Controlling Change on Agile Software Development Projects

Andrew L'Ecuyer¹, Syed Adeel Ahmed^{2,*}

¹School of Continuing Studies, Tulane University, USA

²Department of Physics & Dual Degree Engineering Program, Xavier University of Louisiana, USA

Copyright©2016 by authors, all rights reserved. Authors agree that this article remains permanently open access under the terms of the Creative Commons Attribution License 4.0 International License

Abstract The importance of effective change control when developing software is well known throughout the Information Technology (IT) industry. Any software development project will deal with various forms of change throughout its lifetime, and without effective change control processes in place changes can lead to unforeseen, and often negative, consequences for an organization [8]. Yet despite this, change will almost always occur, and should therefore be anticipated. The change control processes in place on a software development project vary depending on the software development methodology being utilized [3]. Agile software development methodologies have become increasingly popular in recent years, with eighty-eight percent of respondents in a 2013 survey stating that their organization was practicing Agile [1]. According to Winter [1], Scrum is the most popular of the various Agile methodologies. A primary motivation for switching to Agile methodologies such as Scrum has been to better manage changes that were always inevitable, yet difficult to manage when utilizing traditional software development methodologies, such as the Waterfall methodology [4]. As more and more organizations transition to the Agile Scrum methodology, they must ensure processes are in place to properly control all forms of change. This includes changes that fall within the scope of a single project, as well as changes with a scope that extends across the enterprise. This paper explores various forms of change control and provides strategies and methods for effectively controlling and managing all forms of change on Agile software development projects.

Keywords Agile, Change Control, Change Management, Enterprise Change Management, Scrum, Waterfall

development project is effectively controlling change. As Wilson[9] explains, change can occur at different levels within a project, including changes to processes within an organization that impact day-to-day work activities, as well as changes to stakeholder expectations that impact project deliverables. Regardless of the type of change, however, all changes have the potential to greatly impact a project. According to Wilson [9], change “can be both good and bad...but if the implementation of change is not controlled, the outcome of that change can be unpredictable”. Change control is therefore needed to accurately predict the outcome of changes, namely by providing a mechanism to determine the impact a change will have on a project [9].

Existing research on Agile software development methodologies places a strong emphasis on the importance of effectively managing and responding to change. According to Karlesky[10], “the environment surrounding any embedded project is ever in flux”, and budgets, resources, schedules, competition and customer needs can change throughout the life of a project. Karlesky[10] explains that while traditional project management tends to avoid change due to the costs and schedule impacts typically associated with it, Agile methodologies embrace the fact that change will occur, and therefore emphasize the importance of effectively managing and responding to change.

When analyzing existing research on Agile methodologies, it is important to focus on the scope of the changes that are being discussed, as well as the individuals responsible for making decisions related to those changes. Focusing on the latter, Karlesky[10] describes the customer as the primary decision maker on an Agile project, making all decisions related to the project's direction. This statement indicates that the customer would therefore make most decisions regarding changes to a project, such as changes to software requirements or changes to the priority of new features and functionality. Mahapatra, Mangalaraj and Nerur[11] also emphasize the importance of the customer in a project's decision-making process, explaining that when utilizing Agile methodologies, most changes are

1. Introduction

A critical aspect of successfully managing any software

managed by a development team consisting of software developers and the customer. What is interesting when analyzing both descriptions of those responsible for decision-making, however, is that both state that most, but not all, decisions can be made by the development team itself (assuming that the development team includes the customer). This therefore indicates that there are certain decisions that cannot be made by the development team, and must therefore be made externally to the project itself.

In specifying that certain decisions cannot be made by Agile development teams, Karlesky[10] and Mahapatra, Mangalaraj and Nerur[11] indicate that other processes, people, and possibly even other organizations could be involved in the decision making process for certain changes. In other words, certain changes might have a scope that extends beyond a single Agile project and development team. While existing research thoroughly addresses the effective management of changes that fall within the scope of a single Agile project, little research exists on managing these other changes that fall outside of the scope of a single Agile project.

This paper analyzes different strategies for controlling all forms of change on Agile software development projects. This includes analysis of processes for managing change on projects utilizing traditional Waterfall software development methodologies and projects utilizing Agile software development methodologies. Specifically addressed is the management of enterprise-level changes, which are changes with a scope that extends beyond a single project, on software development projects utilizing the Agile Scrum methodology. An in-depth look at the concept enterprise change management (ECM) provides insight into processes and procedures for successfully controlling and managing enterprise-level changes, while also providing strategies for integrating ECM processes with Agile processes. Lastly, design strategies are discussed that can be leveraged by development teams to create software that can easily accommodate and respond to change.

