The Role of the Story Master: A Case Study of the Cognitive Load of Story Management Tasks

Aaron Read
Center for Collaboration Science
University of Nebraska Omaha
aread@mail.unomaha.edu

Nicholas Arreola
Center for Collaboration Science
University of Nebraska Omaha
narreola@mail.unomaha.edu

Robert O. Briggs
Center for Collaboration Science
University of Nebraska Omaha
rbriggs@mail.unomaha.edu

Abstract

Agile software development methods allow design teams to respond quickly to changing circumstances. Much of this flexibility is achieved through increased involvement of the customer who becomes responsible for the development of the specifications of the software and the communication of these requirements to the developers. Reports have shown that the customer is often overloaded in the role. We discuss the customer’s role in terms of cognitive effort and find that much of this effort performed by the customer can be performed by someone else. We explore this possibility through the experiences of two individuals in the “story master” role, a role created as part of a complex software project, as an assistant to the customer. We present the evidence from the case study as a set of tasks performed by the role which reduce cognitive effort. We evaluate the role’s success as an Agile Information Systems Development component using Conboy’s [1] framework.

1. Introduction

Information systems projects continue to be troubled by low success rates [2]. Defects in system requirements account for a major portion of these failures [3]. It is a cognitively demanding task to assure the correctness, completeness, and consistency of requirements for complex systems. It is not uncommon for a large-scale system to have tens of thousands of requirements with complex interdependencies.

Requirements can be made more manageable by representing them systematically with modeling and structuring techniques [4-7]. Modeling a system may require extensive up-front effort to discover and document requirements. Much of this work may be wasted, however, because it is common for requirements to change during the course of a large-scale systems project as the business environment evolves and as stakeholders gain new insights about their problems and possible solutions.

Agile software development methodologies try to minimize the cognitive effort of creating and maintaining system requirements by keeping documentation to a minimum. These methods often capture a high-level summary of a requirement as a user story — a one sentence description of a user’s interaction with some feature of the system [8, 9]. In agile methods, details of the requirements are meant to be communicated directly between the developers and the customers as the software develops. This allows requirements to be moved very quickly from the intentions of the customer to the development of software without going through the effort of formal specifications. For this approach to work, developers and customer are encouraged to be co-located and must invest considerable time in face-to-face conversations.

Several reports have shown that the customer role and related Agile roles such as the product owner are overloaded with work. [10, 11]. In our experience using the Scrum method to develop a complex software application with many detailed software requirements, we have found as well that the product owner is overloaded. Many meetings discussing design details with developers and a UI designer make it difficult to assure that decisions made end up as developed user stories.

Much of the work surrounding user story or requirement management involves assuring that important design details are correctly captured, organized, and communicated. This work can be described in terms of cognitive load and cognitive effort. The Time-Based Resource Sharing (TBRS) model of cognitive load [12] describes the limited information processing capability of short term memory in terms of the number and difficulty of information retrievals that must be performed in a given amount of time. As the number and difficulty of performing information retrievals increase, the amount of cognitive effort increases. The implications of TBRS for requirements management will be discussed in greater detail later in the theory section. Essentially, TBRS allows us to understand how the need to comprehend and work with details leads to more time and effort. We can then understand the cognitive effort
needed to be performed by a customer in terms of the information cues they need to process.

While a customer is central to decision-making and prioritization of user stories, much of the work commonly attributed to the customer role, such as backlog management [13], and the writing of acceptance tests may not need to be performed by the customer. In our study, we explore the feasibility of transferring such tasks to another individual who takes on the role of a story master, through a case study of the experiences of two individuals who assumed the role. Exploring this possibility has potential economic benefits, in addition to having benefits for researchers seeking to understand the nature of cognitive load in Agile Software development. Customers and other roles that support the development of requirements in software development projects such as business analysts and requirements engineers are expensive to train and employ. The story master can provide the benefit of reducing the cognitive load of the customer role with much less cost, since they do not require the training or expertise of the other roles mentioned. This case study illustrates the feasibility of transferring cognitive efforts to another individual. Each cognitive effort is explained in terms of TBRS. We also analyze the agility of this role using Conboy’s [1] framework to illustrate its success as an agile software development method. We discuss our findings and suggest future research.

