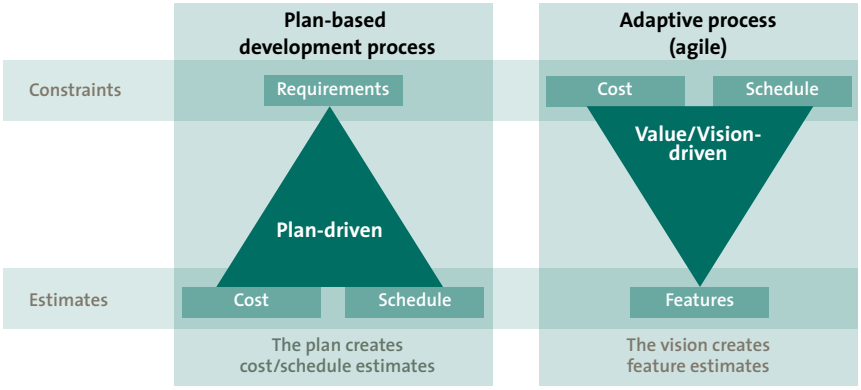

Figure 2. **Estimates & Constraints ([Sligo6]).**

of the solution can be described to clarify the integration and possible challenges in integrating all releases of the solution as well. The *product roadmap* can specify the plan a bit further by defining themes that the team will be working on. Instead of use-cases that define a specific set of interactions leading to a desired result, user stories are defined. These are defined from the perspective and in the language of the customer, focused on the desired result without exactly defining the details on how to achieve this result. The *release plan* will define the schedule for iterations, while specific user stories are detailed per iteration. The *product backlog* keeps the overview of the total number of user stories defined that still need to be completed in an iteration to come. Combined with the (overall) *burn-down chart* this provides a complete overview of progress. To ensure quality criteria are being met and clear 'acceptance criteria' are defined and adhered to during the project, the *definition of ready* and *definition of done* can be specified upfront, detailing respectively a set of criteria for adding a user story to an iteration and a set of criteria for accepting and deploying a piece of software. The definition of done should include:

1. Successful testing and a demo of the functionality, ensuring the quality of the product as well as its alignment with customer requirements;
2. Business acceptance, ensuring the user story is addressed and the delivered software fits the desired scope of the project;
3. Documentation, while limited, should be present to ensure the technical design is documented and to enable hand over to a support team that will continue to maintain the software after implementation;
4. Authorizations should be defined and incorporated in the design of the software. Access in the system should be controlled. During development, there should be segregation of duties between developer and tester to ensure independence and avoid 'tunnel vision'.

There are also a number of deliverables specific to a phase in the iteration. In the iteration planning phase the work for the upcoming iteration is laid out. This can be done through the creation of a *sprint backlog*, where the *user stories* are divided in *story points* and assigned to a specific sprint where they should be completed. During the iteration execution the *sprint backlog* shows which story points *still need to be* completed in that sprint. Progress can also be shown through a *sprint burn-down chart*, which also shows what *has already been* completed ([Rubi11]). To ensure everyone knows what they are doing the sprint backlog can be further broken down in specific tasks through a *task board*, which also aids in managing the workload. In the *daily stand-up* the workload can be discussed and rebalanced where necessary, this way problems can be recognized in time to resolve them. Once a piece of software is tested it can be shown to the customer through a *demo*, which provides the customer more insight to the progress being made and ensures that the expected quality is delivered. In the iteration review, the team and relevant stakeholders review whether the commitment made at the start of the iteration is fulfilled at the end. The pieces of software, also called *increments*, are delivered and need to be measured up to the *definition of done*. To ensure maintainability of the system, the *source code* and *support documentation* can be included in the iteration review. These can be included in the delivery of the increments and as such in the definition of done. The same is the case for security requirements. In the iteration retrospective the past iteration is reviewed to see whether there are any lessons learned or improvements that can be made. A possible measurement of the work done is the *velocity* (number of user stories completed in the iteration). Other (related) measurements are the number of *stories completed* (in combination with what is left in the sprint backlog) and the number of *tests passed*. If the deliverables for each iteration are also released for use in a production environment by the customer, the number of *customer-reported defects* may just be the most important measure as a direct indicator of quality as it is perceived by the customer.

