Pattern Stories and Sequences for the Backlog

Applying the Magic Backlog Patterns

LISE B. HVATUM

REBECCA WIRFS-BROCK, WIRFS-BROCK ASSOCIATES

Agile development processes are driven from a backlog of development items. This paper looks at how to apply patterns from our Magic Backlog collection through the use of storytelling. We follow a project through various scenarios from the early days through significant growth into a mature setting and then, finally to a disruptive transition as the project becomes part of a program. Our goal is to focus on common project issues that we believe are experienced by many people involved in software development, and how the use of combinations of patterns, including our own as well as other patterns and practices, can help resolve or mitigate these problems.

Categories and Subject Descriptors: • **Software and its engineering~Requirements analysis** • *Software and its engineering~Software implementation planning* • *Software and its engineering~Software development methods*

General Terms: Management

Additional Key Words and Phrases: requirements engineering, requirements analysis, agile, product backlog

ACM Reference Format:

Hvatum, L. and Wirfs-Brock, R. Pattern Stories and Sequences – Applying the Magic Backlog Patterns. 24th Conference on Pattern Languages of Programming (PLoP), PLoP 2017, Oct 23-25 2017, 21 pages.

1. INTRODUCTION

In our earlier papers on building "The Magic Backlog" [Hva2015, Wir2016], we presented patterns to build and structure the backlog for an agile development effort using the outcomes of requirements elicitation and analysis. This paper expands on our earlier work by considering pattern stories and pattern sequences to assist readers in the application of these patterns.

Our patterns are targeted at large systems that must support complex operational processes, have hundreds of detailed requirements and possible safety and security concerns, and that may need to support external audits or prove that sufficient testing was done before deployment. We expect these projects to use electronic Application Lifecycle Management (ALM) tools to manage the backlog, such as Doors NG, JIRA, or TFS/VSTS, and that these tools allow backlog items to be linked and to have attributes which enable queries to be formulated based on these attributes to make sense of the current and evolving state of the backlog.

The initial scope of a product and most of the product requirements are generated through requirements gathering using elicitation techniques. This output of requirements gathering is then processed using techniques like story mapping, use cases, and workflows [Hva2015]. There are a number of publications that provide methods and techniques for how to elicit, analyze and process information to reach detailed software requirements [AB2006, Got2002, Got2005, GB2012, HH2008, Wie2009, Wie2006]. Our patterns add to this software requirements engineering body of knowledge by providing practical advice on how to build a good backlog from these requirements for a large and complex product using an ALM tool.

The term "Product Backlog" is part of Scrum terminology and is defined in the Scrum Guide [SS2013]:

"The Product Backlog is an ordered list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product. […] The Product Backlog lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases."

The items in the ordered list are Product Backlog Items (PBIs), a generic term that allows for any representation of a product "need" to be included in the Product Backlog. A common item type is the user story, but to think of the Product Backlog purely as a list of user stories is too simplistic for any large and complex system. Given that the Product Backlog is the "single source" for driving system development, you want it to give you the full picture of the product requirements. For the remainder of this paper we will use the term "backlog" to mean the Product Backlog.

The initial backlog typically has PBIs of varied granularity, from specific detailed needs to rough ideas on a theme or epic level. As development progresses, the backlog contents are prepared and modified to reflect the current understanding of the product and the efforts required to create it. The Scrum Guide [SS2013] states:

"A Product Backlog is never complete. The earliest development of it only lays out the initially known and best-understood requirements. The Product Backlog evolves as the product and the environment in which it will be used evolves. The Product Backlog is dynamic; it constantly changes to identify what the product needs to be appropriate, competitive, and useful. As long as a product exists, its Product Backlog also exists."

Most agile process descriptions are sketchy on how backlogs are developed and maintained, and so give little guidance to the product team for this activity. Requirements engineering does not address this level of detail and although the many practices around elicitation techniques and other requirements activities are very important, there is little help on the actual backlog creation, evolution, and ongoing management.

The backlog is primarily a tool for the team and should be designed for their needs. Within a team we include all members actively involved in the development effort – people who take on the roles of product owner, business analyst, project manager, architect, UX expert, developer, tester, technical writer, etc., whether or not they are full time. Note that although we are dealing with systems whose product needs are too large to "live on a wall", this does not infer that the team responsible for them is distributed, very large, or for a program having multiple sub-projects and teams. So in this paper any reference to teams is meant to mean a single, unified team. We plan to specifically cover backlog management for complex teams (Scrum of Scrums, Scaled Agile Framework, distributed team, etc.) with future patterns.

Our patterns define role-based activities and responsibilities. We expect individuals to be moving between roles depending on what they are currently doing. A product owner could double as a tester. A project manager might do the work of a business analyst, as well as development and testing. One role that especially needs clarification is that of the analyst. A Scandinavian proverb states that, "A beloved child has many names," and this is true of this role. Business analyst, product analyst, and requirements engineer are frequently used. In essence this role is an expert on requirements engineering, i.e. the elicitation and management of requirements. The analyst is not a domain expert, but a skilled resource who knows how to drive elicitation efforts, how to analyze and structure the outcome, how to translate from the business domain to the technical domain, and how to administer and maintain a requirements collection for a product. The role of the analyst often falls on a project manager, and sometimes on the development team. But as products are getting larger and more complex, there is an emerging analyst profession, and more frequently teams include dedicated analysts. Just as the agile tester is integrated into the team, so should the analyst be. The primary audience for our patterns is the analyst.

2. THE BACKLOG PATTERNS

The Magic Backlog patterns collection are patterns that give practical help on building a good quality product backlog for a software project. We initially used the term "Magic Backlog" because agile process descriptions pay relative little attention to the creation of the backlog – so it appears as if by magic. Feedback we got during the writing of our first pattern paper about creating a backlog made us realize that the term has another meaning: done well with the right contents and structure the backlog can do magic to support the team. Readers familiar with the stories of The Magic School Bus [Sch2015] will probably recognize the connection. If you need a submarine, the school bus will transform into one. If you need a microscope, or a fully equipped biology lab, there will be one in the bus. With careful design and preparation of your backlog, it can be as magic as the school bus, supporting your current needs. It provides a technical view of the product to the development team, while the Product Champion can see a business view. It keeps the current plan for the project manager, and the testing structure for QA. It helps you know where you are and where you should be going next.

In two earlier papers [Hva2015, Wir2016] we have documented twelve of the patterns that are part of the Magic Backlog collection. These patterns are fundamental in that they provide the needed basis for backlog creation and management. As mentioned in the introduction, we want to explore additional contexts and expect our patterns collection to grow somewhat in this process. One area we want to get deeper into is the additional demands on the backlog in the case of programs rather than projects. Our example project in this paper does end up in this situation and we have some ideas of how our existing patterns support this context. But we also see additional patterns emerging. We include these new patterns in italics in the patterns overview in table 1 to indicate that we have not yet written these (i.e. we have identified them and will document them in future papers).

The need to structure and manage the backlog and the associated development workflows with some degree of formality increases with project size. Note that the specific context of the Magic Backlog collection of patterns is backlogs for products of significant scope and complexity, with at least a three-year time frame for gradually delivering the full system. These projects have little choice but to use professional tooling and dedicate time and resources in backlog management.

Table 1 presents the patterns currently identified as part of the Magic Backlog Patterns collection in short form. For the full documentation of the first twelve patterns please refer to our two earlier papers [Hva2015, Wir2016].

Pattern name	Description
Frame	How do you organize the main structure of the backlog to best provide the benefits of a quality backlog to a variety of users? Choose a backlog structure that represents a functional breakdown of your system. Create a hierarchical structure and link items in this structure in a way that best represents the product to the backlog users. A functional structure is a model that most likely aligns the understanding for most roles on the development team.
Views	How can the backlog provide representations of a product that is intuitive to a variety of user roles? Create additional backlog structures to reflect alternate views of the product, for instance an architectural view and a quality view. Lower level backlog items can be linked both to items in the functional product structure (the Frame) and to items in the alternate structures. As an example, a User Story can be linked both to a main Feature (in the Frame) and to a Subsystem (in the architectural view).

