The Product Backlog

Conference Paper · May 2019

DOI: 10.1109/ICSE.2019.00036

3 authors:

Todd Sedano
Pivotal Software
28 PUBLICATIONS **152** CITATIONS
SEE PROFILE

Paul Ralph
Dalhousie University
79 PUBLICATIONS **1,038** CITATIONS
SEE PROFILE

Cécile Péraire
Carnegie Mellon University, Silicon Valley
34 PUBLICATIONS **234** CITATIONS
SEE PROFILE

The Product Backlog

Todd Sedano
Pivotal Software
Palo Alto, CA, USA
Carnegie Mellon University
Silicon Valley Campus
Email: professor@gmail.com

Paul Ralph
University of Auckland
Auckland, New Zealand
Email: paul@paulralph.name

Cécile Péraire
Carnegie Mellon University
Electrical and Computer Engineering
Silicon Valley Campus
Moffett Field, CA 94035, USA
Email: cecile.peraire@sv.cmu.edu

***Abstract*—*Context:* One of the most common artifacts in contemporary software projects is a *product backlog* comprising user stories, bugs, chores or other work items. However, little research has investigated how the backlog is generated or the precise role it plays in a project.**

***Objective:* The purpose of this paper is to determine what is a product backlog, what is its role, and how does it emerge?**

***Method:* Following Constructivist Grounded Theory, we conducted a two-year, five-month participant-observation study of eight software development projects at Pivotal, a large, international software company. We interviewed 56 software engineers, product designers, and product managers. We conducted a survey of 27 product designers. We alternated between analysis and theoretical sampling until achieving theoretical saturation.**

***Results:* We observed 13 practices and 6 obstacles related to product backlog generation.**

***Limitations:* Grounded Theory does not support statistical generalization. While the proposed theory of product backlogs appears widely applicable, organizations with different software development cultures may use different practices.**

***Conclusion:* The product backlog is simultaneously a model of work to be done and a boundary object that helps bridge the gap between the processes of generating user stories and realizing them in working code. It emerges from sensemaking (the team making sense of the project context) and coevolution (a cognitive process where the team simultaneously refines its understanding of the problematic context and fledgling solution concepts).**

***Index Terms*—Product backlog, dual-track agile, scrum, lean, extreme programming, user stories, design thinking, user-centered design, feature engineering**

I. Introduction

A *product backlog* (or simply *backlog*) is a list of work items (e.g., user stories, outstanding bugs, various chores) used by software teams to coordinate work to be done. Developers typically build new features, modify existing features, and fix bugs based on the items at the top of the backlog. For many teams, the backlog is the primary project management artifact or primary non-code artifact. Whoever fills and prioritizes the backlog directs the product's development.

Much research has addressed the process of constructing software systems, often based on backlog items. Scrum, Extreme Programming, and Kanban all concern building working software based on some kind of work item list or pool.

However, little empirical research has explored how backlog items are generated, refined, and organized over time. A common but naïve response is that you just ask potential users or clients what they want. This does not explain how backlog items are generated; it just shifts the burden of designing the system onto the user.

Similarly, little empirical research has investigated the roles or functions of the backlog in contemporary software projects. Again, a naïve response is to assume a backlog is some agile analogue of a software requirements specification. As we shall see in Section III-C, a product backlog is not a requirements specification.

While the concept of a backlog comes from Agile methods, the methods literature does not answer these questions. Agile method(ologie)s tend to focus on: the organization of development (e.g., sprints, team member roles); intra-team communication (e.g., stand-up meetings, burndown charts); and development practices (e.g., continuous integration, pair programming). They tend to emphasize working closely with users to co-design the product but do not directly address how to determine what to build.

Pre-agile methods typically recommended "eliciting requirements" from stakeholders (including potential users) and then refining them until the solution is defined. This approach can have two problems:

1) Users typically do not have system requirements. Users have vague, unreliable, conflicting, evanescent preferences, values, beliefs, and guesses about what might make a product successful [1], [2].
2) Design is a creative process—one must *imagine* a solution [3], [4]. Pre-agile methods offer little guidance for imagining good solutions.

The lack of existing research specifically investigating the product backlog motivates the following research question:

Research Question: *What are product backlogs, what are their roles, and how do they emerge?*

This article describes the research method (Section II), summarizes the results (Section III), continues with a brief discussion (Section IV), and overviews related work (Section V). Section VI concludes the paper by summarizing its main contributions.

II. Research Method

We conducted a longitudinal study at Pivotal Software—a large, international software company with twenty offices around the world—over two years and five months. We used

Constructivist Grounded Theory [5], which involves iteratively collecting and analyzing data to generate and refine a theory. We began by asking, "What is happening at Pivotal when it comes to software development?" Data collection included participant observation (that is, simultaneously observing and participating in activities), interviews, questionnaires, and examining organizational documents and records. We analyzed this data using constant comparison and several qualitative analysis techniques, as described below.

Because this study is so large, multiple core categories emerged, including *sustainable software development* [6], *team code ownership* [7], *software development waste* [8], and the topic of this paper: *the product backlog*.

As the *product backlog* core category emerged, we incorporated data from a survey of the company's product designers, performed additional interviews, and continued participant observation to refine the category. We constantly compared emerging findings to data from these three sources until reaching saturation, as described below.

A. Research Context

We studied Pivotal because it is successful, open to research, and interesting in its use and evolution of Extreme Programming. More specifically, we studied Pivotal Labs: a division of Pivotal that helps to transform clients' engineering cultures while building products. Client personnel work with Pivotal personnel at a Pivotal office where they can experience Extreme Programming [9] in an environment conducive to agile development.

Typical teams include about six software engineers, one product designer, and one product manager. In some smaller teams, one person fills both the product designer and product manager roles. Larger projects are organized into smaller coordinating teams, each with one product manager and one or two product designers.

Pivotal has practiced Extreme Programming [9] since the late 1990s. While each team autonomously decides what is best for each project, the company culture strongly suggests following all of the core practices of Extreme Programming, including pair programming, test-driven development, refactoring, weekly retrospectives, daily stand-ups, prioritized backlog, and team code ownership.

We observed eight greenfield and brownfield projects from various domains, including automotive, networking, healthcare, publishing, e-commerce, virtual machine management, and a multi-node database. Each project had one to three product managers, zero to two product designers, and four to 28 developers. Project details are available elsewhere [10].

B. Data Collection and Analysis

This paper analyses more diverse data than is typical in grounded theory studies. Participant observation produced voluminous field notes, supported by internal documents and code artifacts. Intensive interviews of 56 product designers, product managers, and software engineers who had experience with Pivotal's software development process from ten different offices helped us enter into participants' perspectives. ("Intensive interviews" are "open-ended yet directed, shaped yet emergent, and paced yet unrestricted" [5].) An informal online questionnaire, with responses from 27 of 71 total product designers (a 38% response rate), produced further insights into their practices.

The first author iteratively collected and analyzed field notes and interviews; used line-by-line coding [5] to identify nuanced interactions in the data and avoid jumping to conclusions; and reviewed the initial codes while reading the transcripts and listening to the audio recordings. We discussed the coding during weekly research collaboration meetings and recorded insights from these discussions as grounded theory memos [11]. We stored initial codes in a spreadsheet and used constant comparison to generate focused codes.

