REQUIREMENTS Engineering in an AGILE Software Development ENVIRONMENT

W. Allen Huckabee

The Business Capability Lifecycle (BCL) methodology, which was implemented to develop defense business systems, requires a change in requirements engineering processes. Previous software development work by Systems, Applications, and Products on the Global Combat Support System-Army (GCSS-Army) followed the waterfall Software Development Life Cycle (SDLC), which is not acceptable in the BCL methodology. The typical functional requirement statement is not easily changed and introduces problems into an Agile SDLC. In this article, the author posits that Agile-based require-

Image designed by Diane Fleischer

ments (user story and acceptance criteria) best fit the BCL approach. By implementing best business practices and lessons learned from the GCSS-Army project, a typical BCL-led program can achieve significant benefits, such as (a) increased effectiveness in requirements meeting the users' needs; (b) increased performance of customers and software developers; and (c) reduced requirements volatility.

Keywords: *Agile, Business Capability Lifecycle (BCL), Investment Management (IM), requirements engineering, Software Development Life Cycle (SDLC)*

The Business Capability Lifecycle (BCL) is an "overarching framework" implemented by the Department of Defense (DoD) to "rapidly deliver" (Defense Acquisition University [DAU], 2013, p. 3) useful information technology (IT) capabilities to DoD users. The framework mandates the use of iterative development processes to deliver IT capabilities in "18 months from its Milestone B to Full Deployment Decision (FDD)" (p. 4). As the DoD moves toward becoming more integrated using Enterprise Resource Planning (ERP) systems, this article makes the case that the standard requirement statement and Work Breakdown Structure (WBS)-driven waterfall Software Development Life Cycle (SDLC) are not advantageous to the compressed cycle time required by the BCL methodology. In fact, lessons learned from the Global Combat Support System-Army (GCSS-Army), which replaced the existing suite of legacy Standard Army Management Information Systems, suggest that the standard Statement of Requirements-driven development is not as efficient as other methodologies. This article proposes that many benefits can be gained by performing more elaborate requirements engineering processes during the Investment Management (IM) phase of the BCL, using Agile-based user stories and acceptance criteria for integrating the Army's remaining logistics and tactical finance capabilities into GCSS-Army, while following the BCL methodology (DAU, 2013).

This article reports on a case study of project requirement-engineering processes and documentation of an ERP software development project, which seeks to identify the potential benefits of using Agile-based requirements-engineering processes. The project under analysis transitioned from the waterfall SDLC to an Agile SDLC. A limitation of this study is that access to quantitative data was restricted; therefore, such data could not be used in this study. A second limitation is that this article only addresses functional requirement statements, and therefore, quality and technical requirements are not addressed.

This article is organized in the following manner. First, a review of literature discusses common requirements-engineering processes used in typical software development projects. The final sections provide an overview of the requirements engineering process used on the GCSS-Army project, along with some lessons learned and benefits observed, followed by conclusions.

Literature Review

A review of business requirements-engineering literature highlights three general requirements-engineering processes used in the software development process: functional requirement statements (Institute for Electrical and Electronics Engineers [IEEE], 1998, p. 37), use cases (Regnell, Kimbler, & Wesslén, 1995), and Agile-user stories (Layman, Williams, Damian, & Bures, 2006). Paetsch, Eberlein, and Maurer (2003) defined requirements engineering as a process by which valid requirements are "identified, analyzed, and documented for the system being developed" (p. 1). These researchers suggested the main goal of traditional requirements-engineering activities is to "know what to build before system development starts" (p. 1). Generally speaking, this helps in reducing the cost of rework later in system development. Traditional methods typically utilize functional requirement statements, Software Requirements Specification (SRS) documentation, and use cases as methods of describing "what is to be done, but not how they are implemented" (Paetsch et al., p. 1). Additionally, these requirements engineering activities work very well with waterfall methods, but are not effective in iterative SDLCs. However, Paetsch et al. suggested that Agile requirements-engineering methods can be productive in an iterative development environment where software can be delivered faster, with "improved customer satisfaction and frequently delivered working software" (p. 1) utilizing user stories with less formal documentation processes.

Functional Requirements

Functional requirement statements "define the fundamental actions that must take place" (IEEE, 1998, p. 16) in the software system. Additionally, they provide detailed information on how a system should perform and how it should interact with databases and other systems, but do not address user interaction or business value. Detailed design constraints and compliance standards the system must meet are also included in functional requirement statements. Figure 1 provides an example of a functional requirement statement used on the GCSS-Army project. This example was taken from the GCSS-Army requirements database.

FIGURE 1. EXAMPLE OF A FUNCTIONAL STATEMENT OF REQUIREMENTS EXTRACTED FROM THE GCSS-ARMY REQUIREMENTS DATABASE

The system shall allow a user to enter mission and/or usage data.

Source for requirement: ULLS-G—P3–29(1) FD, ULLS-G EM 7.2.3

The example in Figure 1 is a simple one; however, Cohn (2004a) suggests that typical IEEE-style functional requirement statements are "time consuming to write and read, assume everything is known in advance" (p. 5), and lack early user feedback. Functional requirement statements are typically listed as "shall statements," where each requirement starts with "the system shall..." (p. 16). A functional requirement typically includes elements such as:

- Validity checks on the inputs;
- Exact sequence of operations;
- Responses to abnormal operations;
- Effect of parameters; and
- Relationship of outputs to inputs (IEEE, 1998, p. 16).