People	How can you represent the various aspects of your system's users in a backlog? Create backlog items for personas to cover the dimensions of user profiles, and associate the personas with the appropriate functional backlog items. Their descriptions are then readily available for any team member with access to the backlog. Either tag a user story with the name of the persona, or link the persona backlog item to the functional item.
Tales	How can you improve the understanding of how users interact with the system and the impact on dependencies between individual user stories? Include narratives that give a free-form representation of product usage in your backlog. Most likely your narrative will span multiple user stories, and the natural level to link it in is to the feature level. The actual text for the narrative is captured in a document which is then uploaded as an attachment to the narrative backlog item.
Usage Models	How can you improve the understanding of how individual user stories contribute to a business transaction or user goal? Enrich your backlog with models that provide a structured representation of product usage. Each usage model represents a business transaction or a use of the system as a whole to accomplish a complex task. The purpose of the model is to improve your understanding of how the system is used and provide a tool to prioritize, plan, and verify your product deliveries. Possible models are Use Cases and Business Process Models.
Placeholders	How can you represent partly unknown functionality in your backlog? Create temporary backlog items as placeholders to be exchanged for detailed items later, when they have been elaborated. When the detailed items are created, you will want to replace your placeholder backlog item with the new detailed items. If you instead keep the placeholder item and link these details to it, you will increase the levels in your backlog thereby making querying and backlog maintenance that much harder.
Plans	How are the backlog items associated with your plans for delivery? Associate the detailed requirements slotted for the next delivery to an entity representing this delivery. Tools normally associate backlog items with iterations and releases by using a planning-related attribute on backlog items. Backlog contents can then be filtered based on the values of this attribute to produce lists of items for a specific release.
Connections	How can you explore the diverse contents of your ALM system? Create connections from other item types to the appropriate requirements backlog items. You want to establish these connections systematically following a defined model, normally linking tests to requirements, defects to both requirements and to the tests that detect and/or verify the defect resolution, and change sets to the requirements they implement or defects that they resolve.
Answers	How can your team gain insights about the product from the backlog? Create shared queries and reports that can be reused by your team. The primary focus when extracting information from the backlog should be on the direct development team needs, and not stakeholders. The goal is for the core team to always know where they are and be able to prioritize their efforts on the most pressing work.

Pipeline	How can you ensure that you always have some backlog items with sufficient maturity to enter the development process? Design a process that creates a steady stream of prepared backlog items. The process works as a pipeline that steadily refills the backlog with items with enough detail to be meaningful to the developers.
Funnel	How and when do you introduce new product ideas into your backlog? Keep a list of future product ideas to explore that is separate from your Product Backlog. When an idea has been accepted into the product scope and has matured enough to be represented by epics level items, then introduce these into your Backlog. Expect that a good portion of product ideas will never be fully developed. Some may be discarded early after limited investigation either because they cannot be supported by a business case, because they are too costly to develop, or because they just do not fit into the portfolio.
Maintenance	How do you keep your backlog as a reasonably accurate representation of the planned and implemented product? Regularly and consistently maintain the backlog contents. Maintaining the backlog is more than adding details and updating statuses. New contents need to be added as new requirements are elicited. Business priority changes will adjust the user story sequence/iteration planning. A maturing understanding of the product may require refactoring of the structure for the *Frame* and the alternate *Views*. Objects and attributes that the team uses for its planning and metrics need to be updated as the items go through the *Funnel* and the *Pipeline* and then through implementation/verification, making sure that structure and attribute changes caused by new material is consistently applied across the full set of contents.
Remodel	*Refactor the backlog to refocus the contents*
Rules	*Who can do what and how in the backlog*

Table 1: Backlog Patterns Overview

When developing a set of patterns you typically realize that there are patterns in other pattern collections that relate to/work with/expand/overlap with the patterns in your own collection. This is great, because it is all part of our body of knowledge, and finding related patterns often strengthens the trust in your own patterns being useful in a shared context. But you are also faced with some challenges:

- The time and effort in finding related patterns and understanding their interaction with your own. Although there are some preliminary pattern catalogs, most patterns are found by researching the contents of papers, books, and online resources. This is very time consuming and hard to achieve for pattern authors who are not academics.
- The trade-off between incorporating patterns from other authors into your writings versus writing your own version of those patterns. Patterns from other authors may differ in style and context so although they are relevant they may not fit well into your collection. They may need some tweaking to be fully applicable for the type of problem you are addressing. But rewriting a large number of patterns that are already well documented is a bit of a waste and will make your paper/writing bloated.

In this paper we are using a bold font for our own patterns (**Frame**), bold and italics for future patterns in the Magic Backlog collection (e.g. ***Remodel***), and italics with underline for external patterns (*Just Do It*).

3. PATTERN STORIES AND PATTERN SEQUENCES

We decided to use pattern stories and pattern sequences to illustrate typical applications of the Magic Backlog patterns and give the reader a practical understanding of the usage of our patterns for common scenarios. So what are patterns stories and patterns sequences? In volume 5 of the POSA book series [BHS2012] the authors define pattern stories in the context of architectural patterns as:

"Pattern stories are like diaries that tell their readers how patterns guided the development of an entire software system, subsystem, or large component. […] Pattern stories also discuss what problems were resolved in what order during the construction of the software systems whose genesis they describe. In addition, they discuss what patterns were selected to resolve these problems and how the selected patterns were instantiated within the software architectures of these systems."

While their focus is system architecture, our focus is on backlog creation and management. So our pattern stories are about how patterns were applied in software development to construct and maintain a large backlog using an ALM tool. Sometimes our stories tie in patterns and practices from other authors/pattern collections. We have listed the thumbnails (short descriptions) and sources for these in the appendix.

Pattern sequences combine patterns to solve problems having a broader scope than may be solved by individual patterns. Pattern stories typically describe patterns applied, one after another, to deal with the combined problems within a given context. The reader will recognize pattern sequences within a story. Pattern sequences are not necessarily linear; rather the patterns within the sequence may be partially sequential, partially linear, and the order of some patterns may be interchanged. Allan Kelly does an excellent job of explaining and applying pattern sequences in his book, *Business Patterns* [Kel2012]:

"When patterns are applied together they are said to form a patterns sequence. There may be many ways of combining the patterns in a pattern language, and each pattern may appear in multiple sequences. Pattern sequences show the order in which patterns are combined to make a whole. […]. Pattern writers cannot foresee every context, problem or force that may lead to modifications when applying a pattern: human judgment is needed to select and adjust individual patterns and sequences. On other occasions we may find that the application of one pattern forces us to use another. The negative consequences of applying one pattern will create forces, resolving these forces may require the use of another pattern."

This quote brings up yet another concept, that of a pattern language. A pattern language is more than a collection of related patterns. Rather, the individual patterns are linked together to form a whole, where a single pattern works with other patterns to implement a larger pattern, and where the network of inter-linked patterns provide a completeness of solution design for the overall context. Alexander explains it this way in *The Timeless Way of Building*:

"In this network, the links between the patterns are almost as much a part of the language as the patterns themselves."

The POSA authors warn that pattern stories are specific and that readers must generalize from the story. If their context differs from the context of the patterns story, even slightly, the pattern selection and sequences may not be directly applicable for their solution. Another drawback of this approach is that a story can lose the feeling of authenticity because it doesn't include every detail or potential decision; it may appear concocted just to tell a good story. Despite these potential shortcomings, we like pattern stories because they engage the reader.

For this paper we selected a pattern story with a context that should apply to many readers. It tells of general problems faced with managing backlogs as a project grows. This story reflects the authors' personal experiences with real projects.

The following story is about a project as it evolves from initiation to maturity. The project starts small but evolves into a large program. We mine this story for patterns and then visualize and discuss the resulting patterns sequence.