2. Background

In this section we will describe the tasks of the customer in terms of their cognitive effort. We will begin by discussing the TBRS model of cognitive load and how it can be used to describe the cognitive effort of customer tasks. We then discuss the tasks of the customer in terms of cognitive effort.

2.1 The Time-Based Resource Sharing Model of Cognitive Load

In the education literature, cognitive load is classified as extrinsic, intrinsic and endogenous [14]. Researchers use this classification scheme to focus on eliminating material that contributes to cognitive load but do not contribute to understanding. However, this classification scheme does not provide insight into the cognitive mechanisms that give rise to cognitive load. The Time Based Resource Sharing model of working memory (TBRS) proposes such mechanisms [12, 15]. Unlike earlier cognitive models, TBRS proposes that working memory has a single central processor that is single threaded – it can only complete one operation at a time. The central processor must divide its time among several tasks, such as processing information cues, fetching things from long term memory, pushing things from working memory to long term memory, processing concepts in working memory, and refreshing the contents in the contents of working memory. Items held in working memory fade in a few seconds and must be refreshed by the central processor to keep them active in working memory. Each of these operations requires time, so there is a limit to how many operations can be executed in a given slice of time before concepts start to drop out of working memory. According to TBRS, cognitive load is defined as the proportion of time the processor is busy doing other things to maintaining items in short term memory. It can be represented by the equation:

\[ CL = \sum aN/T \]

where \( N \) represents the number of operations, \( a \) the difficulty of those operations, and \( T \) the time taken for a set of operations.

If the logic of TBRS holds, then the number of information cues necessary for a person to complete a task pertaining to system requirements would directly affect the cognitive load imposed by the task. As information cues increase, the number of fetches and pushes to and from long-term memory would also increase, which could cause a cascading of cognitive load. The less expertise or understanding a person has of the information cues, the more difficult it is to process those cues, and thus the negative impact of cognitive load increases.

While cognitive load considers the number and difficulty of operations needed to process information cues in a set amount of time, cognitive effort can be conceptualized as the amount of effort and the number of operations needed to complete a task. In other words, as the quality and quantity of informational cues associated with a specific task increase, so does the level of cognitive effort required to complete that task.

2.2 Tasks of the Customer Role

Regardless of the Agile Methodology followed, the customer is central to the supply, clarification, and validation of requirements. The greater the flexibility afforded by the Agile approach the greater the need for feedback from the customer [16]. The customer, therefore, also has the responsibility to collaborate and communicate feedback frequently with the developers when building the software instead of working through an upfront contract [17]. This is accomplished through the following activities:
• Providing stories [18].
• Maintaining stories in the backlog (In Scrum)[13].
• Prioritizing stories so that the most important features of the software are developed first [18].
• Participating in sprint planning meetings or the planning game where the effort to build the software in the stories is estimated.
• Developing acceptance tests with which to validate the software.
• In XP, the customer is encouraged to be onsite in order to clarify requirements [8].

Literature discussing the customer role in agile highlights the importance of customer tasks associated with getting the requirements right, as many challenges associated with the customer role are related to developing requirements correctly. This is a natural result of the need for requirements to be developed quickly in Agile settings. These challenges include:

• Be knowledgeable enough to formulate requirements [8].
• Developing requirements that result in an end-to-end product [19].
• Developing requirements with enough detail
• Inspecting requirements [20].

The last three requirements are challenges associated with tasks high in cognitive effort because they involve the processing of many information cues which are characteristic of requirements engineering. This is especially true in systems that involve many user stories in which assuring that requirements are correct and complete involves processing large amounts of information.

Although a more effective customer understands the domain of the software, its business value, and is capable of making decisions about the prioritizations of user stories[8], many of these challenging tasks do not need to be performed by the customer role. For example, business analysts may be used to assist in the writing of user stories, and testers assist in validating the software [11].

In other words, the customer is uniquely qualified to perform some of the tasks he is responsible for while some of the work associated with this role may be cognitive overhead—cognitive effort that the customer does not necessarily need to exert. Given the time constraints and pressures faced by the customer, an understanding of these important but re-assignable tasks is of great importance to the Agile software development movement. We explore many of these tasks through a case study of the story master role—a role specifically designed to aid the customer in capturing, managing, and communicating user stories.

