March 2022

PAD Beyond the Classroom: Integrating PAD in the Scrum Workplace

Jade S. Weiss
University of South Florida

PAD Beyond the Classroom: Integrating PAD in the Scrum Workplace

by

Jade S. Weiss

A thesis submitted in partial fulfillment
Of the requirements for the degree of
Master of Rhetoric and Composition
Department of English
College of Humanities
University of South Florida

Date of Approval
March 9, 2022

Keywords: Agile, Project Management, Information Technology, Software Development, Technical and Professional Communication

DEDICATION

This thesis is dedicated to my loved ones who encouraged me to take a break from pursuing this degree. Specifically, my sibling Jas, my best friends Rebecca, and my partner Nick. With their promise of love no matter what, I was able to step away and with that time, learned to do two things I previously thought impossible: remain motivated to do academic work without self-loathing and genuinely understand that I have value as a person beyond goals I set for myself decades ago. I'm not sure where I would be if it weren't for my friends reminding me that breaks were possible and that I deserved them.

I also dedicate this thesis to the allies and incredible folks that inspired me to write on these topics and welcomed me back after my break without batting an eye. Thank you to Jessica Griffith and Emile Deville for making graduate school better, funnier, and friendlier. Thank you to Dr. Lisa Melonçon for helping me see this through and for your thorough, concise insight to the tech comm field, and of course for PAD. It has become my Swiss army knife and will remain that way far into the future. Thank you to Marissa Klippenstein and Dr. Jessica Cook for being administrative heroes, I could not have done it without your help.

Lastly, this thesis is dedicated to me. I overcame unspeakable curveballs to arrive at this point. I am a proficient and successful technical writer and I have grown exponentially past that narrow role. But I am thankful to finally close this chapter officially, with the correct rites and rituals.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Purpose: The "story" format used in Scrum ticket writing is confusing to developers and leads to insufficient ticket content, which lends to miscommunication between team members and administrators, and disrupts workflow from the bottom up. A burgeoning methodology in Technical Writing, Purpose, Audience, Design (PAD) is an alternative ticket format that is easier to teach developers and improves the aforementioned conditions than the existing "story" format. The goal of this paper is to lay out why and how PAD can benefit developers on smaller Scrum teams who are tasked with writing their own tickets. This paper does not offer solutions for larger Scrum teams who operate with a designated Scrum Master.

Method: This paper begins with an introduction to the arguments being made, a section defining terms used in the paper, and basic background information to set the stage. I then review the classic "story" ticket format in detail and establish what the problem is by providing workplace experience as examples of where the "story" format fails in small software development teams. This thesis also includes a review of existing literature on PAD and rhetorical methodology, how PAD can work as a remedy in the workplace, and examples of this working successfully in 3 different software development teams. After, the paper offers an implementation chapter to discuss the basics of how PAD can be automated as a format. This thesis closes with a discussion section that touches on future research suggestions and additional thoughts concerning the continuation and potential application of this research. This paper is a departure from other academic works on the subject because it centers my real workplace experience as a valid alternative source of knowledge in the absence of robust formal research.

Results: The paper posits that PAD is a beneficial ticket format alternative to offer to frustrated developers who are tasked with writing their own tickets. It successfully emphasizes that this isn't a one-size-fits-all solution to all Scrum workplace communication issues but rather a reasonable solution to offer developers, one that there is no reason not to at least try out—especially in small to mid-size tech companies. In addition to this, the thesis offers additional suggestions to those interested in continuing the work and supports the current work of PAD being brought to more programs seeking to produce competent technical writers.

Conclusion: Improving the rules used to write tickets and approach project planning betters communication, speeds up workflow, and eases tension between developers and administrators. Administrators are more satisfied with ticketing that clearly lays out the information they require for project management and developers are happy with a simplified process for something they're tasked with regularly. PAD should be taught in technical writing programs, incorporated in the workplace, and featured as a standard aspect of on-the-job training. In particular, PAD should be offered to Scrum practitioners frustrated with their ticket writing format and has many potential applications beyond this paper

CHAPTER ONE

Introduction

The “story” format used in Scrum ticket writing is confusing to developers and leads to insufficient ticket content, which lends to miscommunication, disrupts workflow from the bottom up, and causes undue tension between team members and administrators. A burgeoning tool in Rhetoric and Technical Writing offers a flexible, highly useful alternative that both remedies a problem plaguing small to mid-sized software development teams and invites further conversation regarding its ability to reduce the gap between the workplace and academic spaces. The purpose of this thesis is to leverage my experience as a technical writer and lay out why and how this tool can benefit developers on smaller software development Scrum teams who are tasked with writing their own tickets and seeks to add to the small but ongoing conversation about the tool’s additional applications.

This thesis does not offer solutions for larger Scrum teams who operate with a designated Scrum Master or ticket handler. This thesis specifically focuses on small to mid-sized development teams of about ten to fifteen developers because that is the environment and team size I have worked with since I entered the industry. Since the tool – PAD: Purpose, Audience, Design, is only in its beginning stages of formal discussion and research, I offer my seven years cumulative (and counting) experience working in software development and my background in rhetoric and technical writing as a valid source of knowledge to formally begin the conversation about practical application. This thesis also does not position itself as a one-size-fits-all solution and acknowledges that this work is far from being over and seeks to emphasize that the lack of discussion intensifies motivation to write. It is also worth noting that this thesis does not discuss Kanban or other similar methodologies found in software development because I have solely worked with Scrum.

This thesis is a departure from other academic works on the subject because it centers my real workplace experience as a valid, alternative source of knowledge in the absence of robust formal research. This work also highlights my ability to offer knowledge of great value back to my academic community, which may offer one strategy to bridge the gap between the expectation of technical writing students, and the actual demands on professionals in the field. Throughout my coursework, there has always been an ongoing conversation about addressing the gap between the workplace and academic spaces. If the point of getting a degree is to acquire a job after or during school, then the curriculum must stay aligned with current workplace values. I have noticed that this gap is something many programs strive to close. I feel a sense of responsibility to share analysis and argument on this topic informed by my own experiences, as PAD has lent heavily to my efficacy as a technical writer and may very well be one of the simplest ways to offer something universally helpful for curriculum designed to produce employable graduates.

After this brief introduction, I will define necessary terms and then provide basic background information on software development strategies. In Chapter Two, I review the classic “story” ticket format, which is a basic tenet of one of the most dominant and widely employed software development strategies, Agile, which will be defined prior. Then, by drawing on my seven plus years in the industry, I discuss why the “story” format prevents many teams from timely and efficient product development. Essentially, I establish where the problem lies in the “story” ticketing format and its effects on small software development teams. Chapter Three is where the main argument of this thesis is made. I review existing literature on PAD, how it can work as a remedy in the workplace, and examples of this working successfully in three different software development teams. Afterwards in Chapter Four I provide two methods and related information to what implementation of the PAD format looks like in Jira and in real practice. Finally in Chapter Five, I offer additional discussion points and new, potential areas of research regarding PAD to continue supporting the burgeoning conversations about the tool by offering more of the numerous ways it has aided me as a technical writer.

Defining Terms

This section provides relevant definitions and basic information on software development and how these terms will be used in this thesis. I will use quotations when I introduce a term formally for the first time, but I will not be applying this punctuation going forward. This section also does not define PAD as that is discussed at length in Chapter Three.

This thesis focuses on the software development industry. I focus on software development because this industry primarily uses Scrum methodologies to create, or “develop,” products. On the IBM website, “software development” refers to the industry focused on developing or maintaining software (What is Software Development, IBM). For the purposes of this thesis, I focus on small to midsize software development teams. I define “small to midsize teams” as teams in software development who are smaller than 15 people *who are charged with writing their own tickets and adhere to Agile methodology.* Typically, teams with more than 15 members have an employee often called a Scrum Master, whose sole job is to write and maintain tickets for the entire team.

To understand the ticketing process, Agile, and the role of Scrum, one must also understand the basic framework of how software is developed. “Software” is defined as the programs, applications, or operating information used by a computer or mobile device (Ausgustyn). Examples of software include Zoom, Discord, etc. Most teams, and what I focus on specifically in the scope of this thesis, use Agile to develop software. The development of Agile will be described in the next section, but for now, I define "Agile" as an iterative strategy of software development. The goal of Agile is to deliver the simplest, most effective products to meet basic user needs in the least amount of time possible. It's important to note that Agile is an *iterative* development philosophy, meaning products are pushed out in small batches, also known as "releases," with the intention of improving and expanding upon those products in further development blockings.

Agile's philosophy, which will be reviewed in the following section, manifests in software development workplaces as "Scrum." Scrum, as defined by Atlassian the industry leader in Agile philosophies, "is a process framework used to manage product development and other knowledge work" (Agile Project Management: Best Practices and Methodologies). Scrum is empirical in that it provides a means for teams to establish a hypothesis of how they think something works, try it out, reflect on the experience, and make the appropriate adjustments. That is, when the framework is used properly. Scrum is structured in a way that allows teams to incorporate practices from other frameworks where they make sense for the team's context. Simply put, Scrum is the framework that helps teams tasked with delivering complex products cooperate together (Drumond). So, if Agile is the "what" of software development, Scrum can be understood as the "how." Agile and Scrum have many different, sometimes canned, ways to accomplish this "how," and usually rely on the Scrum Master to steer the team.

Scrum Masters are those who help enforce the scrum values or frameworks the team has agreed to commit to (What Is a Scrum Master?). Scrum Masters are to remain flexible, but firm on standards and focus on improving workflow across the team. They are typically tasked with enforcing ticket standards and even writing or managing the bulk of the tickets, in addition to facilitating planning meetings and ensuring that every team member understands the product needs and has the resources and time to execute those needs.

In larger or cross-functional teams, Scrum Masters are often in charge of tickets. “Tickets” also known as “issues” are defined and best described as a to-do item, either a la carte, meaning separate from a larger project, or as an important part of the main project. These tickets and how their format can impact workflow and team efficacy is a focus of this thesis.

When there are no Scrum Masters involved with planning, ticketing, and product development, a team can suffer. In smaller teams, the “technical writers” or “tech writers,” or the individuals tasked with creating and maintaining documentation, and enforcing documentation standards in a workplace, may be forced to take on quasi-Scrum Master roles, in addition to their own duties when there is not one present. This is another motivator of this thesis.

The people who use Agile and Scrum to develop software are called developers. When I discuss “developers,” sometimes referenced as “devs,” I specifically mean those who work in software development who are tasked with developing or creating, maintaining, and fixing a company’s software.

One of the main tools developers use to organize and plan software development is through Jira. “Jira” is a platform that aids in facilitating Scrum for software development teams (What is Jira Used For?). Jira is where development teams create their tickets, project plans, and time management. It is a project planning tool developed by Atlassian which positions itself comfortably as the industry standard of Scrum tools. It is important to note here that Jira is an incredibly expensive software and is not easily accessible to academic researchers which one of the reasons why it's remained mainly discussed in the IT industry and not in academic spaces.

A "sprint," also known in action as "sprinting," refers to the time negotiated and agreed to by the development team in which they are required to deliver on the tickets they've committed to. A typical sprint is two weeks— meaning in two weeks' time the devs must complete the tickets they were assigned. If the devs are unable to complete their assigned tickets, they must communicate to their Scrum Master or manager that they have a block and need to move the ticket into the next sprint or into the "backlog." The backlog is a place on the Jira platform, basically a de-prioritized "To-Do" list where tickets are dormant until called upon for a sprint. If a developer finished their assigned sprint tickets early, they may also pull from the backlog to get a jump on new tickets. In this way, Agile allows for some flexibility in the development process via the backlog.

In Jira, tickets in a sprint sit on a "board" made up of columns, also known as "swimlanes". The board is the page where tickets are organized and where devs and admins can view what tickets were committed to and haven't been started, which ones are actively being worked on, which ones are completed, and tickets that are blocked. Devs and admins can click on tickets to view more information and relevant comments from those working on them. The tickets are organized in a certain way to make them easily understandable and accessible to anyone who reads them. When I say, "classic ticket format," "user story," or "story format" I am referring to the traditionally used format in Agile ticket writing (Rehkopf) which reads, "As a [persona], I [want to], [so that]." The personas are used to represent the voice of the user. For example, "As an accountant, I want to be able to insert my client's data into the software, so that I can file income taxes."

In the software development world, the user is one of the most important considerations when developing products. A “user” or a “customer” is the individual who uses the product—and ultimately makes the team money. Putting it all together, developers use the Agile framework, via the Jira platform, to make software to sell to users.