The first author routinely compared new codes to existing codes, refining them and eventually generating categories. We periodically audited each category for cohesion by comparing its codes. When this became complex, we printed codes on index cards to facilitate reorganization. We wrote memos to capture the analysis of codes, examinations of theoretical plausibility, and insights. Eventually, a small number of core categories, including *the product backlog*, emerged. An online appendix contains examples of categories and coding.[1]

We drew on other relevant documents, code artifacts, and questionnaire results to elaborate and saturate categories. Since line-by-line coding of such documents is intractable, we instead wrote memos about them, as recommended by Glaser [12, p. 98].

We continued theoretical sampling on all data sources until no further backlog-related categories were evident (i.e., theoretical saturation). Theoretical saturation was not truly achieved until midway through writing this paper, which is normal for Grounded Theory studies [5, p. 285]. The articulation of the results, below, was the key indicator that saturation had been reached.

Data collection and analysis were continuous but not uniform (e.g., some interviews were too close for much intervening analysis). Similarly, some weeks we did considerable analysis; others very little. This is not ideal but is unavoidable in a longitudinal inquiry where everyone has conflicting commitments. A more detailed account of our data collection and analysis strategy is available elsewhere [10], [13].

III. Results

Generating, refining, and sequencing solution concepts, features, stories, and eventually the backlog appears to be one large, tightly-interconnected process that cannot be divided into neat phases. However, we can divide our observations into interrelated practices (Section III-A) and obstacles (Section III-B) that resonate with participants and then abstract them into a more transferable theory of the backlog (Section III-C).

[1]https://www.researchgate.net/publication/330823601_The_Product_Backlog_-_Grounded_Theory_Codes

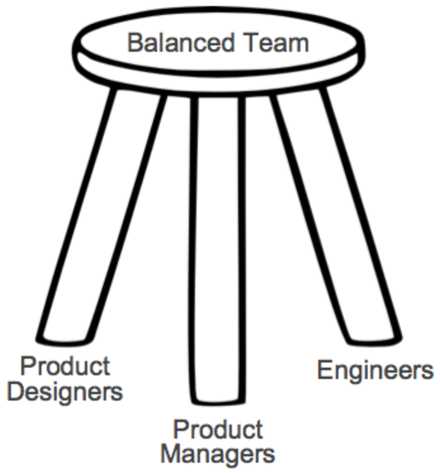

Fig. 1: Balanced Team

A. Practices

We observed 13 practices related to backlog generation and refinement. They are summarized in Table I, and their relationship to the backlog is illustrated in Figure 2. This section elaborates each practice. While we present the practices in a neat, idealized sequence for clarity, in reality their execution order is messy, with parallelism, backtracking, and interdependency.

1) Balanced Teams and Dual-Track Agile: Pivotal uses two important organizational systems: "balanced teams" for people and "dual-track agile" for activities.

A *balanced team*, as illustrated in Figure 1, "is an autonomous group of people with a variety of skills and perspectives that support each other towards a shared goal" [14]. Here is how one Pivotal designer explains balanced teams:

"Design is the voice of the users; its main role is to advocate for the user's needs. Development is the voice of the system; its main role is to help the team understand what's possible and advocate for good system architecture. Product management is the voice of the business; its main role is to advocate for the business's needs and facilitate prioritization of the team's backlog" (Participant 16).

In other words, the teams we observed included members with diverse backgrounds, who combine business knowledge (product managers), design skills (product designers), and full-stack development / testing skills (engineers). All three roles are needed to keep the team balanced, like the three-legged stool shown in Figure 1. If a role is under- or over-represented, the team becomes unbalanced, and the product may suffer.

Meanwhile, *dual-track agile* is a kind of continuous software engineering [15] in which activities are organized into two groups. We call these groups "tracks" because they are continuous and parallel, like train tracks. Track one typically includes research, negotiating with stakeholders, drawing user interface mockups, and writing user stories. Track two typically involves building, testing, architecting, refactoring, deploying, and maintaining the product. **The whole team is involved in both tracks**, but product designers focus on track one while engineers focus on track two. Ideally, track one stays ahead of track two so engineers always know what to do next. These tracks are **not** phases, and the team does **not** iterate between them.

These practices create the need for a backlog. Dual-track agile creates the boundary that the backlog helps to span. Balanced teams create the conditions under which a backlog, rather than a different kind of artifact, is useful for spanning the boundary between the two tracks.

2) Persona Modeling, Stakeholder Mapping, Interviews, and Affinity Mapping: We observed product designers leading the team in defining one or more proto-personas. "A *persona* is a description of a fictitious user based on data from user research" [16, p. 58; emphasis ours]. A *proto-persona* is a description of a fictitious user that is based on intuition or simply made up [17]. Personas typically represent a group of similar users (or potential users) with a picture, motivational quote, demographic information, behaviors, needs, and goals. Participant 56 explains: *"When we're doing exploratory research, we are looking for patterns in people that can form a persona. We are looking for patterns in what these people do that are problematic."*

We also observed some product designers leading the team in drawing a map of a product's stakeholders—especially individuals inside the client organization (e.g., product sponsor, related business units, marketing, executives).

The product designers used their proto-personas and stakeholder maps to determine who to interview.

We observed product designers interviewing clients, current users of an existing system, or people in a particular demographic (e.g., drivers of Korean cars) who might use a new system. All observed projects and nearly all surveyed product designers relied on user interviews. Interviews primarily asked open-ended, non-leading questions; some used interview scripts or topic maps as guides. In some interviews, the interviewee would use the product at home, at work, or at a Pivotal office. The entire team often attended.

While the interviewer asked questions, one or more team members wrote key ideas on sticky notes. Different color sticky notes were used for each interview to facilitate traceability.

Interviewees share their ways of working, beliefs, ideas, values, attitudes, and preferences; however, interviewees do not consult an internal directory of enduring preferences. Rather, the interviewee and interviewer spontaneously co-construct evanescent preferences [2]. The purpose of these interviews is to *understand* the interviewee, not to elicit requirements or discover solutions. None of the product designers we observed use the term "requirements." Participant 42 explained: *"My goal is to listen. Once a client asked me, 'what do you make of all the interviews you've done so far?' I reminded the client, 'at this stage, my goal is to listen and absorb, my processing will come soon."'* Product designers intentionally resist solution-focused thinking (which they call "solutionizing") during interviews.

TABLE I: Practices Related to Product Backlogs

Name	Brief Description
Balanced teams	Balancing team composition with experts in business, product design, and software development.
Dual track agile	Organizing the work into two "tracks." Track 1 typically includes research, negotiating with stakeholders, drawing user interface mockups, and writing user stories. Track 2 typically involves building, testing, architecting, refactoring, deploying, and maintaining the product.
Stakeholder mapping	Drawing a diagram of individuals who are interested in the success of the product.
Interviewing	Semi-structured discussions with stakeholders (e.g., users, product sponsor).
Persona modeling	Creating fictional users (character sketches) to reason about who will use product features.
Affinity mapping	Organizing data from user interviews or ideation sessions to generate insights.
Design studio	Converging on a product concept by iterating between generating and discussing design ideas.
Sketching / mockups	Drawing informal models of graphical user interfaces.
Usability testing / validation testing	Reviewing mockups with users.
Writing user stories	Writing brief, informal descriptions of some aspects of a software system.
Story showcase	Building a shared team understanding of upcoming user stories.
Backlog grooming	Refining and resequencing user stories.
Accepting stories	Evaluating delivered work.

