Patterns to Build the Magic Backlog

LISE B. HVATUM

REBECCA WIRFS-BROCK, WIRFS-BROCK ASSOCIATES

Agile development processes are driven from a backlog of development items. This paper describes patterns to build and structure the backlog for an agile development effort using the outcomes of requirements elicitation and analysis. 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:

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.

1. INTRODUCTION

The focus of this paper is the challenge of developing what the authors have come to think of as the "Magic Backlog" – this well-founded set of detailed software requirements that agile process descriptions seem to start from [SS2013, DBLV2012, GG2012]. Items from the backlog are analyzed, elaborated and changed when starting development of the software, whether it is a Scrum sprint or an item being moved to work-in-progress on a Kanban board. But somehow there is already a backlog where functionality is broken into smaller portions ready for consumption by an agile development team. Who makes this backlog and how? What should be in the backlog? What are the benefits of a well-organized and maintained backlog?

One reason why agile software development processes may seem to start from an existing backlog, with little focus on how this backlog came to life, is that the process of developing the backlog involves roles like Product Owners and Business Analysts that are more peripheral to the actual software development. There are a number of books, blogs, and articles that provide methods and techniques for how to work with stakeholders to elicit requirements, and how to analyze and process the results to reach the detailed software requirements [AB2006, Got2002, Got2005, GB2012, HH2008, Wie2009, Wie2006]. The patterns described in this paper add to the body of knowledge of software requirements engineering by providing practical advice on how to build a good backlog from the outcome of analysis efforts.

The need to formalize the backlog increases with the size of the project. Our paper 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 tools to manage the backlog. The target audience of this paper is primarily the Business Analyst but including other roles involved in defining, refining, and managing requirements for a software product in the context of an agile development effort, including the Product Owner, testers, developers and architects.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission. A preliminary version of this paper was presented in a writers' workshop at the 20th European Conference on Pattern Languages of Programs (EuroPLoP). EuroPLoP'15, July 8-12, Irsee, Germany. Copyright 2015 is held by the author(s). HILLSIDE 978-1-4503-3847-9.

We do not discount useful requirements models with our patterns, but suggest ways to better integrate them into the backlog and present a clearer view of how the system functionality relates to various requirements models and artifacts.

The appendix provides an overview of requirements engineering and techniques used by agile teams to develop software requirements. This defines terminology and concepts for software requirements as used in this paper, and constitutes the overall context for the "Magic Backlog" patterns. Readers who want to refresh their knowledge about requirements engineering and the relation to agile software development, including User Stories and Story Mapping, should read the appendix before continuing the paper. Note that we may use terms in the following sections that are explained in the Appendix (for example 3 Amigos Meeting), so if you find an unfamiliar term please check the Appendix.

2. THE BACKLOG

The term "Backlog" or more formally "Product Backlog" is part of the Scrum terminology. This is the definition from 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 list are Product Backlog Items (PBIs), which is a generic term that allows for any representation of a product "need" to be included in the backlog. A common backlog item is the User Story, but to think of the backlog purely as a list of user stories is too simplistic for any large and complex system. Given that the backlog is the "single source" for driving system development, you want the backlog to give you the full picture of the product requirements. This implies showing the relationship between business requirements and user stories, showing dependencies between requirements on the same level of granularity, incorporating quality requirements and technical requirements, keeping the outcome of discussions with the stakeholders, showing the state of the development and testing, and including defects as they also represent work items in the backlog.

During software development the product backlog is always in flux. Some items are being completed while new items are being added, and existing items are being modified – either elaborated with more detail or changed as the product understanding is maturing. As stated in the Scrum Guide [SS2013]:

"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."

To understand the relationship between the backlog and the requirements process consider the flow of requirements activities shown in Figure 1. This process has six main activity categories: gathering requirements, where the elicitation techniques belong; processing elicitation results; elaborating requirements to produce detailed and validated requirements ready for development; realization of the detailed requirements in the implemented software system; maintaining the backlog which includes dealing with planning, change management, and traceability; and reporting to gain information about the backlog status.

The flow of activities is not a one-pass process, but a continuous flow that continues throughout the development of a product. This flow feeds the backlog with contents – the processed details derived from the requirements elicitation "raw data." Elicitation results are not the only inputs to the process of defining the detailed requirements and creating the backlog. As the product starts to take shape and mature, the backlog will be influenced by user experience design, system architecture, and testing strategy – all of which will cause the creation of backlog items in their own right. 3 Amigos meetings (or iteration planning meetings), which are used to gain agreement and precision on what it means for a story to be accepted as "complete," and team discussions will result in refined backlog items to include more details and acceptance criteria, and may cause items to change and even be split into more detailed items.

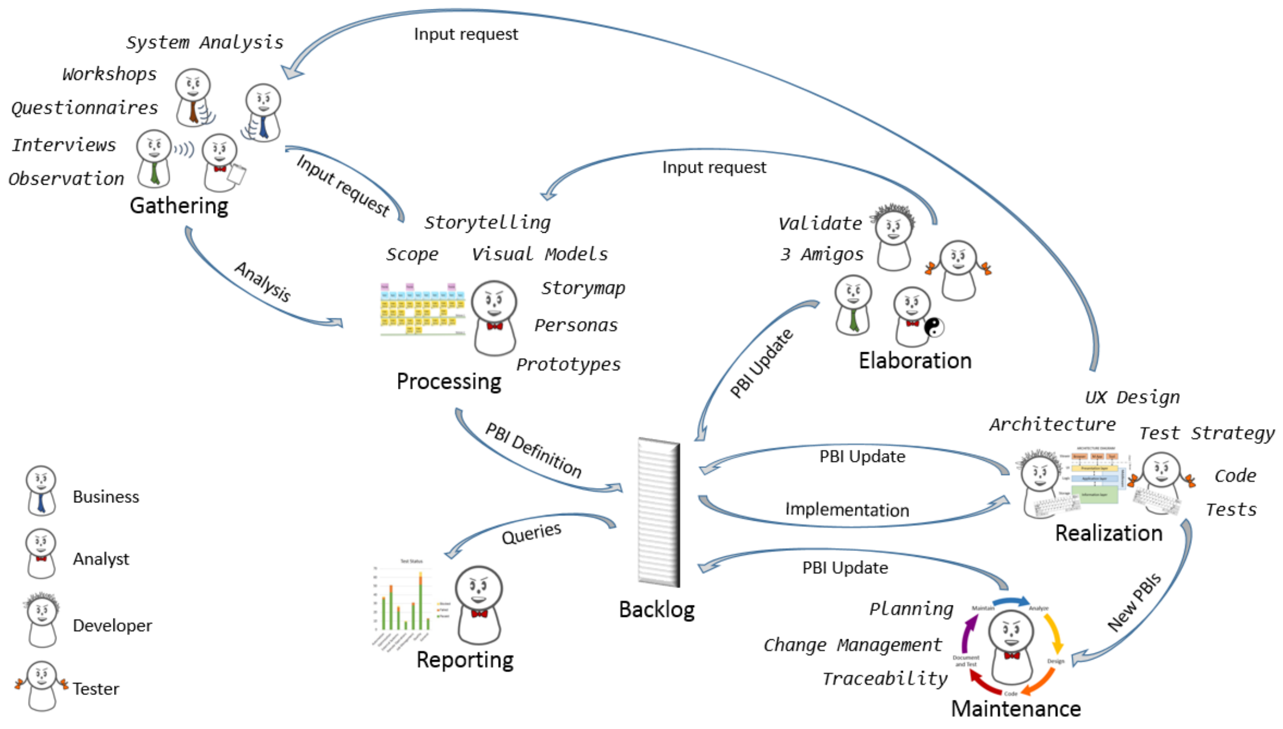


Figure 1: Requirements Engineering Flow

Because the backlog serves as the "single source" for the product development, the structure of the backlog must be carefully planned, evolved and maintained so that it provides the proper level of clarity needed for correct implementation. A smaller backlog can be managed informally (e.g. "stickies" on a wall), but any larger undertaking will need a tool to manage the backlog. You will want a tool that allows for creating items with unique IDs, linking items with specific relationships (tested by, defect, parent-child, etc.), allowing for the items to have descriptions and other attributes, and if you want to create trend reports you will need the tool to record changes over time.

Since the continued grooming of the backlog requires effort, there is a balance to be found between keeping all the details of development and storing only the big picture. In general, one should ensure that the formal backlog contains what is important for long term persistence – what will be consumed in the future when the team members can no longer remember all the details. Typically this is what is needed for regression testing, possible security audits, etc. It is beneficial to capture sufficient details about requirements to enable reuse in a future product. Detail that is only needed for the short term can happily live outside the backlog tool, for example, on a wall-based Kanban board. A good example of out-of-backlog details are developer tasks, unless you need to know two years from now if a database performance bug was fixed by John or Anna. Bottom line, the full backlog may be implemented by a combination of formal tools and informal methods.

There is another aspect of the backlog that can change when a tool is used to manage it. In the Scrum process, a backlog item does not have an explicit existence in the product backlog after it is "Done" (implemented and tested). But with a tool to manage large collections, requirements backlog items can live on in a "Done" state to support regression testing because they are now part of a structure that represents the full product.

Building on Wiegers' definition of quality characteristics for requirements from Table 1 (see Appendix), the following additional criteria should be fulfilled to have a quality backlog when using an electronic tool to manage the backlog:

- ✓ Enable a product overview
  Viewing the backlog should provide a high-level understanding of the product scope and the overall structure of the requirements on all levels of granularity to the product owner, the team and the business stakeholders.
- ✓ Enable a technical view
  The development team needs to view the product differently than simply user functionality. They are usually more comfortable with a view that reflects the system architecture and major software parts.
- ✓ Support navigation
  Easily move around within the requirements, finding the right details and following links between items.
- ✓ Support planning
  Allow associating items with releases and (depending on development process) iterations/sprints.
- ✓ Support answers about progress
  For example burn-down charts, defect status vs. requirements, testing vs. requirements.
- ✓ Provide the ability to associate requirements into collections
  For example all requirements that belong to a workflow or narrative or a use case or a story map.
- ✓ Capture quality criteria for individual requirements
- ✓ Capture quality criteria for requirement collections

Although Product Backlog Item (PBI) is a generic term that allows for any representation of a product "need" to be included in the backlog, many only equate a product backlog item with a user story. This is insufficient to represent requirements for complex products. Below we introduce specific terminology used in the pattern collection to consistently describe backlog elements that organize and represent requirements:

Item or Backlog Item – Any representation of a product need.

Backlog Structure – A multi-level way of organizing the functionality of a product represented as a linked hierarchy of Structure Elements. At the top of this Backlog Structure are structure elements that represent the most general organization of system functionality; at the lowest level structure elements are linked directly to discrete system functionality (typically user stories). Common ways of organizing system functionality are into Epics or Themes.

Structure Element – Any backlog item that represents part of a structured view onto detailed functional or technical backlog items.

Model – A structured representation of another perspective onto the system's functionality, such as a Use Case Model or an Information Model that has a structure and additional requirements.

System Quality Backlog Item – A non-functional and/or quality related requirement of the system, which can either represent a specific Quality Scenario, or a quality requirement that applies to more than one individual functional backlog item.