Functional requirement statements are rolled up into a single "software requirements specification (SRS) document" (IEEE, 1998, p. 4). A typical SRS describes all of the system's technical and functional specifications for products and systems. Paetsch et al. (2003) indicated that the SRS is "unambiguous, complete, correct, understandable, consistent, concise, and

feasible" (p. 3). Software requirements specification documents are typically provided to a program management office as the "baseline" (Paetsch et al., p. 3) as input into a "linear waterfall development activity" (Davies, 2001, p. 46) "before analysis starts" (Jacobson, Spence, & Bittner, 2011, p. 16). Jacobson et al. (2011) further suggested that requirements analysis "starts before implementation," and implementation is completed before the "verification starts" (p. 16), leaving user feedback out of the process until all development and testing has been completed, which is not conducive to iterative SDLCs.

Use Case

An approach used in both traditional and interactive software development projects to describe system requirements is the use case. Use cases allow analysts to solicit and document requirements from the customer with the goal of identifying and describing a number of "typical use cases for every actor" (Regnell et al., 1995, p. 1) interacting with the system. The use case is a component of the Unified Modeling Language, which supports iterative software development processes, thereby allowing an analyst to solicit user feedback early in the development cycle.

Additionally, a use case defines all of the ways of "using a system to achieve a particular goal for a particular user" (Jacobson et al., 2011, p. 4) and "describes the possible outcomes of an attempt" (International Institute of Business Analysis, 2015, p. 398) to accomplish that goal. Additionally, a use case makes it "clear what a system is going to do and, by omission, what it is not going to do" (Jacobson et al., p. 4).

Wiegers and Beatty (2013) provided an example of using use cases in gathering the requirements for a "Chemical Tracking System" (p. 161) in an iterative environment. The researchers suggested that in an iterative environment, waiting until the "requirements specification is complete" (p. 161) is too late to seek user feedback, and suggest that soliciting early and consistent feedback from users is a key success factor in documenting requirements in an iterative SDLC. This is a key difference in iterative processes and traditional processes. For example, Paetsch et al. (2003) conducted a study that compared traditional requirements-engineering methods, use cases, and Agile software development approaches. These researchers indicated that customer involvement was a primary difference between the different methodologies, which can be beneficial to the success of a software development project.

Additionally, use cases are written from the user's perspective to "avoid describing the internal workings of the system" (International Institute of Business Analysis, 2015, p. 398) and are very detailed. According to the institute, there is "no fixed, universal format" (p. 398) for creating a use case. However, Wiegers and Beatty (2013) recommended the use of a template in the form of a Microsoft Word document or spreadsheet with a formal organization. A use case has certain elements that are considered mandatory, which are listed in Table 1.

TABLE 1. MANDATORY ELEMENTS OF A USE CASE

Element	Description	Prior Research
Name or ID	The unique name of the use case.	International Institute of Business Analysis, 2015; Wiegers & Beatty, 2013
Goal	Brief description of a successful outcome.	International Institute of Business Analysis, 2015
Primary Actor or Actor	A person or external system that interacts with the system.	International Institute of Business Analysis, 2015; Wiegers & Beatty, 2013
Preconditions	Any fact that must be true before the use case can begin, which acts as a constraint on its execution.	International Institute of Business Analysis, 2015
Post Conditions; Guarantee	Any fact that must be true for all possible primary and alternative flows when the use case is complete.	Wiegers & Beatty, 2013
Trigger	An event that initiates the flow of events for a use case.	International Institute of Business Analysis, 2015; Wiegers & Beatty, 2013
Exceptions	Any exceptions/messages that must be handled by the system.	Wiegers & Beatty, 2013
Flow of Events	The activities performed by the actor and the system during the use case's execution.	International Institute of Business Analysis, 2015

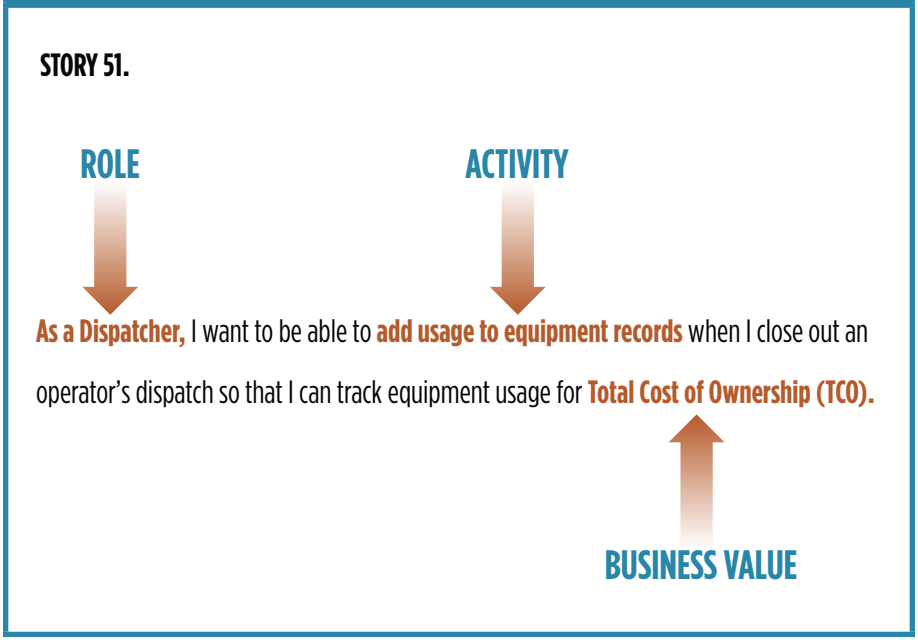
Use cases have some advantages and limitations. For example, Regnell et al. (1995) suggested use cases help deal with the "complexities of the requirements analysis process" by allowing customers and developers to