4. PROJECT BLUEBIRD – FROM A SMALL START TO PROGRAM INCLUSION

Project Bluebird starts small, but with an expectation that it will grow over the next couple of years to a large undertaking. That is, if the product they are creating is as successful as the business owner predicts. Our story follows the development of the Product Backlog for the Bluebird project from its first creation through growth and maturity until it is absorbed into a program level backlog. Through this journey we will see the need for maintenance, restructuring and the ability to support the growing needs of the project.

Illustrating the overall quality of a backlog is a bit of a challenge – how do you show a well organized versus a messy backlog without taking the reader through a number of detailed screenshots? And even more problematic, a screenshot from an ALM tool will look organized, because it is the nature of the tool to present nice clean looking lists. Only people who understand the project will readily see the issues. There may be some obvious mistakes, like a requirement work item used for what is really a task, or missing traceability from user stories to tests. But we really could not find a good way to picture good and not-so-good backlogs to our readers. So bear with us in introducing an alternative way to illustrate backlog organization. We should all be able to relate to our walk-in closet…

Figure 1: The life of a walk in closet

So let us look at the basic needs of the *walk-in closet* and of a project using their backlog:

- Easy to get an overview of the contents
  Enough business clothing to last the week? Do I need to buy more socks?
  I can quickly get an understanding of structure and key elements of the product functionality (much like when looking at the story mapping), see the linked bugs and test cases as well as seeing separate lists of test cases and bugs. I can see if we have enough user stories ready for development.
- Easy to find items
  Pick up what to wear today – shirt and pants in matching colors and nice shoes to go with your outfit.
  Navigate to a bug that was resolved by engineering and need to be verified. Search for user stories that are recently added to see if we need to add more details before sprint planning.
- Easy to add and remove items
  Returning from the cleaner and hanging my shirts in a known place. Picking unused items to donate.
  Recording a few new bugs and linking them to the correct user stories. Cleaning up the regression test plan and removing tests that are no longer valid after we changed the implementation.

A. Early Days – starting a limited backlog from scratch

Maria works in a large organization that has delivered several sizable software products over the last three decades. She is in charge of the new Bluebird project (her company names engineering projects after birds; the marketing department then names the product as soon as the first version is ready to deploy). Eventually it is expected that this project will grow to a team of 60-80 people, possibly with some of the team geographically distributed between 3-4 sites, and also including external contractors as well as internal staff. But so far it is early days, and the 5 people on the team are in exploration mode. Their first task is to agree with the business owner about the first limited delivery (the minimum viable release - MVR).

In addition to Maria, the team members are Tom – a business analyst, Greg – a domain expert, John – the system architect, and Heidi – an experienced developer. The team has access to internal business experts and to a few select customers. The team is located together in Portsmouth, a bit north of Boston, USA.

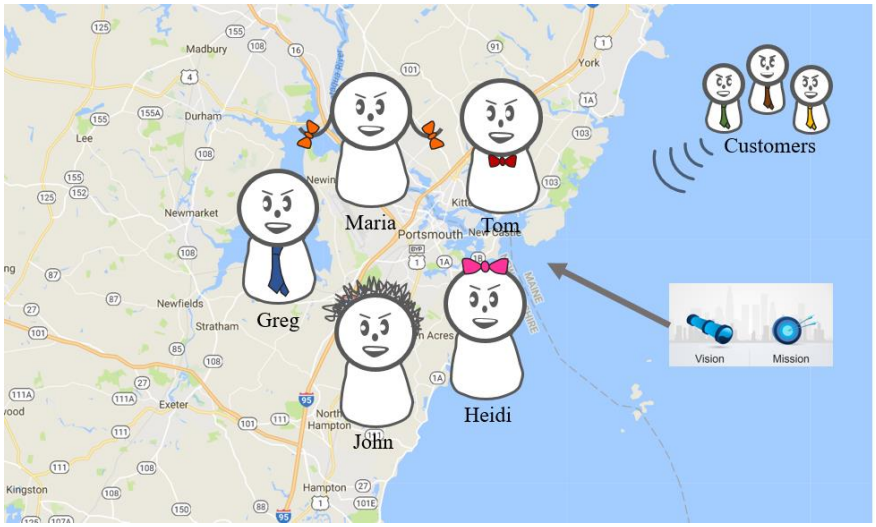

Figure 2: The Initial Bluebird Team

Some early requirements have been gathered at the time the team is formed, and the team is considering how to proceed. The requirements are in the form of an initial business document with a relatively well-formulated vision but little detail, some user interviews, and several internal e-mails between Maria and some of the stakeholders discussing the new product.

Initially, the team decides that their first delivery (MVR) is so small that they do not need formal tooling for the backlog (they will be using Git to manage their source code). They lay out the functionality using story mapping [Pat2014], and use a physical Kanban board [KBN2015] on a wall in their team room for planning and managing their tasks.

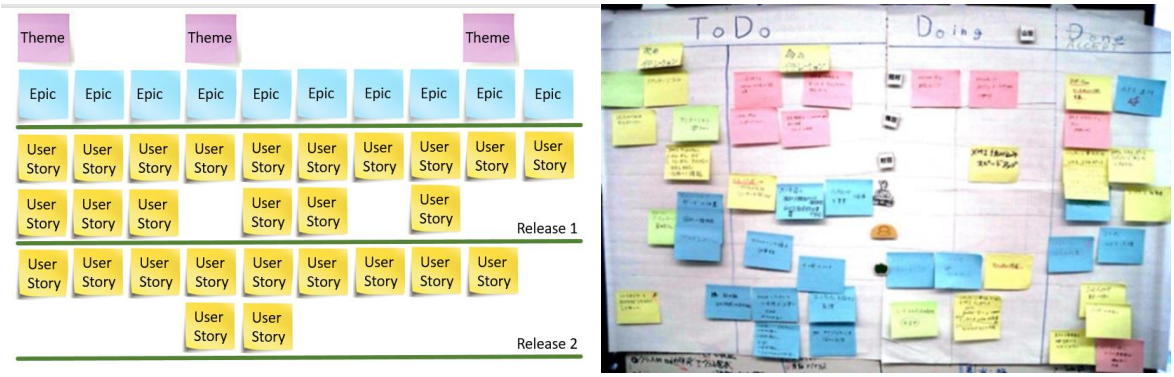

Figure 3: Early Bluebird Story Mapping and Kanban Board

A few weeks later the team adds a QA professional, Nicole, and another developer, Sven. They now start developing test cases associated with the user stories plus a set of transaction-level test scenarios associated with their epic-level requirements. Nicole suggests that the team should consider using an ALM tool. This will allow them to track test results, and to properly log issues that are not immediately fixed. It will also let them automate the generation of some basic metrics to help them know how ready the software is to deploy (quality release criteria). Since the project is expected to grow much larger they will need digital tools in the future so they might just start now to get some experience with then.

The first challenge is to select a tool. There are several ALM tools in the market from major vendors. Some members of the Bluebird team have experience with a few of the tools, but none of them has strong opinions or have ever before participated in the starting up of a project and initiating the use of an ALM tool. After some online research and discussions with colleagues the team decides to use VSTS from Microsoft to manage their backlog. This tool is already available in their company and licensing is covered centrally. Because VSTS is cloud based there is no infrastructure to set up. A couple of days later the team has their project setup in VSTS and can start using the tool.

And this is where total confusion sets in. In principle, the backlog is the team's to-do list. On the Kanban board this is easy – every sticky is something that need to be done, whether it is running a test or implementing a user story. But the tool has so many different items, and each item has a heap of attributes. Any item can be linked to any other item, and there are even different types of links. There are just so many ways things can be done that the team feels lost.

Still, they stay with their decision to try using a tool because they know they will have too much content as the project increases to be able to manage it without an ALM tool. But after a few weeks of team members entering contents however they wish, it is clear they need to agree on some general rules. The on-line Kanban board does not look as expected. When listing out user stories it is hard to get an overview of deployable system functionality because the user stories are a mix of technical needs ("internal plumbing") and features. The granularity varies. And then there are contents that reflect tasks for the team but that really do not contribute to the product, like organizing a visit from stakeholders and setting up a company book study activity.