2. Change Control Analysis

All software development projects are likely to face some form of change throughout their lifetime, such as changes to customer needs and requirements [12]. As Berger and Rumpe[12] explain, change has the potential to impact critical aspects of a software development project, such as a project's schedule and budget. As such, in order to ensure a project's success, the project team must be able to control and manage all forms of change. Change control directly addresses this need by implementing processes to effectively track all changes, and to ensure that project teams sufficiently plan for the implementation of those changes [12].

According to Aiello[8], change control for software development can be implemented in a variety of different

ways. For instance, the *a priori* method of change control can be utilized, which means approval is needed through a formal change control board (CCB) before any changes to software are made [8]. With the *a priori* method software requirements are defined up front, as is the design [8]. Then, requests for change (RFCs) are created to implement the requirements and design, which are then reviewed by the CCB [8]. Once the RFCs are approved by the CCB, they can then be implemented by the development team [8].

Another change control method described by Aiello[8] is *gatekeeping*, in which a CCB reviews RFCs to make changes to production or quality assurance (QA) environments. For example, the CCB might review a request to deploy a new version of an application to a production server, or it might review a request to patch software that is currently deployed to a QA environment [8]. Therefore, while the *a priori* method is concerned with changes to source code, the gatekeeping method is concerned with changes to various production and QA environments. Nonetheless, both approaches highlight important elements of any change control process, namely establishing a process for capturing and documenting changes, as well as processes for then reviewing and adjudicating those changes.

As Aiello[8] explains, change control processes can range from being informal and lightweight, to being very formal and robust. Therefore, change control processes should be implemented that are tailored to the specific needs of the project, organization, and the environment in which the project operates [8]. An important element of any software development project that directly influences how change is controlled and managed, is the software development methodology utilized on that project. For instance, a project may utilize the Waterfall methodology, which according to Crookshanks[4] involves a sequential approach to development in which one phase of development must be completed prior to moving on to the next. This means that first all requirements for the system are defined, which is then followed by creating a detailed design for the system [4]. Once the design is complete the system is implemented, and following implementation testing can begin [4]. Finally, once testing is completed the full system is released [4]. The Waterfall methodology also often requires large amount of documentation at each phase, which is typically due to the approvals required to move on to the next phase (e.g. a design specification must be approved prior to moving on to the implementation phase) [4].

In the case of the Waterfall approach, problems arise when changes are needed. As Crookshanks[4] explains, when changes are needed (e.g. changes to requirements), a RFC process is typically followed, such as the RFC processes found in the *a priori* and gatekeeping change control processes discussed above. According to Crookshanks[4], when utilizing the Waterfall methodology all phases of the software development effort must be revisited to accommodate a specific change. This means

that in addition to analyzing, reviewing and approving the change (e.g. through a CCB), requirements would also need to be defined for the change, which would then be followed by establishing a detailed design for the change [4]. Therefore, due to the need to formally document, analyze and review all changes, as well as the need to produce formal documentation at various phases of the software development effort, implementing changes when utilizing the Waterfall methodology can be time consuming and can require a great deal of effort. This indicates that changes would often be avoided and even frowned upon when utilizing the Waterfall methodology, mainly due to potential schedule impacts that could result from following formal change control processes.

Due the inability to easily accommodate change when using the Waterfall methodology, other methodologies are needed that can effectively respond to and accommodate change. According to Crookshanks[4], Agile software development methodologies can be leveraged to meet this need, which “attempt to address the fact that changes not only happen but can be expected”. Agile software development methodologies incorporate the following four basic values of Agile development, which are defined as part of the Agile Manifesto:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan [7].

The fourth value of the Agile manifesto specifically deals with change control and how teams respond to change. According to Canty[2], this value emphasizes the importance of expecting and even welcoming change, and therefore implementing processes that allow a project to effectively respond to change. As Berger and Rumpe[12] explain, when utilizing Agile methodologies “changes are not delayed or rejected, but rather encouraged since they are driven by the customer itself or by implications due to technical limitations”. Therefore, instead of following a rigid change control process for implementing changes as is often found with the Waterfall methodology, Agile methodologies are designed to be flexible and provide project teams with the ability to respond to change with minimal planning and process [2].