"focus on one, narrow aspect of system usage at a time" (p. 1). Lee, Cha, and Kwon (1998) added that use cases are easy to "describe and understand and are scalable" (p. 1), allowing the customer to trace use cases throughout the SDLC. Like Wiegers and Beatty (2013), Regnell et al. (1995) indicated that one advantage of the use case is that it facilitates active collaboration between the customer and developer, which enables the developer to learn about "potential users, their actual needs, and their typical behavior" (p. 1). However, they further indicated this approach can produce a "loose collection of use cases, which can lack 'synthesis'" (p. 1), which is a weakness. Lee et al. (1998) identified the "lack of rigor" and no "systematic approaches to analyzing dependencies" among the many use cases developed for a system, which impedes "detecting flaws" (p. 1), as limitations to this approach.

User Stories

Another well-known approach to requirements engineering in iterative SDLC environments is the Agile requirements-engineering methodology. In an Agile SDLC, user requirements are captured and recorded as user stories (Layman et al., 2006). A user story removes the formality normally associated with typical requirements engineering activity. They still define what the system is to perform, but from the user's perspective, with a focus on business value (Saddington, 2012). User stories provide a context within which a requirement is to be developed around some "feature, functionality, or capability needed" (Coplien & Bjørnvig, 2011, p. 167). User stories provide a more effective means by which the customer, in coordination with the program office, can link a user requirement to the system's mission-critical functions required to meet organizational goals (Huckabee, 2013). Figure 2 provides an example user story, which is a conversion of the functional requirement statement in Figure 1 to an Agile user story.

FIGURE 2. EXAMPLE OF AN AGILE USER STORY



Note. This user story was created from the functional Statement of Requirements shown in Figure 1.

Wiegers and Beatty (2013) suggested that user stories are concise statements that "articulate user needs and serve as a starting point" (p. 144) for customer and developer collaboration. Use cases are different from functional requirement statements, which focus on a single system task. User stories are an "interaction" (Nazzaro & Suscheck, 2010, p. 2) between the user and the system, focusing on business value. User stories are written or told from the "perspective of the person who needs the new capability" (Wiegers & Beatty, 2013, p. 145). They are informal and written in plain English on an index card. User stories typically describe a process or process step, focusing on a user role (or another system), which performs the process and achieves the business value. User stories can also be broken down into "quantifiable units of development effort" (Breitman & Leite, 2002, p. 3), which can increase the accuracy of estimating scope.

User stories identify critical success factors used to measure system performance during development. However, to be effective, the format of user stories must follow standards in their creation, use, and interpretation. Table 2 describes user story components.

TABLE 2. ELEMENTS OF A USER STORY

Element	Description	Prior Research
Title	Story title	International Institute of Business Analysis, 2015; Rees, 2002
User story number or ID	Unique identifier of the requirement	Rees, 2002
Value statement	Value achieved from the capability	International Institute of Business Analysis, 2015; Nazzaro & Suscheck, 2010; Rees, 2002
Conversation or activity	Action being performed by the system; aids in understanding the features and/or values to be delivered	International Institute of Business Analysis, 2015; Nazzaro & Suscheck, 2010; Rees, 2002
Related story numbers	Relates the current story to other stories	Rees, 2002
Acceptance criteria	Defines the boundaries of the capability; describes system specifications	International Institute of Business Analysis, 2015; Koch, 2005; Leffingwell & Widrig, 2003; Resnick, Bjork, & de la Maza, 2011; Sy, 2007

The most important feature of a user story is its use in promoting collaboration between the customer and software development team about a need or needed capability. Storytelling is a major part of the process where the customer tells a story about a user's need or capability with some acceptance criteria. Cao and Ramesh (2008) conducted a qualitative study of 16 organizations using Agile requirements-engineering processes. They suggested that using user stories in an Agile-based software program creates a more satisfactory relationship between the customer and developer. As iterations of storytelling and demonstrations continue, the requirements will change until all acceptance criteria have been demonstrated and accepted by the customer, tested, and promoted to the production system. Cao and Ramesh also suggested that in an Agile environment, user stories produce "clearer and more understandable" requirements because of the "immediate access to the customer" (p. 64). Leffingwell and Widrig (2003) agree and suggest that when the software developer misunderstands or misinterprets customer needs, trust is reduced, which can result in the "inability of the program manager to resolve budget and schedule conflicts" (p. 782).