We present our findings as a set of insights gained as two different story masters filled the role on the project. Both story masters had no formal training or experience as systems analysts, programmers, or testers.

3. The Case Study

Our case study answers the question: How much cognitive effort does the customer have to perform in providing stories to the development team, that does not necessarily need to be performed by the customer? We answer this question by reviewing insights gained through the experiences of two individuals who have filled the role for 16 months. These insights are grouped according to the cognitive effort they address. We also address the challenges of transferring cognitive effort from the customer to another individual. We first introduce the background of the case study.

3.1 Case Study Background

3.1.1 People Involved. A group of stakeholders agreed to build a new rapid development environment for collaborative software applications called ActionCenters. Drawing on experience with earlier prototypes, they anticipated the project would require approximately 500,000 to 1,000,000 lines of new code. The team had a budget of $4,000,000 USD, which they anticipated would only be sufficient to build the core functionality of the proposed system. Further funding would depend heavily on the degree to which the organization could derive value from first round of development.

The ActionCenters project team consisted of a one customer representative, a system architect, four full-time senior developers, two part time developers, a UI designer, and the story master. Each professional team member teamed with an intern/mentee. Part way through the project, the testing team formed consisting of a test lead and two testers.

3.1.2 The Project. The customer had been involved in developing the requirements for an earlier full-scale prototype of the system. None of the prototype code could be used in the production version of the system because the team decided to implement the new system on a different technical platform. None of the
developers of the prototype system was involved in developing the production version.

Full formal specifications for the prototype system existed. The customer and the prototype development team had spent a year writing those specifications before coding began. Those specifications could not be reused, however, because they were too closely tied to the original development platform. The team found that it was not possible to reverse engineer the requirements and constraints from the detailed system specifications. The customer had a detailed understanding of what the capabilities and constraints of the system should be. Those requirements remained stable over the life of the project.

3.1.3 Selection of an Agile Software Development Methodology. At the beginning of the project, only the customer knew the system requirements. The customer had other duties in addition to the project, and so could not devote full time to communicating the requirements to the development team. The customer and development team therefore agreed to use an Agile development approach for the project hoping to eliminate the cognitive load associated with writing formal specifications. The customer also expected to take advantage of the flexibility of developing software iteratively. None of the team had previous experience with Agile methodologies.

The development team established work practices based on the Scrum Agile Software Development method (Highsmith and Cockburn 2001). The architect also served as the scrum master. The customer and the architect shared the duties of the product owner. The team agreed to develop software in four-week iterations. The customer, the scrum master, and story master would work together to decide which stories should be implemented in each iteration. Each iteration would begin with a one-to-two day planning meeting where the stakeholders would review what had been accomplished and learned in the previous iteration. Then developers would estimate the level of effort required for stories to be built in the next cycle. Developers made their estimates in terms of story points [9]. A story point roughly equated to a half a day of work for a single developer. During each iteration, developers held a daily 15 minute stand-up meeting to report what they had achieved the previous day, what they would achieve that day, and what barriers they faced. The customer was co-located at the university, along with the development team. However, he did not make himself available to the team unless it was for scheduled meetings which usually occurred between the systems architect, UI designer, and story master—the customer was constantly pressed with other duties.

3.1.4 Cognitive Overload of the Customer. The customer originally agreed to write the stories for the project. However, the customer had responsibilities outside the project, and therefore did not have enough time to write well framed stories during each sprint. This cognitive overload would remain evident throughout his involvement with the project. Four sources of cognitive effort are referenced later in the case study.

3.1.5 Creation of the Story Master Role. The team created a new role they called “story master” and hired a person to work with the customer to capture and document stories and enter them into an online story management system. The principle duty of the story master was to assure that the development team had stories to estimate at the beginning of each iteration. The story master therefore participated in design meetings with the customer and UI designer, and made notes about the design decisions and agreements they achieved. The story master worked with the customer and the scrum master to decide which stories should be executed in the next iteration. The story master wrote acceptance tests for each story the developers committed to build during a sprint. In the early months of the project, the story master typically spent 15-20 hours a week in the role.