Maria and Nicole commit to doing some research – how are other teams inside and outside the company structuring their backlog? When they reach out to friends and colleagues on other projects they realize that creating good backlogs takes time and effort, and not in the least experience and some trial and error.. There are many pitfalls, and most of the larger projects have been through serious restructuring and rework of their backlogs before they got something they are reasonably happy with; usually it takes more than one iteration in order to adequately revise the structure. Common pitfalls seem to be including too much detail that needs to be maintained, being unclear in the structure and allowing any team member to add contents without some rules to follow, and not understanding the need to structure the backlog so that it supports creating dashboards that show meaningful development status and progress. They are also told that they are smart to ask for advice early, but even then they should expect to need to refactor their backlog sometime down the road. It just seems to happen to every project at some point. So go lean from the start and expect change.

Some of the internal projects allow Maria and Nicole to study their backlogs and help them to understand how the structure helps the project team with their internal development process. Maria and Nicole decide to "borrow" the model from a project that is similar enough to the Bluebird project and use this to create their own VSTS contents.

When they have a workable preliminary backlog, they organize a full day team session and provide a draft of "how to use VSTS." At the end of the session the team has an agreement for basic rules for how to structure and maintain their backlog. They will create a 3-layer structure with user-facing functionality, and use the story map to populate this. Bugs and test cases, both manual and automated, will be linked to the leaf-level item (i.e. the user story). They also agree on the meaning of the various backlog item attributes

and which ones to use. For example, the "iteration path" attribute is used for planning to assign user stories to iterations, and the "area path" attribute is used to categorize the story as belonging to major system components.

Nicole and Sven spend some additional time creating shared queries and setting up the project dashboard so the team has some core metrics of system deployment readiness. VSTS has some built-in reporting that supports this. Their dashboard includes bug statistics and test results. All of this is possible because items are linked in the tool. During their efforts they struggle with learning how to create good queries in VSTS and how to make sure each query returns the items intended. It does take some practice to use the query editor, and they are sometimes pretty frustrated before they get the hang of it. It turns out they have to tweak the internal model of how all the items are linked to make their dashboards operational (i.e. to make all their queries work properly). They are also frustrated by the tool limitation of only showing trends over the last 4 weeks. For later, they will need to invest in some more reporting functionality. But for now, they agree to stay with what can be shown in the dashboard tiles provided with the tool.

After the dashboard is set up, Nicole and Sven explain it to the rest of the team. They also stress the need for keeping VSTS contents up to date so that the dashboard reflects the current project state. Even so, this is an issue for weeks with Tom and Nicole chasing team members to update their items in a timely fashion. But when Maria starts actively using the dashboard information for project communications, this helps. People finally realize that the information is helping the project team to know where they currently are and where they should go next in their joint planning sessions.

The team experiments for a few days with using the Kanban board feature in VSTS and adding tasks to their user stories. But they soon revert to their wall-based Kanban board as this is more efficient for a co-located team and, not insignificantly, conveys more information by the placement of stickies, use of color coding and handwriting, and added comments and drawings. They find that the wall-based Kanban board just has a lot of personality that is missing from the electronic version supported by the tool. So the team ends up using the ALM tool mostly for high-level planning, test management, and defect management, and not for managing their flow of work. This decision did not affect the dashboards as these were fortunately designed to use only the data that is still being maintained in VSTS, and not on the Kanban wall.

Questions/Decisions: Do we need a tool? How do we get started using an ALM tool? How do we structure our user stories? Do we need traceability and why? How do we share information through the tool?

When the Bluebird team gets to the point where they realize that relying only on their wall of information (their *Information Radiators*) is not working so well, they face a number of decisions. They need a way to manage their growing set of user stories with acceptance criteria, tests, test results, bugs etc. But which tool should they choose and how should they best use it? It may be that their organization already has tool standards, and that they have colleagues who can help them get started. Or they may need to choose a tool and learn from the tool vendor and outside resources. Even if the amount of data is too much to handle without a tool, it has not yet grown so large that they cannot try out a tool and how to structure their contents and change it if it does not work out. So at this point they need to *Just Do It*.

Luckily the Bluebird project has Nicole to help. She has some experience using VSTS from an earlier assignment. She uses the structure from the *Story Mapping* and the initial requirements document to create an initial **Frame** that provides the breakdown of the user functionality and organizes the user stories. She uses the Kanban board to help her assign user stories to iterations and updates the states of each user story to show if is it already implemented or in the **Pipeline**.

This team liked their wall-based Kanban board and quickly decided to keep using it. So they only need the tool to help them with the longer-term high-level contents. But they do want to use the reporting/dashboard features of the tool. Also, to be able to produce their **Answers** they need to link their tests and bugs to user stories to provide the necessary **Connections** to run basic QRC reports/graphics. *Patterns sequence: Just Do It, Story Mapping, Kanban, Information Radiators,* **Frame**, **Pipeline**, **Connections**, **Answers**

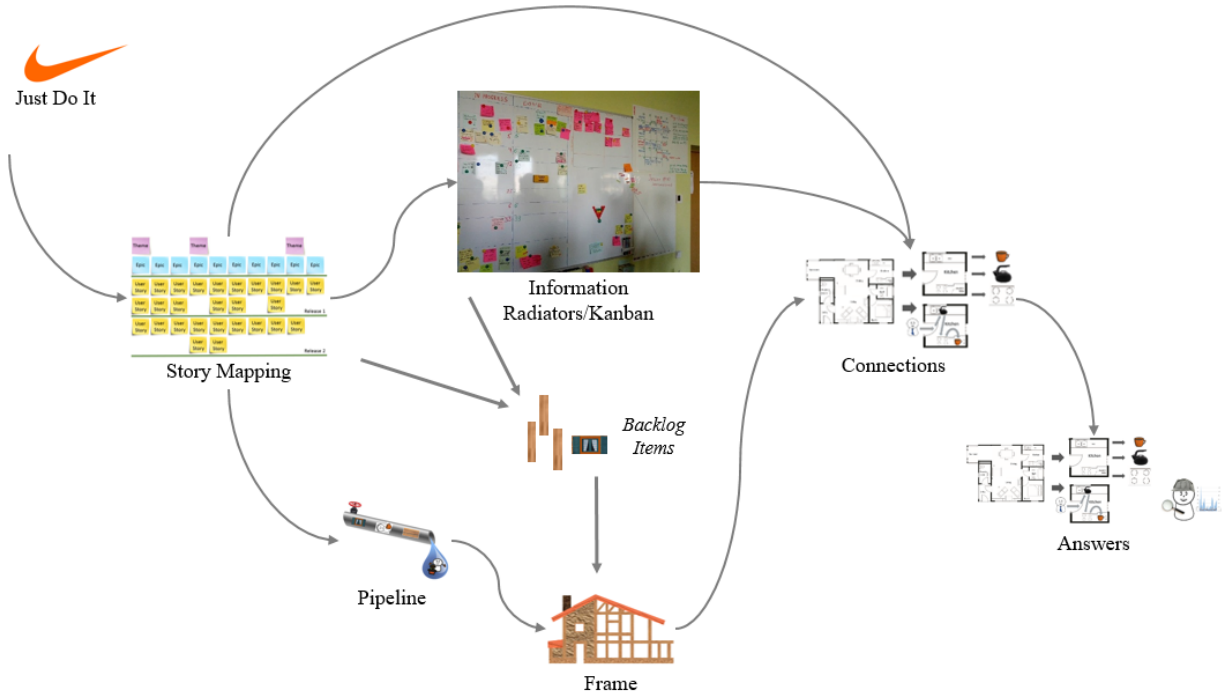

Figure 4: Early Days Patterns Sequence

B. Dealing with growth and increased backlog contents

After a few months, the team has grown to 16 people, with 3 off-site developers working on the front end. As well as increasing the number of developers, the team hired two IT Operations staff, Richard and Fred, to deal with deployments and a UX expert, Theresa.