Acceptance criteria accompany each user story and are defined when the user story is created. Acceptance criteria define when development is complete (Resnick et al., 2011), and when a story is added to a sprint the acceptance criteria can be adjusted. This is where the customer communicates system specifications to the development team. Unlike user stories, acceptance criteria have no defined content or format (Figure 2). However, Nazzaro and Suscheck (2010) suggested that acceptance criteria can be a "test case or a brief description of 'done'" (para. 11). Also, acceptance criteria must be clearly understood by all parties, as it helps in establishing a shared understanding of success. A story's acceptance criteria should include usability requirements, specific performance metrics, and data validation requirements. Including these components in acceptance criteria assists the customer in defining measurable and testable criteria (Koch, 2005; Leffingwell & Widrig, 2003; Sy, 2007).

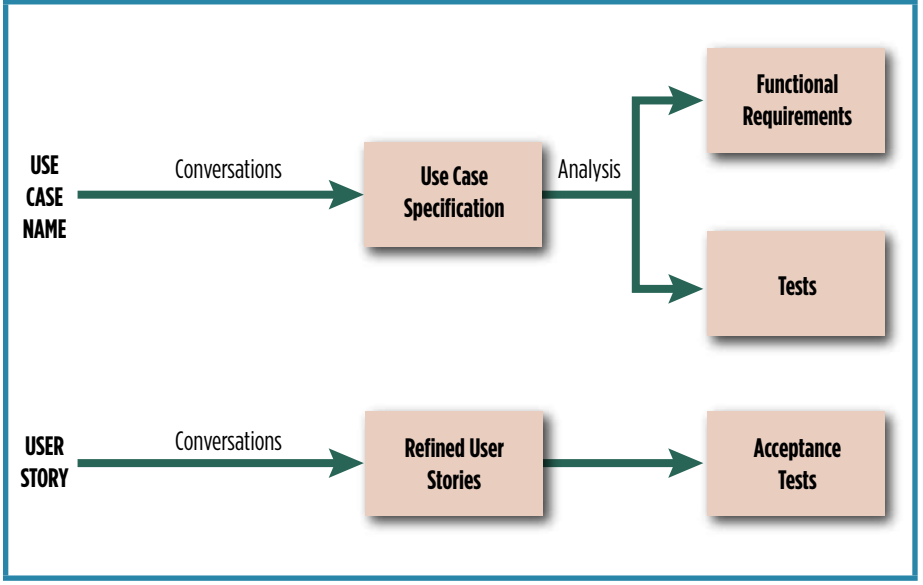
Acceptance criteria that are too detailed can limit collaboration and result in a misinterpretation of a requirement, whereas acceptance criteria with little detail create a scenario where a requirement is missed. The right mix of acceptance criteria will become clear with experience; however, best business practices dictate that not all the details need to be included in the acceptance criteria for a given story. For example, this article suggests that more details about a need or capability can be provided as an attachment, such as a mock-up, spreadsheet, and/or algorithm, and additional criteria can be placed in integrated test cases for validation later in the development cycle (Leffingwell & Widrig, 2003; Nazzaro & Suscheck, 2010; Resnick et al., 2011).

A Comparison of Use Case and User Stories

Requirements engineering literature reveals that use cases and Agile user stories are both advantageous in iterative SDLCs; however, some differences exist. Both use cases and user stories initiate a dialogue with the customer about the desired capability and are both "sized to deliver business value" (Cohn, 2004b, para. 14). Davies (2001) suggested the primary differences between the two methodologies are in the way "their scope is determined" (p. 46) and the artifacts produced during the requirements gathering activities, as well as "consistency" (p. 48). Nazzaro and Suscheck (2010) suggested the primary difference is that use cases communicate system capabilities, while the user story focuses on "customer value" (para.

16). The use case is more formal and detailed, whereas user stories are less formal. The deliverables or artifacts produced using the two approaches vary (Figure 3). Wiegers and Beatty (2013) described these as a "core distinction" (p. 146), which aligns with Davies (2001) in that the artifacts produced from the use case approach include a "use case model, a design model, software development plan, software components, and a test plan and test cases" (p. 48).

FIGURE 3. COMPARISON OF USE CASE APPROACH AND USER STORY APPROACH



Note. Adapted from Wiegers and Beatty, 2013, p. 146. Copyright 2013 by Karl Wiegers and Seilevel. Reprinted with permission.

Davies (2001) suggested that user stories are less formal and written on an index card, and the artifacts produced using user stories are a "story card, engineering tasks, source code with associated unit tests, and acceptance tests and a software release" (p. 48). This aligns with Cohn (2004a) and Wiegers and Beatty (2013) in that user stories are "smaller in scope" (para. 14) than use cases.

The use case methodology is more consistent than the user story methodology because the goal behind use cases is to provide a complete set of requirements documents, whereas "gaps can emerge" when using Agile stories because the development activities in a sprint reflect only "those requirements discussed with the customer" (Davies, 2001, p. 48); it is the

customer's responsibility to ensure that any gaps in requirements are identified during the demonstration of the software at the end of each sprint. However, Nazzaro and Suscheck (2010) would disagree; they suggest that the higher level of collaboration between the customer and developer using Agile stories produces a higher level of detail than use cases.

