# More Patterns for the Magic Backlog

REBECCA WIRFS-BROCK, WIRFS-BROCK ASSOCIATES

LISE B. HVATUM

Agile development processes are driven from a backlog of development items. This paper adds three new patterns to our collection of patterns to build and structure the backlog for an agile development effort using the outcomes of requirements elicitation and analysis. The *Funnel* adds new content to the backlog, the *Pipeline* prepares backlog contents for implementation, and *Maintenance* performs continued care of backlog contents. The need to formalize the backlog increases with the size of the project.

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:**

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.

## 1. INTRODUCTION

In our earlier paper, "The Magic Backlog" [Hva2015], we presented patterns to build and structure the backlog for an agile development effort using the outcomes of requirements elicitation and analysis. This paper adds three patterns to our collection – the *Funnel* adds new content to the backlog, the *Pipeline* prepares backlog contents for implementation, and *Maintenance* does continued care of backlog contents.

Our work is 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, and that these tools allow backlog items to be linked and to have attributes.

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 of 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 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."*

Your entire backlog will contain contents in several stages of completeness, a typical set being:

- Active PBIs – items currently being worked on by the development team.
- Prepared PBIs – items sufficiently mature and understood and with enough detail that they can be taken on by development (i.e. enter into a sprint planning session or 3 Amigos communication).
- Rough PBIs – Future development items like epics which need further analysis and refinement.
- Blocked PBIs – items that cannot be moved forward until some problem is resolved, for instance a licensing or intellectual property issue, a technology decision, or a business uncertainty.
- Completed PBIs – items that are ready to ship (or already shipped).

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 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 that are too large to "live on a wall", this does not infer that the team 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.) in a future paper.

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 a dedicated analyst. 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 need to structure and manage the backlog and the associated development workflows with a certain level of formality increases with project size. The context of these patterns is new development for a product of significant scope and complexity, with at least a three-year time frame for gradually delivering the full system. These patterns specifically target developing a quality backlog [Hva2015]:

Frame – the basic structure provides a product overview and logical navigation

Views – the expanded backlog structure enables multiple views of the contents

People – elaborated personas span the dimensions of the user space

Tales – narratives give insight into how users interact with the system

Usage Models – usage models show structural user story dependencies

Placeholders – epics are replaced by user stories when elaborated

Plans – backlog items are grouped into release candidates

Connections – requirements, code, tests and defects are linked to provide traceability

Answers – helpful metrics such as burn-down charts are generated from the backlog

Pipeline – backlog items are prepared for implementation

Funnel – adding new contents to the backlog

Maintenance – continued upkeep of the backlog

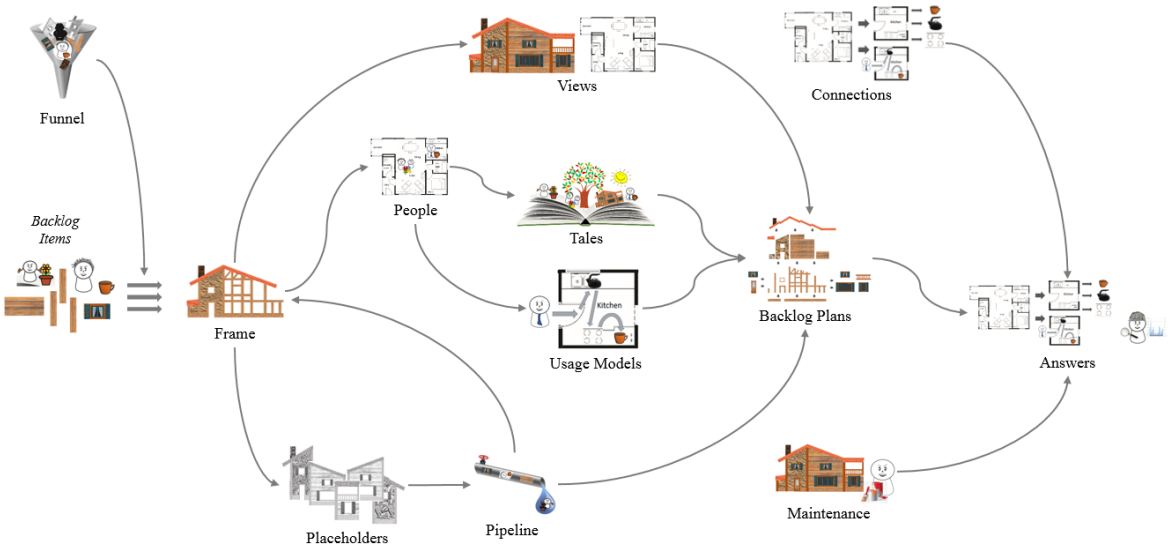
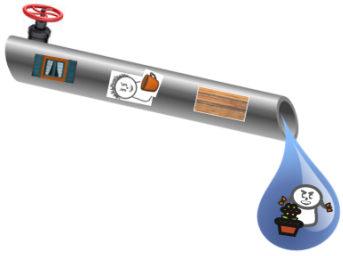