Product designers would stop interviewing when they felt that they understood users' needs around a particular feature set and would resume interviewing when additional information was needed. *"I interview until I do not learn new things from interviews"* (Participant 52). They typically began with about five interviews per proto-persona [18]: *"I have had projects where there are multiple personas so I would say like five for one persona is good"* (Participant 41).

Product designers use interviews to refine their proto-personas into user personas. They believe that personas help the team understand and empathise with users, and therefore create products that better support user needs and goals. *"This builds empathy for your user base. Going through [these practices] really helps with decision making moving forward"* (Participant 41). Later, they use personas for "persona validation;" that is, predicting whether the user group represented by a persona will use a product, feature, or solution concept. After interviewing several people, product designers sometimes realized that the persona would not use the product. The team then had to choose whether to redesign the product for the opportunity provided by the target persona (i.e., "invalidate the product") or find a new demographic that will use the product concept (i.e., "invalidate the persona").

We observed teams synthesizing what was learned during interviews using *affinity mapping* [19]. Teams clustered sticky notes from user interviews on a large whiteboard or wall. They would place and rearrange the notes into cohesive clusters, leading to "insights" about user needs. Ad hoc, informal conversations would often trigger rearranging the notes and clusters. Each new design insight was recorded on a new sticky note and added to a special "design insights" list. After all the sticky notes were on the board and all clusters finalized, the team would discuss each cluster and its related insights on user needs. Some teams perform this exercise one interview at a time to incorporate the data quickly; others do a batch of interviews to allow the conversations to blend and percolate. Many participants refer to this process as "synthesis."

Participants see adding customer requests directly to the backlog as an anti-pattern: it *"creates a Frankenstein amalgam of features. Microsoft products exhibit this. Instead, we want insight-synthesized research. Patterns emerged from the synthesis of data. Insights earned their way"* (Participant 17).

Persona modeling, stakeholder mapping, interviews, and affinity mapping help the team learn about the domain. This knowledge informs the user stories or other items that constitute the product backlog.

3) Design Studio, Sketching Mockups, Usability Testing / Validation Testing, and Writing User Stories: Once product designers felt that they had a reasonably good understanding of user needs, they generated solution concepts to satisfy those needs. For instance, we observed product designers leading the team in what some participants call "design studio." Each member individually sketched mockups of webpages or mobile screens and illustrated how users would interact with them. Teams iteratively examined the mockups and drew new ones, leveraging each others' ideas. Teams eventually converged on a promising solution concept. Designers iterated between refining the mockups and conducting further user interviews (in which they shared mockups with potential users, provided a task or goal, watched the users interact with the mockups, and listened to the user's feedback). This "usability testing" would continue until the new feedback seemed inconsequential. It would either validate or invalidate each feature represented by the mockups.

As a product and the team's understanding of users mature, product designers did less exploratory research and more validation and usability research. They relied more on telemetry from working software. However, when new assumptions or questions arose, designers often reverted to interviewing.

In all projects, we observed product managers writing user stories based on mockups, discussions with product designers, and their own research. User stories were typically written only for features with positive usability feedback.

A *user story* is a brief, informal description of some aspect of a software system. Large, vague stories are called *epics*. While user stories vary widely between organizations, most observed stories included a motivation and acceptance criteria. The brief motivation statement followed the pattern: *As a <user> I want to <action> so that <value>*. This is sometimes called the Connextra template [20]. The acceptance criteria followed the pattern: *Given <context>, when <condition> then <action>*. This is sometimes called Gherkin syntax [21].

When working on a new feature, the product manager would list several story titles (e.g., "Michael logs in," "Michael creates an account," "Michael logs out") in a sequence and then elaborate and refine each story, sometimes combining small stories or splitting up large ones. Product managers aimed to identify the smallest possible features that still provide user value. Stories are typically composed directly in Pivotal's project management software, *Pivotal Tracker*.[2] Stories are often linked to a relevant mockup: *"I try to attach [mockups] to every story. I try not to write a story before a [mockup] is done, and that works out really well"* (Participant 1).

No simple mapping links user interviews, user needs, design insights, mockups, epics, and user stories. The team does **not** *translate* interviews into user needs and design insights, *transform* insights into mockups, *deduce* epics from mockups, or *decompose* epics into smaller stories. All of these verbs misrepresent the cognitive process of designing. Such terms present designing as *symbol manipulation*—like solving high-school algebra problems. You cannot simply rearrange interview notes into user needs and design insights. Each insight entails a creative leap to one small area of an enormous, ambiguous, multidimensional solution space.

When product designers draw a mockup, they are *informed by* their multifaceted understanding of the project context, including their memories of user interviews, notes, design insights, education, experience, preconceptions, and biases. Similarly, when product managers write stories, they are *informed by* their multifaceted understanding of the project context as well as relevant mockups. **Each transition is a predominantly improvised, creative act.**

Much has been written about how design thinking involves—**but is not limited to**—writing, drawing, problem-solving, and decision making (cf. [22], [23]). Critically, the evolving solution concept triggers reframing of the project context. This oscillation between ideas about the project context and ideas about potential solutions is essential to design thinking (see Section III-C).

Design Studio, sketching mockups, usability testing / validation testing, and writing user stories all contribute to the generation of backlog items.

4) Story Showcase, Backlog Grooming, and Accepting Stories: We observed teams meeting to discuss upcoming stories. The product manager and product designer(s) shared their understanding of the stories with the engineers. While the product manager wrote the stories before the meeting, teams often revised stories, combined smaller stories or broke up larger ones, and clarified acceptance criteria. The engineers would examine the acceptance criteria to better understand what 'done' means for each story and discuss the effort required. During these meetings, team members broadened their shared mental model of the work to be done, and everyone would leave with a deeper, clearer understanding of the project context, desired product, and immediate stories.

We call this meeting a *story showcase*. For historical reasons, some participants call it an "iteration planning meeting," even though most projects are iteration-less, and the meeting is mostly about *sharing* not *planning*. Unlike planning meetings in Extreme Programming and Scrum, the team does not commit to the work to be done in a specified period. The product manager may change or re-prioritize any unstarted story.

We observed product managers periodically (e.g., right before a story showcase) examining the backlog to verify that stories were developer-ready and prioritized to deliver the most value to the user quickly.

We did not observe product managers prioritizing the entire backlog. Instead, product managers appear to move just enough high-priority stories to the top of the backlog to stay ahead of the engineers. While the backlog appears as a continuous list, there is an imaginary line between the top *n* stories, which have been recently prioritized, and stories *n+1* onwards, which may have once been prioritized but are now in no particular order. In some instances, we observed engineers getting ahead of the product manager and beginning story *n+1* without realizing that they had moved into the unprioritized section of the backlog.