Crookshanks[4] explains that while many Agile methodologies exist, among the most popular of these frameworks is Scrum. Instead of sequentially working through each phase of a software development effort, as is done when utilizing the Waterfall methodology, Agile Scrum executes these phases in small increments known as sprints, building small chunks of useable functionality at a time [4]. This means that the software is built and delivered iteratively, with sprints typically lasting between two to four weeks, instead of delivering the entire system at once [4].

According to Crookshanks[4], three roles exist within the Agile Scrum methodology. The first role is the product owner, which is an individual that has a need and for a new piece of software (e.g. a customer) and is therefore responsible for generating and prioritizing the software requirements [4]. The next role is the Scrum master, who is responsible for ensuring that all Scrum processes are being implemented properly, and to assist with removing any impediments that might be preventing developers from completing their tasks [4]. The final role is the project team itself consisting of developers [4].

As Crookshanks[4] explains, prior to each sprint a sprint planning meeting is held to determine what requirements will be implemented during the next sprint. When using the Agile Scrum methodology, requirements (also called stories) are tracked in what is called a product backlog [4]. Formal documentation of requirements is not needed when using Scrum, so simple methods for tracking requirements in the backlog are often used, such as tracking requirements on index cards [4]. During a meeting known as sprint planning, requirements in the product backlog are prioritized and clarified by the product owner, and the development team then provides estimates for a subset of those requirements, typically those that are the highest priority [4]. According to Crookshanks[4], this subset of requirements becomes the sprint backlog, which represents the requirements that will be implemented during the next sprint. Once development has been completed for a sprint, a sprint review meeting is held to demonstrate the system to the product owner, who is then able to provide feedback on the existing functionality to the development team [4]. According to Crookshanks[4], this not only allows the product owner to see the progress being made on a system, but also provides an opportunity to reprioritize and adjust requirements in the product backlog.

While the above is not intended to provide a complete overview of the Agile Scrum methodology, it does highlight key Scrum processes that directly apply to change control. Canty[2] provides insight into the changes that can occur when utilizing Scrum, which mainly involve changes to the product backlog. For instance, the priority of requirements in the backlog might change, or the content of a requirement might change (e.g. to clarify a requirement or provide additional details for a requirement) [2]. Further, additional requirements or defects might be added to the backlog, while requirements that are no longer needed might be removed [2]. Since it is the product owner that is responsible for populating the product backlog with requirements, as well as prioritizing and clarifying the requirements, these changes will typically come through the product owner [4]. This means that the product owner can update the backlog regularly (e.g. during sprint planning) without the need to follow any formal RFC or CCB processes. The Agile Scrum methodology is therefore designed with the expectation that change will occur regularly, and includes processes to ensure that regular changes are effectively managed and controlled. And since the product owner is responsible for prioritizing the backlog,

they can be confident that the highest priority changes are implemented during each sprint.

3. Controlling All Forms of Change

What is evident from analyzing the change control processes built into the Agile Scrum methodology is that processes to formally document and approve changes, such as the RFC and CCB processes traditionally found when using the Waterfall methodology, do not exist. This therefore leads the question of whether or not formal change control processes are ever applicable when utilizing Agile methodologies such as Scrum. In other words, do the change controls processes inherit to Scrum accommodate all changes that an Agile project team might face, or are there certain changes that might require additional, formal change control processes.

According to Bogner and Elfanbaum[6], Agile processes are not able to accommodate changes whose scope extends beyond the project itself, which includes the ability to:

- effectively manage internal personnel so appropriate stakeholders are available;
- gather and prioritize the most important features desired by the organization throughout the development cycle;
- adjust the notion of ongoing training in a continuous release environment;
- ensure customer team members are informed by the full breadth of stakeholders required for enterprise acceptance;
- secure timely approval of new technologies that a team would like to leverage;
- And address stakeholder discomfort with cultural, business, social or other nontechnical changes related to software implementation.