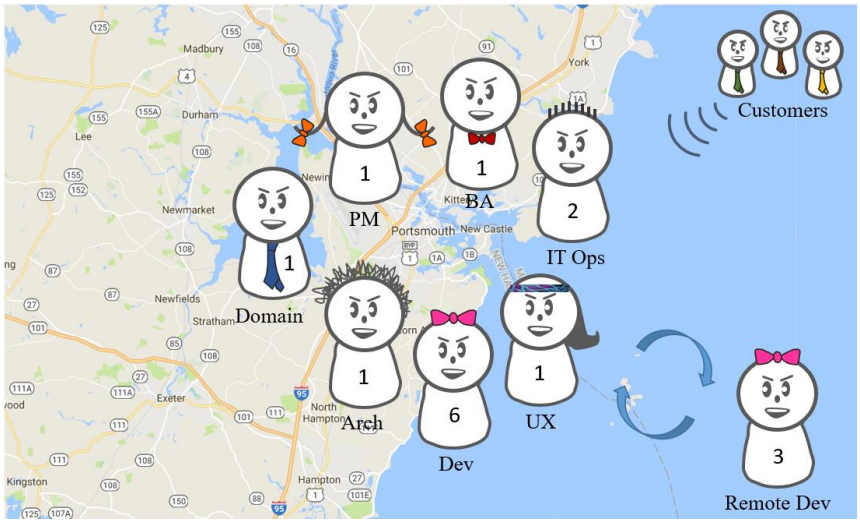

Figure 5: Expanded Bluebird Team

A first delivery has been deployed and is now used by a few select customers who are generating lots of feedback. The team realizes that they have outgrown their simple process and tooling, and that they need to really start using their ALM tool more formally, paying more attention to their communication and planning.

A particular issue that keeps surfacing is that user stories keep being added which do not reflect actual user features. It is clear that the team has a real need to manage backlog contents related to architecture, building the deployment infrastructure etc. But the added user stories affect the queries and metrics with contents that are undesirable for querying about the deployable features of the product.

After this issue has surfaced in several retrospectives, Maria asks Tom (BA), John (Arch), and Richard (one of the new IT Ops guys) to come up with a solution. They propose to create several top nodes in the backlog. The core top node is for the product functionality. Another top node is for technical user stories. These are stories that support multiple user stories and even features. For example, implementing the internal communications layer of the application is a technical user story. A third top node is for the non-functional (quality) requirements structure. Then there is a node for the testing and deployment infrastructure. Finally they propose a top node for "other stuff", giving the team a place to add contents that do not fit anywhere else.

As the BA, Tom gets the "fun" task of creating the new structure and then moving user stories under the right parent node. John helps out with the technical contents and creates a new backlog structure under the technology top node following the architectural view of the system. There is some initial pain to get all the team members to remember to follow the new structure, but when they finally get used to it the improvement in is evident in better query results and dashboard contents.

The team also discussed creating a top node for the remote team members working on the front-end but decided against this for two reasons: they want to keep the remote team as integrated as possible, and since the remote team works on customer-facing functionality, their work is contained under the core top node.

Since most of the team still sits in one location and the front-end team has very close interactions with the local UX expert Theresa, the team initially decides to keep the physical Kanban board as their daily planning tool and use the ALM tool for persisting backlog contents that are important to manage and maintain over time. Typically they enter their user stories into VSTS, but continue to keep their individual tasks informally on the wall space. This eventually causes a lot of friction with the remote team members who feel left out and repeatedly realize they lose out on sharing of information. The challenge is not so much on the more formal documentation that is captured in VSTS items as it is with the informal communication around the physical Kanban board and daily tasks. This eventually leads the team to look at their tooling situation. They end up with a hybrid solution where development that is only done locally remains on the Kanban board, while development involving the front end is now using the online Kanban feature in VSTS. The team also starts using Slack for team communication which also helps the remote team members feel included. And they start doing regular workshops together every 3 months where they meet face to face for at least one week building trust and aligning their work.

Theresa develops personas to give a better understanding of the various user profiles and their needs. Initially she stores these on a shared drive. But after several experiences of people not finding the right profiles or kind of ignoring their existence, she decides they need to be more available and connected to the other material defining the development. She discusses with Tom (the BA) and they agree to link the persona descriptions directly to the user stories they apply to. This way it is easy for the developers to review the correct persona description when implementing a new feature.

Questions/Decisions: How do new types of content fit in with the user functionality? How do we incorporate personas in our backlog? How can we plan for refactoring and technical enablers when our story maps and backlog **Frame** *is only focused on user functionality?*

To deal with the additional contents, Tom creates additional **Views** for the technical enablers, for non-functional requirements, for the test and deployment infrastructure, and one for the stuff that does not fit well under any of the other top nodes. This helps navigation and planning efforts for the developers, and allows the team to add technical items to their **Plans,** such as updating the database technology (stuff like this does not fit in with any particular user story).

The persona descriptions developed by the front-end team are made available to the rest of the team as backlog **People**. Nicole and her QA team are excited about easy access to these descriptions and the ability to link them to particular test cases.

Patterns sequence: **Frame, Views, Plans, People**

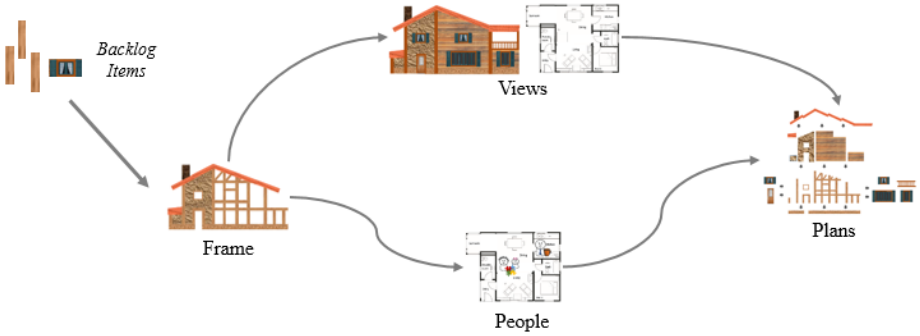

Figure 6: Maturing Project Patterns Sequence

C. Dealing with deteriorating quality of backlog contents

Another year has gone by, the team is now 60 people distributed at 4 different sites.

Figure 7: Globally distributed Bluebird team

Because the team has grown and its distributed nature, the ALM tool now plays a much bigger role in communicating and tracking the work of the team. Maria realizes that they have a problem. Different roles have different needs for the backlog contents they manage. Sub-cultures at different sites are starting to grow. Some groups within the team prefer tight communication and a minimum backlog content, other groups want more formality and enter a lot of details to the backlog, including individual tasks and detailed technical backlog items. Since all team members are adding to the backlog however they see fit, it has started to turn into an unmanageable "monster." One result of this is that it is currently impossible to run queries that provide clear results as to the team's status and the quality of their work. To deal with this the team adds additional attributes and tags to the existing backlog items. But because there is no shared understanding of how this should be done, it just makes matters worse by confusing the backlog contents even more. And some team members are getting really frustrated with the tool.

Maria decides for the team to take some time away from day-to-day development to deal with the unmanageable backlog contents. She involves the whole team in an initial analysis to find the core needs and current problems of the backlog for the various roles and team members. In general, the team members realize the importance of a good backlog and they are all are positive about contributing to improving it.

It is clear that the team needs better "traffic rules" for handling the backlog contents. They agree on how each backlog item should be managed, and actually decide to enforce and restrict some actions to gain better consistency. For example, they agree that when a bug is created, it must contain the steps to recreate it (if possible) and it must be linked to a user story. From now on, only the QA resources are allowed to close a user story under the functionality top node. John (the architect) starts to enforce the structure of the technical contents. This has been the most unstructured area since Tom does not deal with this part of the contents. Maria has the team write up their new rules on the project wiki and makes sure it is agreed between the project locations and that all individual team members are aware of it. They also decide to make the backlog quality and rules a recurring item in their retrospectives so that they can avoid another cleanup of this scale. The two weeks it took to get their backlog back under control and cleaned up according to their improved rules was a painful eye-opener for everybody.

Questions/Decisions: What to do when the backlog has grown out of control and the contents have gotten messy?