Functional Backlog Item – A user story or other functional requirement.

Persona Backlog Item – A backlog item that represents a user persona.

Technical Backlog Item – An internal software requirement, which could represent a technical need or a constraint due to the technical environment that is not exposed to a user.

3. 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.

4. THE BACKLOG PATTERNS

In this section, we introduce patterns specifically targeting the practicalities of developing a good backlog with the quality characteristics listed above (page 4). The patterns are defined in the context of new product development for a product of significant scope and complexity, with at least a three-year time frame for gradually delivering the full system. There is an expectation that an Application Lifecycle Management (ALM) tool is used to manage the backlog, such as Doors NG, JIRA, TFS, etc., and that this tool allows the backlog items to be linked and to have attributes.

One should keep in mind that the backlog is primarily a tool for the development team (QA included) and should be designed primarily for their needs. It is not a tool to communicate with every stakeholder. Rather than trying to use the backlog for too many purposes, we recommend that the team/project manager keep a separate roadmap and business stakeholder-oriented status view. Giving business stakeholders access to the development tools that contain the backlog may cause the development team to be overly careful with their contents and thereby reduce the value of the backlog and its associated artifacts.

Individual Patterns:

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

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

Backlog People – elaborated personas span the dimensions of the user space

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

Backlog Usage Models – usage models show structural user story dependencies

Backlog Placeholders – epics are replaced by user stories when elaborated

Backlog Plans – backlog items are grouped into release candidates

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

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

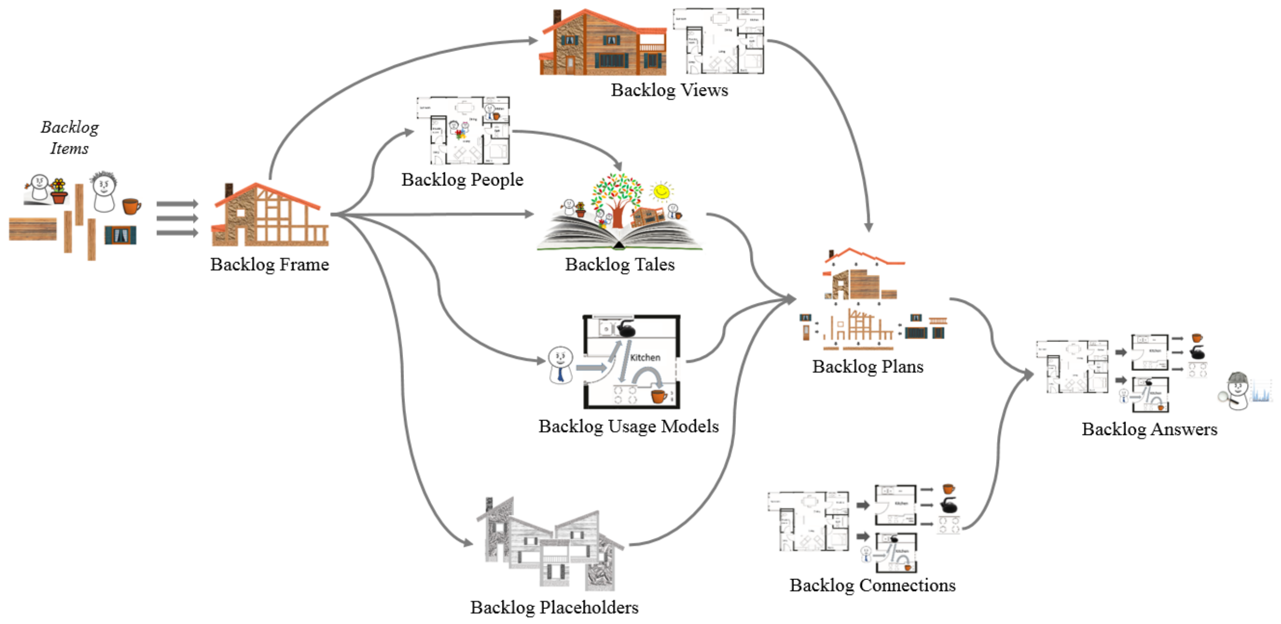


Figure 2: Backlog Patterns Sequence

The Backlog Frame

The Product Backlog is a reflection of your product.

Your project has a backlog with hundreds if not thousands of items. You are using an Application Lifecycle Management (ALM) tool to manage the backlog electronically.

How do you organize the main structure of the backlog to best provide the benefits of a quality backlog to a variety of users?

You want a backlog that supports the extended development team during the product life cycle. You must take into account the different users of the backlog that are involved, their roles (project manager, business analyst, developer, tester, etc.), and support their various activities. Their needs can be quite different. A business analyst often works on collections of items; developers on the other hand frequently address a single item at a time. The main structure of the backlog needs to represent the overall product and frame its details in a way that all backlog users are comfortable with. All users should be able to easily navigate from a high level overview to details that they are specifically interested in.

Your initial requirements elicitation and early architectural decisions have probably left you with a lot of material to go into the backlog: user roles, business goals, requirements of varying granularity, and system technical considerations, just to name a few. In information management terminology, you have a variety of *content types* that could potentially be linked together to represent the product and all the dependencies. Without a strategy for how to approach all this content, you are likely overwhelmed by its sheer volume and diversity. Unless you can slice out a subset of this material as central to the backlog, and organize the backlog accordingly, you risk creating an overly complex and hard to maintain structure.

If you now also have to learn to use an ALM tool, your challenges are really compounded. Most ALM tools give limited guidance on how to create and organize the backlog. When creating a new project in the tool, you select a template for the backlog contents that typically contains a few pre-defined content types such as user stories, features, bugs, etc. But the way to structure the backlog contents is left entirely up to the user by how they choose to manually link backlog objects. When first starting to use an ALM tool, it is not uncommon to be too enthusiastic by creating too many links, entering too much content, and creating too much structure, resulting in a backlog that is overly detailed and difficult to navigate. This can result in a hard to use backlog: specific items can be difficult to find, reports hard to configure, and the essence of the system functionality becomes drowned in detailed items.

In agile process documentation you often see the backlog described as a set of epics and user stories. However, a single-level list of these content types alone does not provide you with a good understanding of the product you are going to build. You need to see the "big picture" and understand how functional items relate to each other and how they contribute to larger pieces of functionality that comprise the whole product.

Therefore, 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. Also, a functional structure typically takes shape earlier in the development cycle than for example an architectural view. Any primary structure for your backlog should be able to represent relationships between other backlog objects, since any activity should be traceable to a business requirement.

Your backlog structure hierarchically frames and organizes backlog functionality. Viewing the top level presents an overview of the system functionality. Following the breakdown of a higher-level item involves traversing parent-child type relationships through to the lowest levels of detail. The top-level items in your backlog structure should represent the high-level business requirements or themes. The second level items are detailed business requirements that represent the main functions of the product as understood by the business stakeholders. The leaves in your frame represent the user stories and other backlog items that are broken down by the development team into the tasks that will realize the product. A set of user stories and other PBIs linked to the same parent requirement forms a set of requirements, that, when implemented, fulfills the parent requirement. Since links between items are easy to change, the structure can be easily modified.

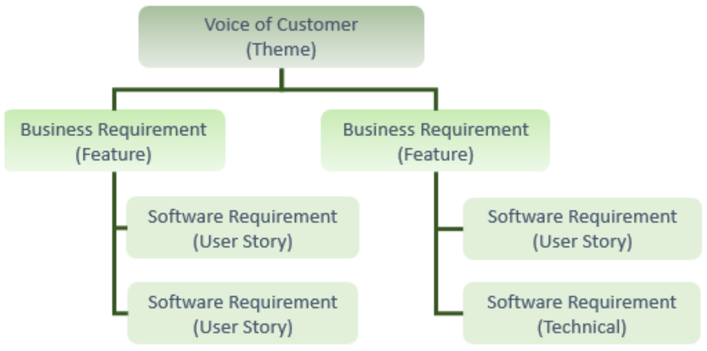
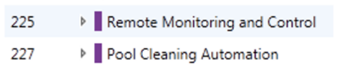

Voice of Customer (Theme)
Business Requirement (Feature)
Business Requirement (Feature)
Software Requirement (User Story)
Software Requirement (User Story)
Software Requirement (User Story)
Software Requirement (Technical)

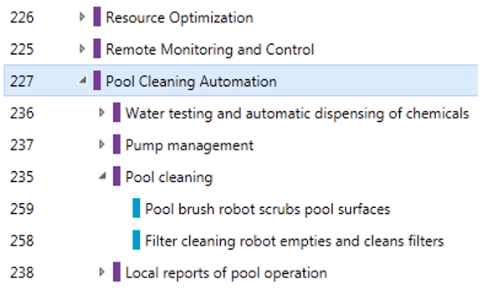
Figure 3: Primary Backlog Structure

You will want to limit the depth of your structure and keep it as simple as you can so as to avoid excessive work maintaining the backlog. A deep structure will bring practical challenges in formulating queries you may want for your dashboard or reports. Our recommendation is to frame your backlog with three levels of requirements, but certainly no more than four.

Example: Sorting through the comments and wishes from the various sources, the user profiles and their needs start to take shape. The business analyst eventually has content enough to make an initial high-level structure of the functionality, dividing it into large domains (or themes) – the debris cleaning, the water management, the remote monitoring, and the customer support. She organizes the backlog using a 3-level requirements structure:



```
226  Resource Optimization
225  Remote Monitoring and Control
227  Pool Cleaning Automation
```



```
226  Resource Optimization
225  Remote Monitoring and Control
227  Pool Cleaning Automation
236    Water testing and automatic dispensing of chemicals
237    Pump management
235    Pool cleaning
259      Pool brush robot scrubs pool surfaces
258      Filter cleaning robot empties and cleans filters
238    Local reports of pool operation
```

The top-level themes reflect the voice of the customer and are similar to the benefits listed in the product vision. Expanding below, user stories are grouped together to form functional blocks. Navigating this structure should be intuitive based on an understanding of the first and second level of system requirements.

The Backlog Views

The backlog can present itself differently, based on user need.

Sunroom
Dining
Kitchen
Powder room
Bath
Living
Entry
Bed 1

You manage a large backlog in an ALM tool where the *Backlog Frame* provides the main structure. You observe that some of the new contents added by team members do not fit well into the functional hierarchy.

How can the backlog provide representations of a product that is intuitive to a variety of user roles?

The backlog is a fundamental tool for your extended development team, and so must cater to the needs of frequent users. The functional breakdown of the *Backlog Frame* was chosen because it is the best solution to provide a common view for all roles in the team. But it is not necessarily the view that is most effective during development, or a view that gives the full representation of what to build for the product.

Your developers have a product understanding that reflects the implementation structure of the system. They would like to group needs and requirements related to the database, the User Interface, the specific APIs, or system components, rather than follow a user-oriented functional structure. So their natural view of the backlog is one where navigation follows a technical hierarchy starting from system to sub-system to module to object or function. While you could link your technical items to user features, this will not support efficient use of the backlog by developers who want to see a structural view that matches the software architecture.