Figure 1: Backlog Patterns Sequence

A running example (The Benson Automated Pool Cleaning System) is used throughout the patterns. This was first introduced in our original paper [Hva2015] and is repeated in Appendix A for new readers.

# The Pipeline

Backlog items are matured just in time.



You have a backlog created from the initial requirements elicitation done for a new product. Development has started and is actively implementing some items in the backlog. The backlog has additional items that are within scope, but not sufficiently understood to begin development. Your organization has access to the business owner on a regular basis.

**How can you ensure that you always have some backlog items with sufficient maturity to enter the development process?**

You want to make sure that the team always has items in the backlog that are well enough understood that they can move into implementation, for example into a sprint or onto a Kanban board. The initial scoping of the project has likely created several larger items such as themes or epics in the backlog. These items will need to be further broken down to the level of granularity typical for user stories and other types of PBIs such as architectural/technical implementation items. Methodologies like Scrum, XP and other agile development processes are sketchy or almost silent on how backlog items are created and prepared until it is time for PBI planning (sprint planning or 3 Amigos activity).

If the PBI planning needs to tackle larger backlog items like epics, this effort can be ineffective, erratic, or incomplete. Too much time spent planning drains resources away from the implementation effort. Developers normally do not have detailed business domain understanding and so are not the best people to drive the elaboration of an epic into user stories. Worst case, the organization does not provide sufficient business resources and leaves the elaboration of requirements to the core development team without adequate support from a product owner. Consequently, higher level business requirements run the danger of being misinterpreted by technical resources and with poorly set priorities can even make their way through the implementation process into a mediocre product.

The work to create enough clarity for larger items in order to break them into suitable sizes for development may be too extensive to complete within the typical PBI planning time. It may involve some amount of research and investigation, for example, to determine what is needed to comply with a regulation or policy, or to gain sufficient understanding of a complex physical system to be controlled. Or it may involve deeper engagement with customers and users to explore potential solution alternatives.

The associated business priority of your backlog items helps the team schedule items for development according to the business impact/value. You need a way to ensure that the business priority is set, and if needed, adjusted in time for the planning of upcoming development.

Even projects with business analysts and domain experts dedicated to developing detailed requirements can experience problems related to the handover to and communication with developers. It can be difficult to provide sufficient detail for the implementation and be clear enough in the descriptions so as to avoid misunderstandings. The documented backlog representation of an item never contains all the knowledge that should be transferred from the business and domain experts to the design,

implementation, and testing team members. The risk of missing information and miscommunication is even higher when the team developing the requirements is in a different location.

Somehow, you need to manage the time dedicated to backlog preparation while still being able to feed the development process with the items that makes most sense from both a business and technical perspective. You do not want to invest in heavy upfront work on requirements and risk elaborating requirements that may never be implemented or that may no longer be valid because of later changes in the product. The danger with elaborating details too early is that unless implementation is imminent, they may become stale and have to be reworked because of overall changes to the product and the business approach.

**Therefore, 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. It is a recognized and properly staffed activity that works in unison with the development process that consumes the items.

The involvement from developers, testers and other team roles in operating the pipeline will help the team look forward and know what is coming later in the project, enabling them to be prepared and adjust the stream to better fit the technical and architectural side of development. Their involvement is also important to ensure that the items contain the type of information and the level of detail required for implementation, testing, and deployment. Participation and ownership in the *Pipeline* process leads developers and testers to not only appreciate the "why" of a new item, but to add their own ideas and innovations in the process.

A backlog with PBIs in various stages of completeness is shown in figure 2. We will use this figure to explain the pipeline process. The *Pipeline* is the specific process for moving items from Rough PBIs to Prepared PBIs. The items that go into the pipeline are for work items for which there is strong certainty that they will proceed into development. The business priority is established during pipeline activities and the selection and sequencing of PBIs going into the pipeline is based on the business roadmap/perceived business value.