Finally, both methods define the boundaries on what is expected to be delivered and define when development is done, as well as help to establish process objectives and thresholds, such as screen refresh rate, printing times, or exporting formats. The detailed nature of use cases is good at "articulating the functional behavior of a system" (p. 401). In contrast, user stories are good in helping to "capture stakeholder needs" and prioritizing development activities, and they serve as a good basis for estimation and project planning, WBS development, requirements traceability, and for "project reporting" (International Institute of Business Analysis, 2015, p. 402).

GCSS-Army Requirements-Engineering Overview

Requirements engineering activities on the GCSS-Army program have changed over the past 5 years. When the program began, requirements engineering activities followed the waterfall SDLC, where a number of requirements in a functional specification document (database version) were handed over to the developer for planning, analysis, and development. These requirements were in the form of functional requirement statements (Figure 1) that defined system operation. The program started with over 8,000 functional requirement statements; however, because of program rescoping activities, the requirements were reduced to just over 4,500. These functional requirement statements limited the program's abilities to interpret the requirements, because many lacked the important business rules required to fully develop a specified capability. Moreover, the functional requirement statements contained limited test criteria; experience from Army logistics subject matter experts was relied upon to develop test criteria to validate requirements, which constrains incremental development.

Often, these functional requirement statements failed to tie system activity to business value or to the organizational goals that users expected, possibly limiting the system's benefits once deployed. Also, functional requirement statements do not allow for change, which is the norm in incremental SDLC activities. In typical incremental activities, requirements are modified during development based on the customer's priorities during a sprint.

Most of the functional requirements found in the Combined Arms Support Command's GCSS-Army requirements database originated from antiquated software end-user manuals of systems no longer in service. For example, the functional requirement statement in Figure 1 was extracted from the Unit Level Logistics System–Ground end-user manual. The replacement of this system began in the mid-1990s with the Standard Army Maintenance System–Enhanced (SAMS-E). Additionally, functional requirement statements such as Figure 1 were never purged or updated. The antiquated statements may still be valid; however, many of the statements are not connected to regulatory guidance and are not process-oriented, which reduces the effectiveness of Business Process Reengineering (BPR). This disconnect adds complexity and error to the planning, analysis, and development processes and can add risk in a compressed development timeline. This can also result in the fulfillment of a requirements list, instead of focusing on delivering capabilities that add business value, or that can be linked to organizational goals (Saliu, 2005). Finally, to overcome these limitations, the Program Manager (PM) GCSS-Army mandated a change in the acquisition strategy for production release 1.1 and beyond.

> ***The Agile methodology is aimed at increasing productivity, reducing requirement volatility, increasing customer satisfaction, and improving software quality focusing on incremental development (Maurer & Martel, 2002).***

In 2009, PM GCSS-Army directed the systems integrator to depart from the waterfall SDLC and adapt the Agile SDLC methodology. Background data supporting the move to the new methodology indicated productivity issues, requirements volatility, and the need for rapid prototyping to meet program scope, schedule, and budget constraints. The Agile methodology is aimed at increasing productivity, reducing requirement volatility, increasing customer satisfaction, and improving software quality focusing on incremental development (Maurer & Martel, 2002). During this change, analysis of functional requirement statements ceased and user stories became the standard for GCSS-Army requirements, introducing new challenges for the program.

Even though the program provided Agile training to project members, moving from functional requirement statements to Agile user stories was a paradigm shift. With this shift, the program office had not established standards for user story development. Without a standard, customers developed user stories with no specific format or criteria by which to validate what was to be delivered. This created an atmosphere where the customer and developer lacked a shared understanding of what defined success with regard to a capability's specification or how user stories were to be interpreted. This lack of understanding of story structure, content, and format created increased requirement volatility in the Wave 1 product release, which started with an approved requirements baseline of just 200 user stories. The volatility in Wave 1 generated over 300 change documents, either modifying existing requirements, or adding requirements that were missed.

By applying best practices to what has been learned about the Agile methodology over the past 5 years to current and future development efforts, a standardized process for creating user stories and associated acceptance criteria can be created. Standardized processes for creating user stories will increase the customer's ability to develop measurable and testable user stories; increase the effectiveness of the systems integrator's planning, analysis, and development activities; reduce the negative impact on the program's scope, cost, and schedule; and deliver a quality product that meets the customer's expectations. These benefits align with findings by Cao and Ramesh (2008) that Agile requirements engineering can "produce clearer and more understandable requirements" (p. 64), with capabilities that are more aligned with the customer needs and can be better prioritized as the customer's needs change.

Best business practices also dictate that a link to other stories be placed in the acceptance criteria. Linking the current story and acceptance criteria to other requirements helps the PM keep scope creep to a minimum. Lessons learned from the GCSS-Army program indicate that the development of one story can impact other stories; therefore, a link is required to reduce the amount of rework or defects later in the SDLC. Additionally, this link provides integration points to existing stories or stories that have not been created. This link is necessary to ensure requirements are completely integrated into the enterprise solution, and it helps in integration and regression testing later in the development cycle. For example, in Figure 4 the Dispatcher role does not track the total cost of ownership, but the role does contribute to the business objective, which adds value for the Army.

