Manage Your Expectations: How has Agile Lived up to its Promise to Decentralize Power?

A Research Paper submitted to the Department of Engineering and Society

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Jacob Rice

Spring 2023

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Advisor

Kent Wayland, Department of Engineering and Society

Introduction

In software projects, it is often necessary to monitor the progress of a software product, as well as its adherence to budgetary and time constraints. To this end, software is usually developed using one of two paradigms for development: plan-driven development and Agile development. In plan-driven environments, this is simple: everything is planned by managers months or even years in advance, and there are comprehensive guidelines in place to ensure compliance with these standards. However, in Agile environments, this changes. Requirements can come and go, and planning is done iteratively rather than all at once before the project gets off the ground. This fundamental difference in development style necessitates a change in management practices for managers to be able to perform these activities in an Agile environment (Aguanno, 2004).

In addition to changing the duties of managers, Agile also creates necessarily flat hierarchies, if not at the organizational level, then at least at the team level (Sochova, 2020). By placing the emphasis on the team rather than the individual, Agile implies that there is no distinction of power levels between people on a team, and all team members share all responsibilities of the software development process. Thus, the power of the traditional manager is split in Agile environments (De Smet, 2018). Oftentimes, the developers even take up some previously managerial duties like task delegation (Maruping et al., 2009). This demonstrates the ability Agile has to deconstruct traditional divisions of power and labor by delegating tasks that are traditionally associated with individuals in positions of power to the team. Thus, Agile sets itself up to break away from traditional management and control schemes.

With this new power redistribution, developers are more often able to choose what they want to do with respect to the product. This has the added benefits of allowing developers to make use of their domain-specific knowledge and increasing overall motivation (Puranam, 2022). Additionally, it has been shown that increasing the amount of control an employee has over their work can lead to reduced psychological strain in high stress environments (Karasek & Theorell, 1990). With up to 94% of software development companies reporting the use of some kind of Agile in the workplace and even more transitioning to Agile (Digital.ai, 2021), it becomes important to understand whether or not Agile truly delivers on its potential to redistribute power in software development companies so that companies can take advantage of the benefits that come by allowing their developers more freedom. Thus, this paper aims to discuss the degree to which these hierarchies change from plan-driven to Agile development, specifically in the context of a government contractor.

Background

Prior to the development of Agile processes, development was done via plan-driven processes. Plan-driven processes attempt to plan everything necessary for development (e.g. timelines, requirements, etc.) out before development, which can be seen in the prototypical plan-driven model: the Waterfall model (Royce, 1987). However, sometimes iteration is necessary, as requirements may (and often do) change (Lynn, n.d.). The Waterfall model, in particular, agrees that iteration may be, at times, necessary, but ideally, this iteration would only occur within the steps immediately before or after the one currently in progress, which limits the speed at which this iteration can occur. Other models like the spiral or iterative model attempt to address this by adding some iteration into the base Waterfall model, but these plan-driven methodologies all ultimately result in iteration being a drawn-out process (Rahim et. al, 2018).

Additionally, these plan-driven processes can take a long time to develop any form of product, which is undesirable, especially as requirements are often constantly changing as a project continues over long periods of time (Lynn, n.d.). Therefore, projects may be abandoned, as they no longer fit the need that they had been designed to fulfill. Thus, in 2001, a group of developers came together to create the Agile Manifesto to fix these systemic issues surrounding plan-driven development.

As the founding document for Agile development, the Agile Manifesto emphasizes several qualities that these developers thought the development process should have. In contrast to plan-driven development, it emphasizes several qualities, the first of which is the "continuous delivery of valuable software." What this means is that, rather than delivering one product at the very end of the development process, developers should instead aim to continually deliver features incrementally to give the customer an idea of how development is progressing. Additionally, the manifesto promotes the idea that changing requirements are welcome (Beck et. al, 2001). Combined, these two qualities of Agile sought to remedy some of the largest downfalls of plan-driven methodologies. Often, how these two properties are realized is through the use of shorter, iterative development processes than plan-driven development can achieve. For example, one of the most common Agile frameworks, Scrum, incorporates this idea by dividing work into shorter "sprints" of work, which are structured similarly to the waterfall model, but on much shorter timescales, with planning occurring in parallel with development rather than prior to development (Schwaber & Sutherland, 2020). This iteration allows for the rapid incorporation of changing requirements into a project as well as the incremental delivery of software that the Manifesto aspires to.