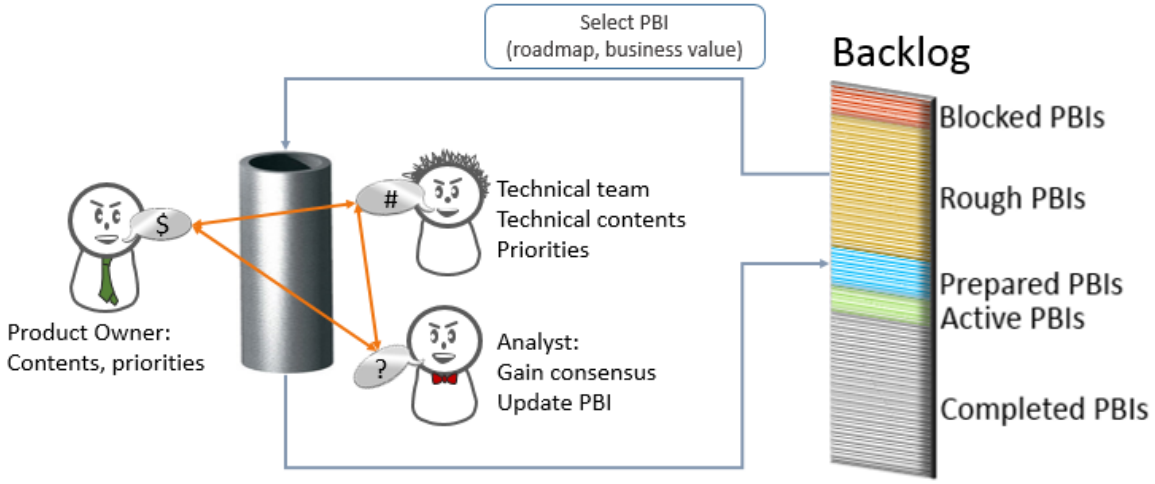



Figure 2: Backlog Pipeline process

The rough PBIs identified through requirements elicitation activities form the contents of the pipeline. They often are vetted through a funneling process that ensures that they align with business and customer needs. Pipeline input may come in bursts driven by elicitation events, however, as the product matures, the flow of Rough PBIs will gradually slow and finally consist entirely of smaller improvements and user defect reports.

As active PBIs become completed PBIs, items that are in the prepared PBI state will be taken on by the technical team and moved to Active. Unless new items are becoming prepared PBIs, the technical team will run out of items to work on. This is what the *Pipeline* offers—a process for creating a stream of new prepared PBIs in a continuous, managed flow.

To be a good quality prepared PBI, we would recommend that the item has:

- a granularity that fits within an implementation cycle or flow
- a clear and concise definition (for example using the Gherkin [CUC2016] technique)
- acceptance criteria
- understood business and technical risk
- a business priority
- sufficient detail for defining development tasks

Ideally, you want the *Pipeline* to feed into the backlog at the same rate as items are developed (i.e. set to Completed PBIs), leaving only a few backlog items as prepared PBIs at any time. This will avoid wasted time spent breaking down and elaborating larger items like epics and themes too early, or even before they are slotted for implementation. It is reasonable to consider breaking out a part of an epic for elaboration/implementation, leaving the rest for later.

The *Pipeline* should not be confused with PBI planning efforts. Both mature and refine backlog items, but the scope and purpose of these activities are different. The elaboration in the *Pipeline* stays within the problem domain, transforming business needs and product ideas that were initially captured at the epic or themes level into well-understood, prioritized items. The PBI planning continues the breakdown of the item into implementation tasks impacted by architecture, technology and team priorities. A well-operating *Pipeline* will greatly contribute to streamlined PBI planning by providing a sound starting point.

While being respectful of the available time of the technical team members, you still want members of the technical team involved with the business roles and domain experts to run the *Pipeline*. This will help the combined team to take responsibility of the product and its quality, and reduce the risk of missing and misunderstood information that can cause implementation mistakes.

The product owner is responsible for providing clear business needs, priorities, and enough detail to allow maturing the PBI. An engaged product owner will frequently review the developed product and incorporate learnings and corrections into the backlog, as well as ensure that business priority is driving the selection of backlog items to be prepared for development. The technical team will add the architectural and implementation considerations, allowing them to best optimize the flow of backlog items into development. This implies that the final sequence of backlog items through the *Pipeline* involves trade-offs between business and technical requirements.

Driving the *Pipeline* process can be seen as a business responsibility, so a natural owner of the activity is the product owner. But if this role is not fully available to the team, a better choice may be to have the project manager role (or some combination of the analyst and the project manager) do the overall orchestration. By applying the principle of consumer-driven planning, meaning that the consumer of the information drives the process to gather and develop it, the role most depending on the availability and quality of the contents will be in charge.