We observed product managers shaping the product direction by resequencing stories based on conversations with stakeholders about user research and business needs. We also observed teams asking product managers to resequence some work to make the story implementation easier (e.g., building user creation before user login).

Backlog grooming is the process of reviewing and refining a backlog. It includes both reviewing stories for clarity and completeness and sequencing (or resequencing) stories.

We use the term *accepting stories* to denote a process we observed in which product managers evaluate delivered work. The product manager would verify that the implemented story functioned according to the provided acceptance criteria. If it did not, the product manager would reject the story (necessitating rework).

Story showcases, backlog grooming, and accepting stories all involve managing the backlog. The story showcase also helps the team develop a shared understanding of upcoming stories in the backlog.

B. Obstacles

The teams we studied routinely encountered numerous obstacles revolving around a general theme of stakeholders misunderstanding the nature of product design. The term *design* can refer to many different things even within software engineering. In the broadest sense, design means specifying the properties of an object [24], including what it does (feature selection), how users interact with it (interaction design), how it looks (visual design), how it is organized (architectural

[2]Publicly available at pivotaltracker.com

design), and exactly how it works (low-level design). Let us consider some specific events from our research.

1) Preconceiving Problems: In a government agency project, Participant 25 felt that the team found the problem they were told to find: *"They told us to go and look at grant life cycle management, so we went and looked at grant life cycle management, and then, ultimately, we created a prototype for managing a grant life cycle..."* The designer was frustrated because they saw better ways of framing the problematic situation, but there was too much momentum around solving the pre-conceived problem.

This misunderstanding concerns the way design problems are framed. Software engineering literature often assumes that the problem to be solved is simply known or given. In reality, stakeholders often agree that a situation is problematic but disagree on the exact nature of the problem(s) [1]. Moreover, framing a problem and generating a solution concept are *the same cognitive process* [25]. Overspecifying a problem for the designer actually impedes effective solution generation [26].

2) Preconceiving Solutions: A more extreme version of preconception is a client arriving with a dubious problem *and* an ill-considered solution concept. In one project, the client wanted a healthcare application. They had in mind a very detailed, extensive feature set and a particular proto-persona who would use these features. After several interviews with prospective users, the product designers invalidated the proto-persona. The client argued that 'the interviewees who gave us that feedback were not our real end users' and identified a new proto-persona, which was also invalidated by research. The client identified a third proto-persona that was again invalidated. The client rejected good user research because it invalidated their business model. The team relented and built the product the client wanted. A year later, the company dismissed their engineers because no one used their product.

"The stakeholders can assume that they know what the problem is and what the solution is before we have really had time to process what we have learned. We will start interviewing. People are saying 'I know it! We do not have to keep doing this research,' and I am saying 'no, no, we have only talked to two people, we do not know what is there yet, we do not know if that is the solution" (Participant 41).

At Pivotal specifically and in the interdisciplinary design literature more generally, solution concepts emerge from a complicated process of studying stakeholders, simultaneously framing a problem and generating fledgling solution concepts, sharing initial results with stakeholders, and elaborating a tentative solution. Clients arriving with premature problem frames or solution concepts, which have not emerged from careful research and reflection, obstruct design.

3) Pressure to Converge: Creative design is closely related to *divergent thinking* [27]; that is, exploring many dissimilar ideas. Insufficiently exploring the design space and converging prematurely on a poor solution concept is therefore a major threat to innovation.

We observed teams spending several weeks researching the problematic context, generating solution concepts, and validating feature ideas with prospective users. In multiple projects, we observed teams iterating on many ideas in a short period, then selecting one with which to move forward. The period of entertaining divergent ideas was always brief:

"The sketching exercises take us about two to five hours. We are divergent with many different ideas, and then we keep doing it until convergence" (Participant 51).

To the participants, their convergence on a fledgling solution concept in a few hours does not feel premature. Participants feel that they resist early "solutionizing," consider many alternatives, and have the flexibility to pivot to another solution depending on the findings. Some designers feel that the diversity of a balanced team leads to more creative solutions: *"Oftentimes, the developers give us the most interesting points of view; they flip the designer's model with their sketching"* (Participant 44).

To an outsider, however, given the complexity of the projects involved, two to five hours does not seem like much time to generate and consider many genuinely innovative solution concepts. What is clear is the tension between time pressure and divergence. Some clients push for an aggressive schedule and dismiss divergent thinking as a waste of time.

Expert designers in other fields exhibit similar patterns [22]. Despite the common advice to generate several alternative solution ideas, expert designers tend to converge rapidly on a single concept and resist radical reformulations [23].

4) Ambiguity: We observed participants struggling with ambiguity on several occasions. Survey respondent 25 said, *"It felt odd to walk in every day not knowing what you would tackle or try to learn."* Indeed, this happens so often that many product designers have grown accustomed to stakeholders panicking over ambiguity. Participant 41 explained that *"people can be uncomfortable with how open-ended and exploratory it is; sometimes the amount of unknown is very uncomfortable."*

A common sequence we observed begins with a client arriving with some idea about "the problem" and a tentative solution, and expecting the team to refine and elaborate them. Instead, user research tends to undermine preconceptions *before* offering new insights. Structure decreases before increasing. Participants often find this stressful and confusing.

"Most of the time, when clients are upset, it is because we have not managed the ambiguity well ... [Retaining their] tangible idea of what the product is going to be is extremely stressful because their [envisioned] product may not actually be what is right for them" (Participant 25).

Even when the team does not have a preconceived solution, embracing ambiguity is challenging. On a disaster assistance project, an experienced product designer expected the process to produce more questions and more ambiguity before finding a solution. As ambiguity increased from broad user data, the team became pessimistic while she was optimistic. The team kept saying "there is nothing to solve," but she persevered and discovered "enough nuggets" to design mockups for two products. The team picked one and released a product that satisfied the users' needs.

Ambiguity is unavoidable when designing systems for complex organizations with multiple, quarreling stakeholders. However, product managers and engineers often lack design thinking skills and familiarity with this process. Agonizing over ambiguity appears to affect team morale:

"Product managers struggled with it the most. They are used to having a goal and being able to work towards that goal ... Pivotal employees who are very uncomfortable with the ambiguity make it hard for the team." —Participant 25

To help the team cope with ambiguity, product designers may *"tell stories and talk about previous experiences; connect the client to former client PMs who have been there; reassure that we are on track and the ambiguity is normal."* (Survey Participant 11). They *"help them understand that we are in this together. We have a process that will help us find the best ideas and solution"* (Participant 52).

Participants appear to become more comfortable as they get more experience with Pivotal's design process.

5) Time Pressure: Meanwhile, time pressure inhibits the entire design process. Interviewing users, researching organizations, synthesizing design concepts, and iterating on mockups is time-consuming. Product designers can feel acute pressure to truncate user research and deliver immature mockups before they have a sophisticated understanding of the project context. Time pressure is often exacerbated by clients not understanding the design process and its value or thinking they have already designed the product. *"Sometimes people feel the anxiety of this taking too long, and push to start coming up with the solution"* (Participant 41).