If the backlog starts being an impediment rather than a tool supporting the development, it is necessary to invest in correcting issues it has. The backlog is an entity owned by the whole team, but that does not mean that everybody should be able to change everything, or that each person should be able to do any modification. The overall structure should be kept under control and not be modified too frequently, as this is disruptive to the people using the backlog. This is not too different from the issues of shared folders and team SharePoint sites. The more people that are involved in changing the contents, the harder it is see the logical structure and organization.

Based on their agreed upon **Views** structure, the team takes the time to ***Remodel*** the backlog and agrees on a set of backlog ***Rules***. The ***Rules*** inform the team how to perform ongoing backlog **Maintenance** to try to avoid ending up in the same misery again.

Patterns sequence: **Views, *Remodel*, Maintenance, *Rules***

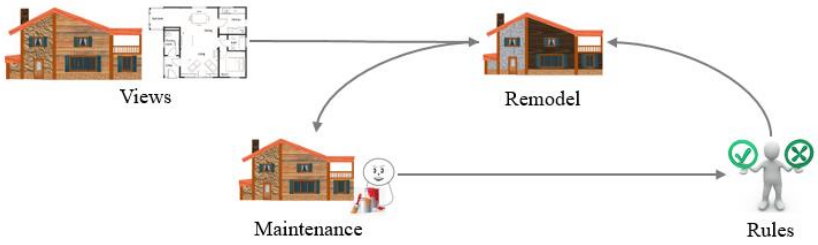

Figure 8: Refactoring and Clean-up Patterns Sequence

D. Becoming part of a program

After 3 years, the organization around the Bluebird project has grown to almost 100 individuals including the customer support team. A company reorganization and a new business focus have led to the Bluebird project to now being part of a larger program. Maria, Greg and John have all transferred out to other projects in the organization, leaving a knowledge gap. A new program manager, Frank, is looking into how to handle the seven projects in the program that all have separate backlogs and different processes and *Rules*.

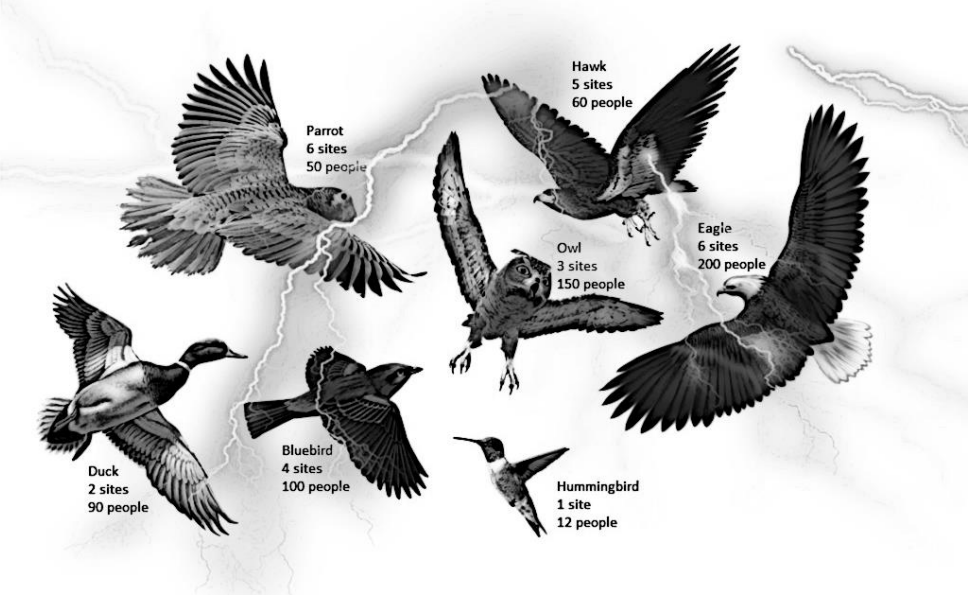

Figure 9: Overall Program with Individual Projects

Initially Frank does not want to interfere too much with the projects. Each project works rather independently but shares the vision of what the final integrated solution will be. They have each developed their backlogs according to their way of working and with their target delivery aiming for a deployment of the first total solution 5 months from today. Some of the projects have been around for 2-3 years and they have large backlogs with lots of history in their contents. But after observing the projects' interactions and realizing the significant problems of regularly updating the overall solution with new and improved features, Frank realizes that the current ways of working need to change. Backlogs only existing on the project level will never lead to a streamlined process which can deliver good quality software frequently and reliably.

The frustration is particularly felt by the teams doing the final User Acceptance Testing (UAT) and preparing the deployment in the production environment. Although each project is closing their User Stories in compliance with the teams' criteria for "Done," it seems far too often that seemingly simple changes and updates to individual projects' code break the system when it is integrated. And no one seems to feel responsible for the bugs and system instability or volunteer to resolve it. If the software works on the project level testing, it is a fight to get developers to take action – each time.

It is also hard to really know what features and user operation are enabled by the software updates. Within the integrated solution, a fully working feature may depend on internal services and user functionality provided by several projects. Since each project has its own schedule, several features may be partially done but are lacking service upgrades to fully function even if the user interface allows the feature to be executed. The UAT on the program level is quite frankly a nightmare to run, and the project teams then deflect any bugs found during program level UAT. They claim that the software works in their environments, so they don't think that their software is the problem. This attitude only adds to the time it takes to get these integration bugs fixed.

Attempts to change the situation with process improvements activities turn political. It is clear that most of the projects are not open to changing their ways, and/or they do not grasp the severity of the situation. They are stuck in a mindset where they feel that as long as they deliver according to their project plan they are "safe" and "performing to expectations."

Realizing that the biggest issue is not to get all teams to use the same practices, Frank decides to start with a very practical approach to streamlining teams' efforts and get them working together. He decides to focus

on aligning the projects for the overall planning. His previous project worked out a very crisp and lean backlog with processes to maintain it this way during development, and he now adopts this approach for a program level backlog. Rather than trying to combine and align the existing project backlogs, he decides to create an independent program level backlog, with a user functionality **View** only. This backlog is not detailed enough for development purposes – that level of detail will be maintained by the individual project level backlogs. This program level backlog is for product planning across the program, and it serves as the backbone for the system integration testing.

When an item from the program backlog moves into development, it will have clear integration level acceptance criteria spelled out. Operational models and personas for all features are attached. The projects have to agree on each project's contribution, and this now drives project-level priorities. After the projects analyze their commitments to the program level item and decide for their implementation items, they create the corresponding contents (typically a set of user stories) in their own project backlogs. They then proceed to follow their own team process to develop and test the needed functionality. However, they also submit code back to the integration level frequently as parts of their implementation is completed to enable system level testing as early as possible. Frank is making sure that a new item is not started from the program backlog until the previous item is completed. Teams that complete their tasks early are asked to help other teams complete their work items.

System integration tests are added to the backlog (linked) during implementation. As soon as the projects have delivered enough of the feature to start testing, the UAT provides feedback that helps in completing the feature.

Over a few months the teams get used to this new way of driving their planning. Each team is keeping their own backlog and their way of working, but are now prioritizing work that enable features in the integrated solution. The UAT team starts having regular demonstrations to the whole development organization which again increases the feeling of ownership of the overall solution.

Another improvement that helps to finally break down the barriers between project teams is the creation of an integration team where a subset of the project team members are rotated through on a weekly basis. This team is responsible to sort out any build and deployments issues, bugs and other findings related to the integrated product. The integration team effort serves as an eye-opener as project team members get exposed to the overall product. One very positive outcome is the agreed push to run more of the project-level testing in the staging (production-like) environment. This eventually leads to far less bugs found during the UAT effort which again reduces time wasted on disagreeing about who caused the bug and friction between the project teams.

Finally, the new backlog enables metrics to be pulled on the system integration to feed a program-level dashboard on product quality and deployment readiness.

Questions/Decisions: What are the additional needs for the backlog when supporting a program, i.e. a set of projects that work together to provide a solution? Does the program need to have one backlog used by all the projects? Do the teams need to apply the same work practices?