The product owner and analyst roles are key to a successful *Pipeline*. These roles do the heavy lifting in capturing and recording the outcome of the elaboration efforts that transform a Rough PBI to a Prepared PBI. They also ensure that the backlog items are consistent and of good quality. Far from producing backlog items as widgets on a factory line, the product owner and analyst need to be able to explain and go along with the work as it is integrated into the active backlog or they will become an impediment to

progress. If they are overwhelmed and falling behind, other team members need to chip in just as when a Kanban queue becomes a bottleneck.

Note that although we are describing this process with clear roles and responsibilities, this need not be a formal organization. An agile development team which shares the manager/architect/developer/tester roles will need to jointly cover the *Pipeline* responsibilities. Regardless of how the work is accomplished, access to the product owner and domain experts remains essential.

The backlog *Views* illustrated that a backlog structure may contain both business requirements and technical requirements. As described earlier, user related requirements often involve the product owner, analyst, and project manager, frequently supplemented by domain or operational experts. For technical requirements, the software or system architect is the likely role that drives the elaboration, and the participants may be some or all of the development team members depending on the size of the team and how it is organized. For system quality requirements, other roles like cyber security experts may need to be engaged. In summary, items need to be elaborated by those who have the expertise to do so.

Practically, there are different ways the pipeline activities can be performed. *Pipeline* work may be done in on-site workshops if all participants are co-located and generally available, or online either in a synchronous (meeting) or asynchronous (through documentation) manner. The best approach really depends on the availability and physical location of the participants. Of course an on-site model with face-to-face communication is the most efficient way to collaborate, but a team that has learned to work well together can also handle a distributed way of working. It is more important to have the right knowledge involved than to meet face-to-face.

Furthermore, the elaboration activities are likely iterative and happening on several levels of abstraction. Any activity that builds a better understanding of the business domain and the business needs is really a part of the pipeline. This may include an analyst building backlog *Models* like operational workflows, or UX designers creating storyboards, domain experts providing *Stories*, etc. Material on this level will normally be supporting several prepared PBIs. For instance, an elaboration activity may take on the main path through a workflow, targeting this path to be implemented and working in the next product deployment. All backlog items derived from this workflow path that are required to complete the path from start to end would share the same high business priority, although some alternatives within the path might be implemented later. The elaboration activity can be focused, and probably includes very specific domain expertise for the selected operation. The UX expert would develop the UI designs for the workflow path. The outcome of the *Pipeline* for this case would be the workflow path, the UI designs, and a set of user stories with a common business priority where each fulfills a piece of the overall workflow.

If the team is using Kanban, the *Pipeline* can be visualized as an additional queue (or set of queues) on the left side of the Kanban board. This queue can be more flexible in size and abstraction level since items often are grouped (i.e. developing a workflow, storyboard etc. for a theme or epic). Another way to view items in the pipeline is through query results from the backlog tool.

There are two factors that are key to making the *Pipeline* a success: having the right people involved, meaning individuals that understand the business and the technologies involved, and having good communication between the people involved throughout the overall process. If development team lacks access to domain expertise, they will be left guessing the requirements details, oftentimes with sad consequences. They might miss the real business value, or have a product that is not being seen and tried by real users before it is too late in development to be able to change the system operation without incurring major costs.

The need for capable and skilled resources in the pipeline activities and transfer of knowledge is most important for new development. For mature products/legacy, the backlog is typically a list of defects

combined with limited new features, and the prioritization and elaboration would not require the same level of business involvement.

*Example*

*For the next deployment, the product owner Ambitious Aaron wants the Benson system to be able to measure the level of chemicals in the pool and report back to the central monitoring system. This is a larger piece of functionality that needs to be broken down into more specific and detailed requirements that are suitable for development planning. It has not been decided yet how chemicals in the pool will be measured, so there are significant unknowns that must be sorted out before any implementation can be taken on.*

*The Benson BA organizes a workshop with Aaron, the chemists supporting the development, the Benson UX designer, and the members of the development team that handle the central monitoring system and the chemical testing module. In the workshop, the group decides on the workflow and some early UX sketches. Over the next 2 weeks, the BA continues to transform the requirement into a clear set of user stories. This is supported by an updated workflow and a UX storyboard.*

*In parallel, the chemists start an investigation into possible equipment to be used to measure the chemicals in the pool, and eventually make a selection. They work with the vendor of this equipment to provide a test setup that the development team can use for testing. The vendor is also providing specifications for how to interface with the equipment programmatically.*

*Finally the new requirement has reached a level of clarity and the specification and the test equipment is prepared enough for the development team to start the implementation.*

# The Funnel

Promising new product ideas need to reach the backlog from a variety of sources.