On the healthcare application mentioned in Section III-B2, the client arrived with a preconceived solution concept, and the product designer felt intense pressure to deliver unvalidated mockups to the engineers. The team coped by having the engineers focus on configuring the tech stack, adding continuous integration and easy deployment to the production environment, and delivering web pages with no styling. In other words, they delivered something small that was very easy to change while the designer refined her mockups. However, this led to substantial stress and rework, which are both intrinsically wasteful [8].

6) Blocking Access to Users: In one extreme case, as Participant 41 explains, *"I could not even create a persona for the customer service agent because we were not allowed to talk to them."* The project was a call center application. The client refused to let customer support representatives be interviewed. The client believed that they were too busy to be interviewed and that call center metrics and case notes would be sufficient to inform the new product.

Without access to the users, the product team could not develop deep insights into how to improve the users' efficiency. They could not create realistic personas or evaluate usability. The absent users fundamentally broke the team's process. To deliver the product, the team had to speculate about the personas, needs, and behaviors. The team could not validate whether the proposed solution would address the users' needs.

C. Theorizing the Backlog

We observed teams organizing activities into two tracks with a soft but unmistakable boundary between them. The first track involves researching the project context and generating solution concepts, mockups, and user stories. The second track involves delivering a software product based on those concepts, mockups, and stories.

This observation leads us to adopt Glaser's *cutting point* theoretical coding family, which he describes as indicating a "boundary, critical juncture, cutting point, turning point, breaking point, benchmark, division, cleavage... This family is a variation of the degree family. Degree focuses on the full range, while here we focus on significant breaks or cutting points on the range. Cutting points are very important in theory generation since they indicate where the difference occurs which has differential effects" [11, p. 76].

As illustrated in Figure 2, the product backlog spans the boundary between the two tracks in dual-track agile. Track one fills the backlog (with stories) while track two empties the backlog (by implementing the stories). Of course, not every team that has a backlog uses dual-track agile, but a soft division between generating and implementing solution concepts is consistent with not only our understanding of Scrum, Kanban, and Extreme Programming but also our past observations of both professional and student developers.

The idea of the backlog spanning the boundary between different kinds of work (and the different kinds of professionals who do the work) is crucial to answering the questions motivating this paper: what is a product backlog, what is its role, and how does it emerge?

1) What a Product Backlog is: The product backlog is an informal model of work to be done. The backlogs that we observed are partially-ordered lists of brief work item descriptions. Some items include effort estimates, dependencies, or pointers to various artifacts elaborating on them. Because backlogs are used in *agile* development, only immediate work may be sequenced.

As a model, the backlog is useful for externalizing cognition and memory. A large software system could have thousands of stories—far too many to consider at once or to remember for long. The backlog allows, for example, a product manager to externalize thoughts about a large number of features, see the big picture, introduce changes, and prioritize effectively. Similarly, the backlog can help engineers remember what to do next, drill down into the specifics of the work to be done (e.g., by looking at a user story, mockup, and acceptance criteria), and recall the details of a discussion several days prior.

A backlog is neither a requirements specification nor a design specification. The backlog predominately communicates the desired order of work items. It lists neither design details nor the needs that features are supposed to meet. The backlog items (i.e., stories) are reminders of conversations, not self-contained specifications.

2) The Role of a Backlog: A product backlog is a boundary object that bridges the gap between generating and implementing user stories.

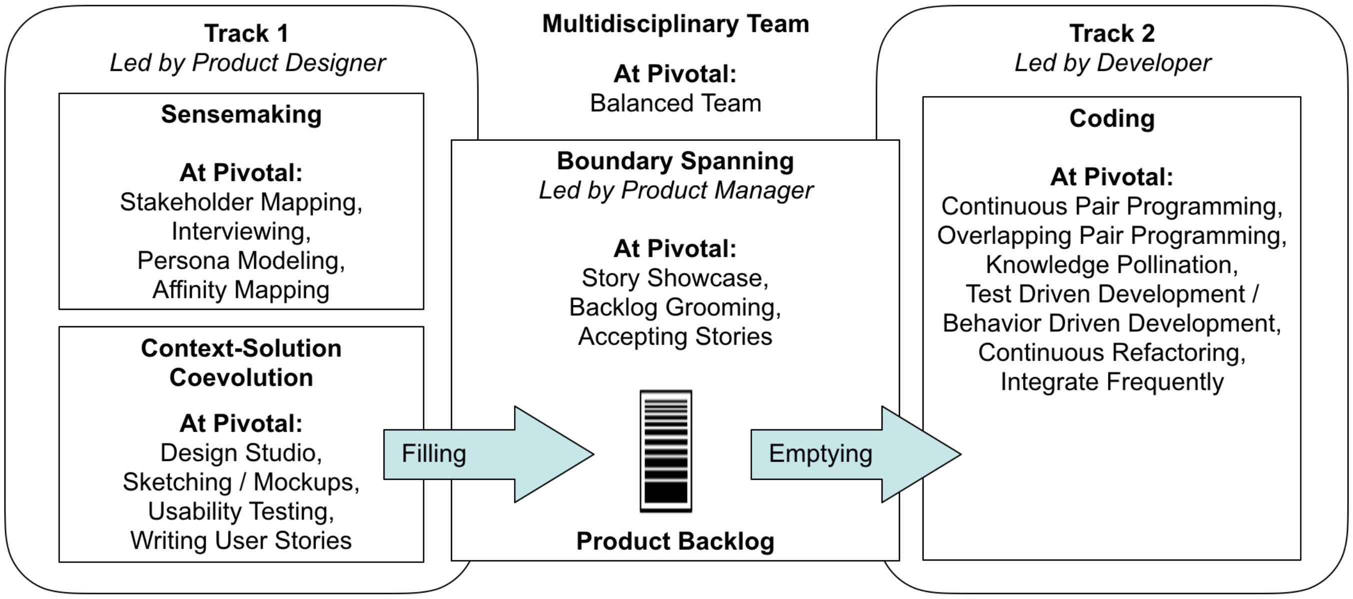

Fig. 2: Theory of Product Backlogs

A boundary object is an artifact that different people can use and interpret in different ways [28]. For example, the same map can be used by hikers to find a campsite or biologists to study animal movements. Boundary objects not only facilitate knowledge sharing and coordination [29] but also help groups cooperate without consensus [30].

We observed teams using backlog grooming and story showcase meetings to build a shared understanding of the product vision and work to be done. User stories both facilitate this conversation and later provide a reminder of it. Seeing the backlog as a boundary object helps us understand the difference between a backlog and a specification: a good specification *replaces* face-to-face communication while a good backlog *facilitates* face-to-face communication.

Conceptualizing the product backlog as a boundary object assumes that it mediates communication between different groups or roles [30]. In our observations, the backlog facilitated communication between professionals with different backgrounds: product designers, product managers, and engineers.