3.2 Review of Insights into the Cognitive Effort of customer tasks

Four major cognitive efforts emerge as we reviewed field notes from both story masters. We discuss how the role evolved to overcome challenges associated with transferring this cognitive effort to another role. At the end of each section, we summarize the challenges and solutions associated with handling each cognitive effort.

1. Capturing stories in meetings
2. Managing the backlog
3. Coordinating development of stories with UI designer and Development team
4. Creating and running acceptance tests

3.2.1 Reducing the cognitive load of capturing user stories

How cognitive effort is reduced: Much of the critical elements of the software’s design and other decisions related to the software were captured during intense, free flowing meetings, in which the customer and other stakeholders would be fully engaged. Any attention shifted away from these engagements in order to focus
on something else, such as capturing a story, was potentially devastating to the goals of the meeting because it would inevitably squelch productivity. When the constant shift in attention would occur, the customer and stakeholders would walk away from the meetings feeling “mentally exhausted.” In terms of TBRs, such complaints are predictable as the work practice just described normally entails shifting back and forth from working memory, the details discussed at present, and long term memory, especially when elements from past discussions and projects needed to be recalled. A reduction in the number of complaints of “mental exhaustion” was noticed as the story master matured into its role, which was to be a team member who prevent the customer and stakeholder’s need for shifting attention during intense, free flowing discussions.

**Evolution of the Story Master Role:** Initially, there was the assumption that the story master would be capturing many more details relating to the software than is currently the practice. The story master attended almost every scrum meeting, weekly status meeting, in addition to design meetings between the customer and the UI designer and interviews with the customer. Meetings involving the customer as well as GUI design meetings were recorded with an audio recorder to be reviewed later, when needed. There seemed to be the expectation that the story master would automatically be able to recognize any detail as a user story. However, this expectation changed quickly for several reasons. First, it was not efficient for the story master to be present in all meetings. The amount of software-relevant details captured in these meetings varied greatly. In the initial interviews with the customer, 458 user stories were captured in two two-hour sessions. On the other hand, almost no details about the software showed up in scrum meetings or status meetings. Roughly a dozen new details would show up in initial six-hour-meetings with the UI designer. At the same time, the first story master found listening for these user stories to be very demanding. Stories could show up at any time and he felt like it was necessary to pay full attention in meetings while without knowing exactly what was supposed to be noted.

The need to improve understanding of how to focus in story master meetings continued with the second story master. After the role transition from the original Story Master to the new Story Master, the process was going smoothly in that expectations were becoming clearer for which meetings the story master should attend. However, it was still unclear as to how the Story Master was successfully contributing to the Agile process within the meetings. One of the Story Master’s challenges was knowing when to capture information that needed to be documented as stories without being a burden on the Customer and other team members. This was challenging because stories that needed to be captured were getting lost in discussions related to design issues, negotiations among the Customer, programmers, and the GUI, as well as deciphering programming language. During such important discussions, the customer, is in a state of cognitive overload, and pausing and restarting the discussions to capture stories only makes such a state worse.

A new work practice was implemented, meetings involved a portion of time when titles of stories would be written on the board whenever the Customer thought of them. During downtime, at the end of the meeting, or at a designated time the Customer and the Story Master would get together and turn the titles into formal stories. These titles were useful because the Story Master would use them to guide his listening for further details related to the titles and they would jog the memory of the Customer when it came down to developing the titles into formal stories. Moreover, they reduced the amount of cognitive effort needed by the Customer because he no longer needed to switch back and forth from capturing complete stories to playing a critical role in team discussions.

Developing an ear for capturing user stories involved learning what details to listen for during meetings and knowing when to interrupt a discussion to prevent from missing critical information. Before the implementation of the new work practice, the Story Master wasted a lot of effort capturing irrelevant information. Instead, what the Story Master needed to learn was to capture decisions that came out of the discussions. Also, it became the responsibility of the Story Master to interrupt the discussions when he thought that certain decisions needed to be captured or information was unclear. The team knew, in advance, to be prepared for this, and actually appreciated the extra clarification.