Your project is underway. There are ongoing elicitation efforts that keep producing ideas for your product. The initial backlog has a *Frame* and has started to be used for development.

**How and when do you introduce new product ideas into your backlog?**

Ideas for additional functionality arise from various sources. Users of the product make requests ranging from small improvements to more substantial suggestions. Some of this input fits nicely into the existing product roadmap, but other ideas challenge the system boundaries or simply do not fit with existing plans. Ideas may also come from the development team as they grow in their understanding of the business domain.

Your business organization is on the lookout for new revenue and will contribute new ideas, especially if the product is successful and performing well. Additional business opportunities can take the product to places far beyond the scope of the original product plan.

New ideas are usually sketchy and need additional analysis and refinement before they can be ready for the refinement of the *Pipeline*. They may need to be explored for both business value and technical feasibility. For larger items there needs to be some degree of understanding of the cost to implement, for example if the additional functionality requires the addition of a database or the development of a new user interface for an additional user role not covered by the initial system, or require adding a new skillset to the team.

A business idea is not necessarily the same as a user need. Users may desire improvements and changes for a product that do not directly increase product revenue. An idea might make the product more attractive and better positioned against competing systems, in which case the improvement could make sense, or it might not bring any benefits that are attractive to the business owner, i.e. not have any business value. Just because you can follow up on every user request does not mean you should.

New ideas may not fit in the current backlog *Frame*, and you may not want to restructure your backlog before you know whether these ideas will be discarded or acted upon.

You want to manage all your product ideas, including the ones that were abandoned, and keep them in mind for future work. To be able to follow up with customers and internal stakeholders, it is good to know where the idea came from and why it was conceived. Your organization may have a system to keep track of this, but if this information is too remote from the engineering team it is more difficult to get the connection between the new ideas and the already structured backlog.

**Therefore, 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. Maybe the idea belongs with a related system in your organization instead.

Development and vetting of business ideas is a responsibly of the product owner's business organization. Ideas that are deemed interesting from a business perspective, i.e. that have enough business value, are defined as potential candidates for development. They need to be scoped out and defined with enough detail that they can be handed over to the team for a technical analysis. At this point (i.e. when you need to involve the engineering team to proceed with the analysis and detailing of the business requirements) they constitute work for the team, and so they should be added to the backlog as part of the "… *ordered list of everything that might be needed in the product …"* If the technical fit and cost to develop is acceptable, the idea continues through the *Funnel*; if not, it is discarded. This investigation process is likely iterative with more detail added through the various analysis activities.

If the new items have a good fit in your current backlog *Frame*, they can be added directly into the current structure – larger items will most likely be added as *Placeholders*. If you do not have a clear place to insert these items you will need to either expand the *Frame* or include a special node for future ideas as shown in figure 3 below. At this point the items need additional work before they can move into implementation – they are what we introduced as rough PBIs in the introduction section.

If you want to manage your rough PBIs in your ALM tool, you can either keep them in a separate part of the backlog, for example assigning them to a named release that could be called "Future", or identify them with a tag. If the decision is to not further pursue items in this collection at some point, they may be moved to a separate sub-category for "discarded ideas" or may be marked in other ways (tagged, attribute, etc.). In figure 3 we have chosen to use a discarded category in the backlog.