With the addition, in the acceptance criteria, of two sentences that link to other stories (roll-up of usage data), a customer can prevent scope creep, errors, and defects downstream in development.

FIGURE 4. EXAMPLE ACCEPTANCE CRITERIA TAKEN FROM GCSS-ARMY REQUIREMENTS TRACEABILITY MATRIX

ACCEPTANCE CRITERIA

Demonstrate that GCSS-Army will 1) allow me to update usage on an end item when a Dispatch is closed 2) allow me to update usage on components when a Dispatch is closed 3) allow me to view the total usage on an end item and/or components 4) demonstrate that equipment usage is provided to LOGSA through the backwards compatibility interface currently in production.

Link to other stories:
No roll-up of usage by equipment category or equipment serial number is needed now (another story).
No roll-up of usage by component is needed now (another story). (POC Jane Smith).

Story Controls:
AR 750-1, DA Pam 738–751, and DA Pam 750-8

Note. AR = Army Regulation; DA = Department of the Army; LOGSA =Logistics Support Activity; Pam = Pamphlet; POC = Point of Contact.

Lessons learned from previous development activities would indicate that some form of controls be placed on Agile requirements and that such controls become a best practice in the development of Agile requirements. Story controls define the boundaries for an Agile requirement. These controls are found in the Army Integrated Logistics Architecture (U. S. Army, 2008) as inputs to operational activities. Story controls consist of Army Regulations, a Department of the Army pamphlet, and field manuals. These controls connect the Agile requirement to the logistics architecture, establish references to the as-is processes, and aid in BPR. Additionally, story controls assist the customer and developer in demonstrating where a software solution can fill capability gaps and in identifying the policy implications brought on by BPR. Controls facilitate the customer's dialogue with the logistics and tactical finance communities on required policy changes. Finally, story controls

benefit the program by providing a shared understanding of specific regulatory requirements, facilitate policy updates and requisite business rules, and prevent scope creep.

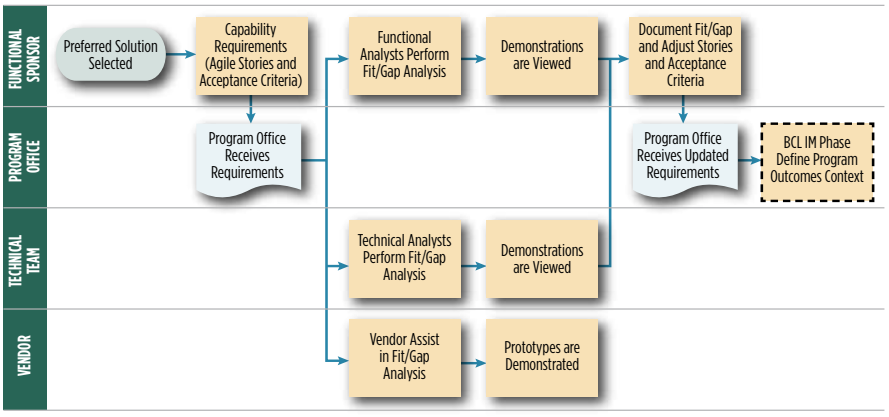
Refining Agile Requirements

The BCL methodology provides a 12-month block of time between Milestones A and B, when program planning occurs. This is when Agile requirements can be refined and become part of the potential program scope and approach documentation, which is part of the prototyping phase. At this point, the sponsoring organization should coordinate with the program office to provide a technical team to work with the functional sponsor in reviewing and refining the requirements through product demonstrations and prototyping. These actions align with findings by Cao and Ramesh (2008) that a benefit of prototyping allows the customer to "validate and refine requirements" to obtain "quick customer feedback" (p. 65). This is an important step that must not be overlooked. For example, performing this analysis enables the technical team to determine how a product can fulfill requirements with out-of-the-box capabilities, limiting the amount of customization required to fulfill the user's requirements, which is one of the goals of the BCL methodology. During the refinement process, the technical team works with the functional sponsor to review requirements; provide specific solutions and recommendations based on requirement analysis, product demonstrations, prototyping, and simulations; and document the solutions' fit/gap. In this study, a fit/gap analysis is the method of comparing as-is "enterprise processes and system functions to adapt local processes to industry best practices" (Pol & Patukar, 2011, p. 2) contained in a software solution. A fit/gap can be performed by different methods; among them are demonstrations, or what Pol and Paturkar defined as "simulations" (p. 2). Once the fit/gap analysis is complete, user stories and acceptance criteria are modified to address the solutions' fit/gap with the user's requirements. This final step reduces program scope and schedule risk by providing the systems integrator with a list of refined requirements for estimation and development.

From a BCL process perspective, the fit/gap analysis should be initiated once the preferred solution has been identified and serve as an input into the Define Program Outcome context. This is because during the business process reengineering activities, the functional sponsor has gained an understanding of the processes to be implemented into the software solution. The outcome of this process should be a set of reengineered process

models with known requirements and potential gaps. Figure 5 describes the proposed Agile requirements-engineering methodology as it relates to the BCL process (DAU, 2013).