According to this list, various changes external to a project team can directly impact that project, such as changes to available personnel or changes to organizational goals and priorities [6]. Additionally, changes might originate on project teams that need to be coordinated across the enterprise, such as changes that require full stakeholder acceptance throughout the organization, changes that require the procurement of additional technologies, as well as changes resulting in additional training requirements within the organization [6]. Sullivan[3] provides additional examples of changes whose scope extends beyond the scope of a single project, such as changes to one business process that require changes in other business processes, business rule changes that impact more than one system, and changes to software modules that impact other software modules [3]. According to Sullivan [3], these changes are considered enterprise-level changes. Regardless of the specific type of change or where it originates, however, any enterprise-level change that impacts an Agile Scrum project must be coordinated with changes to product backlogs that occur through typical

Agile processes. As Bogner and Elfanbaum[6] explain, the inability to effectively manage enterprise-level changes can result in the failure of software development projects within an organization. Therefore, additional change control processes are needed on an Agile projects to accommodate changes whose scope extends beyond a single project.

According to Bogner and Elfanbaum[6], enterprise-level changes can be effectively managed by applying Enterprise Change Management (ECM) processes to Agile projects. ECM consists of implementing processes to analyze and understand the impact changes have across the entire enterprise [5]. In other words, while the scope of change control on an Agile project is typically the project itself, ECM is concerned with changes that must be managed and coordinated across the entire organization. ECM processes are therefore needed to address enterprise-level changes, which means in order to address these changes an organization must have an ECM process in place [6].

According to Sullivan [3], a model for effective enterprise change management comprises five main components, including assets, dependencies, policies, roles and workflows. As Sullivan [3] explains, assets include the objects within the enterprise that change, such as application software, while dependencies represent relationships between assets where one asset relies on one or more other assets. Policies, on the other hand, are concerned with the rules that have been implemented for making changes to assets [3]. Next are roles, which are assigned to the individuals responsible for following these policies and therefore controlling change [3]. As Sullivan [3] describes, an ECM model consists of roles for reviewing, approving, preparing for, and implementing a change. Finally, workflows are the processes that are implemented to “manage the change process, coordinate people and assets, and enforce policies...to enable a consistent, rule-based process for executing change” [3]. Sullivan [3] provides the basic elements of any workflow process, which includes start, intermediate and end states, as well as rules for transitioning between states. An ECM model implemented within an organization to manage and control enterprise-level changes must therefore implement each of these five components.

In addition to the five components of ECM, ECM also has a standard lifecycle consisting of four stages [3]. The first of these stages is change initiation, which consists of either proposing a change to an existing asset, or proposing the creation of a new asset [3]. The next stage consists of reviewing the change, which according to Sullivan [3] can range from simple, lightweight review process, to a formal, process-heavy review with a CCB. Assuming the change is approved during the review stage, the change then moves to the implementation stage during which the actual change is made [3]. Finally, following the implementation stage is the assessing stage, during which the change is assessed and a decision is made regarding whether or not the change should be left in place [3].

The main takeaway from the above overview of ECM as

it relates to change control is the way in which changes are reviewed and approved during the ECM lifecycle, as well as the policies that must be in place to enforce rules for changing various assets within the organization. According to Sullivan [3], certain enterprise-level changes might require formal change control processes. Therefore, formal RFC and CCB change control processes might be applicable for certain changes. However, whether or not formal processes are required depends on the policies that are in place for the asset being changed. For example, Sullivan [3] explains that certain forms of content within an organization might require formal approval, such as changes to documentation that could have legal or regulatory implications. Policies for changing these documents would therefore likely require a formal process for documenting, reviewing and approving the change.

While ECM does not necessarily enforce specific processes for managing enterprise-level changes, analysis of ECM does highlight the fact that enterprise-level changes often require formal approval. Since the change-control processes for specific assets are dictated by policies, an organization must ensure policies are in place for all assets that must be controlled across the enterprise. This means that an organization must identify all assets requiring change control, and then define policies that allow the organization to effectively control those assets. Policies might dictate lightweight change control processes for certain assets, such as changes to code that fall within the scope of a single project, while other policies might enforce formal, detailed processes for changes to other assets (e.g. changes to enterprise architectures that impact multiple systems). In other words, policies might allow projects utilizing that Agile Scrum methodology to handle certain changes through typical Agile Scrum processes, while requiring other changes to utilize formal RFC and CCB change control processes. Ultimately, the specific needs of the organization, as well as the level of control required to manage all changes across the enterprise, influence the change control policies implemented within an organization.