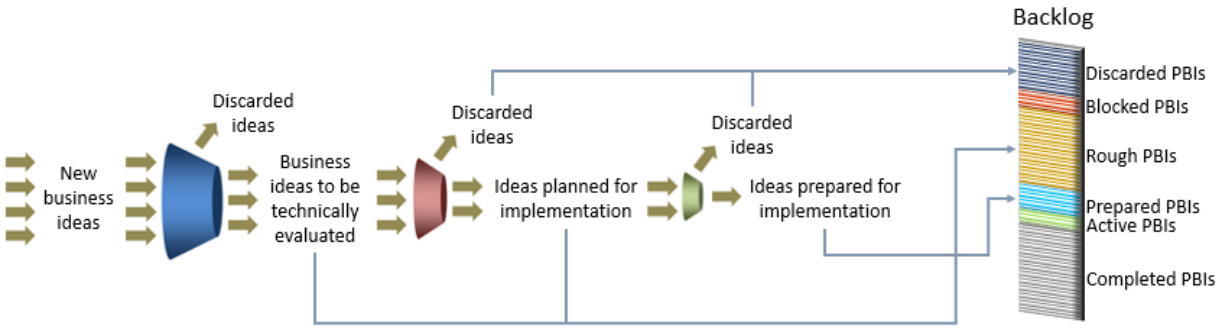



Figure 3: Backlog Funnel process

Although the business function is likely keeping a record of product ideas, we recommend that the Team also keep track of all reasonable ideas that they spent time to explore somewhere in their backlog, even those that were eventually discarded. If the overall contents of the *Funnel* are stored in a bunch of documents and disparate tools like customer portals and business portfolio systems, there is the risk of losing the traceability of the idea. The same idea may reappear several times throughout the lifetime of the product, and unless the people currently involved in the project remember it, you may end up exploring and discarding the same idea multiple times. And you never know what the future will bring. Mining ideas that were dismissed before can be useful either when planning a related product, or because the business or technology has sufficiently changed to support pursuing previously unfeasible ideas.

We suggest that you should keep track of only the reasonable ideas. There is no need to waste time recording items that you consider irrelevant. What you consider reasonable depends on who submitted the idea and to what extent you want to track all input. Any idea from a very important customer likely needs to be captured. If an idea belongs in another product in your organization and is captured there, you may not need to record it.

The deeper exploration of business ideas identified in the *funnel* is outside the scope of our paper. The focus of this pattern is not on the business process to refine and vet new ideas that are in the funnel, but instead on how to manage the flow of these ideas into the product pipeline and backlog.

*Example*

*As the Benson system is becoming popular and gaining market share, a number of requests for additional functionality are coming from both pool maintenance companies and pool owners. Equipment manufacturers see new opportunities to sell their products and are also proposing new opportunities.*

*Some of these ideas are highly unrealistic (clearly outside the scope of the system, highly expensive, etc.) and are discarded without further work. But any potential opportunity that seem interesting is recorded as a possible future improvement. The product owner Ambitious Aaron works with the Benson BA to select ideas that are added to the product roadmap. During this process they also decide to discard ideas that they decide not to invest time and effort into investigating.*

*When growing the roadmap they do consult with the technical team to get a better understanding of the cost of implementation. Ideas that are explored by the team are included in the teams Backlog under a node called Future Ideas. As they get added to the implementation roadmap, these ideas are moved into proposed releases based on their business value. As ideas are evaluated further, some are again discarded and moved under a Discarded Ideas node.*

# The Maintenance

A shared storage where correctness of the stored material matters require regular upkeep.

Your project has been active for a while and you have a backlog with rich contents. You have started to experience problems with the backlog contents not reflecting the current state of the product and of the development status.

**How do you keep your backlog as a reasonably accurate representation of the planned and implemented product?**

Your backlog includes a representation of the work you need to perform to produce the next deployable version of your product. If your backlog is outdated, you lose your knowledge of where you are – what work is done and what work is left to do. Your planning will suffer, and so will your ability to report (whether to the team or to the stakeholders).

As a product progresses through development the understanding of the product both from a technical and business perspective will change, requiring you to update your backlog. Exposure to real users causes the business to realize new opportunities, while other features no longer are as important as initially perceived. As technology is replaced it results in changes to the technical requirements. Your product environment changes – products that you depend on release new versions with modified APIs, products that you planned to integrate with are going obsolete, competitors force you to change priorities to catch up with popular features, etc.

As the backlog content grows, your initial *Frame* will likely become too limited. If navigating the backlog and/or producing *Answers* is getting compromised by the current backlog structure, you will want to update the structure and possibly also the traceability model for your *Connections*.

User stories are rarely totally independent. So when implementing a user story, the team may end up implementing parts of (or even all of) other user stories in the backlog. The team might also decide in agreement with the product owner to fulfill most of the scope/acceptance criteria for a user story but leave a few parts for future development. Since you depend on the correctness of the backlog contents to drive development and provide you team and your stakeholders with *Answers*, these changes need to be reflected back into the backlog.

The backlog is a shared tool for everybody on the team, and the contents are changed and updated by any team member as new contents are added, plans are developed, work is being done, and deliverables are being accepted. This is a continuous and natural flow as part of the development effort. But with so many players, there is a risk that the quality of contents of the backlog will deteriorate over time because the responsibility of updating the contents is unclear or just simply because people forget to record changes, update status, or make links. A test case fails (recorded) and a bug is raised (recorded) linked to the test case, but the link to the requirement is forgotten. Now the metrics of bug distribution by requirement will provide the wrong result. Or a change-set is added to the codebase resolving a defect, but the defect remains in an open state because the developer forgot to update its status. If she also forgot to link the change-set to the defect there is no traceability to show in which build a bug was fixed.