3) Where Backlogs Come From: Backlogs emerge from three interconnected processes:

1) **Sensemaking**. Pivotal product designers typically deploy several practices—including stakeholder mapping, interviewing, persona modeling, and affinity mapping—to learn about the domain. Regardless of whether a project is outsourced, offshored, or in-house, "To convert a problematic situation to a problem, a practitioner must... make sense of an uncertain situation that initially makes no sense" [25, p. 40]. In sociology and management as well as software engineering, this activity is called *sensemaking* [31], [32].
2) **Context-solution Coevolution**. A designer's thoughts oscillate between a tentative problem frame and fledgling solutions, such that understanding of the problematic context and solution concepts coevolve [23], [22], [32]. We observed coevolution in both affinity mapping (where the teams oscillated between writing design insights and re-organizing needs categories) and design studios (where the sketching of mockups triggered reconceptualizations of the project context and vice versa). We observed teams continuing to refine mockups and their understanding of the context after usability testing and while writing stories.
3) **Boundary Spanning**. While the product designers lead track one and engineers lead track two, we observed product managers leading the team in deploying several practices—including backlog grooming, story showcasing and accepting stories—to bridge the gap between the two tracks. Participant 1 explained: *"You meet every week, and you feel like you have this check in... I feel like this helps bridge the gap between what's going on in the design world and the dev world. Having those check-ins really help."* Following previous research, we call this category *boundary spanning* [33], [34].

To be clear, the two tracks of dual-track agile and the boundary spanning between them are ongoing, parallel processes, not linear phases.

IV. Discussion

This section evaluates our study using appropriate quality criteria, including those proposed by Charmaz [5].

1) Credibility: "Is there sufficient data to merit claims?" This study relies on two years and five months of participant-observation, 56 intensive open-ended interviews, an informal questionnaire, and archival records.

2) Originality: "Do the categories offer new insights?" This article provides the first empirically grounded explanation of how product backlogs are generated. The definition of the backlog and the *misunderstanding design* obstacle are also novel.

3) Resonance: "Does the theory make sense to participants?" We presented the results to the organization. Six participants reviewed the results and this paper for member checking [35]. They indicated that the results and theory resonate with their experience; for example:

- *"This paper combines all the things that were in my head and helped me to connect dots"* (Participant 52).
- *"Context-solution coevolution speaks to the back and forth nature of learning about this specific problem (context) and testing a solution's impact (if any) in a continuous way"* (Participant 56).
- *"It gave me a different relationship to the backlog by putting it between the 2 tracks"* (Participant 57).

This process produced no significant changes, which is not surprising because participant observation mitigates resonance problems.

4) Usefulness: "Does the theory offer useful interpretations?" Researchers can better understand agile development by viewing the backlog as a boundary object and model of work to be done. (See also Section VI.)

5) Researcher Bias and Prior Knowledge Bias: Participant observers develop an insider's perspective; they both notice and miss things an outsider would not. Similarly, prior knowledge can not only help the researcher interpret events and select lines of inquiry but also blind the researcher to alternative explanations [12]. We mitigated these risks by recording interviews and having the second and third authors review the coding process.

6) Transferability: Grounded Theory is non-statistical, non-sampling research; therefore, results do not statistically generalize to a population. However, as we organized our observations into higher-level categories, much of what seems specific to Pivotal and dual-track agile receded, leaving processes that appear highly transferable to other teams that use backlogs. Sensemaking and context-solution coevolution have been established by previous research as practically universal phenomena in design generally and software engineering specifically [22], [36], [37]. The research on boundary objects and spanners suggests that boundary spanning is fundamental to contemporary software engineering—or at least projects where professionals with different roles and backgrounds collaborate [38]. It should not matter:

- whether the backlog is held in project management software or made of physical cards;
- whether the team uses dual-track agile, Scrum or another agile approach;
- whether most backlog items are user stories, use cases, or scenarios;
- whether the team adopt the practices (e.g., user interviews, affinity maps) described above;
- whether the team is distributed or co-located;
- whether the code is closed-source or open-source.

Moreover, it does not matter whether or not the system has a graphical user interface and human users. API or command line systems simply need a different kind of product designer. The proposed concepts should transfer well to any team that: (1) adopts an agile approach; (2) has a backlog of work items; and (3) has at least one team member who brings business knowledge or design skills.

However, our depiction of dual-track agile assumes an ill-structured problem domain. In *structured* problem-solving, there is one problem, everyone agrees what the one problem is, and the meaning of the problem is clear (e.g., the travelling salesman problem). In structured problem-solving, sensemaking is trivial, context-solution coevolution does not occur, and boundary spanning is unnecessary. The danger here is mistaking ill-structured problems for well-structured ones. As we have described, just because a client arrives with what appears to be a clear problem description or even a comprehensive design specification, *that does not mean the problem is well-structured.*

V. Related Work

It appears that our current understanding of a product backlog as a list of work items for software developers was invented or at least popularized by proponents of Scrum. Although this is difficult to determine with certainty, the earliest reference to a modern product backlog we can find is Schwaber's presentation at OOPSLA 1995 [39].

The *Scrum Guide* defines "the Product Backlog [as] an ordered list of everything that is known to be needed in the product" [40]. Ambler extends this definition to include different kinds of work items such as training or reviewing other teams' products [41] and therefore calls it a "work item list." Scrum's and Ambler's view of the backlog broadly align with our observations in Section III-C1.

The presentation of the product backlog in this article is consistent with the Scrum literature in several ways:

- The backlog is a collection of work items.
- The backlog is evolving throughout a project.
- The backlog is continually refined or groomed by the Product Owner (in Scrum) / Product Manager (at Pivotal).

However, the Scrum literature presents the backlog in several ways that are inconsistent with the observations and conceptualization presented here:

- Scrum defines the backlog as an ordered list; all of the backlogs we observed were only partially ordered.
- The *Scrum Guide* states: "At any point in time, the total work remaining to reach a goal can be summed" [40]. The backlogs we observed all contained items that were neither estimated nor fully fleshed out.
- Some presentations of Scrum, including the *Scrum Guide*, state that the backlog contains requirements. Requirements formally describe needs. User stories informally describe features. None of the backlog items we observed were requirements. Conceptualizing a backlog as a requirements specification is inconsistent with our observations (see Section III-C).

Beyond the prescriptive agile literature, some studies have investigated how backlog items are prioritized. A systematic review of 13 such studies found no consensus about how items should be prioritized [42] and concluded that "the average quality of studies is low" (p. 5).

Racheva et al. [43] synthesized a lifecycle theory of backlog prioritization from existing literature. However, lifecycle theories are typically inappropriate for modeling human behavior [44]. Indeed, weighed against the observations reported here, this model appears to overrationalize the prioritization process in several ways:

1) It assumes backlog items are requirements. Again, this is not what we observed.
2) It presents the steps of prioritization as a sequential process with no backtracking. In contrast, we observed product managers periodically grooming the backlog.
3) It does not explain where the backlog items come from.

We found only one backlog-focused experiment, which showed that students using the *work system snapshot* conceptual modeling technique wrote better backlog items [45].

Of course, many agile development case studies *mention* a backlog, but we are not aware of any cases that *focus on* the backlog. Furthermore, we are not aware of any research that specifically investigates how backlog items are generated.