An organization must be very careful, however, when identifying assets that require formal, rigid change control processes. Since Agile methodologies favor lightweight change control processes, formal change control processes should only be used when absolutely necessary. According to Cockburn and Highsmith [13], “an agile team working within a rigid organization has as difficult a time as agile individuals working within a rigid team”. In other words, Cockburn and Highsmith [13] indicate that having rigid processes within the organization, such as rigid processes for managing changes, can negatively impact a project's ability to operate effectively and efficiently according to Agile principles and processes. As a result, an organization should aim to remove rigid processes where possible, only utilizing formal change control processes when truly needed.

According to Cockburn and Highsmith [13], Agile

methodologies favor collaboration in order to make informed decisions, instead of focusing on exactly who makes a decision. Therefore, management should strive to create an organizational culture in which projects, individuals, and various other entities can effectively collaborate and communicate with one another to analyze, discuss and make ultimately make decisions related to changes. Mahapatra, Mangalaraj and Nerur[11] highlight the impact the culture of an organization can have on individual projects and project teams, specifically describing the leadership-and-collaboration management styles that are found on Agile teams and within Agile organizations. This includes establishing an environment in which individuals throughout the organization are trusted to make key decisions, without the need to follow formal, rigid processes [13]. According to Mahapatra, Mangalaraj and Nerur[11], “the organizational form that facilitates this shift needs the right blend of autonomy and cooperation to achieve the advantages of synergy while providing flexibility and responsiveness”. In other words, to the fullest extent possible an organization must create a culture that encourages collaboration and communication, while empowering individuals and teams to not only make key decisions, but to empower them to make these decisions with as little overhead (i.e. process) as possible. Despite the emphasis that should be placed on creating this type of culture, however, there will likely be certain types changes within an organization that require rigid change control processes, meaning formal ECM processes will be needed in certain instances.

Since certain enterprise-level changes require ECM processes, it is necessary to determine how these ECM processes can be integrated with Agile Scrum processes. In other words, Agile Scrum teams must be able to integrate and coordinate enterprise-level changes with regular changes to their product backlogs that occur through typical Scrum processes. According to Bogner and Elfanbaum[6], an important aspect of integrating ECM with an Agile project is embedding an ECM subject-matter expert (SME) on an Agile project team. This means that while the typical roles on an Agile Scrum team include a product owner, a Scrum Master and the team itself, an additional role is added in the form of an ECM SME. As Bogner and Elfanbaum[6] explain, the ECM expert should actively participate in all Agile Scrum processes to ensure that all enterprise-level changes are tracked and managed along with all other requirements for the system. This means that ECM SME's should ensure that all enterprise-level changes are tracked in the product backlog alongside any other requirements or defects [6]. As Bogner and Elfanbaum[6] state, “creating ECM stories in the same manner as their development tasks integrates change management into the Agile process”. Therefore, by bringing an ECM SME onto an Agile team, it is possible to ensure that all enterprise-level changes can be effectively managed, tracked and coordinated through typical Agile processes.

For instance, if certain requirements in the product backlog cannot be implemented due to an architectural change requiring approval through a formal ECM process, an Agile team can effectively coordinate any changes in the product backlog that depend on that change by tracking it alongside all other items in the product backlog. In other words, tracking all changes in the same backlog allows the team to easily determine which changes in the product backlog are blocked by the pending architectural change, allowing them to then reprioritize or modify items in the product backlog as needed. The SME can also ensure that all enterprise-level changes are progressing through any applicable ECM processes, while also keeping the team up-to-date on the status of any relevant enterprise-level changes. Ultimately, by including an ECM SME on an Agile team, the team will be able to effectively coordinate and manage all changes that might potentially impact them, regardless of the scope or type of change.

4. Designing Software to Accommodate Change

So far the discussion has focused on processes to effectively manage all forms change, yet it is also important to analyze strategies that can be employed by development teams when designing systems to ensure that changes can be managed and implemented as effectively as possible. According to Thomas [14], it is possible to design applications that can effectively and easily respond to change by utilizing table-driven programming. As Thomas [14] explains, table-driven programming is applicable in policy-driven systems, which are systems in which a set of rules is used to define the behavior of the system. The rules comprising these policies are defined using what is known as a policy language, which allows business analysts, engineers and others within the organization to easily view rules, and therefore the behavior of the system, regardless of their technical abilities [14]. Two key elements of table-driven programming are state tables and decision tables [14]. State tables define the various states that can be achieved based on a current state within a system, while decision tables define the rules that must be implemented when certain actions occur at specific state [14]. Since state tables and decision tables define the rules for various states and actions within the system, together they both represent the policies, and therefore the behavior, of the system [14]. Thomas [14] explains that by avoiding the hardcoding of logic in state tables and decision tables, and instead implementing them as tabular data structures (e.g. tables in a relational database), a project team can better respond to changes being that many changes to application behavior will no longer require changes to application source code. Instead, system behavior can be quickly changed by modifying data in the state and decision tables.