FIGURE 5. PROPOSED AGILE REQUIREMENTS-ENGINEERING METHODOLOGY



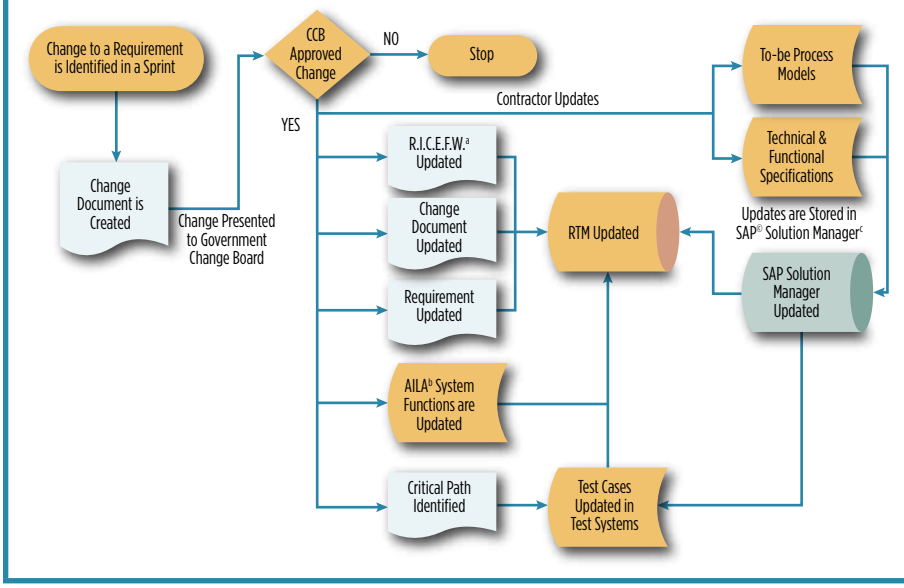
Once the requirements and gaps are identified, the technical team, functional sponsor, and vendor work together to analyze the requirements to demonstrate how the solution can fulfill the requirements and analyze potential gaps to determine whether the solution can fulfill the gaps without customization. The fit/gap results are annotated and the Agile requirements are updated to reflect the new information. The annotated results and updated requirements are then handed off to the program office as input into the Define Program Outcome context (DAU, 2013).

Managing Requirements during Development

One of the most difficult tasks of an Agile project is tracking changes to the Agile requirements baseline. This need for tracking is common on Agile projects, as most requirements generated in the requirements engineering process can be modified based on the customer's priorities while in a sprint. From a capabilities development perspective, lessons learned on the GCSS-Army project show that requirements management and traceability are difficult challenges. To address this challenge and reduce requirement volatility, the PM GCSS-Army has created tools and

a methodology to manage requirement changes and traceability using an online Requirements Traceability Matrix (RTM), as well as commercial software packages used to track requirements as development objects move through the development landscape. The process flow in Figure 6 describes the methodology used to create the online RTM. Because of the iterative nature of an Agile SDLC, the methodology is a critical component of an Agile acquisition project as large as GCSS-Army, and more emphasis must be placed on this process to ensure that user requirements implemented in the solution meet the sponsoring organization's needs.

FIGURE 6. REQUIREMENTS TRACEABILITY MATRIX METHODOLOGY



Note. CCB = Change Control Board; [a]R.I.C.E.F.W. = R-Report; I-Interface; C-Conversion; E-Enhancement; F-Form; W-Workflow. Each of these objects is a development object. [b]AILA = Army Integrated Logistics Architecture; [c]SAP Solution Manager = Systems, Applications and Products Solution Manager.

Proposed Benefits

In addition to the benefits mentioned earlier, implementing the best practices and lessons learned presented in this article will generate advantages for a BCL program. Some of the benefits that can be realized from a more elaborate requirements engineering process include: (a) increased effectiveness in meeting user needs; (b) increased performance of customer and software developers; (c) reduced requirements volatility; (d) a defined

functional and technical scope baseline to be included in the contract documentation at Milestone B; (e) less uncertainty in the estimation process; (f) the potential for a standardized process that can be used DoD-wide; and (g) increased customer satisfaction. Finally, these benefits provide the justification for PMs to use the best business practices recommended in this article.

Conclusions

Change in the requirements engineering processes is required to ensure the success of a BCL-based defense business system development activity. This change is required in part because the BCL approach depends on an accurate and prioritized list of Agile requirements and accurate program scoping so as to facilitate a focus on fielding usable business capabilities as quickly as possible (DAU, 2013, p. 12). Accurate Agile requirements engineering provides the foundation for a successful BCL program because it is more receptive to change. Using story controls establishes the boundaries of the requirement, potential process objectives, and thresholds, and promotes understanding and communication between the customer and developers. Using a standardized and elaborate requirements-engineering process following the Agile software development methodology to develop and refine requirements can provide significant benefits. Finally, following best business practices will help in reducing uncertainty and requirement volatility, thus increasing the chances of success in the short cycle time mandated by the BCL methodology.