The Scrum literature, for example, shies away from how exactly to generate user stories or other backlog items. Scrum simply assigns the job of writing user stories to the "Product Owner" but says little about user research, the creative process of designing the system, or writing and refining these stories [40], [46], [47].

Extreme Programming [9] similarly offers little guidance for determining what to build. In the *Real Customer Involvement* practice [9], an onsite customer provides features through story cards. This does not provide design guidance; it simply transfers design responsibility to customers.

The Human Computer Interaction (HCI) community, in contrast, is chiefly concerned with the design of user experiences for interactive products. HCI incorporated the concept of sensemaking as early as 1993 [48], and many of the practices described above are well-known in user-centered design [49]. These processes and practices remain decoupled from mainstream SE methods [50]. Some in the SE community are consequently under "the impression that the user-centered design approach and usability engineering methods are only for decorating a thin component sitting on top of the software or the 'real' system" [50].

Several systematic literature reviews [51], [52], [53] have recently attempted to correct this misunderstanding and bridge the gap between the SE community's focus on *how to build* and the HCI community's focus on *what to build*. For example, Brhel et al. [51] identify five principles underlying user-centered agile development: (1) separate product discovery and product creation, (2) iterative and incremental design and development, (3) parallel interwoven tracks of design and development, (4) continuous stakeholder involvement, and (5) artifact-mediated communication. Garcia et al. [54] further investigate principle (5) and concluded that user stories are one of the main artifacts mediating communication.

A significant finding that has emerged from these literature reviews, as well as this study, is the idea of interwoven tracks of determining what to build and building it—that is, *dual-track agile*. The term *dual-track agile* originates from articles describing software development at Alias (now part of Autodesk) [55] [56]. Both papers depict two tracks (called the *Developer Track* and the *Interaction Designer Track*), with designers working at least one or two cycles ahead of developers. (We did not adopt these labels because, in our observations, track one was not limited to *interaction* design, and all team members participated in both tracks.)

Recently, several professional articles present various accounts of dual-track agile [57], [58]. These articles suggest that, similar to Pivotal, many companies perceive dual-track agile as a promising approach for bridging the gap between generating solution concepts and building software.

VI. Conclusion

This article reports the first in-depth field study focused on the product backlog. It makes the following contributions:

1) It clarifies that a product backlog is an informal model of work to be done (and not a requirements specification).
2) It explains how a backlog functions as a boundary object: that is, bridging the gap between generating solution concepts and building software.
3) It explains how backlogs emerge from sensemaking, context-solution coevolution, and boundary spanning.
4) It describes 13 specific practices that give rise to product backlogs at Pivotal.
5) It describes design obstacles faced by product designers at Pivotal.

These contributions are important because, despite the ubiquity of product backlogs, no scholarly research has comprehensively explained how they emerge or function. Software teams can use the practice descriptions to improve how they create their backlogs. They can also use the obstacle descriptions to explain to upper management and clients why user research is needed and why structure and clarity often decrease before increasing. Educators can use the theory of the backlog to help students understand what a backlog is and how it works. Researchers can use this work to understand the backlog's relationship to requirements and design decisions. More research is needed to determine how to assess backlog quality and establish its effects on software engineering success.

References

[1] P. Checkland, *Systems Thinking, Systems Practice*. Chichester, UK: Wiley, 1999.

[2] S. Lichtenstein and P. Slovic, *The Construction of Preference*. Cambridge, UK: Cambridge University Press, Aug. 2006.

[3] N. Maiden, S. Jones, K. Karlsen, R. Neill, K. Zachos, and A. Milne, "Requirements engineering as creative problem solving: a research agenda for idea finding," in *Proceedings of the 18th IEEE International Requirements Engineering Conference*. IEEE, 2010, pp. 57–66.

[4] D. H. Jonassen, "Instructional design models for well-structured and III-structured problem-solving learning outcomes," *Educational Technology Research and Development*, vol. 45, no. 1, pp. 65–94, 1997.

[5] K. Charmaz, *Constructing Grounded Theory*. SAGE Publications, 2014.

[6] T. Sedano, P. Ralph, and C. Péraire, "Sustainable software development through overlapping pair rotation," in *Proceedings of the International Symposium on Empirical Software Engineering and Measurement International Conference on Software Engineering*, ser. ESEM, 2016.

[7] ——, "Practice and perception of team code ownership," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE. ACM, 2016.

[8] ——, "Software development waste," in *Proceedings of the 2017 International Conference on Software Engineering*, ser. ICSE '17. IEEE, 2017.

[9] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[10] T. Sedano, "Sustainable software development: Evolving extreme programming," Ph.D. dissertation, Carnegie Mellon University, 2017.

[11] B. Glaser, *Theoretical Sensitivity: Advances in the Methodology of Grounded Theory*. Sociology Press, 1978.

[12] ——, *Doing Grounded Theory: Issues and Discussions*. Sociology Press, 1998.

[13] T. Sedano, P. Ralph, and C. Péraire, "Lessons learned from an extended participant observation grounded theory study," in *Proceedings of the 5th International Workshop on Conducting Empirical Studies in Industry*, ser. CESI '17. IEEE, 2017.

[14] J. Thomson. Trust and balanced teams. [Online]. Available: https://medium.com/product-labs/trust-and-balanced-teams-919456ad57cf

[15] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, no. C, pp. 176–189, Jan. 2017.

[16] S. Madsen and L. Nielsen, "Exploring Persona-Scenarios - Using Storytelling to Create Design Ideas," in *Human Work Interaction Design: Usability in Social, Cultural and Organizational Contexts*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 57–66.

[17] M. Farleo. Personas vs proto-personas. [Online]. Available: https://newmediadenver.com/personas-vs-proto-personas

[18] M. Seaman. The right number of user interviews. [Online]. Available: https://medium.com/@mitchelseaman/the-right-number-of-user-interviews-de11c7815d9

[19] J. Kawakita, *Hassouhou-Souzousei kaihatsu no tame ni - A way of thinking that developes creativity*. Chuokoron-Shinsha, 1967.

[20] A. Alliance. (2017) Role-feature-reason. [Online]. Available: https://www.agilealliance.org/glossary/role-feature/

[21] M. Wynne and A. Hellesoy, *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Bookshelf, 2012.

[22] N. Cross, "Design cognition: results from protocol and other empirical studies of design activity," in *Design knowing and learning: Cognition in design education*, C. Eastman, W. Newstetter, and M. McCracken, Eds. Oxford, UK: Elsevier Science, 2001, pp. 79–103.

[23] N. Cross, K. Dorst, and N. Roozenburg, *Research in Design Thinking*. Delft University Press, 1992.

[24] P. Ralph and Y. Wand, "A Proposal for a Formal Definition of the Design Concept," in *Design Requirements Engineering: A Ten-Year Perspective*, K. Lyytinen, P. Loucopoulos, J. Mylopoulos, and W. Robinson, Eds. Cleveland, OH, USA: Springer-Verlag, Jun. 2009, pp. 103–136.

[25] D. A. Schön, *The reflective practitioner: how professionals think in action*. USA: Basic Books, 1983.