Thomas' [14] concept of table-driven programming aligns nicely with the concept of a rules engine, which according

to Browne [15] is a place, separate from the application itself, where business rules can be evaluated. A business rule is defined as "a statement that defines or constrains some aspect of the business [and]...is intended to assert business structure or to control or influence the behavior of the business" [16]. According to this definition, business rules represent policies and rules within an organization. Since business rules can define or constrain certain aspects of the business, business rules can directly impact and influence individual projects, including software development projects.

The rules engine provides a central place in which all business rules applicable to a specific application can be easily viewed, analyzed, modified and then utilized by the application itself [15]. Similarly to the concept of a policy language described above, business rules should be represented in such a way that they can be understood by anyone within the organization, regardless of their level of technical expertise [15]. Going back to the concept of table-driven programming, a rules engine could be utilized to define rules that might be found within a decision table, therefore defining behavior within a system. What is important, however, especially from an Agile perspective, is that rules engines allow an organization to easily and quickly make changes to business rules without requiring changes to source code. Instead of updating source code, a business analyst can simply update the rule in the rules engine, and this change will automatically be reflected in any application utilizing that business rule.

For instance, consider a simple example in which an application is being developed that includes functionality to calculate the commission that a salesperson receives for a sale. Without a rules engine, the rules for determining the proper commission would likely be hardcoded into the application itself. Assume that at the time of implementing this functionality the development team does not know the exact rules for calculating the commission, and are waiting on a decision from upper management (possibly through a formal ECM process) regarding what the specific business rules should be. The development team could hold-off on implementing this specific functionality until a decision is made, or they could hardcode the rules based on currently available knowledge, knowing that the code will likely need to be updated once a decision is made. However, if the former approach is used the team might be delayed in implementing critical functionality, while with the latter approach the software will need to be updated once a decision is made (which would include the overhead of creating a new release, testing, deployment, etc.). A third option, however, is to utilize a rules engine to define the rules for calculating the commission. By utilizing a rules engine, the team no longer needs to be concerned with getting the exact rules right for calculating the commission, and can instead create rules within a rules engine that will assume this responsibility. As a result, the team can fully implement the required functionality using the rules engine, knowing that as soon a decision is made regarding the exact

business rules for calculating a commission the rules engine can be quickly updated, and the application will then immediately reflect that change.

While the use of table-driven development and rules engines might not apply to all projects and scenarios, it does highlight the fact that certain design decisions can directly facilitate a project's ability to remain agile and effectively respond to various forms of change. These strategies prevent changes from blocking the implementation of critical functionality, while also ensuring that certain changes can be implemented with as little overhead as possible. Development teams should therefore analyze various aspects of a system to determine ways in which an application can be designed to allow the project to quickly and efficiently respond to change.

5. Conclusions

In conclusion, the ability to effectively control and manage change is absolutely critical when developing software. Without effective change control processes in place, changes can be unpredictable and therefore have the potential to negatively impact a project or organization. Various models for implementing effective change control processes exist, such as the *a priori* and gatekeeping processes that require formal documentation and analysis of RFCs, as well as formal approval through a CCB. However, the change control processes being implemented on a specific project depend heavily of the specific software development methodology being utilized. For instance, when utilizing the Waterfall methodology formal change control processes are often used. Agile methodologies, on the other hand, such as the Scrum methodology, utilize lightweight change control processes in which changes are easily accommodated without the need for extensive documentation or formal review.

While Agile processes are able to effectively accommodate change, especially when compared to Waterfall change control processes, there are certain changes that cannot be effectively managed through typical Agile processes. Specifically, enterprise-level changes, which are changes whose scope extends beyond a single Agile project, cannot be managed through typical Agile processes. Additional processes, which come in the form of ECM processes, are needed to manage these changes. When implementing ECM processes, an organization must create policies that include rules for controlling changes to various assets across the enterprise. For instance, certain assets might require formal change control processes, while others might require simple and informal change control processes. This means that on Agile teams, certain changes can be accommodated through typical Agile processes, while other changes, specifically enterprise-level changes, will require ECM processes. Agile teams must therefore be able to successfully coordinate both forms of change.