The more detail the team has in their backlog, the more time it will take to maintain the contents. Although it may feel like wasted effort to keep backlog items updated, the effect of not doing so can easily cause more waste. If the contents get significantly out of date, the job of bringing it up to date will be costly. The worst scenario is that the effort to update the backlog becomes too costly and the capability of automated retrieval of accurate backlog *Answers* is lost.

**Therefore, 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.

An important part of the upkeep involves keeping user stories true to the actual scope of implementation. If a user story was only partially implemented, the original story needs to be split into one story reflecting the implemented scope, and one or more user stories for the remaining scope. If implementing a user story caused partial implementation of another user story, then this user story not only needs to be split into the implemented scope and remaining scope, but because this user story was not exposed during the sprint planning or 3 amigos activity, its definition (details and acceptance criteria) need to be updated. This work ideally should have been done during planning, but more likely items were missed or the intended update never happened and so the need for clean-up should be caught in the regular upkeep process.

The required level of precision for the various backlog items needs to be agreed on by the team. Team members must also commit to keep backlog items up-to-date and accurate. The degree of precision and accuracy depends on the work style of the team, their communication level, whether they are co-located or not, and of the nature of the product under development, i.e. size, safety concerns, auditability requirements, etc.. Bottom line, incorrect contents when consumed will cause waste. If a User Story change was agreed by a subset of the team, but the persons doing the final implementation were not included, they waste their time implementing the wrong solution. If changes to acceptance criteria were not communicated, acceptance tests end up being insufficient, leading to defects not being found before deployment.

In finding the correct balance between effort and correctness of the contents, the team needs to focus on who and how the backlog contents are being consumed. Whoever is using the information captured in the backlog items to perform implementation/testing/reporting or any other key team task needs the correct contents, or their work will be flawed. This is not a question of “communication over documentation”. Whether the backlog is a set of stickies on the wall or the contents in an electronic backlog management tool, it is an integral part of the team communication.

Although ownership of the backlog rests with the whole team, it is good for a larger team to have an assigned keeper of the backlog. This role ensures that team members know how best to make their changes, that the backlog keeps its intended/agreed upon structure, does the structural refactoring of the backlog when necessary, and takes care that the contents are kept up to date. This keeper may be the project manager (although this role is often too busy to handle the details), the business analyst, or it may be someone in a QA role. The business analyst role and the QA role both have a vested interest in that the contents are in good shape for metrics/reporting purposes. In particular, this role should always be present at an event where backlog items are moved into development, and is also responsible for updating the backlog with the changes and additional details that surface as developers gain the clarity needed for implementation.

As we have mentioned in earlier patterns [Hva2014], the more detail you want to track in the backlog the more effort it requires to maintain it. If you keep your backlog implementation lean and not too detailed you better about to manage this workload. So think carefully before you decide to track attributes such as assigned to, tasks, or even detailed estimates and actual time to implement PBIs. Are these metrics important?

The contents of the backlog in your ALM tool matters. Although nothing beats the effectiveness of oral communication, it comes with a higher degree of unpredictability as it depends on people's memory. Any reader of the Checklist Manifesto [Gaw2009] will appreciate that some level of structure in capturing and executing tasks is beneficial, even for highly trained, experienced people.

If the backlog is maintained on a regular basis following a clear and consistent organization, then the development team will work from a list of items that reflect the current understanding and priorities of the product under development. Although the regular upkeep requires some investment, it is well worth the effort as it allows the team to keep the pace and always be in control of their progress. The lack of surprises reduces the stress on the team. Backlog maintenance is like flossing – daily maintenance will avoid root canal moments.

There are several sources of change that cause the need to do backlog *Maintenance*, from smaller tasks like updating status, to major work like refactoring the backlog *Frame*. One could argue that there are project triggers that should result in this update work being done, but items are missed, good intentions not always followed, and mistakes are done. This is why regular *Maintenance* is needed.