Finally, developing a formal structure for building stories streamlined the process for knowing what details to capture because it helped categorize them. This story structure consists of: A Location + User + An Action + Goal or Purpose + Effect. Location refers to where the story is occurring in the system, such as the Rich Text Editor, the user refers to the type of user such as the Administrator, an action, such as Right Click + Menu + Copy refers to what is being done by the user as a means to the goal or purpose, the Goal or Purpose refer to the user’s objective such as Copy [a line of text], and the Effect, such as Text Goes to the Copy Buffer is what happens when the previous four elements are satisfied.
Challenges:
- The Story Master idle in many meetings.
- It is difficult for story master to recognize details as stories.
- The Story Master not able to be present at all meetings.

Solutions:
- Develop an ear for capturing stories
- Write Titles of stories, fill in the details of the stories later.
- If the Story Master is not present, have the customer write down the stories and discuss them later with the Story Master
- Develop a structure for stories to assure that the correct details are captured.

3.2.2 Managing the Backlog for the Customer

How cognitive effort was reduced: Aside from prioritizing and clarification in planning meetings, the customer does not need to be involved in the maintenance of the backlog. The story master decreases cognitive effort of the customer and the team by de-cluttering and organizing stories, finding stories when needed, labeling the stories in a meaningful way, and moving the specific stories needed for the planning session to an established area. In addition, the story master removes stories that are obsolete stories and duplicated stories. All of these tasks involve processing information cues in order to understand the relationships between stories. This can amount to a time consuming and messy task. In this project, the backlog often held 300-400 items.

Evolution of the Story Master Role: The primary tasks of the story master associated with managing the backlog were to make sure that stories were understandable, to prevent duplicate stories from being in the backlog, and to ensure that stories were prioritized. This was accomplished by using feature groups and categorizing the stories under subprojects, and to delete obsolete stories. In addition, epics needed to be broken down into stories and dependencies needed to be created and checked for accuracy. Having someone primarily responsible for managing the backlog helped to reduce the duration of the sprint planning sessions because prior to these sessions, the Story Master would meet with the Customer and the Programming Architect to prioritize the stories and rate them on complexity, and to move them to the top of the backlog. This was also a check to make sure that all the stories needed to be estimated were available. This task could usually be done in one relatively short meeting. This whole process is most effective when a week or so prior, the Story Master meets with the Customer to get final approval for any new stories. These meetings were important even with the backlog well organized, since it was difficult to keep it clean of redundant or out-of-date stories resulting from changing thinking about how the system should be designed. In addition to organizing the backlog effectively, it was also important to make the user stories in the backlog memorable. Especially earlier in the project, team members would have a hard time recalling what a story meant, even if it was only just over an iteration old. Having a formal structure for building stories (mentioned in the section on reducing the cognitive effort of capturing stories) also helped with understanding a story’s meaning and mitigated the negative outcomes of memory decay over time because story details that were usually forgotten using the previous work practice were now being remembered. In addition to the previously mentioned benefits, individuals with little exposure to the system could more easily understand the meaning of a story captured using this structure, which was critical when bringing in new employees, such as the testers, later in the project.

In addition to meeting with the Customer and Programming Architect prior to each sprint planning session, the Story Master would meet with the UI Designer to obtain “wireframes,” or in other words, a basic, black and white draft or blueprint of specific features of the system that corresponded with specific stories that were going to be estimated, when applicable. Having the corresponding wireframe attached to the story immensely helped the team recall details when estimating because it added more meaning to the story description by providing a visual.

Challenges:
- Changing thinking about stories results in out-of-date and redundant stories.
- Memory decay makes some user stories unrecognizable.

Solutions:
- Have meetings a week before the end of the sprint to assure that enough user stories are prioritized and ready for the sprint planning meeting.
- Have a formal structure for user stories,
- Include wireframe diagrams of functionality represented in user stories.

3.2.3 Reducing the cognitive effort of communicating stories

How cognitive load of customer is reduced: The Story Master can reduce cognitive effort of communicating
user stories to the developers and UI designers. Several members of the team, especially the developers, need to know important details about upcoming user stories. Explaining and clarifying the details of these user stories requires that the customer think about the delivery of several information cues which results in higher cognitive effort.