Literature Review

In order to understand how the power to make decisions has changed, we must first understand what duties of the different parties have traditionally been in the two styles of development. We will begin first with the role of the manager. In Agile, the primary role of management is to provide the development team with all of the resources they need to succeed in an Agile environment (Sarpiri & Gandomani, 2017). This may include training and coaching, but it also includes things such as setting up an environment in which self-organization is possible. That is, the project manager is responsible for removing obstacles to the successful operating of the team (Rothman, 2010), whether those be systemic organizational issues that limit the ability of the team to work together in a way that best suits them or even, if necessary, helping developers learn how to work with others. Further, the role of the product manager is not explicitly defined in Agile, and some of the responsibilities traditionally viewed as being the domain of the project manager have been distributed to the team at large (Fernandez & Fernandez, 2008). In general, the role of the project manager in Agile is to create an environment in which everyone involved feels that they can safely and effectively carry out development in Agile ways. In addition to ensuring that the process works, the project manager is responsible for more traditional management duties, such as budget management, managing project risk, and communicating with stakeholders, among others (Hati, 2023). In the traditional software development project, the project manager is in charge of initiating, planning, and then monitoring the progress of the project (Tarne, 2007). Further, in traditional projects, the project manager is responsible for the division of tasks between team members (Sergeev, 2016).

It is also important to understand why hierarchy persists and why reducing hierarchy may be beneficial. According to Harold Leavitt, a professor at the Graduate School of Business at Stanford, hierarchical organizations provide structure and clearly defined duties and

responsibilities to an organization (2003). Thus, he claims that hierarchies may provide some order and structure that humans, at some base level, seek out. However, in the same article, he claims that, despite the clarity of structure and organization that high levels of hierarchy provide, they may sometimes lead to abuses of power by those who rank higher in the hierarchy against those who rank lower. Additionally, in highly dynamic environments, high levels of hierarchy can bog down the ability of the organization to operate successfully (Mintzberg, 1989). On the other hand, however, flatter hierarchies, such as those seen in Agile, tend to respond faster to the changes that arise in dynamic environments. Thus, the flatter hierarchies that Agile promises (Sochova, 2020) may help organizations to react better to the dynamic nature of software development.

It has been shown, however, that, in creative projects, such as video game development, the introduction of Agile methodologies creates a flatter hierarchy in theory, but, in practice, a "soft" hierarchy can form (Hodgson & Briand, 2013). That is to say, team leaders and others in managerial positions seem to exhibit some degree of soft control over decisions made when creating the project. In addition to the soft control exhibited by team leaders and other managers in the flatter hierarchy of the project, the influence of higher management still exists, further reinforcing the notion of a hierarchy in the workplace. Thus, in these cases, Agile may fall short of its potential to redistribute power in the workplace. While game development is a subset of software development companies, these results may not necessarily translate generally to software development because of the more specialized structure of a game development company and the modifications that need to be made to Agile processes to adapt for use in this environment.

Methods

A large portion of the analysis of difference between hierarchies presented in this paper draws on Actor Network Theory (ANT). ANT is a method for defining and understanding the relationships between many, heterogeneous "actors" (Cressman, 2009). These actors can be either human or technological, and all actors are connected to each other via a series of relationships, creating a coherent network. In particular, ANT is useful for this work as it allows us to clearly illustrate the many connections between people and processes that are present in a software development environment. Thus, we can use any networks obtained during analysis to help us further our understanding of how duties and responsibilities shift over the course of a transition to Agile development.

In order to investigate this question, I conducted a series of interviews with employees of a particular government contracting company who have some experience in an environment in which a transition to Agile from plan-driven methodologies is currently or has already occurred. The questions asked of these employees were primarily focused on the processes used during their time at this firm, as well as the people who are involved in each of those processes. Specifically, I focused on the parts of the software engineering process where developers and project managers are in close contact with each other, leading to more opportunities for the delegation of responsibilities away from the project managers to developers. Using their responses, I used ANT to construct networks of people and processes from before transition and after transition. Using these networks, I synthesized the team and organizational hierarchies from the two networks and compared the two to see if the introduction of Agile delivers on the flatter hierarchies that it promises.