In some cases, it can be impossible to link technical requirement items to the *Backlog Frame* in a way that makes sense. For example, if your team is building the infrastructure of a large system they will have a large number of work items that reflect technical requirements to internal software modules. Although a technical solution is ultimately there to support the user functionality derived from the business needs, there is no direct one-to-one mapping from technical to business requirements.

Additionally, you may have backlog items that have no obvious connection to any business requirements. These can be requirements induced by your testing strategy, like a simulator or a mock-up. Another category of backlog items that may not have a straightforward connection to user functionality is system quality requirements. Some quality requirements may be integrated into the functional items as acceptance criteria, such as a specific performance target for a user story. But it becomes more difficult if you have a performance requirement that is aggregated over a number of different functions. Depending on the type of system and quality requirements, you may want to explicitly manage these and so include them directly in the backlog.

If your product is a system (not a software application) there may be hardware selection/integration or full-scale hardware development. Mechanical, electrical and software engineers on the team will all have a different structure that represents their understanding of the system. Underlying requirements may be shared between these domains and you want them to be visible through your backlog views.

Therefore, create additional backlog structures to reflect alternate views of the product.

For a technical or architectural view, the top level of the structure represents the main technical parts of the system. As you get to lower levels of the view, you can link to a combination of technical requirements and functional items (e.g. the same leaves as in the functional backlog but also additional leaves to represent additional technical requirements). This means that detailed backlog items can be linked both to the functional product structure and to its technical or architectural structure. When adding additional views to the backlog an additional level can be added on the top to easily navigate between the various views.

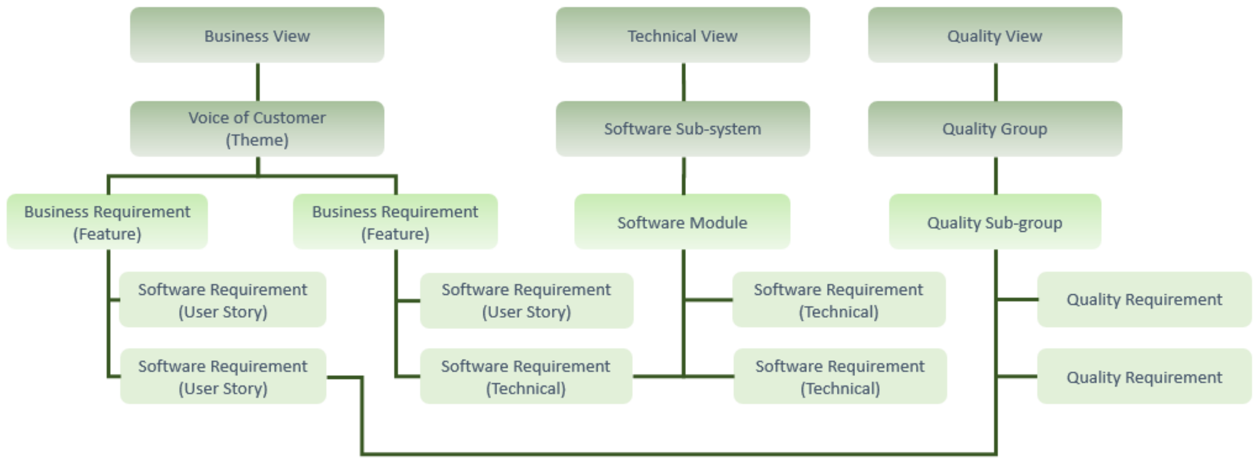
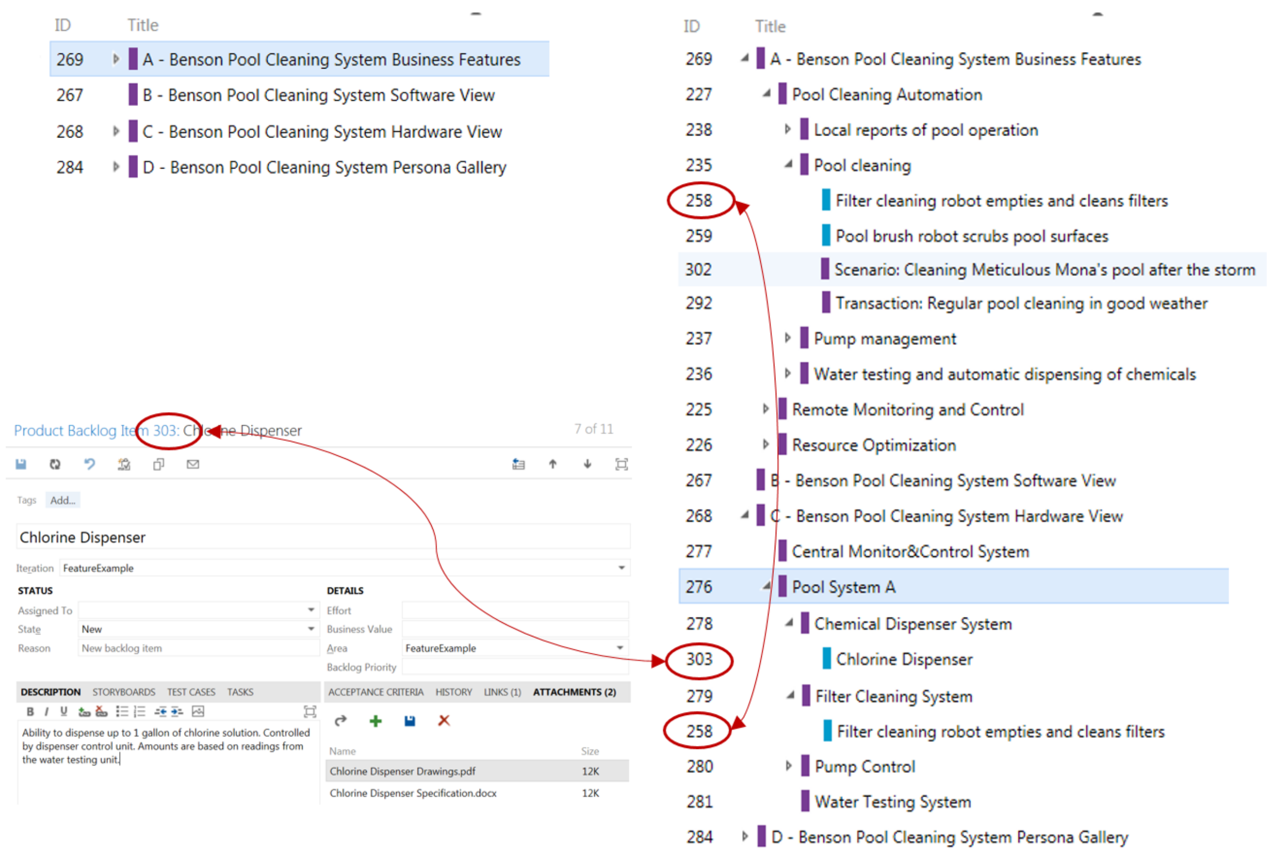

Business View
Technical View
Quality View
Voice of Customer (Theme)
Software Sub-system
Quality Group
Business Requirement (Feature)
Business Requirement (Feature)
Software Module
Quality Sub-group
Software Requirement (User Story)
Software Requirement (User Story)
Software Requirement (Technical)
Quality Requirement
Software Requirement (User Story)
Software Requirement (Technical)
Software Requirement (Technical)
Quality Requirement

Figure 4: Multiple Backlog Structures

Rozanski and Woods [RW2013] define several viewpoints (or ways of structuring a system) that are useful to organize a system's architecture: a Functional view represents the system's functional elements, their responsibilities, interfaces, and primary interactions; an Information view describes how the system stores, manages and distributes information; a Deployment view captures dependencies the system has on its runtime environment; and an Operational view describes how a system is operated, administered, and supported when it is running in its production environment. Attached to each of these views can be one or more models, which provide more detailed views of the system. For example, static information structure models, information flow models, or information lifecycle models can all be part of an Information view.

While you may consider Rozanski and Woods viewpoints as possible ways to technically structure your system, there can be other, simpler ways to structure your backlog. Creating additional views should depend on the product and on the wants and needs of the team. Any view you add should be of value to the team and ease their understanding of the system and its requirements. Adding any new view will cause more work. You need to carefully link items to ensure they are found through all possible navigation routes. Your tool also needs to support multiple types of links for backlog items.

Example: As the team takes on the realization of the system, it becomes clear that working with a backlog that only represents the user functionality is not practical for the technical staff. The BA expands the backlog structure with a new set of top-level nodes to add a hardware and software architectural view as well as a top node for the persona gallery. She links the user stories to the additional structure so that the technical resources easily see what user stories are to be fulfilled by system modules. The technical staff will then add their own details (technical requirements/specifications) to the backlog items in the description attribute and/or as new backlog items.



ID	Title
269	A - Benson Pool Cleaning System Business Features
267	B - Benson Pool Cleaning System Software View
268	C - Benson Pool Cleaning System Hardware View
284	D - Benson Pool Cleaning System Persona Gallery

Product Backlog Item 303: Chlorine Dispenser 7 of 11

Tags Add...

Chlorine Dispenser

Iteration	FeatureExample		
STATUS		**DETAILS**	
Assigned To		Effort	
State	New	Business Value	
Reason	New backlog item	Area	FeatureExample
		Backlog Priority	

DESCRIPTION STORYBOARDS TEST CASES TASKS

Ability to dispense up to 1 gallon of chlorine solution. Controlled by dispenser control unit. Amounts are based on readings from the water testing unit.

ACCEPTANCE CRITERIA HISTORY LINKS (1) **ATTACHMENTS (2)**

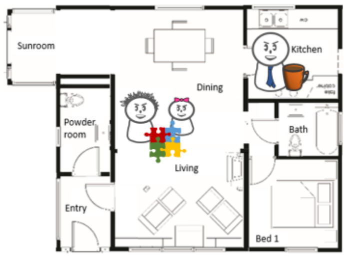
Name	Size
Chlorine Dispenser Drawings.pdf	12K
Chlorine Dispenser Specification.docx	12K

ID	Title
269	A - Benson Pool Cleaning System Business Features
227	Pool Cleaning Automation
238	Local reports of pool operation
235	Pool cleaning
258	Filter cleaning robot empties and cleans filters
259	Pool brush robot scrubs pool surfaces
302	Scenario: Cleaning Meticulous Mona's pool after the storm
292	Transaction: Regular pool cleaning in good weather
237	Pump management
236	Water testing and automatic dispensing of chemicals
225	Remote Monitoring and Control
226	Resource Optimization
267	B - Benson Pool Cleaning System Software View
268	C - Benson Pool Cleaning System Hardware View
277	Central Monitor&Control System
276	Pool System A
278	Chemical Dispenser System
303	Chlorine Dispenser
279	Filter Cleaning System
258	Filter cleaning robot empties and cleans filters
280	Pump Control
281	Water Testing System
284	D - Benson Pool Cleaning System Persona Gallery

In the above view, one user story is shown linked to both a business feature and a hardware feature. A chlorine dispenser backlog item created by the hardware development team has further specifications and technical drawings added as attachments. (Note for TFS: the above view illustrates the true linking of items in the TFS database, but due to visualization restrictions in TFS only parent/child relationships can be shown in multi-level views. Also, only one parent item is allowed for each backlog item. The second link from 258 to 279 is a "related" link and is copied into the view to illustrate our solution).