Evolution of the Story Master Role: Initially stories were developed between the story master and customer, then taken to the UI designer, and then presented to the developers in the sprint planning meetings. Problems with this workflow were discovered as the functionality to develop became more complex. In the first iteration, development focused on the login screens for the tool. The functionality did not require much design work as they were typical to many software applications (e.g. login and user management). After iteration 2, however, as the team began moving towards designing and developing parts of the software that needed more design input, the UI designer requested that stories come out of his designs and discussions with the customer instead of designing the user interface to fit the story. Stories very often contained information about how user needs should be in addition to containing information about what the needs of the user were. These stories unnecessarily constrained the UI designer. Additionally, the story designer could not work independent of the customer since most of the time he required the customer’s knowledge of the user’s needs when developing the software. It became a rule that stories be developed after meetings with the UI designer. This was not a hard rule, however. Sometimes stories would be developed which did not have the UI designer’s blessing. Currently the UI designer generates many of the user stories from sessions with the customer.

The developers also had an interest in stories before they were submitted for sprint planning meetings as well for a couple reasons. First, UI design decisions were often a concern for the UI developer on the team. To save development time, the development team reused existing UI components built in EXTJS (Extended Javascript). The nature of the UI and the functionality could be altered to take advantage of the reuse opportunity. Secondly, the customer included inputs and outputs of the story. It became apparent that it would be useful to keep track of more details about them. To increase the likelihood that important details were included as often as possible, a format was developed which assured that the customer included inputs and outputs of the story. This format helped the customer think in enough detail. It also created expectations for how much the developers should expect in terms of details.

Challenges:
-User stories contain “What” information—simply communicating them to the UI designer doesn’t work. Customer must be consulted to get at the “why.”
-Systems architect needs some heads up on what stories will be developed next.
-Important details are sometimes omitted from communication of stories between developers and customer.

Solutions:
-Allow UI designer and developers to have input on how a story should be implemented.
-Allow a format that assures that important details are communicated.

3.2.4 Reducing the Customer’s Cognitive Effort of Creating Acceptance Tests

How cognitive load is reduced: Creating tests involves assuring that the details of how the story should be tested are included in acceptance tests. This involves processing and generating several information cues about the user stories details, until enough details have been made explicit as an acceptance test.

On a weekly basis, the Story Master and the Systems architect would meet for about an hour to discuss various issues such as test cases and an update on progress. When discussing test cases, sometimes the Story Master would need help testing stories related to the system’s architecture where observable effects were not as obvious from the user perspective. The progress update was mostly a checkup to see how things were going, and a time to discuss areas where more focus was needed and future plans for the software. This weekly meeting was helpful because the Story Master and Systems Architect did not often work
in the same area. In fact, the Story Master performed most of his work from home.

**Challenges:**
- Accepting the software requires consultation with the systems architect to understand how the software was implemented in some cases.

**Solutions:**
- The Story Master can meet with the systems architect regularly to determine the implementation of the user stories.

4. Analysis

In this section, we will analyze the success of the story master role using Conboy’s [1] framework for assessing the agility of an information systems development component (ISD). Conboy defines the agility of an ISD as:

“The continual readiness of an ISD method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality, and simplicity), through its collective components and relationships with its environment” (p. 340).

From this definition the framework is derived:

1. To be agile, an ISD method component *must* contribute to
   one or more of the following:
   (i) creation of change
   (ii) proaction in advance of change
   (iii) reaction to change
   (iv) learning from change
2. To be agile, an ISD method component *must* contribute to one
   or more of the following, and *must not* detract from
   any:
   (i) perceived economy
   (ii) perceived quality
   (iii) perceived simplicity
3. To be agile, an ISD method component *must* be continually
   ready i.e. minimal time and cost to prepare the
   component for use.

We will analyze each task performed by the story master in terms of the framework. Note that the analysis of the role is in its current state.

**Reducing the Cognitive Effort of Capturing Stories**

+ While the customer must still affirm user story titles with the story master following meetings, the story master facilitates the creation of change by allowing the customer more bandwidth to focus on design decisions during meetings.
+ The user story capturing process evolved in a way that conserved quality while at the same time increasing economy. Story titles caught the gist of user stories and could be noted by the UI designer in case of the story master’s absence. The quality of the stories was conserved because the customer could verify and flesh out the stories in an easy-to-remember format later.