Behavioral Control

One of the core aspects of agile methods is its reliance on social processes ([Maru09]), introducing a need for soft controls. To ensure that knowledge is transferred during the iterations, collaboration is optimal and people are working towards the same goals (goal congruence), it is important to have the right people, clearly communicate the goals and create a common purpose. *Knowledge* and *social skills* that are required should therefore be defined when selecting the members of an agile team. *Story owners* are assigned to the stories in the iteration planning phase to ensure that the time invested by the customer is spent efficiently and customers are knowledgeable of the stories assigned to

them. During the iteration execution the software is both developed and tested. Therefore *separated development and test environments* are required as well as *separation between tester and developer*. This can be done either through authorizations in the application (landscape) or through procedural controls like a 4-eyes principle.

Clan Control

With a lack of specific task-related behaviors and a focus on teamwork, clan control should not be underestimated. Business and IT alignment needs to be ensured, as well as long-term vs. short-term objectives and individual and team goals. Goal congruence should be stimulated as much as possible through e.g. team incentives and rewards, their relation to organizational performance, or peer-reviews. During the iteration review, all increments produced in the iteration can be presented to the stakeholders enabling *fast feedback* and ensuring that the increments reflect the requirements of the users correctly. Additionally, a risk assessment can be incorporated into this stakeholder review and evaluate the compliancy with security requirements as well. Through the fast feedback, corrections or adjustments can be made before the product is finished, ensuring higher quality and customer satisfaction.

The types of control as well as the auditable aspects are summarized in Table 3.

Product Review

In addition to the above mentioned controls that can be implemented and reviewed during the agile process, there are also controls that enable auditability of the product itself. The basic controls that should be in place can be defined beforehand. These would follow the guidelines of the information security policy and contain directives in access management, incident and problem management and back-up and recovery. These should be in place for the first piece of working software and ready for review. Every subsequent piece of software should be developed following a strict change management process, which incorporates testing as described above and includes a risk assessment which should review the impact on existing controls as well. Security requirements should be incorporated in the definition of done and only software that complies with this definition should be brought into the production environment.

Effective Use of Agile Processes

In every project there is a need for control. Management goals are to maximize value and minimize risk. While agile software development can add tremendous value, it also has a lot of consequences for the environment it operates in. Organizations should therefore ask themselves: is agile the right choice for us?

Deciding whether agile is the right choice

Agile provides an organization with flexibility, deliverables that are available at great speed, success that can be celebrated within a shorter timeframe, higher quality, as well as increased employee morale and user satisfaction. So why not use it whenever you can? While it all sounds like a dream come true, an agile process still carries a significant risk of failure. Agile processes need to be aligned and require certain enablers to increase their likelihood of success.

	Iteration Planning	Iteration Execution	Iteration Review	Iteration Retrospective	Overall
Outcome Control	• Sprint backlog • User stories • Story points	• Sprint backlog • Sprint burn-down chart • Taskboard • Daily standup • Demo's	• Increment • Definition of Done • Source code review • Support documentation • Security Review	• Velocity • Stories completed • Tests passed • Customer reported defects	• Product vision statement • Overall design • Product roadmap • Release plan • Product backlog • Burn-down chart • Definition of ready • Definition of done
Behavioral Control	• Story owner	• Independent testing			• Knowledge requirements • Social skills
Clan Control			• Fast feedback		• Goal congruence

Table 3. **Control points in iterations and during the project.**

Enablers

The enablers for agile can all be deduced from the Agile Manifesto as it was developed in 2001. They have been incorporated in a great graphic by Lynne Cazaly, as shown in Figure 3.