The last term that is worth defining in this introduction is “administrators.” Administrators are consistently present in software development. They are the managers and other folks that oversee what the devs are doing. Sometimes referred to as “admins,” administrators typically want to know exactly what a dev is doing and how they are spending their time to ensure deadlines and user needs are met. Administrators sometimes interface directly with the user base of a product and work with a Scrum Master to ensure the team can deliver. Admins are normally a source of pressure on a software development team as satisfying their needs means developers are doing their jobs correctly.

The following section will more deeply explore the background of software development, and why Agile is the most dominant framework used in software development.

Background

There have been many methods developed to better meet the demands and needs of the market, development team, and user. The Waterfall method was one of the first, and most widely used, methods of software development. At its core, the Waterfall method is a way to break down projects into phases (Hughey). Progress therefore falls “downwards” as each phase of the project depends on the success of the previous phase See Figure 1 below.

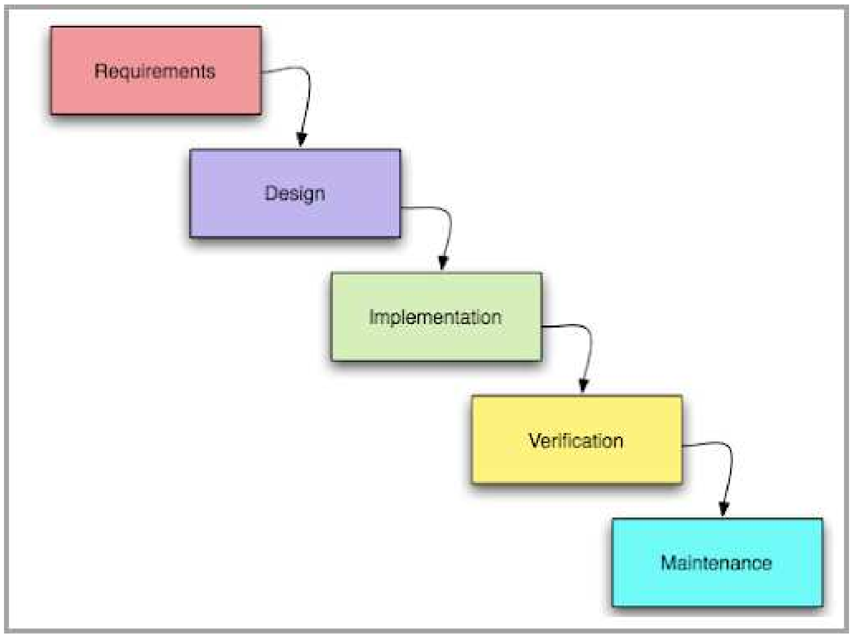


Figure 1. Waterfall Method

However, as technology, software, and user demands changed, the Waterfall method became more and more obsolete. Waterfall, though sufficient for one-off products, was not capable of adapting to long-term projects. For example, a team may develop a single video game, but as updates and expansions are demanded by the market and users, there is no set way to modify the product once its original development is complete. Meaning, Waterfall is not an iterative process, and leaves little room for creating multiple versions or updated releases of a product. It also lent its hand in chaotic ticketing practices and overwhelming task backlogs. With no standard procedure in place to revisit and improve upon concepts effectively as a team, developers are essentially expected to get paid to drown. It was a thrilling time in software development but arguably the most stressful.

From the failures of Waterfall, Agile, and eventually Scrum was developed to meet the needs of a rapidly accelerating market. Unlike waterfall, Scrum is iterative, supports multiple versions and releases of a product, and can adapt better to a user-centric software strategy while still giving ownership and control over the workflow, development, and pacing to the developers. Methodologies like Scrum and the workflow maintained by its standards and structures (like ticket formatting, sprinting, etc.) make it possible to overcome industry challenges and satisfy users' changing needs, if implemented well for any given software development team. As provided for the waterfall method, see Figure 2 below for Atlassian's visual representation of Scrum's iterative workflow. To remain focused in this thesis, I focus on the ticket writing and management practices which are included in the Sprint Planning section of the Scrum chart as seen below.

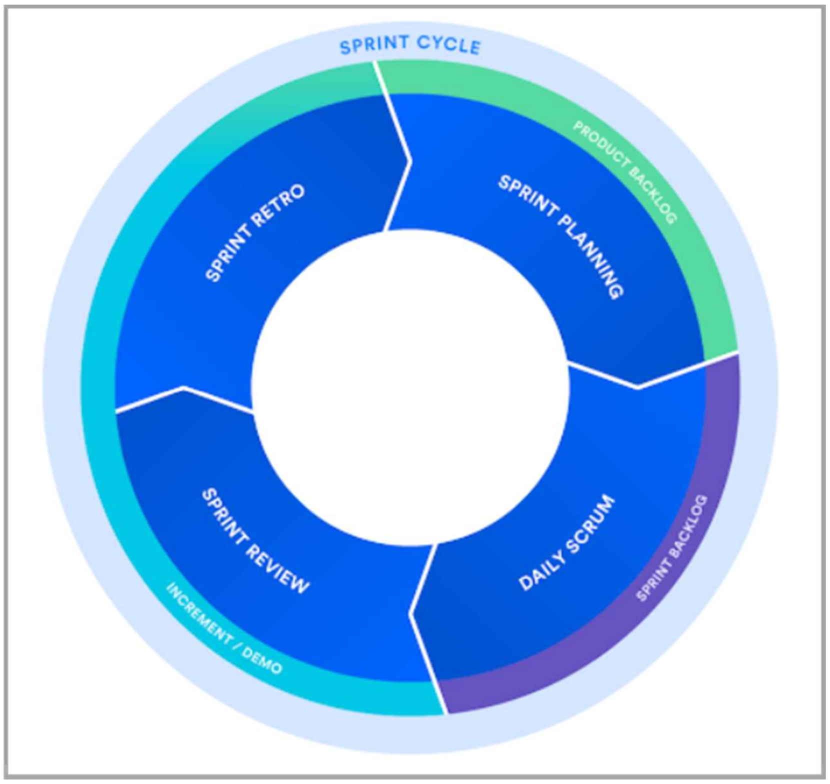


Figure 2. Scrum Chart

To provide additional and arguably entertaining background on Scrum's commitment to standardize iterative work while remaining flexible, recall Agile philosophies mentioned in the defining terms section. The original Agile manifesto was drafted up by seventeen developers working in software at the time who had reached their wits end with the industry's inability to adapt to the accelerating market. Dubbed the "Snowbird 17" because they pursued this work at a ski lodge, they launched an open-source site February 2001 that contained the agreed upon philosophies and their signatures vouching for their shared commitment to generating a shared mindset amongst chaos. This site is still up and running at agilemanifesto.org which is where the original documents live. The site remains heavily referenced by software devs even now. The group admits it was difficult to come to agreement, but they were rewarded with a philosophy that helped guide software development into a new, more effective era. Agile centers around twelve shared principals (Highsmith):

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcoming changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

There are clear connections between Agile’s core principles and how Scrum manifests in the workplace. For the purposes of this paper, I will discuss the principles that matter most to this thesis. See principal one, “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.” The consequences I discuss in this paper directly impact a team’s ability to uphold this principle. Ineffective ticketing practices impact workflow immediately. Note principals four, five, nine, and ten. Related directly to this thesis and confirmed through my experiences in software development, administrators and devs are truly expected to work closely together on a daily basis. Further, it is the lack of attention to technical excellence and design at the ticket level, *where all the work begins,* paired with poor ticketing standards that inhibits an admin's ability to understand their developer’s progress, therefore making it difficult to trust their devs to get the work done. This results in breaking the tenth principle because it leads to extra meetings and discussions. There are more ways to discuss how these principles inform Scrum, but to remain on topic, Scrum and the Agile philosophies it strives to uphold, help teams survive and produce quality work. I say this to stress that Scrum is going to remain an industry standard and therefore, efforts related to improving upon its practices has value to the larger development landscape but also immediately in my own workplaces.

Also important to emphasize in ongoing development of software methodologies is the increasing emphasis on user demands and input on products. Recall principal one above, “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.” When considering the history of software development, the focus is typically on the perspective of the user. The progress is measured by how many new products they have experienced in their lifetime, the benefits it has awarded them, commonly discussing learning curves and related challenges to the user. It is only natural that in software development culture, teams look towards the user, constantly thinking about what users need, what problems they might face, and what keeps them excited to continue engaging with a product. It is this user-centric mentality that made simpler workflows like the Waterfall Method (Figure 2) obsolete because they could not keep up with the urgent demand to satisfy customer needs and solve any technical issue they may experience.

When I entered the industry in 2016, I was not privy to the way this focus on users manifested in software development. I began as an IT associate at a service desk, sometimes referred to humorously as a, “service desker” for several years, then shifted to technical writing roles within various departments focused on software development and support. There was an overlap with these roles while I was completing my undergraduate and graduate degrees in rhetoric and technical writing which made it impossible to ignore the impacts and effects that constantly interfacing with user feedback had on workflow and communication. As an employee who values communicative working environments because they reduce tension, this overlap also made it difficult to ignore where I could make changes to resolve these conflicts. This is also what I believe makes this thesis valuable to the academic community: I offer a feasible, low-risk solution to the workplace, which could be leveraged as a benefit to current curriculum and teaching programs.

From the moment I began working in software development, I noticed that developers struggle when their ticketing and project planning is messy, ill-communicated, or highly pressured. The challenges and problems they solve are complex, quickly changing, and risky which is what makes ticket writing important but also highly time consuming for developer roles. At its worst, the wrong move or miscommunication can bring down an entire system which administrators work very hard to avoid. Effective ticket writing reduces the chances of mistakes because there is little to no question about what the developer is achieving. With this, it is less likely for someone to disrupt a developer in a delicate environment or miscommunicate about what servers are being fiddled with during that ticket. It is also less likely for administrators to force too many priorities at once because developers can refer to the tickets they've committed to at the start of the sprint as their limit. This again highlights why this thesis has value to the workplace. Older coworkers and mentors have talked a lot about Waterfall Method woes and emphasized that these problems became more and more evident as customer feedback was increasingly prioritized. They often bring up how thankful they are for Scrum providing a foundation to improve upon and a common language which simplifies communication for all parties involved. These histories and conversations, my background, and real opportunities to offer solutions to my teams ultimately led me to write this thesis.

This thesis acknowledges and supports the idea that Scrum, although a vast improvement from the Waterfall Method, can still be a source of issues in development, specifically in small to mid-sized companies. In the spirit of Agile philosophies, those who practice Scrum diligently understand that the structure needs adaptation in different circumstances. When Scrum is adhered to militantly without critical consideration, it results in poor communication, disrupted workflow, and a tense work culture which are the problems this thesis seeks to remedy. To further support Scrum's position as an industry standard that has value improving upon, many software development teams have transitioned to working from home due to COVID-19 and have found themselves adopting Scrum practices in order to adapt to the changing workplaces.

Every tool matters when it comes to software development. Specifically, having options to lean on in different situations and work environments. If Scrum acknowledges that development work is ever-changing, then a variety of different tools or formats to utilize depending on the situation is crucial. Jira embodies this well by providing endless customization and the ability to configure tickets, reporting, measuring, down to the field and drop-down options. Scrum Masters are well versed in these options and know how to traverse the endless freedom. Developers functioning without a Scrum Master or bare minimum ticket handler struggle with this. Many of them stick to and enforce the basic, classic templates offered which leads to those aforementioned consequences: poor communication, disrupted workflow, and increased workplace tension. Without easy format alternatives to offer developers working without the right resources, they struggle to find ways to adapt to their changing environment and overcome the complications of poor ticketing. This thesis positions PAD as just another alternative to offer developers as a solution.

Both as a student and during my seven years in software development, I have utilized PAD in the workplace to improve ineffective ticketing— something I've observed as a consistent obstacle for small software development teams. Through these intersections I see great value in beginning a formal argument that PAD offers a succinct, more understandable alternative to classic story writing. In my own experience, the results of introducing and using PAD are clear:

improved communication between developers and overseeing administrators, smoother workflows, saved time and effort via providing required information in a preferable format, and subsequently reduced tensions when working on delivering a product. In Chapter Two, I provide a detailed review of classic story writing to establish the main problem this thesis seeks to address.