The Backlog People

Including system users in the backlog makes it personal.



Your backlog items reflect the needs for a variety of people who will use your system. Your detailed requirements are captured as user stories for the defined user roles.

How can you represent the various aspects of your system's users in a backlog?

The format of the user story is not rich enough to give a good understanding of your system's users. It provides the functional user actions with acceptance criteria, but it does not explain the different needs and preferences among users having the same role. For detailed specifications and for creating test scenarios it is valuable to understand the variance in their needs, for example, between a novice user and an experienced user. Also, you may need to support variations in an operational workflow to accommodate different work styles and skills.

Some user stories are valid for several roles, which can be combined together in generic roles like "new user" or "any user" or just plain "user." This simplifies the task of developing the requirements and reduces the number of detailed requirements in the backlog, as you do not need to repeat user stories for each user role. But at the same time, you lose the ability to trace user functionality to each user role so that you can properly address the needs for each role during development and testing.

Your Business Analyst may have created user profiles and persona descriptions, most likely in Word documents stored in a project repository. If these user profiles are even slightly difficult to access, they won't be kept in mind by the rest of the team.

Therefore, create backlog items for personas to cover the dimensions of user profiles, and associate the personas with the appropriate functional backlog items.

By creating individual backlog items for the personas, their descriptions are readily available for any team member with access to the backlog. With most ALM tools there are two ways to associate the personas with the functional backlog items in the *Backlog Frame*. The lightweight solution is to tag a user story with the name of the persona. In this case there is no direct link to the persona backlog item, as the tag is just created reusing the title of the persona item. The alternative is to fully link the persona backlog item to the functional item. This solution ensures the synchronization between persona items and functional requirements but with the extra cost of maintaining these explicit links.

The personas should span not only the major user roles, but also give diversity on experience level and personality types. The goal is to create personas that, as a combined set, will support testing of the product with enough variation to cover most usage scenarios.

Working closely with personas that have specific user roles and meaningful personality traits can give the developers a rich personal connection to their future user base. But be aware that the more personas you have, the more links you need to create and maintain. Also, when user stories are split or additional work items are added to support users with specific roles and expertise, existing links between user stories and personas will need to be refactored as well.

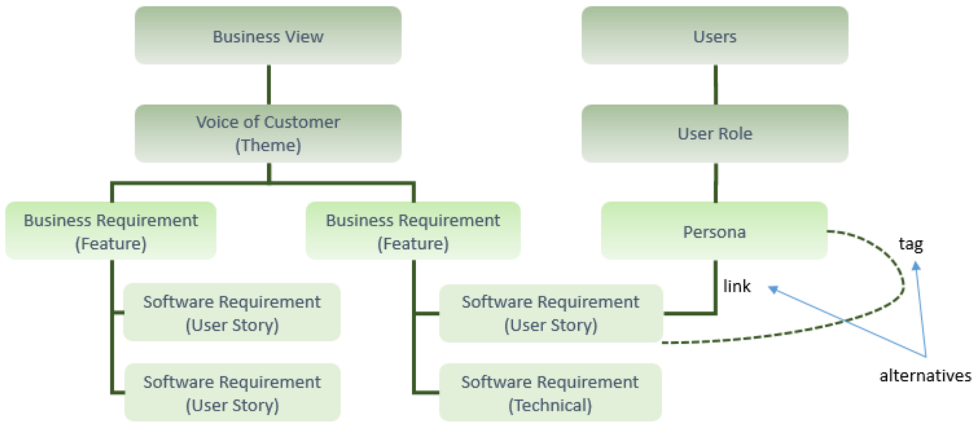
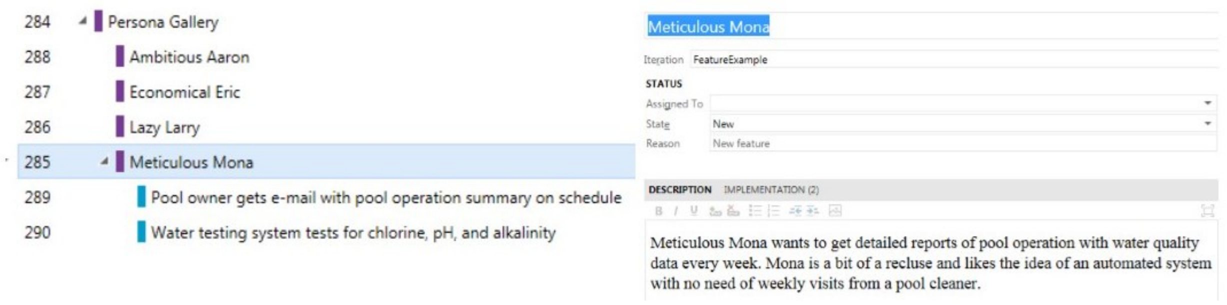

Business View
Voice of Customer (Theme)
Business Requirement (Feature)
Business Requirement (Feature)
Software Requirement (User Story)
Software Requirement (User Story)
Software Requirement (User Story)
Software Requirement (Technical)
Users
User Role
Persona
tag
link
alternatives

Figure 5: Associating Personas with User Stories

Example: Pool owners are far from being a homogeneous group, so three different user profiles (personas) were defined to represent a variety of pool owners with different skills, interests, and personalities. Another persona was added to represent the pool cleaning business owner:

> ***Meticulous Mona** wants detailed reports of pool operation and water quality data every week. Mona is a bit of a recluse and likes the idea of an automated system with no need of weekly visits from a pool cleaner.*
>
> ***Lazy Larry** wants to spend his time drifting in the pool with a beer cooler and no worries about the practicalities of being a pool owner. He is not concerned about the cost of the pool operation.*
>
> ***Economical Eric** is a bit of a handyman and wants to do his own monitoring and maintenance with the minimum support from the pool maintenance company (actually he could have maintained the pool manually had it not been for the fact that he travels extensively and does not want people in his yard while he is away)*
>
> ***Ambitious Aaron** owns a pool construction and maintenance company and wants to be the preferred supplier for the high-income population of pool owners in the area. He believes an automated pool cleaning system has a certain coolness factor for his customers, it can reduce his cost for manual labor, and enable him to sell a bunch of additional equipment when building new pools, as well as more chemicals for the maintenance since the automation means he can handle more customers with less staff.*



284	Persona Gallery
288	Ambitious Aaron
287	Economical Eric
286	Lazy Larry
285	Meticulous Mona
289	Pool owner gets e-mail with pool operation summary on schedule
290	Water testing system tests for chlorine, pH, and alkalinity

Meticulous Mona

Iteration: FeatureExample

STATUS

Assigned To:

State: New

Reason: New feature

DESCRIPTION IMPLEMENTATION (2)

Meticulous Mona wants to get detailed reports of pool operation with water quality data every week. Mona is a bit of a recluse and likes the idea of an automated system with no need of weekly visits from a pool cleaner.

For example, the description attribute for Meticulous Mona lists her particular needs and preferences. Alternatively this backlog item could link to more elaborate persona descriptions either in document form or online in other tools us by UX designers.

The Backlog Tales

Once upon a time a User performed some Actions to reach a Goal.

Your backlog has a large number of user stories for a variety of user roles. Although you have a decent *Backlog Frame* and a technical *Backlog View*, you realize you do not have a good understanding of how users are interacting with the system.

How can you improve the understanding of how users interact with the system and the impact on dependencies between individual user stories?

Narratives are a powerful way to document how a system is used by different roles and by users with different characteristics. When capturing and studying narratives of users operating your system, you will probably recognize user stories from your backlog, or find new user stories that you need to add to the backlog, so there is a strong relationship between the narratives and your product backlog items. Although by nature, narratives have an unstructured content, and it may be difficult to associate them with specific requirements.

Your narratives are most likely captured in Word documents and stored in a content management system. For the narratives to make an impact on the usability and functional flexibility of your system you want your developers to have easy access to the documents and to see how each narrative applies to the backlog items currently under development.

Therefore, include narratives that give a free-form representation of product usage in your backlog.

How you include a narrative in your backlog depends on the level of detail in your narrative, and also on how you want to find it (through navigation and queries). Most likely your narrative will span multiple user stories, and the natural level to link it in is to the feature level. Figure 6 shows one possible way to incorporate narratives in a backlog structure. In this case, a backlog item is created to represent the narrative, and this is linked to the appropriate feature. The actual text for the narrative is captured in a document which is then uploaded as an attachment to the narrative backlog item. Either a full link or a tag will show the connection between a narrative and the personas involved in the narrative.

From this basic structure there are a number of possible elaborations and/or alternatives:

- The narrative may be linked to the theme level instead of to the feature if this better reflects the scope of the narrative.
- There may be multiple narratives linked to the same feature, each of them providing insights into product usage for a subset or all user stories linked to the feature.
- You can link detailed requirements to the narratives and make explicit which user stories need to be implemented to fulfill the narrative.
- The narrative can be captured within the backlog item itself, or be in a separate document attached to or referenced from the backlog item.

- If you capture the narratives in separate documents, you may want to maintain them outside the ALM tool and reference them in a backlog item. This way the people contributing to the narratives will not need access to your ALM system.
- Creating a work item for the narrative allows you to set attributes for the narrative like the planned release and a business priority, and to associate it with test cases. If this is not needed on your project you may add the narrative as an attachment or as a reference directly from your feature work item.

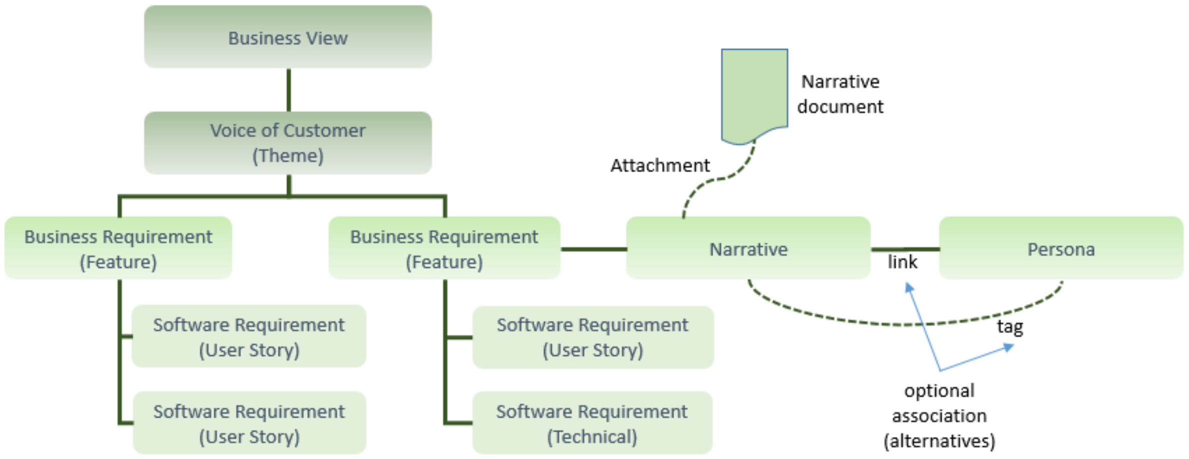
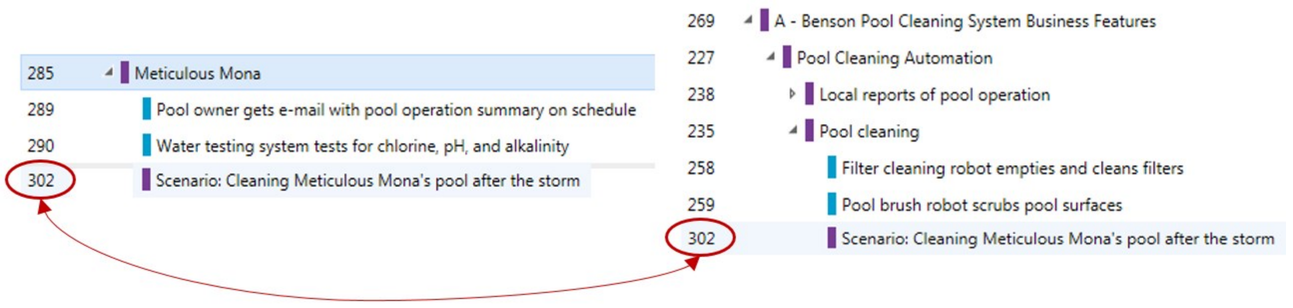