The program-level backlog that Frank creates is a higher-level **Frame** that is also another **View** – one that focuses on the top level product created by the projects under the program. If the program had started off with a set of new projects, it might have worked to create one single backlog that encompassed all the project details. But since there is so much history involved in this case, Frank chooses to avoid a large amount of rework to existing backlogs, leaving individual project backlogs but also the work processes internal to each team as they are (a project team's backlog structure and work style tends to be closely related).

The **Usage Models** and the **People** descriptions in the program backlog helps the projects understand how their parts are contributing (or need to contribute) to the whole of the product. Before these were

documented, there was often confusion about user roles and interactions, both for naming the individual users and applying the correct terminology for the user interfaces.

Being able to provide **Answers** for the overall product was another challenge before the program-level backlog was created. But now there is a good structure to provide an overview of what parts of the system are ready for deployment. Some quality metrics (for instance bug counts) need to be pulled from the individual projects to complete the data, but it is easy to add in this metrics automation since all the data is in the same tool.

Patterns sequence: **Frame, View, Usage Models, People, Answers**

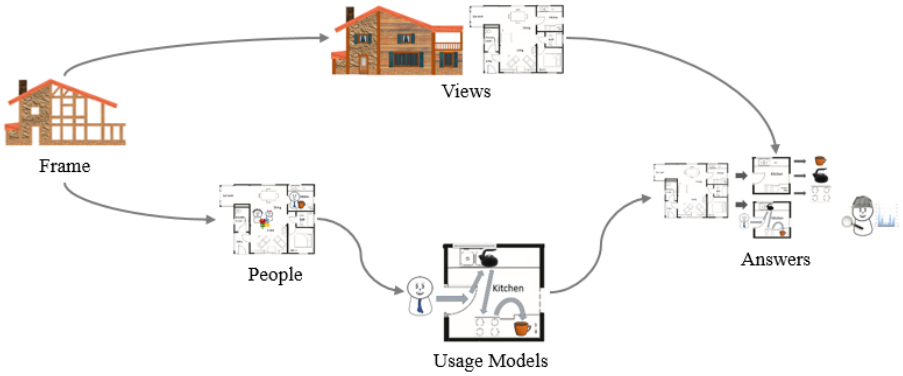

Figure 10: Program Level Patterns Sequence

5. FUTURE PATTERN STORIES

We have also identified at least two additional pattern stories to elaborate in a future paper. One is for a project that starts off large with an entirely different set of problems—moving from an old technology platform and large existing codebase to a cloud-based implementation with more configurable features for specific custom installations. Another story is about a project that has an increasing complexity around incorporation of software from multiple vendors and product configurations.

6. SUMMARY

The purpose of this paper is to provide additional insights into the use of the Magic Backlog patterns by using storytelling and patterns sequences, with additional comments and illustrations. A challenge with patterns sequences is that they are not really strict sequences, but rather a thread of related patterns that work together to create a solution that incorporates a number of individual patterns. There is a lot of flexibility in the order which individual patterns might be applied, and their application may be iterative and/or in parallel.

One of the challenges we hope to experiment with in future pattern stories is how best to keep readers' interest. We think it would be boring to read a series of stories with "made up" details and only slightly varying story arcs. We could write even shorter stories, capturing only the gist of the teams' decisions and their path forward. But based on feedback from our writers' workshop, they said they liked our storytelling and to keep it interesting we should add more drama. We plan on exploring several options for engaging our readers more deeply with the consequences of decisions. Perhaps we could have them explicitly make a decision and then follow an alternative story path to learn about its consequences. Or we might ask them to weigh in on what they value in making a particular decision.

An illustration including the patterns in the Magic backlog collection that are documented until now is shown below. We aim to expand the scope of our existing collection to better support program, distributed teams, and other types of software development. When more complete, our collection may become a pattern language, but as of today we feel this is too early.

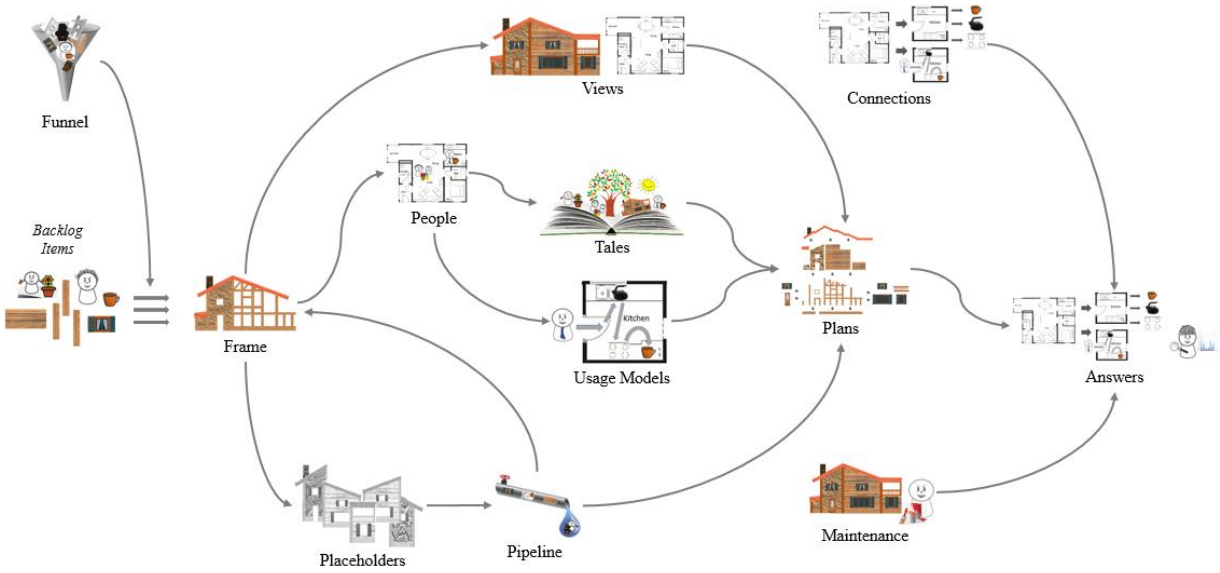

Figure 5: The Magic Backlog Patterns

7. ACKNOWLEDGEMENTS

Many thanks to our shepherd, Michael John, who especially challenged us to do more story telling about the difficult parts, where the team and the backlog had to grow and not lose focus. We would also like to thank our PLoP 2017 workshop participants for their feedback: Tim Yao, Tayyaba Nafees, Robert Biddle, Matt Griscom, and Eduardo Guerra. To gain a better understanding of software requirements and the processes around requirements engineering we have consumed a lot of literature, and we especially appreciate Karl Wiegers' writings on Software Requirements, Jeff Patton's work on Story Mapping, the Scrum Guide by Ken Schwaber and Jeff Sutherland, and Ellen Gottesdiener's and Mary Gorman's workshops and books.

REFERENCES

[AB2006] ALEXANDER, I. and BEUS-DUKIC, L. 2006. *Discovering Requirements: How to Specify Products and Services*. Wiley (ISBN: 978-0-470-71240-5).

[Agi2015] AGILE ALLIANCE 2015. *Agile Alliance Glossary: Backlog*. https://www.agilealliance.org/glossary/backlog/

[Amb2014] AMBLER, S. 2014. http://www.agilemodeling.com/artifacts/userStory.htm

[BC2012] BEATTY, J and CHEN, A. 2012. *Visual Models for Software Requirements*. Microsoft Press (ISBN 978-0-7356-6772-3).

[BHS2007] Bushmann, F., Henny, K., and Schmidt, D. 2007. *Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages.* Wiley. (ISBN-13: 978-0471486480)

[Bra2016] BRANDENBURG, L. 2016. *How to Create a Product Backlog: An Agile Experience*. http://www.bridging-the-gap.com/an-agile-experience-my-first-product-backlog/

[Coc2001] COCKBURN, A. 2001. *Writing Effective Use Cases*. Addison-Wesley (ISBN 0-201-70225-8).

[Coc2008] COCKBURN, A. 2008. *Information Radiator*. http://alistair.cockburn.us/Information+radiator

Addison-Wesley (ISBN 0-201-70225-8).