Synthesizing Hierarchies

This paper will focus primarily on the interviews conducted with two people who have experience in an environment where an organization is transitioning from plan-driven development to Agile development, using their experiences as case studies of these transitional environments. Throughout this paper, these people will be referred to by the pseudonyms Blaine and Flint. Blaine has a great deal of experience with helping various government organizations transition from plan-driven to Agile methodologies. Thus, he has a great deal of experience in the transitional environments we are interested in, and, given his diverse experience, his viewpoint speaks to a more general case of Agile implementation in government organizations. On the other hand, we have Flint, who helped a single government organization transition from plan-driven to Agile development and has worked in other Agile environments.

Blaine

From Blaine's interview, I identify three primary human actors for a plan-driven development environment: the Product Management Office (PMO), the Product Manager (PM), and the developers. The PMO's primary purpose is to generate requirements and, if the development team is currently overloaded or moving too slowly, hire outside contractors to do the work. The requirements that the PMO generates are then passed to the PM, who then passes them on to the developers. The PM's primary goal is to ensure the successful delivery of the product. To this end, they are also responsible for the creation and management of the schedule, which drives the tasks that are done and the timeline in which they are done. Further, this schedule is responsible for allocating developers to tasks, so the PM has a direct hand in determining which developers are to do what. The PM is also responsible, according to Blaine, for performing intermittent quality checks while the developers are working to ensure that the product is being created properly.

The developers, on the other hand, have relatively few powers. The general expectation of the developers in the plan-driven environment is that they complete their tasks and wait until they are assigned a new task. Developers are also able to identify problems and whether changes need to be made to develop the product. What they are not able to do, however, is carry out those changes without approval. Often, developers would have to request the ability to make changes through a change control process. This change control process would be handled by the PM, who, if the change was simple and would not pass a certain threshold of change to cost or time to complete, could approve the changes. However, changes that met that threshold would be escalated to a review board, which would then approve or deny this change request. Thus, developers had little influence on the work they were doing and the decision-making process that occurred over the course of that work. These interactions between the primary human actors and nonhuman processes which are, themselves, actors create the actor-network in Figure 1.