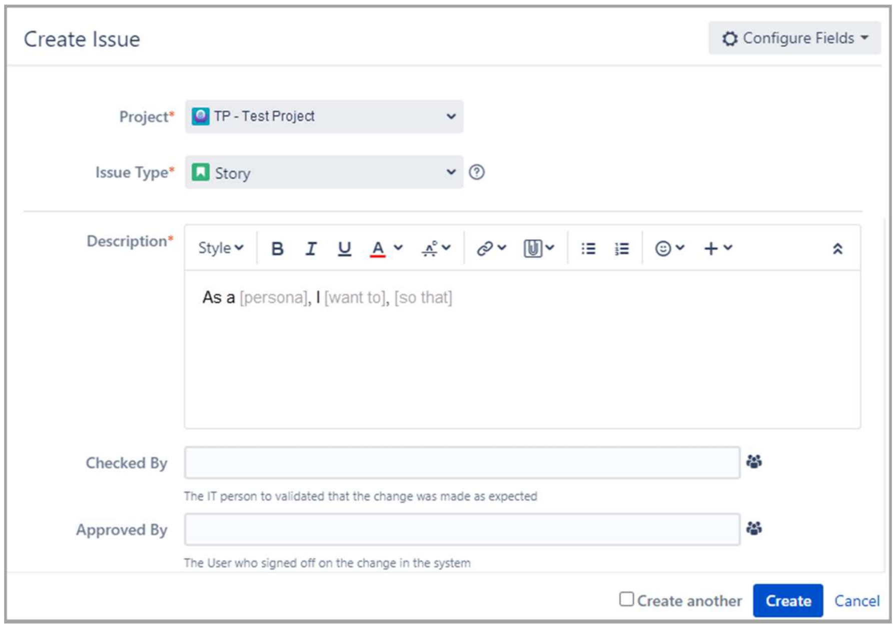
CHAPTER TWO: REVIEW OF CLASSIC TICKET FORMAT

In Chapter Two, I provide a detailed review of standard ticket practices, the classic ticket format, and what that looks like in Jira. Then, I discuss how this format can fail in three different software development workplaces. As previously touched on in Chapter One, tickets are also known as stories or issues. This section is important because it lays out the specifics of the ticket format and its observed failures to effectively discuss how PAD can work as a solution to this format's shortcomings later in the thesis.

Jira is the leading Scrum platform in software development at this time, so this chapter is entirely based on ticketing in Jira. It is relevant to note that there are other platforms who use very similar formats, but their user interfaces and outcomes may vary. Discussing these would go beyond the scope of this thesis. To understand how the classic story format works and why it fails, it helps to understand how developers and administrators use tickets and how tickets determine workflow in an ideal Scrum workplace.

Devs and admins use Jira as a project management tool. Jira is mainly used to organize big projects and break those projects into smaller, bite-sized tasks, or tickets. Unrefined project ideas are fed to a Scrum Master from admins and whoever they serve, whether that be users, internal clients, or other departments at a company. Admins communicate to the ticket handler how much time and effort should be allocated to the project based on workplace resources (i.e., time, staffing, ability…). From there, a Scrum Master breaks down the project parameters and expectations into tickets. Afterwards, developers meet with the Scrum Master to determine the number of hours, sometimes known as "points," that will be assigned to each ticket. What this means is that devs make an estimate about how many hours it will take them to complete each ticket. Devs work closely with the Scrum Master to get the right number of points to keep pace with administrator expectations during a given sprint. Essentially, the project corresponds to user needs and desires. Then, the administrators set priorities so Scrum Masters can work within those parameters to scope the project and communicate to developers via ticket writing and estimating how long it will take to deliver on those expectations. Even more simply, workflow is determined by how projects break down into tickets and how well those tickets are completed.

Tickets are virtual notecards full of useful information and accessed on Jira. When administrators set up their Jira configuration, they are given many customizable field options when determining ticket format. There are countless configurations because of this capability but most often, the standard structure looks a lot like Figure 3, a ticket mockup



Create Issue

Project*: TP - Test Project

Issue Type*: Story

Description*: As a [persona], I [want to], [so that]

Checked By:
The IT person to validated that the change was made as expected

Approved By:
The User who signed off on the change in the system

Figure 3. Story Format Ticket

A ticket's basic components are the project key, issue type, description, and a few approval fields. This mockup is based on what a dev or team member sees when they create a new ticket in Jira.

To align a story with a project’s demands, users select the project key from the drop-down and enter in the required information. When the classic story format is used, developers are invited to enter in text where it is grayed out. This deletes the gray text and replaces it with their entries. When the dev or team member is satisfied and clicks Create, the ticket is generated. A generated ticket is then assigned to the correct project with an identifying number and placed in the backlog. From there, Scrum Masters can assign tickets to certain developers when planning a sprint. Once a ticket is assigned, the ticket moves out of the backlog and onto a board where it can be actively worked on and tracked by the team, devs, and admin during the course of the sprint.

The story format asks developers to insert a persona, what that persona wants, and what it achieves for them. What I have observed on small to mid-sized teams operating without a Scrum Master is that the concept of a persona is not formally taught in any capacity to the developers. This leaves them without guidance on what a persona should represent at their place of work and most always leads to confusing tickets.

The story format invites devs to consider their audience as a persona to “get into their shoes” and imagine it from the persona’s perspective, recall the previous accountant example from chapter one. A team that utilizes Scrum effectively would have a document outlining the different personas developers will encounter in their work, but without this key document, devs have little clue on how to use this format. A quality story format sentence would be: “As a junior staff member in accounting, I want to be able to send bulk emails from my work account, so that I can contact larger departments more easily.” or “As a new employee, I want the right permissions and passwords, so that I can access my accounts.” A Scrum Master would normally write that sentence based on the unrefined project ideas discussed with administrators and then add additional information with the developers to flesh it out. In the absence of this role, an ideal workflow becomes murky and burdensome. Developers are asked to write the tickets from scratch not only without proper training but based entirely on unrefined project goals and missives. The additional task of writing tickets becomes a burden on already stressed developers, and can cause tension and miscommunications between devs and admins.

Additionally, *completed* tickets with time estimates and related project information dictate where and how developers spend their time. The longer ticket creating takes, which in many cases already falls outside of a developer's scope, delays the number of ticketed tasks that can get done. Ticket hours are usually measured as hours in a day. This means devs are capable of and being paid for approximately eight hours of work each day. Some tickets may only need one or two hours, others may need more than a day's work.

Project planning and ticket writing take a considerable amount of time on small teams where there is no filter between unrefined administrator ideas and the developers. On teams with little to no formal training on Scrum formats, I have witnessed developers take more than a day to properly scope out projects which always leads to them needing to move tickets over to the next sprint. This ultimately disrupts a workflow much like a rock dropping into a pool of still water, tickets moved into the next sprint will bump more tickets into the next and so on. It causes endless disruption and results in wasted or misallocated time. I have observed the story format failing in three different workplaces and found the consequences of each failing to be the same. Each was a small to mid-sized development team utilizing Scrum but, in some capacity, not working with a Scrum Master or ticket handler and deferring the task to the devs. In the following section I discuss these workplace examples to describe the failures of the story format in these small to mid-sized teams. I will be generalizing my experiences to protect proprietary information as required by my numerous non-disclosure agreements (NDAs).

Team A was one of my first IT roles. It was a small service desk team tasked with supporting students and staff in a university setting. At the time I was an undergraduate in the Professional Writing, Rhetoric, and Technology (PWRT) program. I was in a service desk role, meaning I wrote my own tickets to handle clients. I worked there for a year without using Scrum methodologies before the team moved towards integrating Scrum. We had very loose ticketing standards prior to this transition, but the team worked happily with the formats they did have. Team A had a culture of communication that served effectively in the absence of standardization. In total, there was only one meeting of light training prior to implementing the transition. Very little time was spent talking about the new story ticket format. Developers immediately felt threatened by this and vocalized that story format was uncomfortable and inappropriate to their existing workflow and practices. There were many meetings called to discuss personas, but administrators were unwilling to offer help when creating them. The manager, struggling to adapt to the new role of Scrum Master, was bogged down with much more documentation tasks than was originally expected of the position and was unable to offer tangible support to the developers.

The team experienced heightened tensions due to this transition in general, but especially in adapting to the new persona-oriented ticketing format. The team vocalized the feeling that time was being wasted and communication was degrading across team members and administration. For my part, I adjusted to the formats easier because personas and audience consideration were not new concepts to me due to my curriculum experiences in PWRT. But I agreed with the developers: the story format was alien to them and without appropriate support, they felt over-pressured and frustrated.

Frequent meetings were called to discuss confusing stories that admins or devs did not understand. The new Scrum Master stated often that writing all the tickets and being asked to keep pace with administrator priorities was unrealistic. Outside of meetings, the Scrum Master and developers felt that time was being wasted writing stories and that the way they were writing them was leading to unnecessary communication. Administrators were unwilling to let the team revert to old ticket practices, so managers were made to enforce the standards. The result of improper training and the new ticketing formats inappropriate for Team A’s needs led to tension and interrupted workflow. The story format simply failed to meet anyone’s needs.

Team B was later in my career and took place at a start-up company of about 50 people where the software development team was less than a quarter of the whole company. Team B had a Scrum Master for some time but decided to remove the role. At first, administrators blamed developers for not being able to complete a task they deemed simple and quick. But admins soon realized that they severely underestimated the number of working hours the previous Scrum Master dedicated to ticketing. Administrators were slow to recognize that a highly efficient Scrum Master, who handled all ticketing tasks, was essential to maintaining the quality of development.

Developers, in the meantime, struggled to keep up with the heavy ticket load. In comparison to team A, team B’s administrators required lengthy, detailed tickets with many additional fields on top of a description. It was a fast-paced environment that required a lot of communication and a smooth workflow to keep their sites running. Over time, more and more frequent “alignment” meetings were called in which administrators sat devs down and discussed their tickets in an effort to understand what they were communicating since the stories alone were not understandable. Developers were reluctant to discuss this during any of these meetings. The dev team resented the loss of their Scrum Master and increased workload. However, outside of these formal meetings with administrators, developers frequently identified that the story format was failing to meet, or even represent, their needs. The problem was this: the rigid ticketing standards were failing to communicate the requirements of the task and administrators failed to be flexible or give proper training to improve standards. Without a Scrum Master, this enforced ticketing format caused wasted time spent on writing, far more miscommunication than necessary, and increased workplace tensions.

Team C is one of my more recent technical writing contracts. This was a notably more corporate experience than team A or B, meaning the body of internal clients this software development team served was large and historically situated. The team itself was still small and was made up of about 15 developers and a manager. Just before I joined this team, they made the transition to Scrum. I took on the role hoping to be in an efficient Agile environment, but I quickly recognized a team struggling to adjust. This corporate workplace, like many others, hyper-focused on the needs of administrators and higher-ups. Every task seemed to be prioritized the same, leaving little wiggle room to appropriately time manage with the new ticketing standards. This team experienced a harder transition than team A because they weren't even using tickets at all before. Each had their own unique workflow that was comfortable for them. Many of my coworkers balked at Scrum and moved through the motions just because they were required to. Story writing was often incomplete, or half filled out with no additional comments—leading to admin confusion and unclear dev requirements which inhibited quality assurance testing and eventually, the quality of the product being delivered. As a result, devs were commonly taken away from their work to discuss their insufficient tickets. This was frustrating for a team that considered themselves capable before this structure was enforced.

Administrators were told that a full-fledged Scrum Master role separate from the existing manager and developers would help, but they were reluctant to acknowledge that request. Because of this reluctance, devs were continually forced to handle ticket writing and spent a lot of time gathering to sort through it together. Much like team A and B, the consensus of Team C was that time was being wasted writing all of these stories in such an inflexible format. These tickets failed to hit the key points that admins consistently asked for: *Why are the devs doing that task? How are they completing the task? When will the task be done?* The frequency of meetings affirms the failures of the story format by showing that additional communication was almost always needed; the stories were unable to stand alone in front of administrators. The team was verbal about the increase in tensions because of the mentioned factors during meetings.

In each of these different workplace environments, the story format failed to improve on or maintain existing communication practices and often led to misallocated time. In other words, these constraints made it difficult to uphold several of the Agile principals and resulted in a less quality or significantly delayed product. Because of this, tensions rose higher and higher in each workplace as developers and admins alike became more desperate for a solution. Before I discuss how PAD helped remedy the consistently observed failures of the story format, I must introduce PAD in full detail. In the next chapter, I will review existing literature on PAD to appropriately segue into my main argument about how PAD is a flexible, realistic solution to the aforementioned consequences.

CHAPTER THREE: REVIEW OF EXISTING LITERATURE AND HOW PAD REMEDIES INEFFECTIVE TICKETING PRACTICES

In this chapter, I review existing literature that touches on what the Technical and Professional Communication (TPC), and rhetorical fields are discussing, and find valuable regarding project management and other industry topics discussed such as team communication and workflow. Through this review, I move to make a claim that for TPC to address the gap between academia and industry, we must continue past the work of identifying the problem and engage in the discussion of direct application. I do this by introducing integrative literature reviews of TPC and practitioner journals spanning more than a decade, then I discuss what is available in the field concerning direct application of rhetorical tools in the industries TPC practitioners hope to serve. Afterwards I discuss how the industry/academic gap can be addressed and then move towards introducing how PAD can remedy ineffective ticketing practices.