Business View
Voice of Customer (Theme)
Business Requirement (Feature)
Business Requirement (Feature)
Software Requirement (User Story)
Software Requirement (User Story)
Software Requirement (User Story)
Software Requirement (Technical)
Narrative document
Attachment
Narrative
link
Persona
tag
optional association (alternatives)

Figure 6: Associating Narratives with Requirements

The purpose of the *Backlog Tales* is primarily to improve the understanding of system usage. To keep the workload to a minimum while getting the benefits from the *Backlog Tales,* we recommend you choose a lightweight implementation of this pattern, and make links between your narratives and other product backlog items only when they clearly add value.

Example: To get a better idea of how the functionality fits together and what is interdependent the BA starts writing short stories about the personas and their life within the context of the automated pool cleaning system. Initially these are happy stories about the easy life of people who no longer need to clean their pools. But then she starts adding some trouble into their lives, like when Larry came home from Toronto and realized his pool was leaking and the water bill was going through the roof, or when Eric's kids held a pool party that threw the water quality out of whack. Events like rainstorms and freezing nights get their own stories, like the one for Mona when her pool got filled with debris and the water quality was severely impacted by a massive storm:



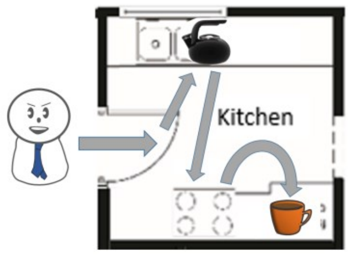
285	Meticulous Mona
289	Pool owner gets e-mail with pool operation summary on schedule
290	Water testing system tests for chlorine, pH, and alkalinity
302	Scenario: Cleaning Meticulous Mona's pool after the storm

269	A - Benson Pool Cleaning System Business Features
227	Pool Cleaning Automation
238	Local reports of pool operation
235	Pool cleaning
258	Filter cleaning robot empties and cleans filters
259	Pool brush robot scrubs pool surfaces
302	Scenario: Cleaning Meticulous Mona's pool after the storm

The illustration shows the same scenario linked to the feature level requirements (Pool Cleaning Automation) and to the persona work item (Meticulous Mona). Having a Backlog View for the persona gallery is great for the testing effort, as functionality and scenarios for a specific persona is nicely grouped and can be explored.

The Backlog Usage Models

Models help create clarity by showing how interdependent details are related.



Your backlog has a large number of user stories for a variety of user roles. Although you have a decent *Backlog Frame* and a technical *Backlog View,* you still have challenges in grouping work items to plan functional increments that provide business value.

How can you improve the understanding of how individual user stories contribute to a business transaction or user goal?

Without some way of defining a group of user stories that need to be fulfilled to achieve a business goal, you risk doing a lot of development work without providing a usable solution to the customer. The functional hierarchy in the *Backlog Frame* does not provide this type of insight. Rather than fulfilling all requirements within a functional area, more likely a deliverable with business value will consist of user stories following some path through a Workflow, with the "happy path" being most important, but not sufficient. So in addition to the "happy path" you will need the basic functionality for a set of functions that are used to fulfill a complete business transaction. For example, to buy an item online you need the key elements of viewing, selecting, paying for, and finally receiving the item. More sophisticated variations can be developed later, but if any of the core functions are missing then the other functions are useless, no matter how well they are implemented.

For release planning, it makes sense to associate business priority, quality criteria and business rules to overall representations of business transactions rather than at the level of individual user stories. Planning and prioritizing at this level results in a more meaningful dialogue with the product owner and requires less work to manage business priorities for individual work items. The same reasoning applies to your user acceptance testing. You want to evaluate and test complete transactions, not just individual user stories, and be able to assess efficiency of operation and usability at the granularity of a completed user goal.

Therefore, 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. A Use Case and a Business Process Model are two possible models you may have created to represent structured user activities to achieve a goal.

When including a new usage model in the backlog, you will want to link it to the level of requirement that best fits the overall contents of the model. This link could be to a theme or to a feature. Your model is likely captured in some sort of documents, which could be Use Cases written using a template, a Visio diagram, or an output file from a business process modeling tool. When creating the backlog item for the model, you can either add the descriptive document as an attachment, or store the description model in a content management system and create a link to this document from your new model backlog item. This allows models to be updated by authors with no access to the ALM tool. Your content management system is probably version controlled, and you may want to create the link so that it always refers to the latest version of the model description.

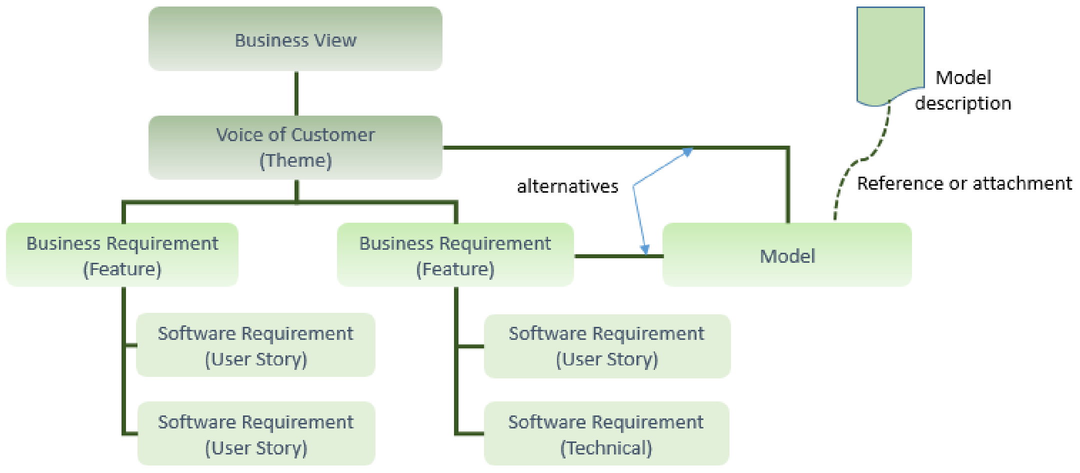

Business View
Voice of Customer (Theme)
Business Requirement (Feature)
Business Requirement (Feature)
Software Requirement (User Story)
Software Requirement (User Story)
Software Requirement (User Story)
Software Requirement (Technical)
alternatives
Model
Model description
Reference or attachment

Figure 7: Associating Models with Requirements

A usage model representing a logical grouping of requirements will need separate structural support in your backlog. It cannot be handled simply as another feature/second level requirement in your existing *Backlog Frame* because there may be a many to many relationship between any model element and specific user stories. As an example, consider the structure shown in Figure 8. Here a specific feature with six user stories has two workflows defined. Each workflow has four of the user stories associated with it, and user story C and D represent some user action/functionality that applies in both models.

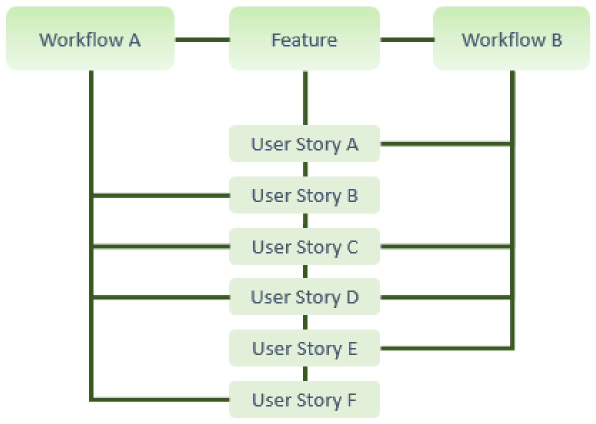


Figure 8: User Stories Linked to Multiple Models

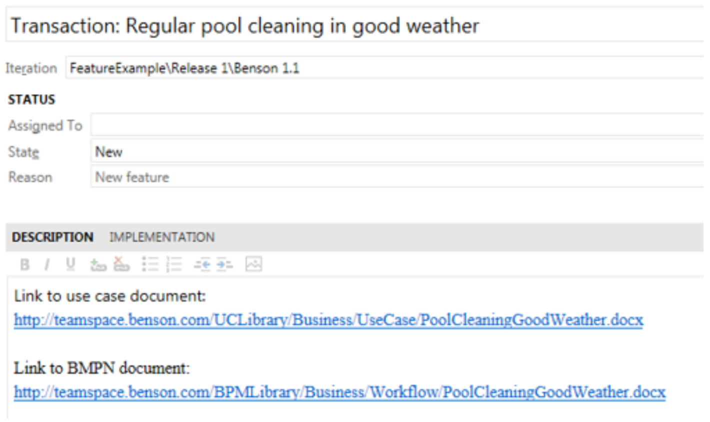
Since your model element is a backlog item, you can give it a business priority and apply this to all requirements associated with it. Any software requirement linked to multiple Use Cases or Workflows will inherit the highest priority of the models it is linked to. So in the situation shown in Figure 8, if workflow B has a higher business priority than workflow A, user story C and D will inherit the priority of workflow B.

Sometimes, it may be more appropriate to set separate business priorities for specific paths through a Workflow, or separately prioritize the main scenario and alternate scenarios in a Use Case. If that is the case you will need to apply the business priority to the individual software requirement, but you can still use the usage model to assist you in resolving the individual priority. This approach is a good aid to the "slicing" performed in agile projects, i.e. finding narrow (minimum) scope end-to-end functionality that brings business value.

By developing more complete usage models in your requirements elicitation, you reduce your risk of missing requirements as you analyze flows of user/system interactions. The usage models can help you keep a consistent and complete set of requirements because you can visualize the dependencies between the detailed requirements. When changing a requirement, you can see the impact on other requirements through your usage models.