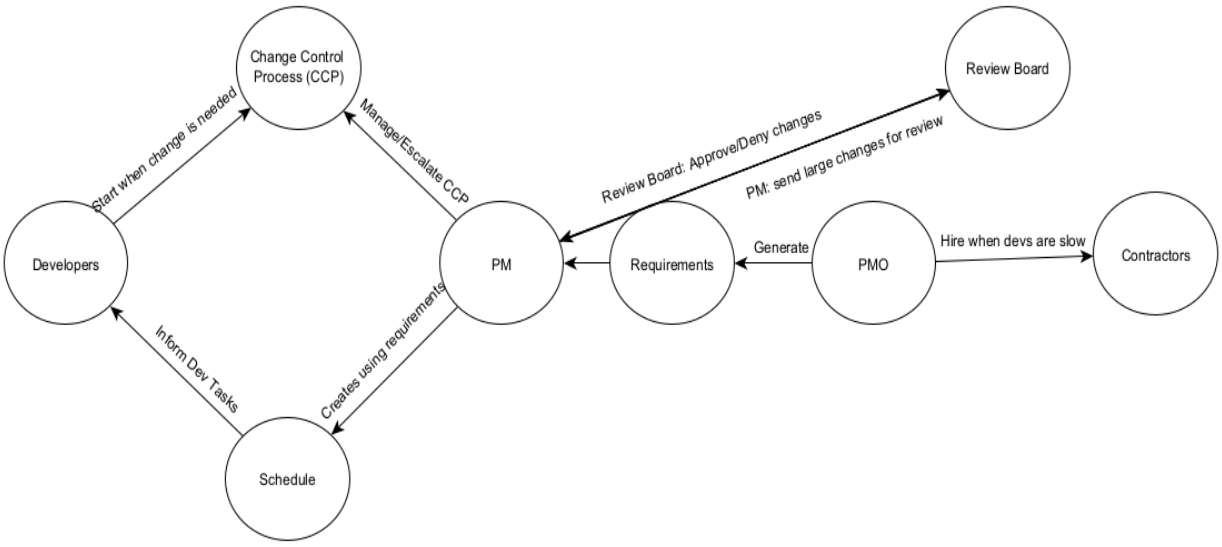

Figure 1 - *Plan-driven Actor-Network from Blaine's Interview*

From Blaine's interview, I identify three main human actors which interact in an Agile development environment. These are the developers, the scrum master, and the product owner (PO). Together, these three actors constitute the Agile team. Contrary to the plan-driven environment, where only one person (the PM) is responsible for the work the team does, the Agile team collectively takes responsibility for the work to be done. Of course, given the nature of development, there is some separation of duties. The Scrum Master is primarily responsible for several things. First, they are responsible for the removal of blockers. These blockers are anything that causes development to stop entirely (Dulin, 2022) and can range from a lack of requirements to outside requests to halt development for some reason. Additionally, the Scrum master is responsible for ensuring that development proceeds smoothly and, along with the PM, the assignment of developers to tasks.

In the Agile Team, the PO serves as the primary point of contact for the customer. Together, the PO and the customer work together to determine what work needs to be done, in what order, and then placing that work into a pre-specified unit of time known as a sprint. However, unlike the plan-driven case, this is done in parallel with ongoing development, rather than in series. Each sprint has a set of work assigned to it, known as its backlog, which the PO prioritizes the tasks in. The developers, on the other hand, are primarily responsible for the completion of the tasks assigned to them by the PO or Scrum Master. Further, if a developer completes a task earlier than expected, they can take on additional tasks to progress the sprint, giving them more decision-making power with regards to the work they do. In the case that the need for a change to the product arises, a more senior developer on the team is responsible for working with the PM to gauge how much what is planned for the current and future sprints will

have to change to accommodate the change in the product. The interactions between these human and non-human actors are illustrated in Figure 2.