An organization must put careful consideration into

determining what changes require formal change control processes, and which changes do not. This is mainly due to the friction that exists between formal, rigid change control processes, and agile processes that favor lightweight change control processes and the ability to anticipate and effectively respond to change. In other words, rigid change control processes could directly impact a project's ability to operate according to Agile principles, meaning formal change control processes should only be utilized when absolutely necessary.

In order to effectively manage all forms of change that could potentially impact them, an Agile team must integrate ECM processes with Agile processes. This is done by including an ECM SME on the Agile team. The ECM SME can then ensure that all enterprise-level changes are tracked along with all other change in the Agile team's product backlog, therefore allowing the team to manage and track all changes in a single location. With all changes in a single backlog, a team can easily see the relationships and dependencies that exist between various changes, giving them a clear picture of the impact all changes will have on their project. Therefore, by effectively integrating ECM processes with Agile processes, Agile project teams can effectively control all forms of change, ensuring that the impact of all changes are well understood and properly managed.

Lastly, while so it is possible to manage all forms of change from a process perspective by integrating ECM process with Agile processes, development teams must also ensure that systems are designed to accommodate change to the fullest extent possible. In order to accomplish this, certain strategies can be employed when designing a system. For instance, table-driven programming and rules engines can be utilized to facilitate an organization's ability to modify system behavior, being that they provide mechanisms to change system behavior without the need to modify application source code. Development teams must therefore consider agility when designing all systems, ensuring that strategies to further enable the project and organization to accommodate and respond to change are utilized and leveraged wherever possible.

REFERENCES

- [1] B. Winter. *Agile Performance Improvement: The New Synergy of Agile and Human Performance Technology*, Apress, United States of America, 2015.
- [2] D. Canty. *Agile for Project Managers*, CRC Press, United States of America, 2015.
- [3] D. Sullivan. *The Definitive Guide to Enterprise Change Management*, Online available from <http://www.realtimepublishers.com/book?id=41>.
- [4] E. Crookshanks. *Practical Software Development Techniques: Tools and Techniques for Building Enterprise Software*,

Apress, United States of America, 2015.

- [5] The Enterprise Portfolio Management Council. Project Portfolio Management: A View from the Management Trenches, John Wiley & Sons, United States of America, 2009.
- [6] M. Bogner, D. Elfanbaum. The Agile Approach to Change Management, Online available from <http://www.baselinemag.com/c/a/IT-Management/The-Agile-Approach-to-Change-Management-440827>.
- [7] Manifesto for Agile Software Development. Online available from <http://Agilemanifesto.org>
- [8] R. Aiello, L. Sachs. Configuration Management Best Practices: Practical Methods that Work in the Real World, Addison-Wesley Professional, United States of America, 2010.
- [9] R. Wilson. A Comprehensive Guide to Project Management Schedule and Cost Control: Methods and Models for Managing the Project Lifecycle, Pearson FT Press, United States of America, 2014.
- [10] M. Karlesky. Agile Project Management, Online available from http://www.researchgate.net/profile/Michael_Karlesky/publication/229042037_Agile_Project_Management/links/5512b1c70cf270fd7e3332b1.pdf.
- [11] R. Mahapatra, G. Mangalaraj, S. Nerur. Challenges of Migrating to Agile Methodologies. Online available from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.2745&rep=rep1&type=pdf>.
- [12] C. Berger, B. Rumpe. Supporting Agile Change Management by Scenario-Based Regression Simulation. Online available from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.380.1000&rep=rep1&type=pdf>.
- [13] A. Cockburn, J. Highsmith. Agile Software Development: The People Factor. Online available from <http://tg-tatiana-oquendo.googlecode.com/svn-history/r25/trunk/CoHi-01.pdf>.
- [14] D. Thomas. Agile Programming: Design to Accommodate Change. Online available from http://info.lnpu.edu.cn/website/jpkc/rjgc/ywzl/Agile%20Programming_%20Design%20to%20Accommodate%20Change.pdf
- [15] P. Browne. JBoss Drools Business Rules, Packt Publishing, United Kingdom, 2009.
- [16] Defining Business Rules ~ What Are They Really?. Online available from http://www.businessrulesgroup.org/first_paper/BRG-whatIsBR_3ed.pdf