[26] P. Ralph and R. Mohanani, "Is requirements engineering inherently counterproductive?" in *Proceedings of the 5th International Workshop on the Twin Peaks of Requirements and Architecture*, Florence, Italy, May 2015, pp. 20–23.

[27] P. J. Silvia, B. P. Winterstein, J. T. Willse, C. M. Barona, J. T. Cram, K. I. Hess, J. L. Martinez, and C. A. Richard, "Assessing creativity with divergent thinking tasks: Exploring the reliability and validity of new subjective scoring methods." *Psychology of Aesthetics, Creativity, and the Arts*, vol. 2, no. 2, p. 68, 2008.

[28] S. L. Star and J. R. Griesemer, "Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39," *Social Studies of Science*, vol. 19, no. 3, pp. 387–420, 1989.

[29] B. A. Bechky, "Sharing meaning across occupational communities: The transformation of understanding on a production floor," *Organization Science*, vol. 14, no. 3, pp. 312–330, 2003. [Online]. Available: https://doi.org/10.1287/orsc.14.3.312.15162

[30] S. L. Star, "This is Not a Boundary Object: Reflections on the Origin of a Concept," *Science, Technology, & Human Values*, vol. 35, no. 5, pp. 601–617, 2010.

[31] K. E. Weick, K. M. Sutcliffe, and D. Obstfeld, "Organizing and the Process of Sensemaking," *Organization Science*, vol. 16, no. 4, pp. 409–421, 2005.

[32] P. Ralph, "The Sensemaking-coevolution-implementation theory of software design," *Science of Computer Programming*, vol. 101, pp. 21–41, 2015.

[33] H. Aldrich and D. Herker, "Boundary spanning roles and organization structure," *Academy of management review*, vol. 2, no. 2, pp. 217–230, 1977.

[34] M. L. Tushman and T. J. Scanlan, "Boundary spanning individuals: Their role in information transfer and their antecedents," *Academy of management journal*, vol. 24, no. 2, pp. 289–305, 1981.

[35] J. W. Creswell and D. L. Miller, "Determining validity in qualitative inquiry," *Theory Into Practice*, vol. 39, no. 3, 2000.

[36] J. Kolko, "Abductive thinking and sensemaking: The drivers of design synthesis," *Design Issues*, vol. 26, no. 1, pp. 15–28, 2010.

[37] P. Ralph, "Software engineering process theory: A multi-method comparison of Sensemaking-Coevolution-Implementation Theory and Function-Behavior-Structure Theory," *Information and Software Technology*, vol. 70, pp. 232–250, 2016.

[38] N. Levina, "Collaborating on Multiparty Information Systems Development Projects: A Collective Reflection-in-Action View," *Information Systems Research*, vol. 16, no. 2, pp. 109–130, 2005.

[39] K. Schwaber, "SCRUM Development Process," in *Business Object Design and Implementation: OOPSLA Workshop Proceedings October, Austin, Texas*, J. Sutherland, D. Patel, C. Casanave, J. Miller, and G. Hollowell, Eds., London, UK, 1997.

[40] K. Schwaber and J. Sutherland. (2015) Scrum guide. [Online]. Available: http://www.scrumguides.org/

[41] S. Ambler. (2010) Agile best practice: Prioritized requirements. [Online]. Available: http://agilemodeling.com/essays/prioritizedRequirements.htm

[42] A. Silva, A. Silva, T. Araújo, R. Willamy, F. Ramos, A. Costa, M. Perkusich, and E. Dilorenzo, "Ordering the product backlog in agile software development projects: A systematic literature review," in *Proceedings of the 29th International Conference on Software Engineering & Knowledge Engineering, SEKE 2017*, Pittsburg, USA, 07 2017.

[43] Z. Racheva, M. Daneva, A. Herrmann, and R. J. Wieringa, "A conceptual model and process for client-driven agile requirements prioritization," in *Research Challenges in Information Science (RCIS), 2010 Fourth International Conference on*. IEEE, 2010, pp. 287–298.

[44] P. Ralph, "Toward Methodological Guidelines for Process Theories and Taxonomies in Software Engineering," *IEEE Transactions on Software Engineering*, 2018.

[45] N. Bolloju, S. Alter, A. Gupta, S. Gupta, and S. Jain, "Improving scrum user stories and product backlog using work system snapshots," in *Proceedings of the Americas Conference on Information Systems*. Boston, USA: AIS, 2017.

[46] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*. Prentice Hall, 2001.

[47] K. Schwaber, *Agile Project Management with Scrum*. Microsoft Press, 2004.

[48] D. M. Russell, M. J. Stefik, P. Pirolli, and S. K. Card, "The cost structure of sensemaking," in *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, ser. CHI '93. New York, NY, USA: ACM, 1993, pp. 269–276. [Online]. Available: http://doi.acm.org/10.1145/169059.169209

[49] B. Martin and B. Hanington, *Universal Methods of Design: 100 Ways to Research Complex Problems, Develop Innovative Ideas, and Design Effective Solutions*. Rockport Publishers, 2012.

[50] A. Seffah and E. Metzker, "The obstacles and myths of usability and software engineering," *Commun. ACM*, vol. 47, no. 12, pp. 71–76, Dec. 2004. [Online]. Available: http://doi.acm.org/10.1145/1035134.1035136

[51] M. Brhel, H. Meth, A. Maedche, and K. Werder, "Exploring principles of user-centered agile software development: A literature review, systematic review paper," *Information and Software Technology*, vol. 61, no. C, pp. 163–181, May 2015. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2015.01.004

[52] D. Salah, R. F. Paige, and P. Cairns, "A systematic literature review for agile development processes and user centred design integration," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '14. New

York, NY, USA: ACM, 2014, pp. 5:1–5:10. [Online]. Available: http://doi.acm.org/10.1145/2601248.2601276

[53] C. Salvador, A. Nakasone, and J. A. Pow-Sang, "A systematic review of usability techniques in agile methodologies," in *Proceedings of the 7th Euro American Conference on Telematics and Information Systems*, ser. EATIS '14. New York, NY, USA: ACM, 2014, pp. 17:1–17:6. [Online]. Available: http://doi.acm.org/10.1145/2590651.2590668

[54] A. Garcia, T. Da Silva, and M. Silveira, "Artifacts for agile user-centered design: A systematic mapping," in *Hawaii International Conference on System Sciences*, 01 2017.

[55] L. Miller, "Case study of customer input for a successful product," in *Proceedings of Agile 2005*. IEEE, 2005, pp. 225–234.

[56] D. Sy, "Adapting usability investigations for agile user-centered design," *J. Usability Studies*, vol. 2, no. 3, pp. 112–132, May 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=2835547.2835549

[57] K. Albrecht. (2015) Dual track agile: Focusing on customer value. [Online]. Available: https://medium.com/kevin-on-code/dual-track-agile-focusing-on-customer-value-a2e39312585b

[58] J. De Litchenberg. (2017) Dual-track agile: Why messy leads to innovation. [Online]. Available: https://www.mindtheproduct.com/2017/04/dual-track-agile-messy-leads-innovation/