When starting to employ an agile approach, it will have a big impact on your existing IT organization and the business at large. An agile approach is centered on the customer, making the customer part of the process and the team. This means a cultural shift where IT needs to open up and invite the customer into their world. The customer needs to have a positive stance on IT and the value it can deliver, in addition to a willingness to invest time and resources in the process. Both developers as well as customers need to be fully dedicated to an agile team.

Apart from dedication, the organization needs to have an understanding of the continuous change it will encounter. A big part of agile is the frequent delivery of working pieces of software. While that enables successes to be celebrated often, it might also invoke resistance to the constant change people experience.

An agile team has great levels of autonomy and works under pressure, both in terms of time as well as the quality of the deliverables. To achieve success under those circumstances you will need a highly knowledgeable team, with the right technical as well as business knowledge, and motivated team members who are extremely collaborative. Constant communication is a must, as well as a geographically centralized team.

Apart from the social/cultural factors, the technical requirements should not be forgotten either. High knowledge levels are required to develop software with simple code, enabling fast creation as well as limited requirements for (complex) technical documentation. It also requires a system environment that enables fast, intensive testing. In addition to rigorous unit testing, integration testing should ensure that a new piece will not obliterate any previous work done and the system will function properly as a whole.

Business and IT alignment

Agile development will not suit all needs and will not be the answer to all failed plan-based projects. Organizations should critically evaluate whether their business needs require an agile approach and whether their organization is aligned with an agile approach.

Figure 3. **The Agile Manifesto illustrated ([Caza14]).**

Organizations acknowledge the benefits of both approaches more and more and recognize that choosing one of the approaches exclusively may not be the optimal solution. As the CEO of a large Canadian retailer indicates, he needs both approaches in his organization to address the need for a stable environment for maintenance and improvement of his legacy systems as well as a flexible, innovative environment to maintain his competitive position in the market. To be able to combine these approaches in one organization, he and many others are choosing a bimodal approach.

Bimodal operations in IT refer to a structure where two IT organizations exist within one overall organization. As [Gart14] defines it, it is "the practice of managing two separate, coherent modes of IT delivery within the same enterprise, one mode focused on stability and the other focused on agility". Each mode has their own focus, way of working and requires a different set of enablers. These

	Mode 1	Mode 2
Goal	Reliability	Agility
Value	Price for performance	Revenue, brand, customer experience
Approach	Waterfall, V-model, "high-ceremony IID"	Agile, Kanban, "low-ceremony IID"
Governance	Plan-driven, approval-based	Empirical, continuous, process-based
Sourcing	Enterprise suppliers, long-term deals	Small, new vendors, short-term deals
Talent	Good for conventional processes and projects	Good for new and uncertain projects
Culture	IT-centric, removed from customer	Business-centric, close to customer

Table 4. **Bimodal IT = Mode 1 + Mode 2 ([Gart14]).**

modes need to be in constant communication with each other as they will need to rely on each other's work at times as well as be able to cooperate in the same (IT and business) environment. Table 4 illustrates the differences between the modes.

In order to successfully deliver IT solutions, organizations should first examine which mode is most in line with their business needs. It may be that making use of only one of the modes will deliver the highest value to the organization, but in many cases a combination of the two will be most effective. A bimodal approach, especially in larger organizations, is well worth considering.

Conclusion

Agile software development methods are a promising approach to address the fast-changing market conditions and customer demands. Organizations should be careful in employing these methods though, as they are not suited to every organization. A combination of agile and plan-based developments may be more beneficial. An upcoming movement in larger organizations is to employ two modes of IT, where one pillar makes use of plan-based approaches, and the other makes use of agile approaches. This enables both the need for a stable environment (Mode 1) and the need for flexibility and opportunity for innovation (Mode 2).

To stay in control over software development processes, the first requirement is to ensure the approach is in line with the organization's processes, needs and culture.