While there are limited sources discussing direct application of rhetorical tools in the workplace and the outcomes, the field of technical and professional communication (TPC) have some ongoing conversations concerning rhetorical approaches to the workplace. More specifically, these conversations explore how rhetorical approaches are beneficial to both academics and practitioners to assert that as many industries move towards focusing on project management and related methodologies, these approaches are more necessary to teach and engage with. Friess and Boettger, and Lauren and Shreiber both completed large integrative reviews of existing literature produced by both academics and practitioners. Each review shows that there is a gap between the workplace and academia. Further, both the work of Friess and Boettger, and Lauren and Schreiber take the position that when considering project management or other current workplace practices related to the management of people, communication, or documentation, utilizing a rhetorical lens helps lessen that gap because it helps to thoroughly consider the factors important to the industry such as processes, workflows, tools, workplace hierarchy, involved stakeholders, etc. However, neither review offers direct information on what their positions look like applied in the field.

Additionally, there are even fewer sources in TPC related to Scrum or Agile in the workplace. One resource, "Scrum Language Use in a Software Engineering Firm: An Exploratory Study" also by Erin Friess discusses the Scrum language used at a software engineering firm. Friess recognizes that there is very little literature out there concerning Scrum and Agile and works to discern how Scrum is used, talked about in the workplace, and the relationship between the two. The study found through observing several Scrum meetings at an engineering firm that although the firm followed Scrum practices more than half of the time, several components were consistently dropped or unused. Friess takes the position that the research and observation work completed in this study should encourage further analysis into Scrum language use in the workplace— something required to better teach Scrum in curriculum to the benefit of engineering students (Erin Friess, Scrum Language, page 130). Although Friess' work is focused on Scrum, it makes no official move to improve Scrum processes or offer tangible solutions to those working with it. Further, Friess's mention of how many parts of Scrum were routinely dropped does not make the connection to Scrum's founding tenant: Agile should be flexible enough to adapt to a team's unique needs.

Rhetorical approaches to the workplace are somewhat in practice currently but remain fairly general and are not consistently found in all programs. Key technical communication textbooks such as Richard Johnson-Sheehann's *Technical Communication Today* and Markel and Selber's *Technical Communication* — both used to teach engineers and other STEM-related programs— center around rhetorical theory. But these texts still exist mainly in academic spaces and do not offer direct rhetorical solutions to those already working in the field. Each text remains general; neither drills any deeper into the details of specific platforms, interfaces, or workflows that are common in today's software landscape. This review finds that while academics in TPC have long used rhetorical theory as a way to analyze and gain understanding into workplace writing (Campbell, K. S., & Naidoo, Orwig, M. L. (2020), Walwema, J. (2020)) and that rhetorical approaches are slowly being integrated, the gap between what is being taught in TPC programs and what the industry needs is wide (Friess & Boettger and Lauren & Shreiber).

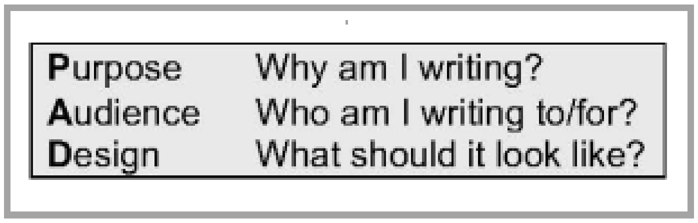
Although there are some ongoing conversations in research spaces regarding rhetorical approaches to the workplace and few discussions about Scrum and other methodologies found in the workplace, there is severely limited research on actual project management techniques focused on application or improvement. If the driving goal of TPC is to be a cohesive field (Friess & Boettger, page 426), thus requiring to not only identify but also address this gap, then the field must engage with more specific applications in the workplaces they hope to serve. Specifically, TPC must consider the workplace as a more crucial part of its audience, not just from an analysis standpoint, but from an actionable and practical one. As it stands, no existing scholarship makes the explicit connection between Agile and Scrum to specific rhetorical tools such as PAD. Further, no existing scholarship discusses the outcomes and realities of utilizing

PAD in any industry. Offering current information and discussion about this is one of the main contributions of this thesis.

As touched on above, there is a lack of literature on Purpose, Audience, Design because the academic and workplace conversations about this tool are only just beginning. All current academic materials on PAD were produced at the University of South Florida (USF) within the PWRT now known as Professional & Technical Communication (PTC). The creation of PAD is solely credited to Dr. Lisa Melonçon, whose class is where I was first introduced to PAD as an important part of my technical writing tool belt as a student.

At this time there is only one published textbook resource, *A Rhetorical Approach to Workplace Writing* which teaches PAD and offers implementation examples for students (Zarlengo, et al.). The text was written by USF TPC educators alongside Dr. Melonçon with the intention of using *A Rhetorical Approach to Workplace Writing* as a supplementary text in their curriculum. I was taught from previous versions of this text and its most current version is in circulation and use at USF. The overall scarcity of additional, meaningful works exploring the practical application and instruction of PAD matters; there are too few resources on PAD. The only other material that exists is a programmatic document written for TPC educators. It is safe to say that PAD is a new conversation in TPC and even more novel to the workplace and wider industry. As I write this thesis in 2022, the potential benefits of either teaching PAD in graduate programs or applying it to the workplace has yet to be measured or studied. Because of this, I lack robust supporting literature for my argument, but I offer my years of workplace experience as a valuable, alternative form of knowledge and institution of praxis.

A Rhetorical Approach to Workplace Writing covers a myriad of topics but what is most important to takeaway is that it identifies both technical and professional writing as a “rhetoric in action," (Zarlengo, et al., 8). By understanding writing as rhetoric in action, Dr. Melonçon posits that practitioners can use *rhetorical awareness*— the act of assessing any writing situation confidently and moving towards the production of quality work. Rhetorical awareness encompasses the following: an understanding of purpose, audience, and how that information impacts design. Dr. Melonçon emphasizes that without using rhetorical awareness, a practitioner is more likely to miss the point of a task and produce less effective work. The best definition that the text offers is that PAD is a "mnemonic device that will help you (the tech writer) analyze a rhetorical situation and make rhetorical decisions to produce effective professional and technical writing projects" (Zarlengo, et al., 21). A simple graphic of this rhetorical analysis may look like the following Figure 4:



Purpose	Why am I writing?
Audience	Who am I writing to/for?
Design	What should it look like?

Figure 4. How to Analyze a Rhetorical Situation

Essentially, PAD invites the writer to consider *why* they are writing and *who* they are writing for to determine what that document should look like. To expand on this, PAD is both a methodology and a workflow that gives writers a starting point for questioning or effective rhetorical analysis of a situation. PAD can be seen as a sort of hybrid strategy for technical writers: a methodology because it is a particular set of procedures (Oxford) in TPC and a workflow because the order of its steps are not arbitrary. A practitioner must address and thoroughly explore purpose and audience in order to approach design thoughtfully. In fact, an essential argument of *A Rhetorical Approach to Workplace Writing* is that each step is inextricably linked in the rhetorical situation overall. To work, each step and line of analysis of the PAD method must be explored with equal attention and importance (Zarlengo, et al., 24).

The other material available is a programmatic document that outlines what TPC teachers should cover to introduce and teach PAD as an effective tool. The document reviews the service course offerings, their rhetorical and theoretical groundings for the pedagogical orientation of the program, and touches on PAD as the rhetorical theory portion of the curricula. The most important idea to take away from this programmatic document is its stated purpose for teaching and focusing on PAD: “Centering the program around the rhetorical theory of purpose, audience, and design allows students to repeatedly analyze the rhetorical situation and adjust to changing parameters, similar to addressing the unknowns of a future workplace environment.” The rhetorical document also goes on to say, “Establishing the purpose and audience of a document requires research and thorough analysis. The concept of design brings in the impact of document appearance, use of visuals to explain information, and the type of document (or genre) that needs to be written to achieve document goals. Taken together, purpose, audience, and design (PAD) gives students a robust rhetorical lens through which they can analyze and respond to any communication situation.”

In other words, PAD is specifically taught to equip students with the ability to navigate any writing situation they may encounter and its format with confidence. PAD invites a careful workflow of rhetorical analysis in a specific order, which lends to this tool's efficacy and practitioner skill-building. This programmatic document became a tenant during my time studying at USF. Indeed, this thesis may be the first to combine PAD, first person industry experience, and its effects on both prospective technical writers and developers, as well as those currently in the industry.

When discussing PAD in class, Dr. Melonçon emphasized that PAD is a flexible, adaptable tool that could be used in more ways than could be covered in class. She acknowledged that the constant state of change and uncertainty in TPC workplaces were exactly the reason why equipping students with tools to adapt no matter the writing situation was so crucial. The programmatic document and textbook both serve to emphasize the sentiment that tech writing as a field is ever-changing, ergo, appropriately adaptable tools are required to ensure student success post-program. As a student that valued effective working practices, this methodology and workflow stuck with me, and its applications were numerous in my navigation as a young practitioner and remain so as a more experienced one now. To refocus on the purpose of this thesis, PAD quickly became my personal method of organizing and analyzing workplace situations where guidance was slim to none. In academia, I believed that I would always be working under another writer, on a team, or at minimum, have guidance from my higher ups on what I should be writing. What I found is that typically, technical writers function alone because the work is commonly devalued. Because of this, PAD immediately became a survival tool for me as a young practitioner, *especially* in ticket writing. With each role I took on in Teams A, B, and C, I leaned on PAD and found an appreciation for clear, quality ticketing to communicate effectively and maintain timely delivery of products.

To provide a relevant example now that ticketing procedures have been covered in Chapter Two, when I am tasked with a new writing situation; I am required to create a ticket to track and communicate my work to the rest of the development team. *I begin by analyzing why I am writing. I ask purpose focused questions such as, am I writing to support an existing or new application? If this is an existing tool, what features am I highlighting? If this is a new tool, is there any existing documentation on the subject? Am I writing to advertise the product and* *increase engagement? Or am I writing to specifically instruct users on how to complete a certain task? I then ask who is my audience? Do they know how to access this product? Are they seasoned technology users or are they new to the product? Do they have any known learning curves or accessibility needs?* Only after asking these questions can I effectively approach the design of the document from an informed standpoint.

How PAD Can Manifest as a Ticket Format

Recall the bug ticket example in Chapter Two. To summarize, developers who used the story format to write bug tickets were often left with lacking tickets such as, “As a person from the email team, I want this bug fixed, so that I can do my job.” With nothing else in the ticket, administrators were left asking questions about the work, how it would be completed, and how much time it would take. Oftentimes I observed that managers would ask devs to leave “additional comments" describing their work, but developers were reluctant to do this, emphasizing that the ticket format should accomplish that without needing extra work.