> ***Following best business practices will help in reducing uncertainty and requirement volatility, thus increasing the chances of success in the short cycle time mandated by the BCL methodology.***

References

Breitman, K., & Leite, J. (2002). *Managing user stories.* Paper presented at the International Workshop on Time-Constrained Requirements Engineering, Essen, Germany, September 9.

Cao, L., & Ramesh, B. (2008). Agile requirements engineering practices: An empirical study. *IEEE Software, 25*(1), 60–67.

Cohn, M. (2004a). *Advantages of user stories for requirements.* Retrieved from http://www.mountaingoatsoftware.com/articles/advantages-of-user-stories-for-requirements

Cohn, M. (2004b). *User stories applied: For agile software development.* Retrieved from http://www.mountaingoatsoftware.com/uploads/presentations/User-Stories-Applied-Agile-Software-Development-XP-Agile_Universe-2003.pdf

Coplien, J. O., & Bjørnvig, G. (2011). *Lean architecture: For Agile software development.* West Sussex, UK: Wiley & Sons.

Davies, R. (2001). *The power of stories.* Retrieved from http://ciclamino.dibe.unige.it/xp2001/conference/papers/Chapter11-Davies.pdf

Defense Acquisition University. (2013). *Defense acquisition guidebook.* Fort Belvoir, VA: Author.

Huckabee, W. A. (2013). *The relationship between effective strategy and enterprise resource planning (ERP) systems business processes: A critical factor approach* (Unpublished doctoral dissertation). Minneapolis, MN: Capella University.

Institute of Electrical and Electronics Engineers. (1998). *IEEE recommended practice for software requirements specifications.* New York, NY: Author.

International Institute of Business Analysis. (2015). *BABOK v3: A guide to the business analysis body of knowledge.* Toronto, Ontario, Canada: Author.

Jacobson, I., Spence, I., & Bittner, K. (2011). *Use Case 2.0: The guide to succeeding with use cases.* Retrieved from http://www.ivarjacobson.com/Use_Case2.0_ebook

Koch, A. S. (2005). *Agile software development: Evaluating the methods for your organization.* Norwood, MA: Artech House.

Layman, L., Williams, L., Damian, D., & Bures, H. (2006). Essential communication practices for extreme programming in a global software development team. *Information and Software Technology, 48*(9), 781–794. doi: http://dx.doi.org/10.1016/j.infsof.2006.01.004

Lee, W. J., Cha, S. D., & Kwon, Y. R. (1998). Integration and analysis of use cases using modular Petri nets in requirements engineering. *IEEE Transactions on Software Engineering, 24*(12), 1115–1130.

Leffingwell, D., & Widrig, D. (2003). *Managing software requirements: A use case approach* (2nd ed.). Boston, MA: Pearson.

Maurer, F., & Martel, S. (2002). Extreme programming. Rapid development for Web-based applications. *IEEE Internet Computing, 6*(1), 86–90.

Nazzaro, W., & Suscheck, C. (2010). *New to user stories?* Retrieved from http://www.scrumalliance.org/community/articles/2010/april/new-to-user-stories

Paetsch, F., Eberlein, A., & Maurer, F. (2003). *Requirements engineering and agile software development.* Paper presented at the 2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Toulouse, France, June 25–27.

Pol, P., & Paturkar, M. (2011). *Methods of fit and gap analysis in SAP projects.* Bangalore, India: Infosys.

Rees, M. J. (2002). A feasible user story for agile software development? *Proceedings of the Ninth Asia-Pacific Software Engineering Conference* (pp. 22–30), Queensland, AU, December 4–6.

Regnell, B., Kimbler, K., & Wesslén, A. (1995). Improving the use case driven approach to requirements engineering. *Proceedings of the Second IEEE International Symposium on Requirements Engineering* (pp. 40–47), York, UK, March 27–29.

Resnick, S., Bjork, A., & de la Maza, M. (2011). *Professional scrum with team foundation server 2010.* Hoboken, NJ: Wrox.

Saddington, P. (2012). *Agile pocket guide: A quick start to making your business agile using Scrum.* Somerset, NJ: Wiley.

Saliu, M. O. (2005). Understanding story-driven development processes. *IEEE Software, 22*(6), 103–105.

Sy, D. (2007). Adapting usability investigations for agile user-centered design. *Journal of Usability Studies, 2*(3), 112–132.

U.S. Army. (2008). *U.S. Army posture statement: Army integrated logistics architecture.* Retrieved from http://www.army.mil/aps/08/information_papers/transform/Army_Integrated_Logtistics_Architecture.html

Wiegers, K., & Beatty, J. (2013). *Software requirements* (3rd ed.). Redmond, WA: Microsoft Press.

Biography

Dr. W. Allen Huckabee is a consultant with LMI providing technical expertise to the test director of Global Combat Support System-Army at Fort Lee, Virginia. Dr. Huckabee provides support to ensure the acquisition program is effective and suitable for combat use. Before joining LMI, he served as a capability developer for GCSS-Army. Dr. Huckabee earned his MBA in Business Management from Saint Leo University and his PhD in Organization and Management with Specialization in Project Management from Capella University.

(E-mail address: phdhuckabee@outlook.com)