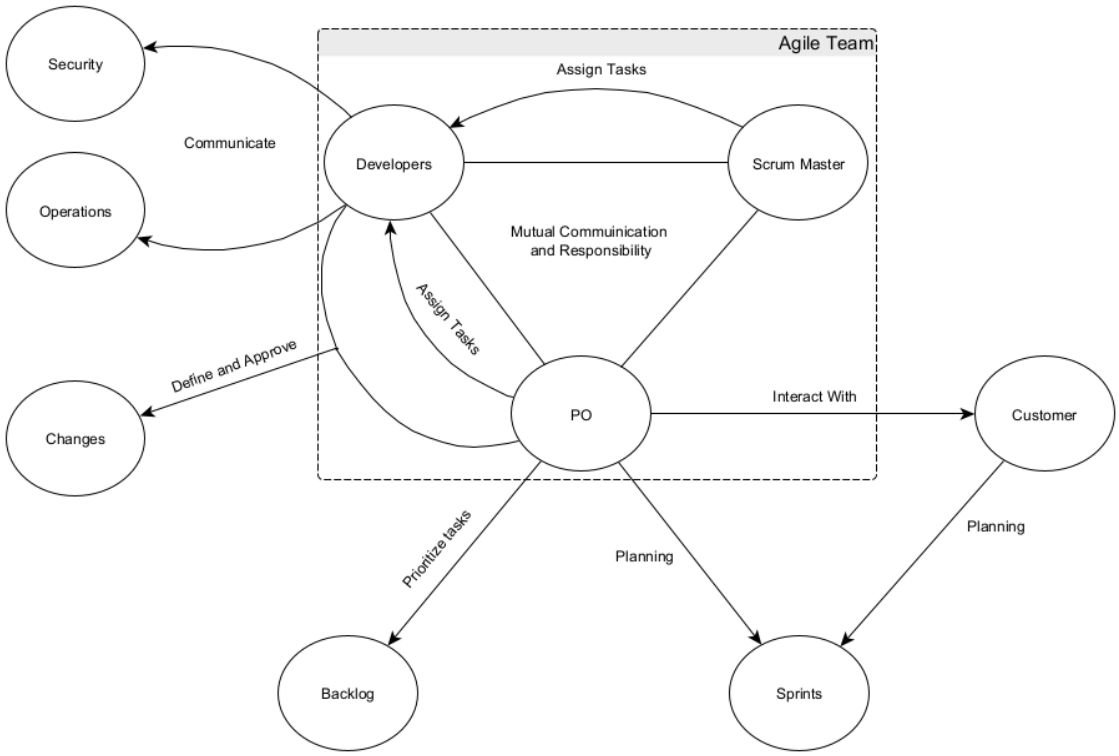

Figure 2 - *Agile Actor-Network from Blaine's Interview*

Compared to the plan-driven case, Blaine's picture of Agile is very clearly one of a flattened hierarchy. In the plan-driven case, the PM makes many of the decisions, and this decision-making process, in the worst case, can flow down from the highest levels of management. In Agile, however, most of the decision-making happens at the level of the Agile team, in which the developer, Scrum Master, and PO play equally important roles. Of course, there is some level of hierarchy inherent in the system. This is seen most clearly by the fact that a senior developer will work with the PM to gauge the damage caused by a change rather than the developer that necessitated the change. However, the picture of Agile, according to Blaine, appears to be one of

mutual responsibility and heightened communication between the different roles, which leads to development being more of a joint project between people of different skillsets rather than a single manager or team leader dictating what can and cannot be done.

Flint

When analyzing Flint's interview, I identified two primary actors in the plan-driven environment: the management team and the developers. We begin by discussing the management team and the processes in which they were involved. To begin the development process, users would submit a written request to management asking for the product to be made. From this written request and their discussion with the users as well as their own input, the management team would create the requirements. These would be analyzed, and management would use these generated requirements to develop a development plan. Once again, the plan is central to the development process, and management is creating this plan. This plan is composed of some set of tasks and the order in which they are to be done. Thus, prioritization of all tasks is done by management at the start of the development process.

The other primary actor in this scenario, developers, on the other hand, did not have as much power in decision-making as management did. When constructing a team of developers, management would select a project lead who was responsible for leading the rest of the developers in the development effort. Thus, this project lead was responsible for assigning developers to tasks. Even in the primary areas where one would expect developers to have some kind of decision-making power, management still held sway. If a developer identified a change that needed to be made, then they would have to ask management, and management would have to approve it. If developers wanted to use a library to perform a certain task, then they would

have to follow a similar process. In fact, in this network, there is a large amount of downward information and decision-making flow from management teams to the development teams. These interactions between actors are illustrated in Figure 3.