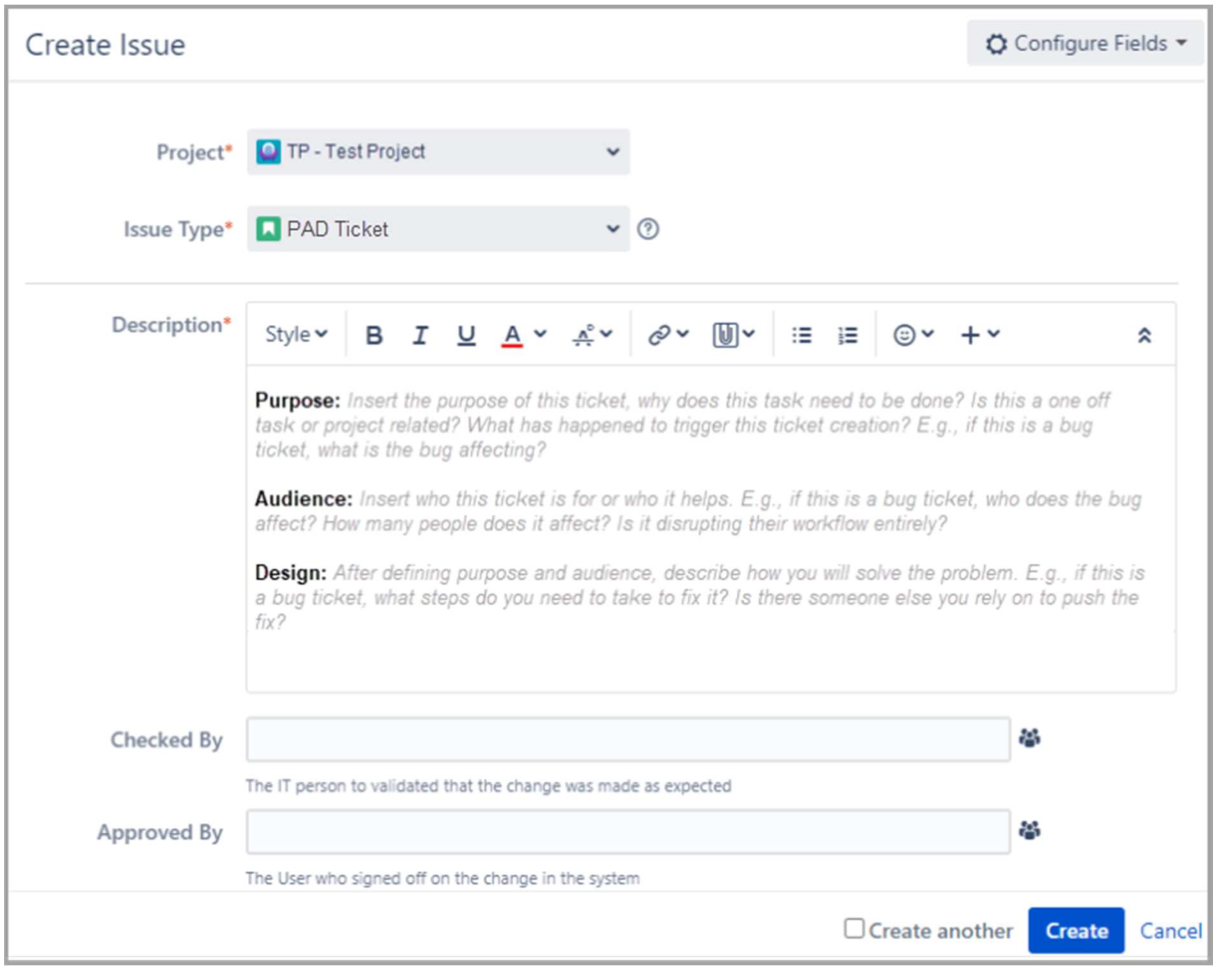
Bugs are very common situations in the software development workplace so they are an appropriate example of how PAD can help a dev write a more effective ticket. I argue, due to the constant point of tension between the story format, the frequency of ticket writing, and the depth of information administrators require, PAD is a more appropriate ticketing solution than the story format. Using the same bug example, when developers have used PAD to write bug tickets, they’ve provided more detail, ensured the audience needs are met, that the task is achieved, and that administrators are satisfied with what the ticket communicates.

If the story format manifests as a one sentence ticket description, PAD manifests organically as more detailed and pointed content. Jira provides the ability to create canned formats for tickets, the story format being their most used and most advertised on the platform.

When I have automated the PAD ticket format, purpose, audience, and design are all bolded headers to individual sections with instructional text that disappears when the developer inserts their own information. Commonly, I lack the time to train each developer one on one, but I have found that instructional text offers an easy way for them to understand what the ticket needs to contain. In Jira, instructional text is grayed out and italics, in my basic mockup below I use italics to indicate this feature:

> **Purpose**: *Insert the purpose of this ticket, why does this task need to be done? Is this a one-off task or project related? What has happened to trigger this ticket creation? E.g., if this is a bug ticket, what is the bug affecting?*
>
> **Audience:** *Insert who this ticket is for or who it helps. E.g., if this is a bug ticket, who does the bug affect? How many people does it affect? Is it disrupting their workflow entirely?*
>
> **Design:** *After defining purpose and audience, describe how you will solve the problem. E.g., if this is a bug ticket, what steps do you need to take to fix it? Is there someone else you rely on to push the fix?*

Recall the previous image of the ticket in Jira. See Figure 5 below for a clearer look at what PAD looks like as a ticket:


Create Issue

Project*: TP - Test Project

Issue Type*: PAD Ticket

Description*:

Purpose: *Insert the purpose of this ticket, why does this task need to be done? Is this a one off task or project related? What has happened to trigger this ticket creation? E.g., if this is a bug ticket, what is the bug affecting?*

Audience: *Insert who this ticket is for or who it helps. E.g., if this is a bug ticket, who does the bug affect? How many people does it affect? Is it disrupting their workflow entirely?*

Design: *After defining purpose and audience, describe how you will solve the problem. E.g., if this is a bug ticket, what steps do you need to take to fix it? Is there someone else you rely on to push the fix?*

Checked By:

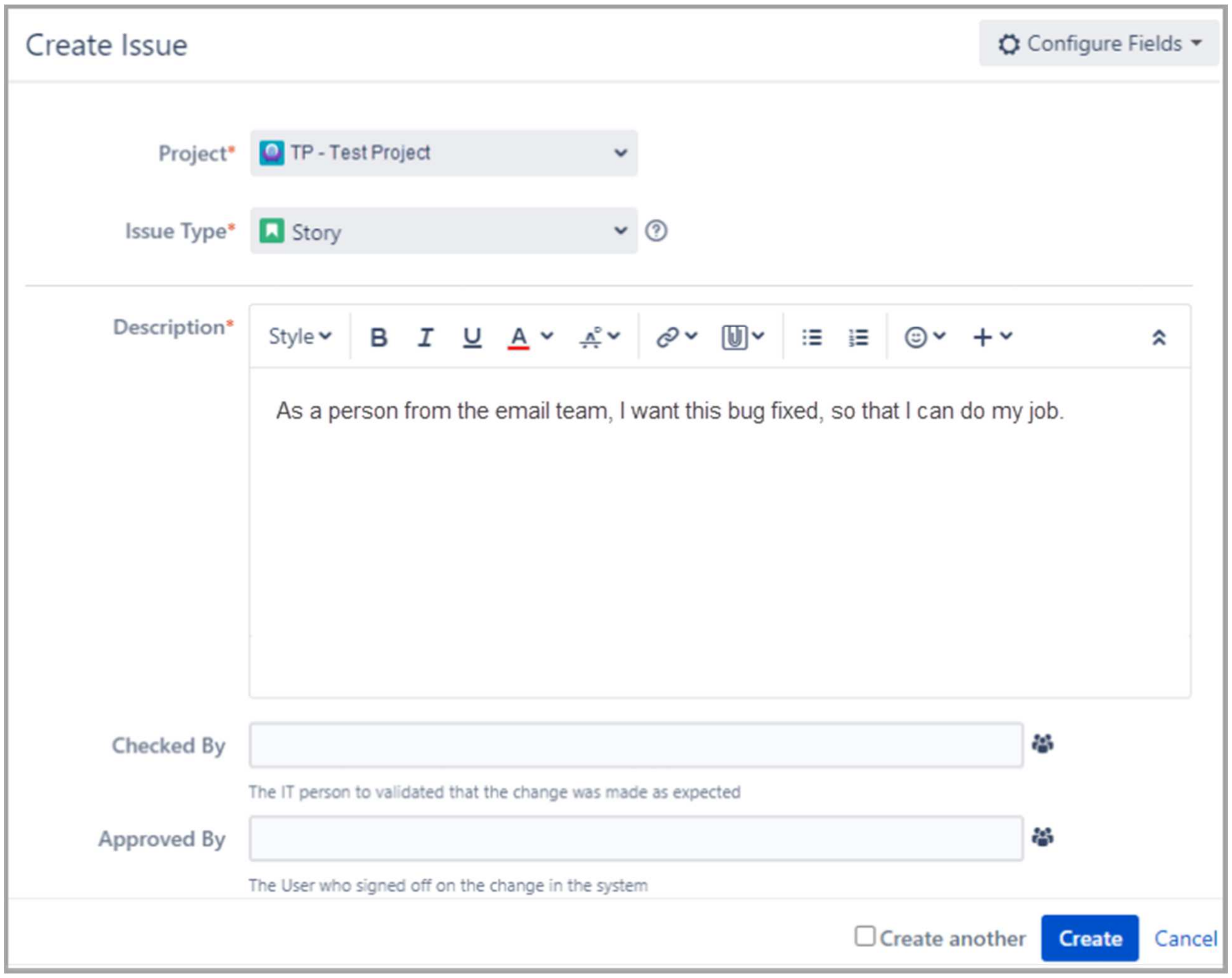
The IT person to validated that the change was made as expected

Approved By:

The User who signed off on the change in the system

Figure 5. PAD as Ticket Format

More details of implementation will be covered in Chapter Four. The essential thing to take away from this example is that the PAD format is simple. This format leaves a lot of flexibility to the devs which they overwhelmingly appreciate, while also satisfying administrators more than the classic story format. When automated and laid out clearly, like above, the devs are provided more detailed instructions in comparison to the story format. For a more detailed comparison that illustrates how each format yields different results, see the difference between two tickets below, one using the story format and the other using PAD, and their observed outcomes. These were modeled from real Jira tickets circa 2020 at Team B, the start-up company. Content was modified to protect proprietary information.


Create Issue

Project*: TP - Test Project

Issue Type*: Story

Description*:

As a person from the email team, I want this bug fixed, so that I can do my job.

Checked By:

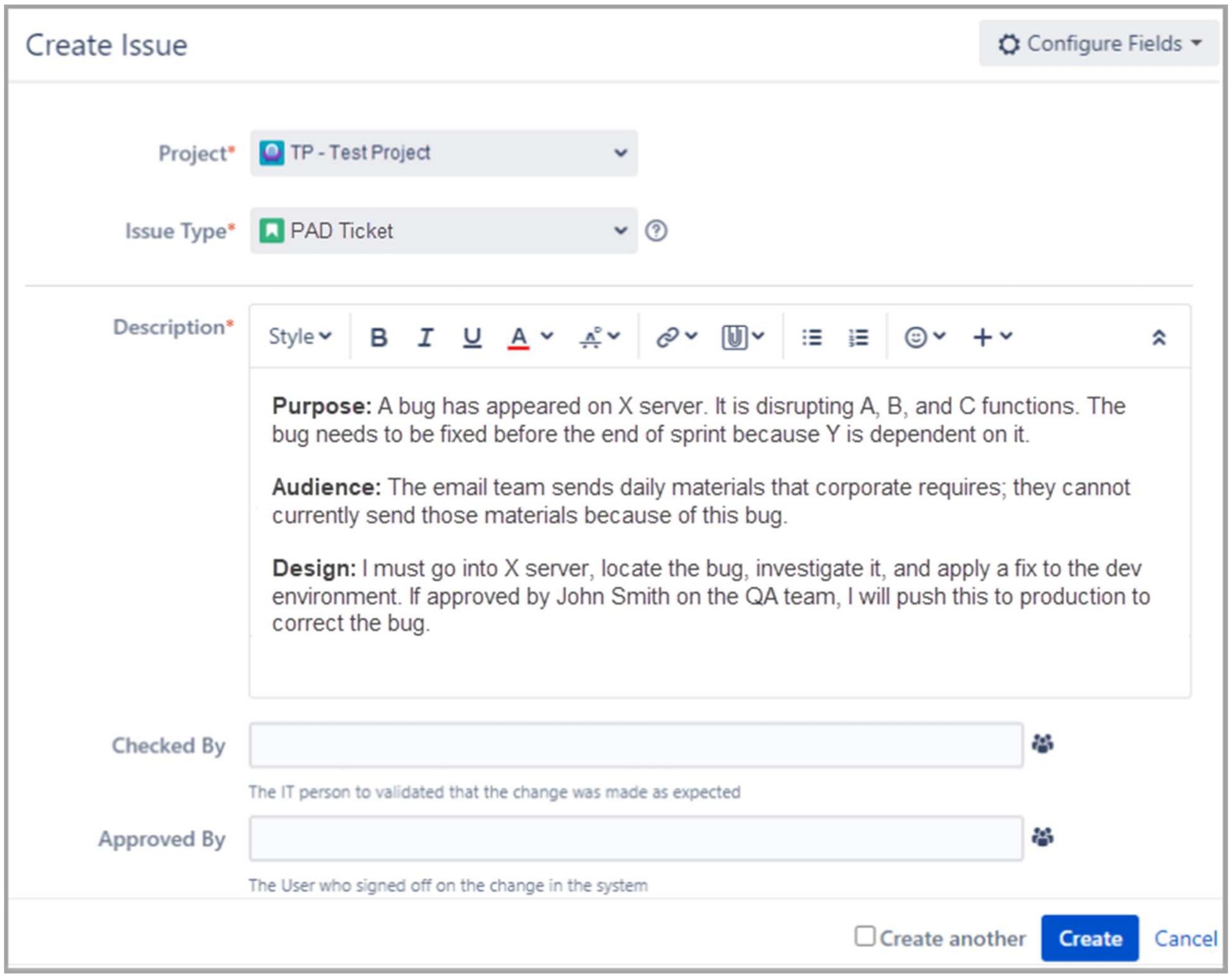
The IT person to validated that the change was made as expected

Approved By:

The User who signed off on the change in the system

Figure 6. Ticket A - Using Story as Ticket Format

Ticket outcome: additional meetings scheduled to clarify exactly why the bug needed to be fixed, who the bug was affecting, and how the bug was ultimately corrected. More time was spent communicating between team members and administrators to ensure everyone was on the same page. Led to tension because the devs were overwhelmingly apathetic to the format and were frustrated that a format was being enforced when it so rarely met administrator needs and required extra time from them that was already scarce to begin with.