**Reducing the Cognitive Effort of Managing the Backlog**

+ The customer did not have to manage the backlog, again allowing him to focus on story creation. The backlog was kept free of obsolete and duplicate stories so that the customer could quickly reference and negotiate user stories with developers in planning meetings. This contributed to the customer’s ability to prepare for opportunities to change presented in planning meetings.
+ The backlog also increased overall economy without sacrificing quality or simplicity, as the customer did not have to perform maintenance tasks.

**Reducing the Cognitive Effort of Communicating stories**

In the end, the user story did not act in this role. The backlog became the sole point of communication of user stories.

**Reducing the Cognitive Effort of Creating Acceptance Tests**

+ The customer was able to get a sense of how the software worked without needing to understand whether or not the minute details of the software worked.

**Transition from Original Story Master to Current**

In order to discuss the continual readiness of the story master role, we talk here about the second story master’s assumption of the role.

During iteration eight, a new Story Master was brought on the team to replace the original Story Master, who had been assigned different responsibilities within the organization. Far unlike the
original Story Master who had a background in information systems and was working on a Ph.D. in management information systems, the new story master had a background in psychology, and was a first year graduate student working on a master’s degree in industrial and organizational psychology. In addition, the new Story Master had absolutely no prior exposure to the Agile software development methodology. At the time of this transition, the role had hardly matured, and there was still uncertainty as to how the team would benefit the most from the Story Master role.

The immediate responsibilities assigned to the Story Master were: maintenance of the backlog, capturing of stories, and creation of test cases. To become familiar with these responsibilities, and as a form of training, the new Story Master shadowed the Original Story Master for one iteration, and then they switched positions during the subsequent iteration. In other words, the new Story Master took an active lead while being closely monitored by the original Story Master. This training procedure was successful as indicated by the fact that the new Story Master felt comfortable in the position and the other team members were pleased with his performance. Evaluation of performance, however, was mostly based on the new Story Master’s work ethic and how well he adapted to the project, and not on the betterment of the Agile process. The lack of focus on the latter was likely due to lack of understanding and maturation of the story master role.

In short, this may have been a weakness of the role, especially at first when it was still evolving. It took both story masters several iterations before they could recognize stories to capture and to be able to create adequate stories. Several aspects of the story master process have improved this however. The template for stories mentioned in the story capturing section helped with this, in addition to the practice of capturing story titles.

**Overall Assessment of Agility**

The story master role contributes to change mainly by facilitating the customer’s ability to create, prepare for, and react to it. It also contributes to quality without compromising economy. While we do not have quantitative evidence of the comparative economic cost or benefit of the story master role, it is highly likely that the role is more economical than a customer role or many of the other possible roles that would replace it. Examples of such roles could include the product owner, a business analyst, a developer, or the scrum master. Each of these roles has a wider skill set than the story master. As we have shown, the story master does not require extensive educational or domain training to fill.

**5. Discussion and Conclusion**

We presented the story master role to illustrate aspects of the customer role of Agile software development methods that may be considered cognitive overhead—cognitive effort that the customer does not need to perform. While the story master was successful in reducing the cognitive effort of capturing stories, maintaining the backlog, and performing acceptance tests, the development of the role to fulfill these tasks did not come without challenges. Many of these challenges have been addressed through the evolution of the work practices of the role. However, some of the difficulties of the role can only be overcome through experience.

We evaluated the story master role in terms of its agility as an Information Systems development component and found that it contributes to the development of change in a way that is more economical than having the customer perform these roles alone and does not detract from the quality of the work performed by the customer. The story master is most applicable to projects with substantially complex requirements. When there are fewer requirements, there may not be enough work for the story master to perform. Additionally, this project was not subject to changing requirements from external factors. The user stories changed frequently due to new insights gained about how the software should be design. Our case study is also unique in that the story master role supported a customer who had a knowledge of all the requirements.

While we were able to assess the value of the Story Master as an Agile Software Development method, we emphasize that this is exploratory research. The main contribution of this study was to identify the existence of a phenomena in a real life context: transferrable cognitive effort. Future research should seek to validate the decrease in cognitive effort by the customer through a set of studies that provides for a baseline comparison of the customer’s cognitive effort with and without the role. Future research should also explore the effectiveness of the story master role in other settings to see if our findings are replicated as well as whether or not the story master role is constrained by other factors than project complexity.

**6. References**