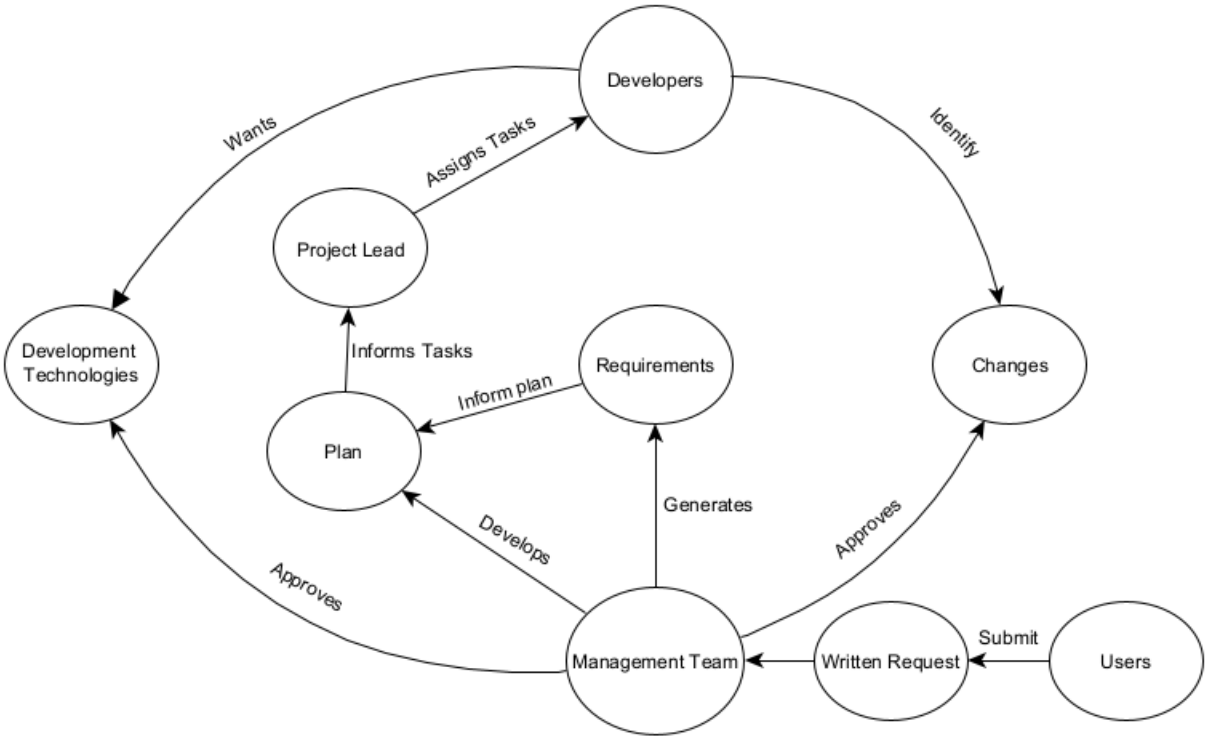

Figure 3 - *Plan-driven Actor-Network from Flint's Interview*

In the Agile environment, I identify the same two major actors as in the plan-driven case: the developers and the management as well as a third, additional actor, the Scrum Master. In the Agile environment, the role of management is somewhat simpler. Once again, the management is responsible for taking in user requests and breaking the project down into tasks. Rather than placing these tasks into a single plan, however, the management would prioritize these tasks and place them into a backlog. Here, the role of the management ends until the next cycle of iteration begins.