Project*: TP - Test Project

Issue Type*: PAD Ticket

Description*:

Purpose: A bug has appeared on X server. It is disrupting A, B, and C functions. The bug needs to be fixed before the end of sprint because Y is dependent on it.

Audience: The email team sends daily materials that corporate requires; they cannot currently send those materials because of this bug.

Design: I must go into X server, locate the bug, investigate it, and apply a fix to the dev environment. If approved by John Smith on the QA team, I will push this to production to correct the bug.

Checked By:

The IT person to validated that the change was made as expected

Approved By:

The User who signed off on the change in the system

Figure 7. Ticket B - Using PAD as Ticket Format

Ticket Outcome: No additional meetings or extraneous communication needed. Administrators were satisfied knowing who it was affecting, when the bug fix was to be expected, and had confidence that the developer understood the gravity of the situation.

As illustrated, PAD ensures the task is achieved and is communicated in a way that is realistic for the developer and meets the needs of administrators. Harkening back to Scrum methodologies and what ticketing achieves, the agency and ability to independently complete tasks alongside team deadlines and dependencies afforded through effective ticketing is highly valued. PAD proved to be useful in solving the consistent issues that arose with teams A, B, and

C when trying to enforce the story format; poor communication which lent to wasted time which resulted in workplace conflict. I now recall the examples used in Chapter Two about teams A, B, and C. Each team represents a different workplace and context, but I discuss them again now to highlight the failures of the story format and how this failure is not isolated, or solely due to poor management or lackluster training. I posit that offering PAD as an alternative ticketing format for developers helps reduce poor communication, disrupted workflow, and tensions between developers and administrators.

PAD in the Workplace: A Remedy for Ineffective Ticket Practices

In Team A, a service desk workplace made the transition from no ticketing or formal methodology to Scrum and its required ticketing practices. The developer's main complaints were focused on how uncomfortable and inappropriate the format felt to their existing workflow and practices. When I was provided the opportunity to consult on ticketing practices, my offering of PAD as an alternative format eased many of their minds. From bugs to more complex releases, my coworkers were able to spend less time pondering or researching, *what is a persona? How do I write in the first person from that coworker's point of view? Why do I have to do this?* And were able to jump straight to writing what was required for them because the format made more sense to them.

It is difficult to describe, but with ample experience under my belt I can confidently say that those who are drawn to the IT umbrella as developers, programmers, quality assurance practitioners, or even the administrators in the field are all individuals who prioritize and rely on direct, no-frills communication. The word “story” lent to their inability to wrap their heads around personas and the point of the format was lost on them. In other words, it did not fit their needs and because of this, the tickets they produced were poor. PAD sat better with my team and when introduced, took to the format flyingly. Slowly but surely less time was spent in duplicate meetings where the only goal was to sort through and translate poorly written tickets. Feedback from this team was scarce because this was a less official process. At the time I was still in a service desk role and was not a true technical writer. However, it was my background, degree, and in-hand solution of PAD that afforded me the opportunity to confidently suggest a ticket format alternative. The most relevant proof that PAD reduced the consequences of ineffective ticket writing was the reduction in excess meetings and the apparent return of relaxed team culture in contrast with the high-stress, high-tension meetings I was attending prior to introducing PAD.

On team B, management made the decision to cut the role of Scrum Master because of the small size of the start-up team. However, management realized that they severely underestimated the workload and impact Scrum Masters have on an Agile software development team. Consequently, I was asked to step in to assist in ticket writing to reduce the workload of overburdened developers. At first, the administrators did not believe that the devs needed assistance with such "easy" work like ticket writing. But the increase in alignment meetings (where team members and administrators would gather and discuss the meaning of tickets in question) became proof enough that a long term, standardized solution was needed. Once I stepped in and scoped the work, I recognized again that PAD would likely provide a viable solution. At this point I had unofficially used PAD as a ticket format on Team A, so I moved towards standardizing the PAD format in Team B through Jira's ability to customize and automate ticket formats based on the team's unique needs. I created a canned format nearly identical to Figure 7 in the How PAD Manifests as a Ticket Format section. From here, I took a quick 15 minutes to brief the devs on this change and emphasize that it would simplify the workflows enforced by administrators. I updated admins on this format change and was interested to observe how this recommendation affected the team. Since this was my second application of it formally— meaning it was being used by other coworkers and not just myself in a research vacuum— I was able to take more pointed observations as things unfolded.

What followed indicates that PAD can correct situations the development world sometimes views as inevitable or unsolvable. Developers on team B, much like team A, took to the format quickly. Feedback I received from administrators praised PAD's ability to get at the point directly and coax the right information out of developers. Simply put, headaches were reduced on all sides of the team. Feedback from developers heavily focused on the time saving factor and that the new formatting smoothed out communication issues between them and admins; general morale was up. From this point, the project of improving ticketing standards was dropped and left undiscussed. This does not indicate lack of engagement but rather is like when folks sit down for a delicious dinner and are so engaged with what's on their plate, the table descends into silence. In the fast-paced world of software development, when a problem is solved, administrators rarely waste time circling back to expand on the conversation. What is fixed is fixed and they move on immediately. I ended up leaving this position, but ensured my automated standards stayed in place. As someone who values best practices when they're able, I did circle back to talk casually with old coworkers and was pleasantly notified that my standards were still in place, worked like a charm, and kept excessive meetings and conflict down to a minimum.

Team C was my most recent experience in employing PAD as a solution to ineffective ticketing practices. Since it was a more corporate setting than Team A or B, pressures and tensions in Team C were much higher. Corporate culture typically revolves around and hyperfocuses on what administrators and higher ups want prioritized. This manifested with Scrum as constant attention and pressure around “correct” ticket writing. This emphasis led to the same consequences witnessed when working with the other teams: poor communication, wasted time, and workplace conflict. I spent most of my time at this role offering my assistance with ticketing but was regularly redirected to my assigned project work instead. For a short time, administrators moved towards a “who, what, where, when, why” format believing this classic template would address the problems. However, the admins did not discuss this change with the developers, but rather introduced the new format unilaterally, leading to a slew of meetings to handle disgruntled developers and irritated administrators. Rather than address each “who, what, where, when, why” in their tickets, developers left much of the template blank. Only filling in the who and the what, never the where, why, or when.

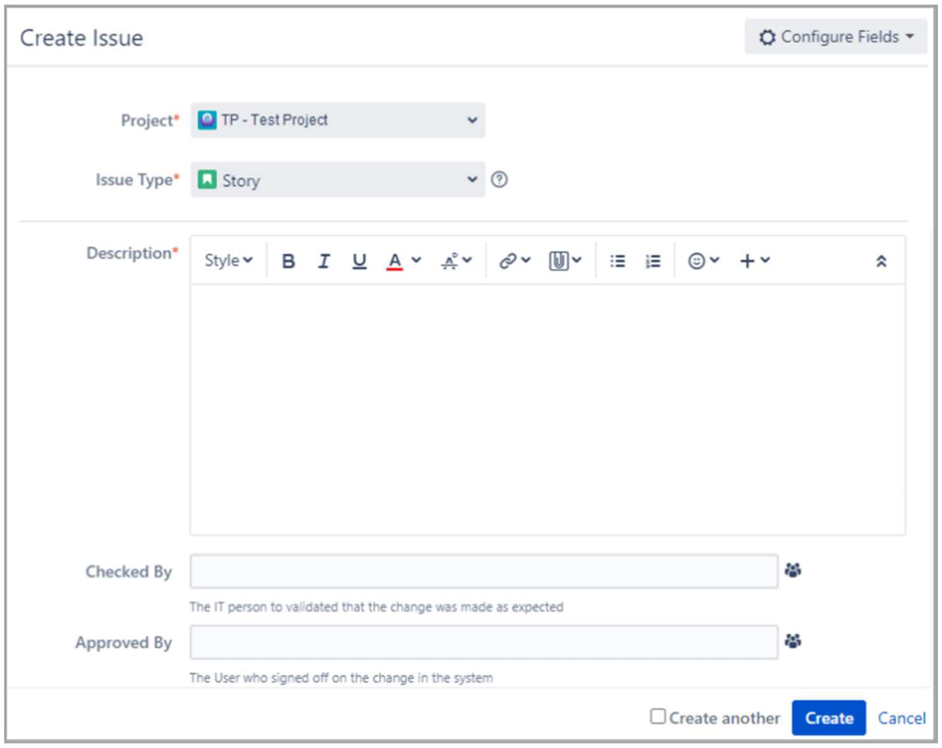
When I asked developers what they didn’t like about the format, they noted it felt too much like reading class for no discernable reason. They did not feel that the “when” was important and didn’t always feel inclined to elaborate on the “why” because to developers, when there is a problem, it needs to be fixed and they don’t particularly care for the specifics. The excessive meetings, lagging workflow, and subsequent tension continued. Eventually, administrators relented and asked to see what I suggested. I re-created the canned template on this company’s iteration of Jira, presented it, and was given some permission to test it out with different coworkers in isolated environments. This meant that their ticketing practices were not being observed by the whole team and the workplace research occurred in a sort of silo. This is where my offering of PAD as a solution differed from the other two experiences. Corporate environments are notoriously resistant to change and, in this case, were reluctant to allow a technical writer to speak on or change their workplace standards.

However, despite their resistance, PAD was successful again. The four team leads I was tasked with testing PAD on enjoyed the format so thoroughly in contrast to the story format they were frustrated with, that they disseminated it to their other team members immediately. In the end, Team C, myself included, were taken aback when the administrators decided to abandon the effort. The eventual explanations were vague: team leads and developers should complete tickets as they see fit but the team should not waste time improving Scrum standards or testing out new formats. This leeway in ticketing resulted in many of the developers using PAD on their own from my simple copy/paste template. It was frustrating that I was not allowed to push my canned ticket format to production in Jira for team use. But despite these circumstances, the devs continued using the basic PAD formatting I introduced— proof enough that frustrated developers benefit from P.A.D. in any capacity. What this additionally argues, is that even when the company body may not be fully on board with the practice, it can still help in unofficial ways by offering an alternative approach to those confused by either the more standard persona formatting or the "who, what, where, when, why" format.

Overall, team A, B, and C, all experienced the same workplace issues of poor communication, wasted time, and subsequently increased conflict because of the failures of the story format in their workplaces. This highlights again that small to mid-sized software development teams operating without a Scrum Master or ticket handler all face a time management issue, leading to other problems. When a developer doesn't have enough time to attend to their assigned work and are made to spend extra time ticket writing, tensions rise. When an administrator enforces a poor ticket format and does not offer alternatives, devs are more likely to write lacking tickets or seek other alternatives on their own. In all cases, when either of the above problems arise, workflow and time management are always disrupted because the ticketing practices are not working as ideally intended. These examples, although not formally researched or recorded, still serve as a valid source of knowledge because it comes from the perspective of a trained PAD practitioner hailing from USF's PWRT program and a seasoned technical writer in the field who employed what they learned in PWRT curriculum to succeed at work. In the next section I offer practical implementation suggestions in a move to encourage others to continue working with the application of PAD in software development workplaces.

CHAPTER FOUR: Implementing PAD as a Canned Ticket Format in Jira

To effectively illustrate what the previously discussed examples of teams A, B, and C looked like and move towards making this information accessible to others looking to continue this work, I will outline how the tickets appear in Jira. In addition to this, I offer insight into how one might configure these fields in Jira based on real experience and Atlassian's resources on ticketing. For the purposes of this thesis, I will describe implementation generally because I assume that those who work with Jira know how to access specific resources from Atlassian's support site. The basic ticket format looks like Figure 8 below:


Create Issue

Project*: TP - Test Project

Issue Type*: Story

Description*:

Checked By:

The IT person to validated that the change was made as expected

Approved By:

The User who signed off on the change in the system

Figure 8. Basic Blank Ticket

From top to bottom, there are the following fields enabled:

- Project drop-down
- Issue Type drop-down
- Description text box
- Checked By user search box
- Approved By user search box

Jira offers many other automated fields to add or remove from the basic story format such as priority level, labels, department, assignee, quality assurance (QA) steps, etc. When fields are customized, the user is offered the option to save over an existing ticket template or create a new one. Doing either would make that format available in the Issue Type drop-down. This thesis acknowledges that PAD may not work in all scenarios, therefore I recommend creating a new Issue Type labeled "PAD Ticket" or something similar to easily integrate it into ticket writing practices. To do this, access the settings on a Jira homepage and navigate to Issues. From here users will find Custom Fields under Fields. Users can also access this area more quickly for existing ticket formats by clicking the Configure Formats option, seen outlined in red below, when creating a ticket. See Figure 9 below.