An alternative way to incorporate usage models into the backlog is to create a top node for each type of model (for example one for use cases, another for workflows), and keep the main structure of the models as specific parts of the *Backlog Frame*. The next lower level would contain the model elements themselves. This way one can easily find a list of related models, such as a list of all Use Cases. Each model could still be linked to themes or business requirements. Just keep in mind that when integrating models into the backlog, there is additional work required to establish the model elements and maintain the links to requirements.

Example: The short stories and the user/functionality mapping provide good input for creating visual requirements models. The BA's favorite model for this type of system is the business process model (BPMN), since it helps a lot in identifying missing user stories. She creates several models for different user scenarios, and starts to populate the backlog with the added user stories from the models. For some of the more complex functionality the team feels that the user story format is too limited in content, and they work with the BA to develop Use Cases that add a better understanding of challenging parts of the system.


Transaction: Regular pool cleaning in good weather

Iteration: FeatureExample\Release 1\Benson 1.1

STATUS

Assigned To:

State: New

Reason: New feature

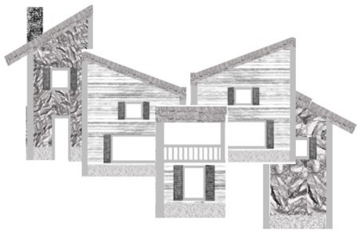
Link to use case document:
http://teamspace.benson.com/UCLibrary/Business/UseCase/PoolCleaningGoodWeather.docx

Link to BMPN document:
http://teamspace.benson.com/BPMLibrary/Business/Workflow/PoolCleaningGoodWeather.docx

One way to add models to the backlog is by creating documents that are controlled and maintained in a content management system (for example in SharePoint), and then link them to a backlog item representing the model. As seen in the illustration for the Backlog View, transactions are linked to the feature level requirements (in this case to the Pool Cleaning feature) along with scenarios.

The Backlog Placeholders

When the time is right, expand upon it.



You are in the middle of developing a large new software product. Some of the system needs are not yet elaborated so there are no user stories ready for the product backlog to represent these needs.

How can you represent partly unknown functionality in your backlog?

For the purpose of planning and reporting you want the backlog to represent the full scope of your product. But content is elaborated as development progresses, meaning that you do not have a complete set of user stories at the start. At the same time, it is not sufficient to just base the planning and reporting on the User Requirements level while you have some preliminary contents on a more detailed level that you want to include.

You have a *Backlog Frame* for the product functionality and to add content to it, this content must fit into the defined structure. Sticking to a defined structure is important for navigation and to enable you to easily find *Backlog Answers*. This means you cannot have varying granularity in details at the same level of the backlog.

Therefore, 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.

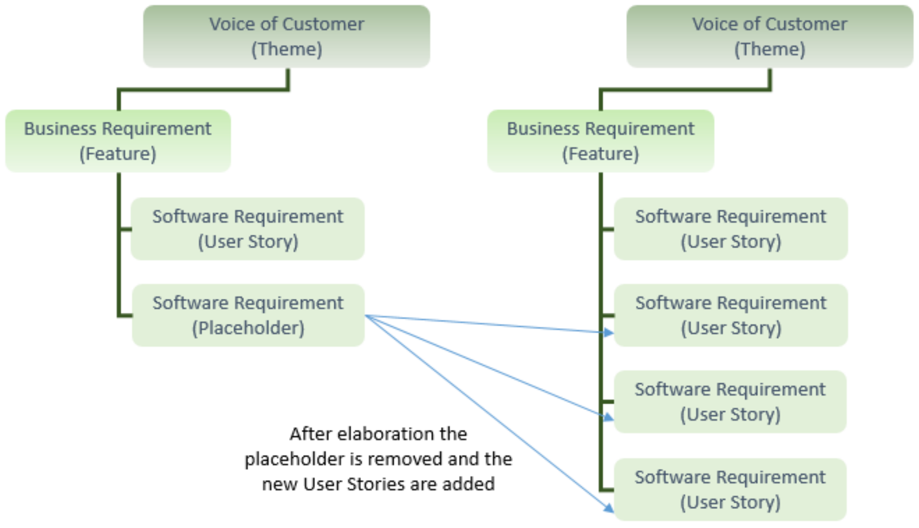
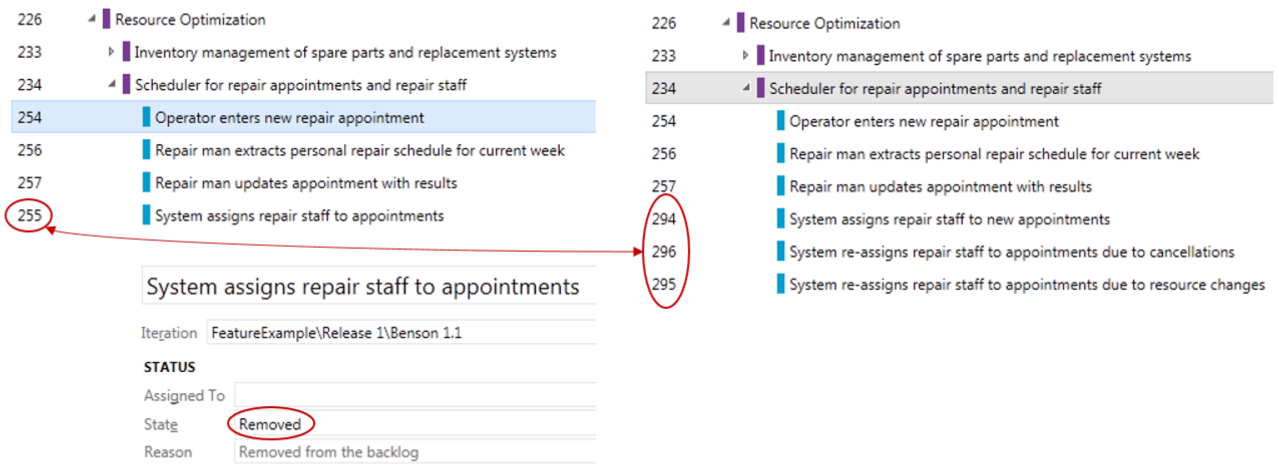

Voice of Customer (Theme)
Business Requirement (Feature)
Software Requirement (User Story)
Software Requirement (Placeholder)
After elaboration the placeholder is removed and the new User Stories are added
Voice of Customer (Theme)
Business Requirement (Feature)
Software Requirement (User Story)
Software Requirement (User Story)
Software Requirement (User Story)
Software Requirement (User Story)

Figure 9: Backlog Elaboration from Placeholder to New User Stories

Note that a User Requirements backlog item such as a feature or theme that has no children (user stories) is not a placeholder, because you know in this case you will add linked items below when you elaborate your user stories. A placeholder is specifically a leaf-level backlog item that you create as a temporary backlog item. Actually, many placeholders are created as original leaf-level requirements that need to be elaborated and split up into more discrete requirements. When you take a placeholder item into a 3 Amigos Meeting, it might end up being split into several new leaf level items as you elaborate more on the details.

Example: As the team learns more they replace big items in the backlog with more details.



226	Resource Optimization
233	Inventory management of spare parts and replacement systems
234	Scheduler for repair appointments and repair staff
254	Operator enters new repair appointment
256	Repair man extracts personal repair schedule for current week
257	Repair man updates appointment with results
255	System assigns repair staff to appointments

System assigns repair staff to appointments

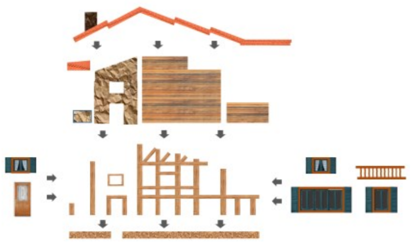
Iteration	FeatureExample\Release 1\Benson 1.1
STATUS	
Assigned To	
State	Removed
Reason	Removed from the backlog

226	Resource Optimization
233	Inventory management of spare parts and replacement systems
234	Scheduler for repair appointments and repair staff
254	Operator enters new repair appointment
256	Repair man extracts personal repair schedule for current week
257	Repair man updates appointment with results
294	System assigns repair staff to new appointments
296	System re-assigns repair staff to appointments due to cancellations
295	System re-assigns repair staff to appointments due to resource changes

For the backlog item "System assigns repair staff to appointments" the team realized that this item could be split into three more specific items that can be individually developed. The initial item is removed from the backlog and replaced by the three new items. This way the number of levels does not grow.

The Backlog Plans

The Product Backlog items are slotted for delivery in specific releases.



Your backlog has a large number of items organized in a *Backlog Frame*. The product roadmap delivers new releases of the product either through an iterative development process like Scrum or through a flow-based process using a Kanban technique.

How are the backlog items associated with your plans for delivery?

The development team wants to plan product deliveries from the backlog and make these plans visible. The *Backlog Frame* or the alternative *Backlog Views* do not provide structural support for the planning of new releases. Adding an additional view for planning will increase the complexity of the backlog and make it harder to maintain.

When working on a backlog with a large number of items, you want to work on subsets of the contents and see a partial view such as all the requirements for a particular release or iteration. You also want to have an easy way to modify a planned iteration or release knowing that its contents will change frequently.

You want to minimize waste and not plan the details of releases before you need to (i.e. when starting the development).

Therefore, associate the detailed requirements slotted for the next delivery to an entity representing this delivery.

A common way for backlog management tools such as JIRA or TFS, and others, is to 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.

You only plan for the immediate future by setting the planning attribute for backlog items that are actively in development or in the pipeline for development within a short timeframe (agreed upon for the current release). All other unfinished backlog items are part of the main backlog, and are not associated with any iteration or release. This way, there is minimal re-planning needed as scope changes during development.

Set the planning attribute for the system requirements level (leaf level) backlog items, not on requirements higher up in any backlog structure. This allows for requirements on the business level to be delivered gradually through multiple deliveries.

If your team is using Scrum, you may want to plan at the granularity of iterations within the next release. If you're using Kanban, you may set the planning attribute to reflect the release level and use another attribute for the state that the item is currently in (new, work-in-progress, under test, done etc.).