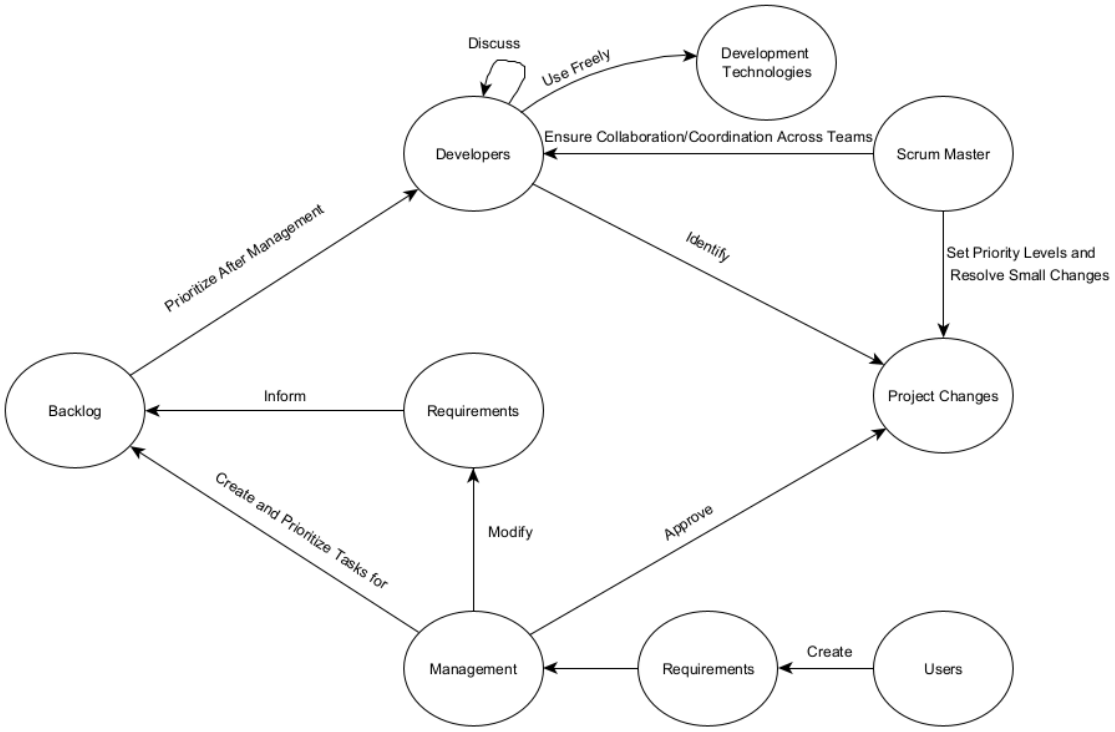

Figure 4 - *Agile Actor-Network from Flint's Interview*

The Agile team is composed of a Scrum Master and the other primary actor, the developers. In Figure 4, we can see the interactions that occur in the Agile environment between management, developers, and the Scrum Master. The role of the Scrum Master in the Agile team is to ensure collaboration between and coordinate across the different teams working on a project. Unlike in the plan-driven case, developers are more able to make decisions about how the system is implemented. If a group of developers wishes to use a library (a pre-written codebase) to accomplish a certain function, they can in the Agile case do so freely without having to contact and get permission from management. If they need to make a smaller change, they can do so without contacting management. However, the developers do not have the power to unilaterally make decisions about what is to be done. While developers can identify changes,

they are prioritized by the Scrum master. This priority is set based on the overall impact that making this change would have. If making this change would impact teams other than your own, then the Scrum master would kick the change back up to management to have it approved. However, to avoid this, there is often discussion happening between developers on different teams. This discussion allows developers to ensure before going to management that their change will not impact other teams, allowing for coordination amongst different teams without having to go through the middleman of management.

There is some level of similarity between the networks of the Agile and plan-driven cases that Flint describes. For example, users always go through management to begin the development process, and management will always work with users to extract requirements and send those requirements down to developers along with some of their own requirements. Further, management is also responsible for the definition of the tasks to be done and the prioritization of those tasks in both cases, though the form of how this happens is slightly different between the two. Where they differ, however, is in the ability of the developer to make decisions. In the plan-driven case, any decisions must be, ultimately, made by management. This leaves developers unable to make such simple decisions as using a library to accomplish a task without having to go through management. However, in the Agile case, some of the decision-making is "pushed down" to the developer level, allowing developers to use a library if they choose or even make small changes to the work that is done, if they determine that this change would not take a large amount of time or effort and this change would not affect other teams. Thus, while the bulk of decision-making about what to do and in what order happens at the higher levels of management, there is some decision-making that occurs at the developer level as well. Thus, there is still some organizational hierarchy, with management being at the top of that hierarchy, though the

introduction of Agile flattens it to allow for some smaller decision-making to occur at the developer level.

Conclusion

Given the networks developed in the previous section, it is evident that Agile does deliver on its promise to flatten hierarchies in a software development environment at least a small amount. However, the extent to which flattening occurs depends largely on the organization which is implementing Agile. That is, if the organizational hierarchy does not change to reflect Agile principles, or if, in practice, one role takes on more than they should to become a more leading role, then the hierarchies that exist in a plan-driven environment may persist, even after transitioning to an otherwise Agile style of development. Perhaps this is exacerbated by the bureaucratic nature of many government organizations which rely on hierarchy to organize work. However, even in the case of these highly bureaucratic organizations, Agile still provides some flattening of hierarchy, at least at the level of the development team, if still maintaining much of the usual hierarchical structure of the organization.

One limitation of this work is its reliance on interviews. While they provide valuable data, they are somewhat reliant on the ability of the participant to remember all that they want to say. Thus, sometimes more minute details can be left out, and minutiae missed. To combat this, future iterations of this study could instead conduct a more ethnographic investigation, which allows the researcher to question the participants directly, as is possible during an interview, while also being able to take note of how people interact with each other, body language, and other factors that get lost in an interview. However, despite this limitation, we can conclude that Agile, despite

potentially being bogged down by bureaucracy at the highest levels of management, can still provide developers with a degree of freedom that allows them to do their best work efficiently.

References

Aguanno, K. (Ed). (2004). Managing Agile Projects is Different. In *Managing Agile Projects*. (pp. 69-74). Multi-media Publications Inc.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. & Thomas, D. (2001). *Manifesto for Agile Software Development.* Manifesto for Agile Software Development. https://agilemanifesto.org/

De Smet, A. (2018). The agile manager. *The McKinsey Quarterly*, *3*, 76-81.