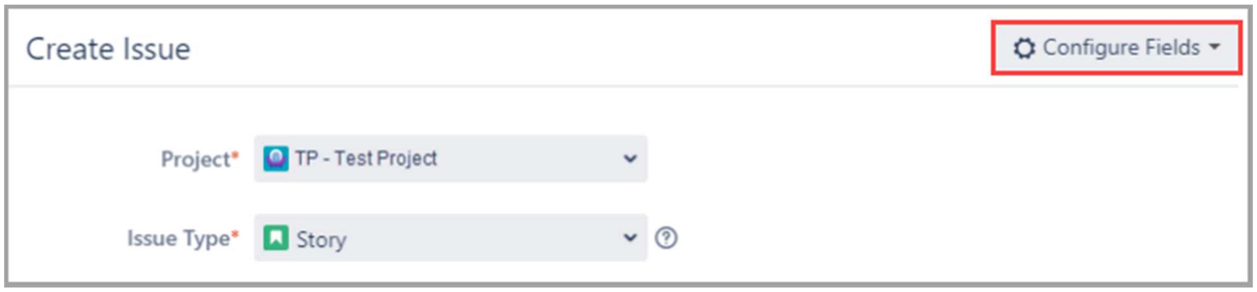


Figure 9. Configure Fields Drop-down

Jira simplifies the ticket customization process by providing on-screen instruction as well. Once in Custom Fields, users can create a variety of fields to their liking.

There are two ways to implement the PAD format in a ticket that I have experimented with in the workplace either for my own tickets or the rest of the dev team. The first implementation method utilizes the instructional text options in the existing description field (which is also automatically included when using most of Jira's canned formats). While in Custom Fields, create a new custom text box field with the PAD text marked accordingly for normal vs. instructional text. This differentiation exhibits as normal text that stays put and grayed out text which disappears when the dev clicks into it for editing. Further, this style of instruction offers direct guidance each time a new ticket is generated. This is helpful because training time is severely limited when adopting new practices, meaning on screen instructions help fill the void when thorough teaching cannot take place. Users can copy from the text below, which was previously seen in Chapter Three and mark it appropriately or create something similar that they believe may suit their team better. Like before, italicized text indicates where instructional text type should be used.

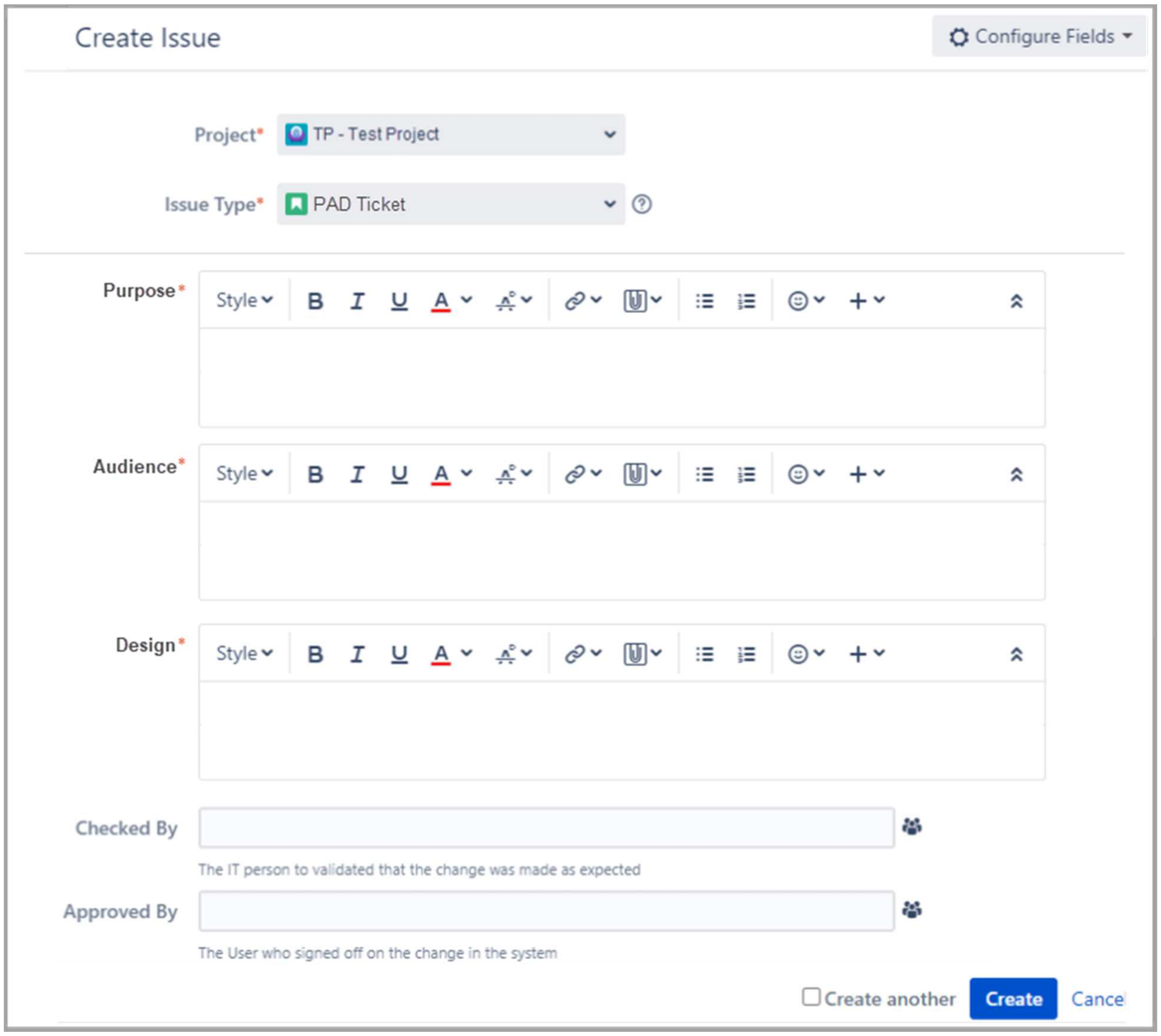
Purpose: *Insert the purpose of this ticket, why does this task need to be done? Is this a one-off task or project related? What has happened to trigger this ticket creation? E.g., if this is a bug ticket, what is the bug affecting?*

Audience: *Insert who this ticket is for or who it helps. E.g., if this is a bug ticket, who does the bug affect? How many people does it affect? Is it disrupting their workflow entirely?*

Design: *After defining purpose and audience, describe how you will solve the problem. E.g., if this is a bug ticket, what steps do you need to take to fix it? Is there someone else you rely on to push the fix?*

Once completed, the user saves the custom field and selects checkboxes for which formats it should be included in. For this example, the user would select PAD Ticket in order to utilize it as a ticket format. It is important to note that the existing description field would need to be removed. These options can be accessed in a similar way Custom Fields was accessed. From this point, any developer who selects the PAD Ticket option from the Issue Type drop-down will immediately be able to engage with the format. If the format ends up being unsatisfactory for the team utilizing it, developers can simply stop selecting PAD Ticket and instead choose Story or another suggested format from the drop-down when creating tickets. This flexibility and ease of access is crucial to implementing in Jira because otherwise PAD becomes a burden, much like the traditional story format.

A second way to implement PAD is to separate each of its elements, purpose, audience, and design, and break them into their own text box fields each with its respective instructional text. If a team was interested in more granular progress data or more thorough quality checks at every level of the format, separating out the fields would help achieve this. It was at one point in my experience requested that reporting include who was reviewing each portion of the ticket. Jira reporting takes its data from the tickets, how fast they progress, time of completion, who touches and edits them, and more nodes. It offers robust visualization methods with the option to customize so that administrators can understand how their team is doing at a deeper, more meaningful level. This second method of implementation may be useful for teams looking to do this. Much like the first example, users would access Custom Fields and create a custom text box field with instructional text for each element of PAD. Ensuring the text is appropriately marked as normal and instructional text where appropriate will ensure that PAD's automation encourages developer independence and efficacy. See Figure 10 below.


Create Issue

Project* TP - Test Project

Issue Type* PAD Ticket

Purpose*

Audience*

Design*

Checked By

The IT person to validated that the change was made as expected

Approved By

The User who signed off on the change in the system

Figure 10. PAD Implemented as Three Fields

If desired, users could choose to apply a Checked By or Approved By field underneath each of the broken-out text boxes to ensure there is an assigned user each time a ticket is being worked on. By providing numerous fields, the reporting administrators generate will be more detailed. I do not recommend this format unless administrators require additional reporting from the development teams. I feel the first implementation option is more than appropriate but in the spirit of options and alternatives, it is good to provide format implementation which worked for me when I was required to provide that level of detail.

Due to Atlassian's product price range, I was unable to actively generate new material for this thesis and am instead relying on materials I drafted while I had access to Jira. Much of the images presented were created or mocked up during my work to adopt PAD in the workplace or nabbed for portfolio content. Jira's price per individual makes research nearly inaccessible to solo practitioners operating without a team or even an academic body to back them up. I believe this is important to note because if someone wishes to continue this work, the biggest inhibitor is access to the platforms where tickets are written and managed. Meaning much of this workplace research will fall to practitioners with current access to Jira. If this implementation is moved to an openly visited site such as Atlassian's forums where practitioners share work arounds and solutions for Jira problems, the above canned format becomes more accessible and can be implemented with little to no training because of the instructional text format. This is important for teams operating without a Scrum Master, the main audience of this thesis, because low barriers to implementation increase the likelihood of use. Since Atlassian's forums are available to the public online, for practitioners with access to Jira, adopting PAD is made easy.

Take-Aways

Scrum's classic ticket format works well for many teams but for small to mid-sized teams operating without a Scrum Master, it fails to meet expectations and results in poor communication, disrupted workflow, and subsequently, heightened tensions. The story format was an effective starting point since it is an industry standard, but clearly needs an alternative to remain productive in varied circumstances. Through my work, I found that PAD generated numerous benefits for its users when offered as an alternative to the existing story format for three small software development teams and argue that this is valid means to continue work on these topics.

Teams A, B, and C experienced the same consequences due to the failures of ticketing practices but despite the situational and structural differences of their companies, each team achieved the following by employing PAD in some capacity: smoother communication between devs and admins, less disrupted workflow, and reduced tensions related to the prior frustrations. Put simply, PAD worked well and was valued by devs working without Scrum Masters and their admins required to keep tabs on dev team progress. For that reason, PAD should be an encouraged offering for software development teams seeking solutions for problems related to lacking a ticket handler.

In the workplace, formal academic research is harder to get approved. Most smaller workplaces lack the resources to take time away from work to participate in a study or feedback rounds so closing the gap between technical writing curriculum and the workplace is admittedly challenging. Despite my academic background, I myself lack the resources and time to pursue a more thorough study. However, as a technical writer trained in rhetorical approaches who must hold a job at all times, I took the opportunity to leverage PAD where I could. In doing so I attempted to solve a specific workplace problem I faced and will likely continue to face in my line of work. In some important ways, my leveraging of PAD in the workplace was a success. I offer the previous implementation chapter as a starting point and a hopeful move in supporting further work on the subject by other technical and professional writers working in software development.

I acknowledge that this thesis lacks empirical evidence, but it does step beyond the theoretical into practical application in the field. In the absence of formal research and the resources to do so, this thesis positions working experience as an appropriate source of knowledge. Adopting PAD is low stakes and if a team finds it to be less useful, then they can simply revert back to story format or employ another alternative. Tickets are written so frequently and as previously discussed, generated in an automated manner, making implementation of alternative formats incredibly easy. There is little to no reason not to share useful tools between the workplace and academic spaces, especially when there are fruitful rewards. In the last chapter, I offer additional discussion and research suggestions for hopeful future work on these topics.

CHAPTER FIVE: DISCUSSION AND ADDITIONAL RESEARCH SUGGESTIONS

In this chapter, I discuss both workplace and academic applications of PAD, then conclude with potential research applications to continue the work of this thesis. As discussed throughout this thesis, the tension between academia and the software industry is amended through access. This access can look like rhetorical tools, the incorporation of low-risk formats like PAD, and continuous training and dialog between both sides of the gap. My hope is that this final chapter, as well as this project as a whole, encourages other industry workers and scholars to do the work of closing the gap between academia and industry, improve workplace operations, and better prepare students and workers of all levels to meet the needs of a demanding, changing, and always-expanding software field. First, I discuss how the implementation of canned PAD formats uphold Agile's founding principles in an attempt to convince readers of all backgrounds that PAD, as a single piece of the puzzle, can help the workplace adapt, stay on task, and improve inter-team and cross-functional communication.