*Example*

*After the workshop that defined the new user stories for the capability to measure the level of chemicals in the pool, the Benson BA updates the backlog with a new business requirement and all the newly identified user stories. She realizes that there is some overlap with existing user stories and proceeds to correct this. It requires a bit of restructuring of the backlog Frame. She also modifies some of the wording to stay with a consistent terminology for all the chemically related backlog items.*

*The next day there is a planning event where some of the acceptance criteria is refined for a couple of user stories, and these items in the backlog are corrected during the planning meeting. The new acceptance criteria cause the QA engineer to review and update three existing test cases and add two new ones.*

*The Benson BA discovers that new defects added recently have not been linked to user stories and so are not exposed by the query the team is using to do their weekly triage of defects. She immediately corrects this so that the next triage event will be done from the correct list of open defects.*

## 3. COMMENTS

As stated in the introduction, this paper is part of a larger work on Product Backlog creation and management for software development. We intend to continue writing more backlog patterns. In particular, we are interested in patterns for coordinating activities of larger and distributed teams, potentially working from multiple backlogs.

Although the term Product Backlog originates from Scrum, we find it has become a term adopted for agile methodologies in general [Rad2016, Agi2015, Bra2016, Mul2016, Wik2016]. Consequently, our patterns deliberately address this wider audience in addition to teams that adopt Scrum. Our patterns can be useful to any software team that actively works with a backlog of work items.

Scrum methodology, with its focus on time boxes, tends to describe meetings or discrete events which result in moving from one phase to another. For example, Mike Cohn has a good description of preparing items for the sprint planning in a meeting he calls backlog refinement [Coh2015]. His description includes some activities of our *Pipeline* and *Maintenance* patterns but with an important distinction—he describes backlog refinement or backlog grooming as a meeting that takes place three days before the end of the current sprint.

Rather than a discrete event, we see refinement as an ongoing process happening in parallel with but slightly ahead of development. There may or may not be an official meeting to groom the backlog. Scrum practitioners might hold a meeting to confirm the status of ongoing refinement and plan their next chunk of work. Those who use Kanban or other agile approaches that operate from a backlog of work items can also find our patterns useful. Our patterns support the continuous flow of items into the backlog, which aligns with their preferred way of working.

With our *Maintenance* pattern we distinguish between the tasks of keeping the backlog contents up to date (which includes linking, rephrasing, modifying and in general cleaning up the backlog contents) from the equally important task of preparing items from a business perspective, described in the *Pipeline* pattern. A Scrum practitioner might consider both of these activities as being part of backlog grooming.

## 3. ACKNOWLEDGEMENTS

Many thanks to our shepherd, David Kane. Without your attention to details and extensive comments we would not have made so much progress. You made us rewrite and rethink our patterns, and we are grateful for your advice and insights. We would also like to thank our PLoP 2016 workshop participants for their feedback: Miyuki Mizutani, Eri Shimomukai, Shéhérazade Benzerga, Haruka Mori, and Norihiko Kimura. Despite our paper being on a topic that was unfamiliar to most of you, you had prepared by thorough reading and your comments helped clarify and improve our work. 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.

## APPENDIX A – INTRODUCING THE RUNNING EXAMPLE

The example case used throughout the patterns in this paper is based on an imaginary development effort since confidentiality issues block the authors from using a real-world example. The flow of activities are still realistic, and based on our combined 50+ years of system development experience. The requirements in the example were first developed to evaluate requirements/backlog management tools. In the running example, we use illustrations from the ALM tool Microsoft Team Foundation Server (TFS).

*Example: The Benson Automated Pool Cleaning System*

*Living in the southern part of the US, I have a pool. I also had a pool boy, a trustworthy and hardworking high school kid who started his own pool cleaning company after working as a life guard at the community swimming pool one summer. My pool was crystal clear and life was good. But nothing lasts forever. High school completed, my "perfect" pool boy left for college in another city. A few months and several mediocre pool cleaning companies later I found myself in a permanent role of being my own pool boy. And I started dreaming of an automated pool cleaning system. Here is the story of building the backlog for the perfect pool cleaning system and yes, it is named after my perfect pool boy…*

*A first round of elicitation activities consisted of interviewing friends who maintain their own pools, a couple of professional pool cleaners, and the owner of a company that would sell and operate the automated pool cleaning system. This produced a vision statement, some high level goals, and unstructured data on user profiles, system parts, functionality, cost models, and legal aspects. And some funny stories from my friends' more or less successful pool cleaning activities.*

***Vision**: "The Benson Pool Cleaning System keeps your pool water crystal clear and correctly balanced with no effort from the owner. Equipment and Chemicals are monitored remotely and replenished and serviced based on automated system reports."*

***Main goals**: Remove Debris, Maintain Water Quality, Remote Monitor Equipment Operation, Schedule Maintenance, Low Cost, Safe Operation.*

The example is continued in the individual patterns.

## 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).

[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).

[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.

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

[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

[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.