Digital.ai. (2021). *State of Agile Report*. (Report No. 15). Digital.ai. https://digital.ai/resource-center/analyst-reports/state-of-agile-report/

Dulin, M. (2022). Blocker vs Impediment (It's Not Just Semantics). *AgileAmbition*. https://www.agileambition.com/blocker-vs-impediment/

Fernandez, D. J., and Fernandez, J.D. (2008) Agile Project Management – Agilism Versus Traditional Approaches. *The Journal of Computer Information Systems*, *49*(2), 10-17.

Hati, S. (2023). Project Manager Role in Agile: Job Description, Responsibilities. *KnowledgeHut*. https://www.knowledgehut.com/blog/agile/the-role-of-a-project-manager-in-an-agile-environment

Hodgson, D., & Briand, L. (2013). Controlling the uncontrollable: 'Agile' teams and illusions of autonomy in creative work. *Work, Employment and Society*, *27*(2), 308–325. https://doi.org/10.1177/0950017012460315

Karasek, R., and Theorell, T. (1990). The Environment, the Worker, and Illness: Psychological and Physiological Linkages. In, *Healthy Work: Stress, Productivity, and the Reconstruction of Working Life.* (pp. 83-116). Basic Books, Inc., Publishers.

Leavitt, H. J. (2003). Why Hierarchies Thrive. *Harvard Business Review*. https://hbr.org/2003/03/why-hierarchies-thrive

Lynn, R. (n.d.). The History of Agile. *plainview.* https://www.planview.com/resources/guide/agile-methodologies-a-beginners-guide/history-of-agile/

Maruping, L. M., Venkatesh, V., & Agarwal, R. (2009). A Control Theory Perspective on Agile Methodology Use and Changing User Requirements. *Information Systems Research*, *20*(3), 377–399.

Mintzberg, H. (1979). The Structuring of Organizations. In D. Asch & C. Bowman (Eds.), *Readings in Strategic Management.* (pp. 322-352). *Red Globe Press London.* https://doi.org/10.1007/978-1-349-20317-8_23

Puranam, P. (2022). Should Employees Be Allowed to Choose What They Want to Do? *Insead Knowledge.* https://knowledge.insead.edu/leadership-organisations/should-employees-be-allowed-choose-what-they-want-do

Rahim, S., Chowdhury, A. E., Hakim, S., Nandi, D. & Rahman, M. (2018). ScrumFall: A Hybrid Software Process Model. *International Journal of Information Technology and Computer Science.* 10(12), 41-48. https://doi.org/10.5815/ijitcs.2018.12.06

Rothman, J. (2010). Agile Managers: The Essence of Leadership. *Cutter Business Technology Journal*. https://www.researchgate.net/publication/291461836_Agile_managers_The_essence_of_leadership

Royce, W.W. (1987). *International Conference on Software Engineering '87: Proceedings of the 9th internation conference on software engineering.* https://blog.jbrains.ca/assets/articles/royce1970.pdf

Sarpiri, M.N., & Gandomani, T.J. (2017). How Agile Managers Affect the Process of Software Development?. *IJCSNS International Journal of Computer Science and Network Security, 17*(5). Retrieved Febrauary 3, 2023, from http://paper.ijcsns.org/07_book/201705/20170538.pdf.

Schwaber, K., and Sutherland, J. (2020). *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game.* Scrumguides. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf#zoom=100

Sergeev, A. (2016). *Team Roles in Waterfall Methodology*. Hygger. https://hygger.io/blog/team-roles-in-waterfall-methodology/

Sochova, S. (2020). Hierarchy. *Agile and Scrum Blog.* https://agile-scrum.com/2020/04/01/hierarchy/

Tarne, B. (2007). *Don't throw the baby out with the bathwater: How to combine and use both agile and traditional project management approaches.* [Paper presentation]. PMI® Global Congress 2007—North America, Atlanta, GA. https://www.pmi.org/learning/library/combine-agile-traditional-project-management-approaches-7304