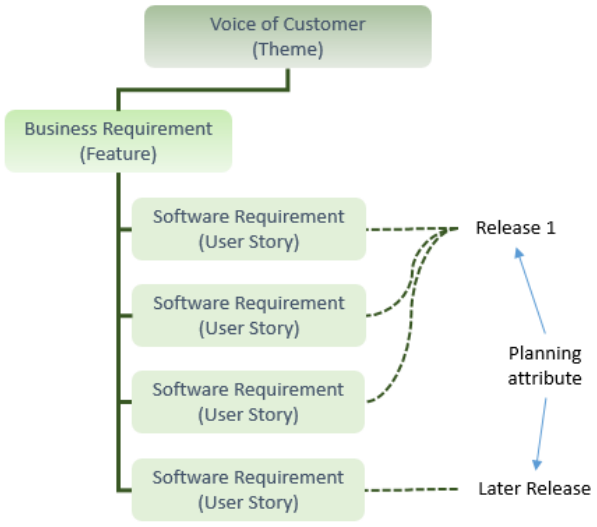
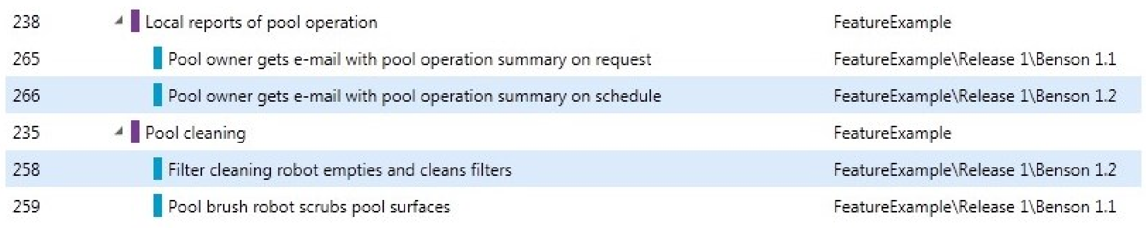

Voice of Customer (Theme)
Business Requirement (Feature)
Software Requirement (User Story)
Software Requirement (User Story)
Software Requirement (User Story)
Software Requirement (User Story)
Release 1
Planning attribute
Later Release

Figure 10: Planning Attribute for User Stories

Alternatively, planning could be accomplished by adding an additional *Backlog View* and associated backlog structure for planning purposes. But since releases and iterations are simply ways of grouping backlog items, they do not need to be backlog items in themselves (they do not need descriptions and other attributes). So instead, by assigning release and iteration attributes to backlog items, you can avoid having to maintain yet another set of links between items.

Example: After discussing with stakeholders about the business priorities it is decided to start development of the water management system first. This part can be delivered as a stand-alone system, it requires limited hardware development (far less challenging than the filter cleaning robots), and there is sufficient clarity of the functionality in the backlog. This said, the BA does agree to add requirements to cover the safety aspects, and she does an elicitation effort dedicated to water management quality requirements. As backlog items are selected for development there is a 3 Amigos Meeting to ready the items and define acceptance criteria and testing details. The plan for delivery is updated throughout development as the team progress on the realization, with the changes recorded in the history of the backlog items, and the current delivery plan shown by the release attribute of the backlog items.



238	Local reports of pool operation	FeatureExample
265	Pool owner gets e-mail with pool operation summary on request	FeatureExample\Release 1\Benson 1.1
266	Pool owner gets e-mail with pool operation summary on schedule	FeatureExample\Release 1\Benson 1.2
235	Pool cleaning	FeatureExample
258	Filter cleaning robot empties and cleans filters	FeatureExample\Release 1\Benson 1.2
259	Pool brush robot scrubs pool surfaces	FeatureExample\Release 1\Benson 1.1

The backlog is always in flux as items are added and modified, for instance through Backlog Placeholders being refined into details, changing business priorities, or for technical reasons that may push items to later releases. The above view shows backlog items slotted for delivery in 2 versions of the Benson system. Note that we only plan for the leaf level items, as we use the feature levels mainly to create structure. Backlog items do not need to be assigned to a specific release. It should be sufficient to identify items planned for the current, next, and all future releases.

The Backlog Connections

Traceability tells you how the contents of your backlog are related.

Your ALM system contains other project artifacts like code, defects and tests that have a relation to the requirements in the backlog.

How can you explore the diverse contents of your ALM system?

As the development and testing progresses on a large project, the amount of and diversity of content will grow significantly. A typical ALM system will contain source code with associated change sets, test cases and test results, and defects, in addition to the requirements.

The *Backlog Frame* and the *Backlog Views* provide the mechanism to relate requirements with each other, not only showing the breakdown from high level business requirements to detailed user and technical requirements, but also giving you a way to navigate and search/query subsets of a larger requirements collection. Now that you have all these additional "backlog item types" you want to have the same capabilities for exploring relations and dependencies that you have for the requirements. You want to see how a release test plan breaks into smaller test plans (test suites) and further into test cases, or what change sets are part of the latest product build.

Furthermore, you want to see relations between the various content types, for example what defects you have logged against a particular user story, or the test results of all tests cases that test the user stories belonging to a particular feature.

Therefore, 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 as shown in Figure 11.

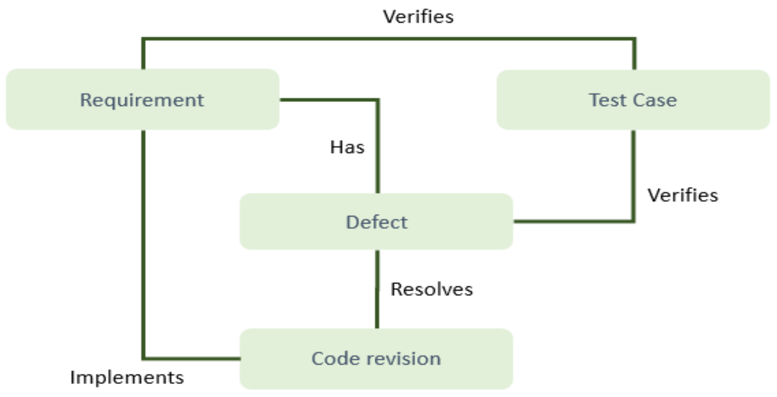


Figure 11: Traceability Model

Whenever new content is created, such as a test case or a defect, the person creating it must make sure it is properly linked following the agreed model. If not, your backlog will have a number of links but not provide consistent navigation or query abilities.

This established structuring is referred to as traceability, and it is a recognized quality of a backlog:

"Not only should the requirements themselves be traced, but also the requirements relationships to all artifacts associated with requirements, such as models, analysis results, test cases, test procedures, test results and documentation of all kinds. Even people and user groups associated with requirements should be traceable." — Wikipedia on requirements traceability [Wik2014d]

"Properly implemented, traceability can be used to prove that a system complies with its requirements and that they have been implemented correctly. If a requirement can be traced forward to a design artifact, it validates that the requirement has been designed into the system. Likewise, if a requirement can be traced forward to the code, it validates that the requirement was implemented. Similarly, if a requirement can be traced to a test case, it demonstrates that the requirement has been verified through testing. Without traceability, it is impossible to demonstrate that a system has been fully verified and validated." [KS2009]

Figure 12 shows one implementation of the overall traceability between the different item types in an ALM system:

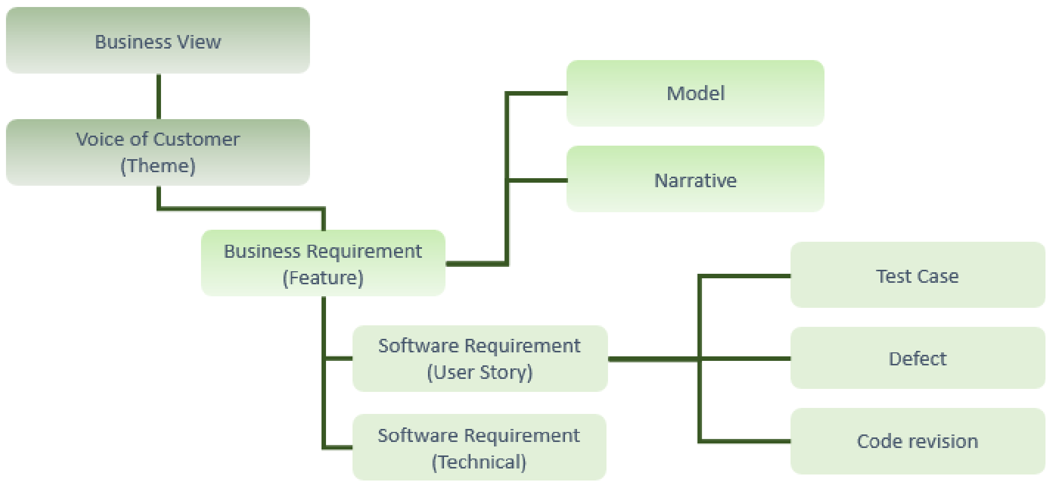
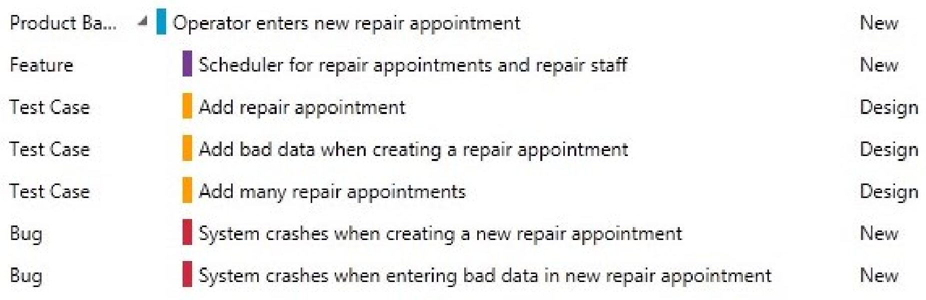

Business View
Voice of Customer (Theme)
Business Requirement (Feature)
Model
Narrative
Software Requirement (User Story)
Software Requirement (Technical)
Test Case
Defect
Code revision

Figure 12: Overall Backlog Traceability

The connections should be made only on the lowest level of detailed item in the backlog that makes sense. A defect should be reported on (linked to) a specific system requirement. A test should be associated with a specific system requirement unless it is explicitly written for a higher-level backlog item such as a model item, in which case it should be linked to that model item. By linking test case, defect and code revision elements to the lowest level of granularity of the backlog items that they encompass, it makes it easier to analyze the overall implementation status of your system. For example, you will know how many defects were reported against a specific system requirement. Because you already established traceability between the requirements in the *Backlog Frame*, you can easily find the defects that have been raised against a specific business requirement by selecting all defects associated with its child requirements.

The more links and connections you make, the easier it is to ask specific questions. But at the same time, it becomes more tedious to maintain all these connections. This applies to any backlog pattern that adds structure and links between items. Additional structuring of your backlog provides information, but at a cost of maintaining the network of links. If you need to evolve an existing structure, it requires "refactoring" and "relinking." The difficulty of doing this depends on how easy it is to see and manage the links in your ALM tool.

Example: At some point the development team starts sketching an initial system architecture and user experience design, and starts to work with QA on how the system can be tested. This helps to improve the initial user stories, and provides additional items and details for the backlog. Throughout the development the team maintains their backlog and keeps their traceability between requirements, testing, code, and defects.

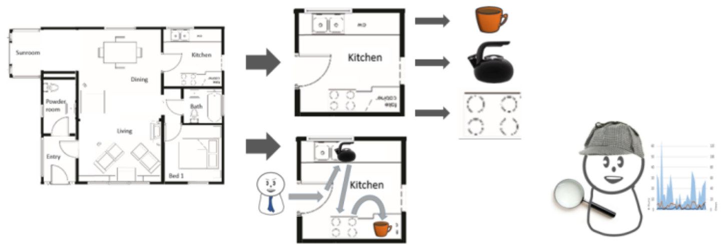


Product Ba...	Operator enters new repair appointment	New
Feature	Scheduler for repair appointments and repair staff	New
Test Case	Add repair appointment	Design
Test Case	Add bad data when creating a repair appointment	Design
Test Case	Add many repair appointments	Design
Bug	System crashes when creating a new repair appointment	New
Bug	System crashes when entering bad data in new repair appointment	New

For the Benson backlog items, each item has a parent feature, a set of (possibly automated) test cases that fully test the item, and a list of bugs. Higher-level test cases may be linked on the feature level together with the Backlog Tales and the Backlog Models.