Implementing PAD as a canned format, as discussed in earlier chapters, is a key opportunity to not only improve workflow, but also stays true to the heart of Agile's defining principles. Think back on principle one: "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software." When a dev team can't maintain workflow because their ticketing and communication are ineffective, they are unable to provide early and continuous releases for their products. Customer and user satisfaction depends on quality, software delivered consistently. What this principle also underpins is that these deliverables are most possible with highly effective teams. The industry needs flexibility and *access* to the tools that make high-quality software possible. Rhetorical strategies like PAD are easy to introduce and implement and can have extremely positive results.

Additionally, PAD helps teams achieve or uphold many of the Agile principles such as principle nine, “Continuous attention to technical excellence and good design enhances agility.” Tickets are written so frequently that their practices play a heavy hand in shaping workplace culture. PAD— by its very structure— asks the user to consider purpose and audience first before approaching design. Through this, the developer is assisted in maintaining their attention on technical excellence and helps keep a steady pace at work. When design is thoughtfully approached, less time is wasted circling back, realigning values, and enduring excessive meetings and therefore, agility is enhanced across the team. In the same manner, administrators may find PAD a useful format in drafting project planning roadmaps for the year. These are documents stored in Jira in some capacity that guide the team through its priorities each year. Typically, they are messy, full of unrefined ideas, and lack central driving themes to motivate workflow. For each priority administrators identify, PAD or similar methods could be leveraged as a way to clarify the purpose and audience of the priority so that the development teams can approach design more informed. As I continue working in Scrum, this work may be possible in the future.

In the open-source spirit of the Snowbird 17, I may move towards bringing PAD to the Atlassian forums in hopes of introducing it to more workplaces that need solutions for upholding the Agile principles they wish to embody. I believe that these forums, although arguably neglected by Atlassian themselves, are where most Scrum practitioners go to find help. Even if they are not using Jira or any of Atlassian’s tools, they offer robust free resources about Agile and Scrum. Small dev teams that experience challenges when operating with less-than-ideal resources require tools to adapt and overcome. If PAD is introduced to these professional open-source spaces via sharing, perhaps using the implementation above as a starting place, it would be well positioned and easily accessible for teams seeking solutions. In this way, a tool originating in rhetoric and technical writing finds direct workplace application and becomes a part of a methodology canon in the making.

Beyond simply introducing PAD into my own workplaces or on industry support channels and forums, I assert that PAD should be a curriculum standard for all undergraduate and graduate technical writing programs. If tech writing programs and other academic bodies wish to close the gap between what is being taught and what workplaces require, considering the perspective of students utilizing what they learned at work is valuable. Referencing back to the main text reviewed in Chapter Three, A Rhetorical Approach to Workplace Writing emphasizes that professional and technical writing hinges on actions and practicality (page 7). Further, it emphasizes that effective technical communication benefits greatly from recognizing its intersections with rhetoric which is after all, “the art of effective communication” (page 13). Through past and current action, I have found that PAD is an effective way to recognize a marriage between rhetorical lenses and practical application. If PAD is “a mnemonic device that will help you analyze a rhetorical situation and make rhetorical decisions to produce effective professional and technical writing projects” then this thesis serves as a piece of evidence that supports the claims that PAD helps students “put rhetoric in practice.” In other words, technical writers and other professional writing students in the USF TPC program should feel confident that the rhetorical approaches and tools being taught are useful beyond the classroom in lucrative, successful ways.

I believe it is worth taking the opportunity to touch on the ways I have leveraged the tool in different workplace situations because it helps to prove its value in tech writing curriculum. Other than being a useful ticket format, I used PAD for many writing tasks where I had little to no guidance. When I first began my first technical writing role in the finance industry, I was tasked with writing a variety of documents I had never encountered before. One of which was a business continuity plan (BCP) which was necessary for companies with physical campuses operating where emergencies could happen. Smaller businesses or those with less urgent functions may not need such detailed documents but for financial firms operating in the south where hurricanes could potentially disrupt business functions severely, BCPs are needed to mitigate risk and fallout. My instructions were to give it a first go and then bring it to the table for administrators to review and tweak. The internet was not of much help, but PAD gave me a proper starting point, purpose and audience are of the utmost importance in regard to emergency planning. I knew to focus on the fact that it was an emergency document (purpose) meant to keep communication and functions up and running between admins, those that work with them, and the many clients they served (audience), therefore the document needed to contain detailed contact, equipment, and physical address information for the entire company (design). PAD enabled me to approach an insurmountable task confidently.

In all of my technical writing roles, PAD also provided me with a more substantial ability to conduct source matter expert (SME) interviews in comparison to my peers that did not have the tool. I eagerly shared it with them when given the opportunity and it eased anxiety when interviewing SMEs. Considering the audience thoroughly is a crucial component of PAD and in the PWRT program I attended, practicing this method of interrogation was common. Considering purpose only tightened down my ability to formulate appropriate questions. I ask, what *is your* *team's experience with X software at this time? What functions or features on the homepage do you want your team to understand in release 1.3?* or *can you walk me through what you consider best practices for general ledger reconciliation for X role?* instead of *What do you want in this document?* With PAD in my toolbelt, I was able to ask informed, direct questions which yielded more detailed interviews and intentional documentation.

In another role, PAD enabled me to create incredibly specific release documentation for specialty departments at a company. One of which was a committee formed from high performing administrators whose role was to provide preliminary feedback about a release's potential success. This would help the application developers and technical writers produce better products in the long run since they were essentially a testing group. I had previously worked with release documentation, but the audience was in the hundreds, so the documents were fairly general. This special department required release documentation that catered to their every quirk specifically because the departments they oversaw were small, niche, and committed to excellence. Although difficult to pin down, I was able to conduct two very productive SME interviews without issue and therefore understood the parameters of the document more clearly. The group was satisfied with the document and did not require additional rounds of edits to understand the material as they had in the past.

With numerous workplace applications that go beyond the scope of this paper, PAD should certainly be taught in more technical writing programs and at minimum be continued in USF's PTC program with added evidence that it is highly valuable to many parties. Any technical or professional writing program that wants to help students with job retention, satisfaction, and general competence should consider the rhetorical approach to satisfy these goals. No one program can perfectly prepare a student for the ever-changing, fast paced professional realm but a flexible, adaptable tool is arguably a valuable solution. PAD continues to prove its value to me, and I will utilize and share it every time an opportunity presents itself. From solving Scrum ticket problems for small software development teams, to preparing a company of sixty thousand employees to handle an emergency, to interviewing data architects, PAD offers a helpful methodology and workflow for approaching any documentation task. It has the potential to take its place as an essential rhetorical tool taught to technical and professional writers and as an openly shared resource on both Atlassian forums and so that dev teams struggling without a Scrum Master have an effective alternative to turn to.

Finally, I will offer future research suggestions in order to continue this work. I offer three potential research applications for PAD focusing on three different audiences: workplace dev teams, graduate students, and workers in UX/UI design. I focus on three different audiences to highlight PAD's flexible nature and its capacity for integration of cross-functional teams and professionals of any experience level, be they experts or completely new to the software industry.

First, I suggest a study looking at tracking the user experience on teams adopting canned PAD formats into their ticket-writing process. The user, in this instance, would not be the end-customer of the software, but the actual software developers themselves. After training on PAD and introducing the canned ticket format in Jira, the researcher may conduct one on one interviews with team members to gauge their perceptions of PAD, its usefulness, and the overall experience. Researchers may look into the three main pain points introduced in this thesis as well: understanding of the ticket process, communication, and perceived workplace tension.

Second, I recommend introducing rhetorical learning strategies to graduate students and interviewing these students to gauge their understanding of rhetorical tools, including PAD.

Then, researchers may follow these students' post-graduation and as they enter the workplace, continue to interview them. Potential questions may be: *How often do you use PAD and other rhetorical strategies in the workplace? How prepared did you feel to enter the industry? How prepared do you feel 3 months, 6 months, 1 year in the industry?*

Finally, there is so much potential to introduce PAD and other rhetorical approaches beyond tech writing and strict software development industries. A researcher may study how PAD and its application in other industries that focus on production, like graphic design or UX/UI design could impact a practitioner's ability to create designs that better suit their user's needs. This could yield interesting results due to graphic design and UX/UI design's existing investment and focus on audience analysis.

The application of PAD is truly endless. I hope this thesis encourages practitioners towards incorporating PAD into their own work or curriculum, and that this thesis is the first of many papers exploring its real-life application and benefits.

REFERENCES

Augustyn, Adam. “Software | Definition, Types, & Facts.” Encyclopedia Britannica, www.britannica.com/technology/software.

All Answers Ltd. “The History Of The Waterfall Model Information Technology Essay.” UKEssays, 31 Dec. 2021,www.ukessays.com/essays/information-technology/the-history-of-the-waterfall-model-information-technology-essay.php.

Campbell, Kim Sydow, and Jefrey S. Naidoo. “Rhetorical Move Structure in High-Tech Marketing White Papers.” Journal of Business and Technical Communication, vol. 31, no. 1, Jan. 2017, pp. 94–118, doi:10.1177/1050651916667532.

Drumond, Claire. “Is the Agile Manifesto Still a Thing?” Atlassian, www.atlassian.com/agile/manifesto. Accessed 17 Feb. 2022.

Drumond, Claire. “Scrum - What It Is, How It Works, and Why It’s Awesome.” Atlassian, 2018, www.atlassian.com/agile/scrum.

Editor. “Agile Project Management: Best Practices and Methodologies.” AltexSoft, 18 Sept. 2019, www.altexsoft.com/whitepapers/agile-project-management-best-practices-and-methodologies.

Friess, Erin. “Scrum Language Use in a Software Engineering Firm: An Exploratory Study.” IEEE Transactions on Professional Communication 62 (2019): 130-147.

Friess, Erin, and Ryan K. Boettger. “Identifying Commonalities and Divergences Between Technical Communication Scholarly and Trade Publications (1996–2017).” Journal of Business and Technical Communication, vol. 35, no. 4, Oct. 2021, pp. 407–432, doi:10.1177/10506519211021468.

Highsmith, Jim. “History: The Agile Manifesto.” The Agile Manifesto, agilemanifesto.org/history.html. Accessed 17 Feb. 2022.

Hughey, Douglas. “The Traditional Waterfall Approach.” Umsl.Edu, www.umsl.edu/%7Ehugheyd/is6840/waterfall.html. Accessed 17 Feb. 2022.

Johnson-Sheehan, Richard. *Technical Communication Today: Pearson New International Edition*. 4th ed. Pearson, 2013. Web. 17 Feb. 2022.

Lynn, Rachaelle. "The History of Agile." *Planview*, 21 Nov. 2019, www.planview.com/resources/guide/agile-methodologies-a-beginners-guide/history-of-agile.

Lauren, Benjamin and Joanna Schreiber. "An integrative literature review of project management in technical and professional communication." Technical Communication 65 (2018): 85-106.

"Methodology Noun" Oxfordlearnersdictionaries.com, 2022, www.oxfordlearnersdictionaries.com/us/definition/english/methodology. Accessed 5 Mar. 2022.

Orwig, Marcy Leasum. "Rethinking Soft Skills Through Front-Stage and Back-Stage Genres." Business and Professional Communication Quarterly, vol. 83, no. 2, June 2020, pp. 223–233, doi:10.1177/2329490620905905.

Rehkopf, Max. "User Stories | Atlassian." Atlassian, 2019, www.atlassian.com/agile/project-management/user-stories. Accessed 20 Jan. 2022.

Rigby, Darrell, et al. "The Secret History of Agile Innovation." Harvard Business Review, 27 Aug. 2021, hbr.org/2016/04/the-secret-history-of-agile-innovation.

University of South Florida. "PTC Programmatic Foundations." University of South Florida, 2021.

Walwema, Josephine. "A Values-Driven Approach to Technical Communication." TechComm, Technical Communication Online, 5 Mar. 2020, www.stc.org/techcomm/2020/03/03/a-values-driven-approach-to-technical-communication.

Zarlengo, Tanya, et al., editors. A Rhetorical Approach to Workplace Writing. Edition 8, United States of America, University of South Florida, 2021.

"What Is Software Development?" IBM, https://www.ibm.com/topics/software-development. Accessed 1 Feb. 2022.

"What Is Jira Used For?" *Atlassian*, www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for. Accessed 1 Feb. 2022.