[Coh2004] COHN, M. 2004. *User Stories Applied*. Addison-Wesley (ISBN 0-321-20568-5).

[Coh2015] COHN, M. 2015. *Product Backlog Refinement (Grooming)*. https://www.mountaingoatsoftware.com/blog/product-backlog-refinement-grooming

[CUC2016] https://cucumber.io/docs/reference

[DBLV2012] DEEMER, P., BENEFIELD, G., LARMAN, C. and VODDE, B. 2012. *The Scrum Primer.* http://www.scrumprimer.org/

[Din2014] DINWIDDIE, G. 2014. *The Three Amigos Strategy of Developing User Stories*. http://www.agileconnection.com/article/three-amigos-strategy-developing-user-stories

[Gaw2009] GAWANDE, A., 2009, *The Checklist Manifesto.* Picador (ISBN 78-0312430009).

[Got2002] GOTTESDIENER, E. 2002. *Requirements by Collaboration*. Addison-Wesley (ISBN 0-201-78606-0).

[Got2005]GOTTESDIENER, E. 2005. *The Software Requirements Memory Jogger*. GOAL/QPC (ISBN 978-1-57681-060-6).

[GG2012] GOTTESDIENER, E. and GORMAN, M. 2012. *Discover to Deliver: Agile Product Planning and Analysis*. EBG Consulting (ISBN 978-0985787905).

[HH2008] HOSSENLOPP, R. and HASS, K. 2008. *Unearthing Business Requirements: Elicitation Tools and Techniques*. In Management Concepts (ISBN 978-1-56726-210-0).

[Hva2014] HVATUM, L. 2014. *Requirements Elicitation using Business Process Modeling*. 21st Conference on Pattern Languages of Programming (PLoP), PLoP 2014, September 14-17 2014, 9 pages.

[Hva2015] HVATUM, L. and WIRFS-BROCK, R. 2015. *Patterns to Build the Magic Backlog*. 20th European Conference on Pattern Languages of Programming (EuroPLoP), EuroPLoP 2015, July 8-12 2015, 36 pages.

[KBN2015] *What is Kanban?* http://kanbanblog.com/explained/

[Kel2012] Kelly, A. *Business Patterns for Software Developers.* Wiley. (ISBN-13: 978-1119999249)

[KS2009] KANNENBERG, A. and SAIEDIAN, H. 2009. *Why Software Requirements Traceability Remains a Challenge*. CrossTalk: The Journal of Defense Software Engineering. July/August 2009.

[MR2005] MANNS, M. and RISING, L. 2005. *Fearless Change: Patterns for Introducing New Ideas.* Addison-Wesley. (ISBN 0-201-74157-1)

[Mas2010] MASTERS, M. 2010. *An Overview of Requirements Elicitation*. http://www.modernanalyst.com/Resources/Articles/ /115/articleType/ArticleView/articleId/1427/An-Overview-of-Requirements-Elicitation.aspx

[Mul2016], MULDOON, N. 2016. Backlog grooming for Kanban teams in JIRA Agile. http://www.nicholasmuldoon.com/2016/02/backlog-grooming-for-kanban-teams-in-jira-agile/

[Pat2014] PATTON, B. 2014. *User Story Mapping*. O'Reilly (ISBN 978-1-491-90490-9).

[Rad2016] RADIGAN, D. 2016. The Product Backlog: *Your Ultimate To-Do List.* https://www.atlassian.com/agile/backlogs

[Rin2009] RINZLER, J. 2009. *Telling Stories*. Wiley (ISBN 978-0-470-43700-1).

[RR2006] ROBERTSON, S. and ROBERTSON J. 2006. *Mastering the Requirements Process*. Addison-Wesley (ISBN 0-321-41949-9).

[RW2013] ROZANSKI, N. and WOODS, E. 2013. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives* (2nd Edition). Addison-Wesley (ISBN 978-0321718334).

[Sch2015] SCHOLASTIC, 2015. *The Magic School Bus*. https://www.scholastic.com/magicschoolbus/books/index.htm

[SS2013]SCHWABER, K. and SUTHERLAND, J. 2013. *The Scrum Guide*. http://www.scrumguides.org/

[Sut2014] SUTCLIFFE, A. G. (2014): "Requirements Engineering" in Soegaard, Mads and Dam, Rikke Friis (eds.), *The Encyclopedia of Human-Computer Interaction*, 2nd Ed.," Aarhus, Denmark: The Interaction Design Foundation. Available online at https://www.interaction-design.org/encyclopedia/requirements_engineering.html

[Wie2009] WIEGERS, K. 2009. *Software Requirements* 2nd Edition. Microsoft Press (ISBN: 0-7356-3708-3).

[Wie2006] WIEGERS, K. 2006. *More about Software Requirements*. Microsoft Press (ISBN: 0-7356-2267-1).

[Wik2014a] WIKIPEDIA 2014a. *Business Process Modeling*. http://en.wikipedia.org/wiki/Business_process_modeling

[Wik2014b] WIKIPEDIA 2014b. *Business Process Model and Notation*. http://en.wikipedia.org/wiki/Business_Process_Model_and_Notation

[Wik2014c] WIKIPEDIA 2014c. *Requirement*. https://en.wikipedia.org/wiki/Requirements

[Wik2014d] WIKIPEDIA 2014d. *Requirements traceability*. https://en.wikipedia.org/wiki/Requirements_traceability

[Wik2016] WIKIPEDIA 2016. *Kanban Board.* https://en.wikipedia.org/wiki/Kanban_board

[Wir2016] WIRFS-BROCK, R. and HVATUM, L. 2016. *More Patterns for the Magic Backlog*. 23rd Conference on Pattern Languages of Programming (PLoP), PLoP 2016, Oct 24-26 2016, 18 pages.

[Wit2007] WITHALL, S. 2007. *Software Requirement Patterns*. Microsoft Press (ISBN: 978-0-735-62398-9).

[YWA2014] YODER, J.W, WIRFS-BROCK, R. and AGUIAR, A., *QA to AQ Patterns about transitioning from Quality Assurance to Agile Quality*. 3rd Asian Conference on Pattern Languages of Programming (AsianPLoP), AsianPLoP 2014, March 5-7 2014, 18 pages.

[YW2014] YODER, J.W and WIRFS-BROCK, R., *QA to AQ Part Two Shifting from Quality Assurance to Agile Quality "Measuring and Monitoring Quality".* 21st Conference on Pattern Languages of Programming (PLoP), PLoP 2014, September 14-17 2014, 20 pages.

[YWW2014] YODER, J.W, WIRFS-BROCK, R. and WASHIZAKI, H. *QA to AQ Part Three Shifting from Quality Assurance to Agile Quality "Tearing Down the Walls".* 10th Latin American Conference on Pattern Languages of Programming (SugarLoaf PLoP), SugarLoaf PLoP 2014, November 9-12 2014, 13 pages.

[YWW2015] YODER, J.W, WIRFS-BROCK, R. and WASHIZAKI, H. *QA to AQ Part Four Shifting from Quality Assurance to Agile Quality "Prioritizing Qualities and Making them Visible"*. 22nd Conference on Pattern Languages of Programming (PLoP), PLoP 2015, October 24-26 2015, 14 pages.

APPENDIX A. EXTERNAL PATTERNS

These patterns and practices from external sources (books, pattern collections, online resources) are used in this paper combined with patterns from the Magic Backlog collection to build the patterns sequences. We chose to treat the practices as a form of patterns because they are of a nature where they could be presented in patterns form (i.e. we perceive them as patterns just not documented using a patterns form).

Name	Type	Source	Reference
Information Radiator	Practice	Alistair Cockburn's Crystal Methodologies	[Coc2008]
Kanban	Practice	Wikipedia article on Kanban Board	[Wik2016]
Just Do It	Pattern	Manns/Rising in Fearless Change	[MR2005]
Story Mapping	Practice	Jeff Patton in Story Mapping	[Pat2014]

Table 2: External Patterns Overview