When the choice is made to make use of an agile approach, the focus of control shifts from the process to the product, at a feature-level. The nature of an agile approach requires the incorporation of soft controls in any management approach or audit plan, as the process relies heavily on organizational culture, team autonomy, collaboration and knowledge of a limited number of individuals. As these control types shift when moving from plan-based to agile methods, the audit framework will need to be adjusted as well, focusing on different deliverables and distinguishing between overall project deliverables and goals, and iteration specific deliverables and goals. With some adjustments 'auditing agile' is however not necessarily a contradictio in terminis.

References

[Bloc12] M. Bloch, S. Blumberg and J. Laartz, *Delivering large-scale IT projects on time, on budget and on value*, McKinsey Quarterly, 2012.

[Cape94] T. Capers Jones, *Assessment and Control of Software Risks.* University of California: Yourdon Press, 1994.

[Caza14] L. Cazaly, *The Visual Agile Manifesto*, 2014, http://www.lynnecazaly.com.au/support/

[Drur12] M. Drury, K. Conboy and K. Power, *Obstacles to decision making in agile software development teams*, The Journal of Systems and Software, 85(6), 2012, pp. 1239-1254.

[Dyba08] T. Dyba & T. Dingsoyr, *Empirical studies of agile software development: a systematic review*, Information and Software Technology, 50, 2008, pp. 833-859.

[Gart14] Gartner, *Bimodal IT: How to Be Digitally Agile Without Making a Mess*, 2014, https://www.gartner.com/doc/2798217/bimodal-it-digitally-agile-making

[Gart15] Gartner, *Kick-Start Bimodal IT by Launching Mode 2*, 2015, http://www.gartner.com/technology/reprints.do?id=1-2OYIM-DT&ct=151006&st=sb

[Goh13] J. Goh, S. Pan and M. Zuo, *Developing the Agile IS Development Practices in Large-Scale IT Projects: The Trust-Mediated Organizational Controls and IT Project Team Capabilities Perspective*, Journal of the Association for Information Systems, 14(12), 2013, pp. 722-756.

[Kirs96] L. Kirsch, *The management of complex tasks in organizations: Controlling the systems development process*, Organizational Science, 7(1), 1996, pp. 1-21.

[Maru09] L. Maruping, V. Venkatesh and R. Agarwal, *A Control Theory Perspective on Agile Methodology Use and Changing User Requirements*, Information Systems Research, 20(3), 2009, pp. 377-399.

[Neru05] S. Nerur, R. Mahapatra and G. Mangalaraj, *Challenges of migrating to agile methodologies*, Communications of the ACM, May 2005, pp. 72-78.

[Rall13] Rally Software Development, *The impact of agile quantified*, 2013.

[Rubi11] E. Rubin and H. Rubin, *Supporting agile software development through active documentation*, Requirements Engineering, 16(2), 2011, pp. 117-132.

[Slige06] M. Sliger, *Relating PMBOK Practices to Agile Practices - Part 2 of 4*, Agile Connection, 13 April 2006, http://www.agileconnection.com/article/relating-pmbok-practices-agile-practices-part-2-4

[Suth10] J. Sutherland, *Scrum Handbook*. Boston: Scrum Training Institute Press, 2010.

[Tush96] M. Tushman and C. O'Reilly, *Ambidextrous Organizations: Managing Evolutionary and Revolutionary Change*, California Management Review, 38(4), 1996, pp. 8-30.

[Yu14] X. Yu and S. Petter, *Understanding agile software development practices using shared mental models theory*, Information and Software Technology, 56(8), 2014, pp. 911-921.

About the author

C.J.M. Hattink MSc CISA is a Senior Consultant at KPMG LLP in Canada. She started her career with the IT Advisory practice of KPMG in The Netherlands. After completing her post-master in IT Auditing and Advisory at the Erasmus School of Accounting and Assurance, she moved to Canada to broaden her horizon and gain international experience. She continued to work for KPMG with a focus on addressing project risks in IT implementations. She strives to combine organizational change management aspects with a risk-based approach in any IT transformation she encounters.