The Backlog Answers

A well-kept backlog provides a lot of useful knowledge for a team.



Your backlog is well structured with an elaborated *Backlog Frame* and *Backlog Connections* that create traceability to the requirements from other project artifacts.

How can your team gain insights about the product from the backlog?

Your backlog contains the detailed data for the health of your product: it knows the state of test cases run on the last build, it knows how many major defects are logged against requirements planned for the next release, and it knows how many items are completed and how many are remaining for your next delivery. This information can be extracted by queries in the ALM tool. Creating good queries demands a thorough understanding of how the team has structured the backlog contents, and depending on the tool used there is a certain learning curve required to formulate good queries.

To see trends over time you need to re-apply the same queries. Team members creating a variety of queries can easily come up with confusing outcomes because although similar, their queries may not be identical. For reporting and trend analysis, you need to access the contents of your backlog consistently over time.

Therefore, 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 merely to report status to high-level stakeholders. Of course it is nice to have an automated stakeholder report directly generated from the ALM tool, especially for the project manager who will save time and effort in reporting. But the highest value is for the core team to always know where they are and be able to prioritize their efforts on the most pressing work.

Figure 13 shows a graph based on TFS contents that show time-based state of the backlog items. This backlog was designed to answer the following questions:

- How large is the scope in number of backlog items?
  The graph shows that it is increasing for a while and then stabilizing. This could be because of added functionality, or because *Backlog Placeholders* were turned into detailed backlog items.
- How far are we?
  This is shown by the number of new PBIs remaining (burn-down plot) and by the number of PBIs that are verified.
- Do we keep work in progress (WIP) to a minimum?
  The development WIP is kept stable at around 5-6 items, and the verification queue averages 4.
- How many of our requirements have sufficient test case coverage?
  This is increasing throughout the project. Test cases could be unit tests and/or user acceptance tests.

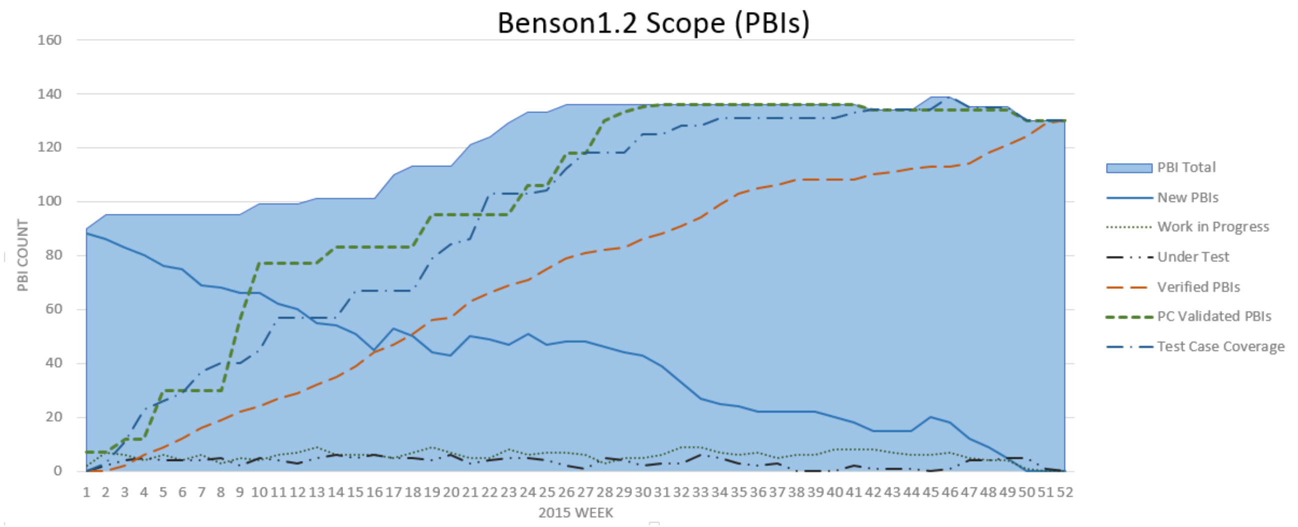


Figure 13: Time-based View of the Backlog Items

Another interesting graphic could be a plot of test status vs. system features as shown in Figure 14. This would tell the team not only how they are doing in testing, but also how many tests are developed for the functional areas of the system, and if any of the tests are blocked from running (could be missing testing resources, bugs, or missing functionality). Another plot shown in Figure 14 of the defect distribution per functional area adds to the understanding of how complete each functionality area is at the current time.

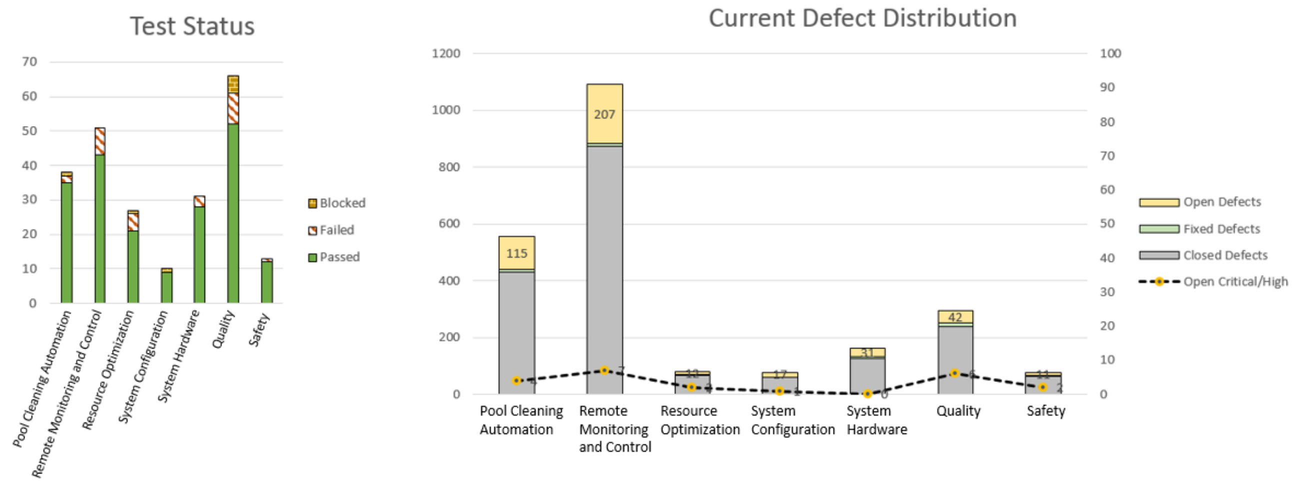
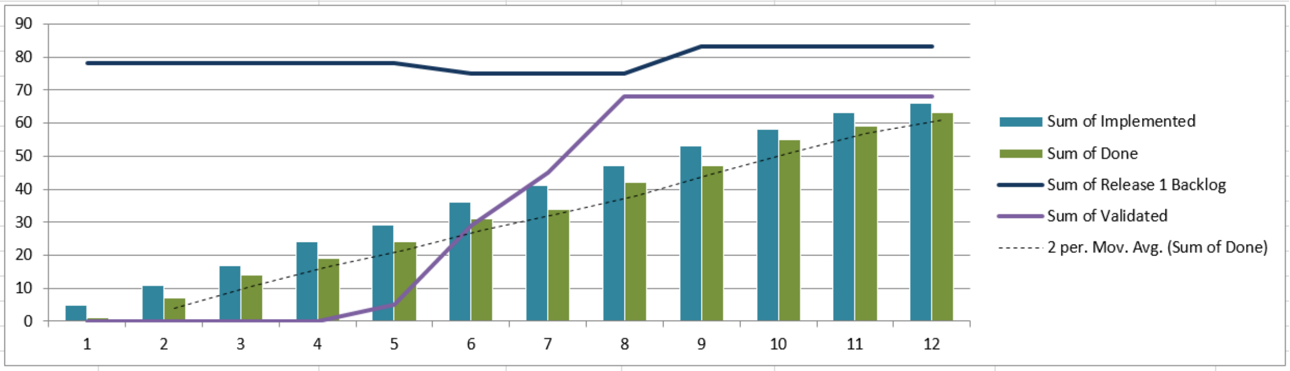


Figure 14: Test Status and Defect Distribution Views

Producing these and other graphs on a project dashboard can be done with automated refresh from ALM tools when the backlog is designed to support the queries that can pull the data for dashboard display. This again requires a sufficient *Backlog Frame* with *Backlog Connections* that are adequately structured and maintained.

In the above examples, the *Backlog Frame* has a 3-level requirements structure. The leaf requirements have a state attribute to determine if they are new, under development, in verification, or completed. Test cases and defects are linked to each leaf requirement, and have state attributes as well to determine pass/block/fail and open/fixed/closed states. The plots are done for the mid-level requirements by traversing the linked requirements and summarizing the results.

Example: Defining and maintaining the right linking of the backlog items is essential to pull knowledge out of the backlog. The Benson team has defined some basic metrics that they believe will help them planning their development effort, like the plot of the current state of the backlog:



Comments/Discussion

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 this paper made us realize that the term has another meaning: done well with the right contents and structure it 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.

Incorporating quality aspects into the backlog is a great way to make sure these are proactively considered rather than being an afterthought. In a recent set of papers about transitioning from Quality Assurance to Agile Quality [YWA2014, YW2014, YWW2014, YWW2015] the authors include a number of patterns that are relevant to the backlog design process. *Integrating Quality into your Agile Process* and *Qualify your Backlog* incorporate *Quality Stories* and *Agile Quality Scenarios* into the backlog. *Fold-out Qualities* adds specific quality criteria as acceptance criteria in the backlog user stories. The *System Quality Dashboard* can be fed from queries that provide *Backlog Answers*.

The number of books on requirements is a good indication of how important requirements are for product development – and for project management in general. This paper is not an attempt to reproduce content from books on software requirements, but rather to focus on the process of building and continuously feeding a healthy backlog of requirements built on a requirements engineering foundation provided by the literature on software requirements.

5. ACKNOWLEDGEMENTS

We are grateful to our shepherd Jason Yip for his questions. They caused some substantial rewrites (and pain) but really contributed to an improved paper! We are also grateful for the feedback from our writer's workshop participants Till Schümmer, Joerg M. Haake, Satomi Oi, Takashi Iba, Arisa Kamada, Yuma Akado, Taichi Isaku, Stefan Holtel, Elissaveta Gourova, and Joe Yoder at EuroPLoP 2015 who provided further insights and ideas for improvement and caused a new round of rewrites but all for the better – thank you! 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.

6. REFERENCES

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

[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)

[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)

[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

[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 978-0-9857879-0-5

[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.

[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

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

[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

[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, 13 pages.

APPENDIX

Wikipedia [Wik2014c] defines a requirement as, “a singular documented physical and functional need that a particular design, product or process must be able to perform. […A requirement] identifies a necessary attribute, capability, characteristic, or quality of a system for it to have value and utility to a [stakeholder].”

a. Requirements

Requirements can be defined on various levels of granularity and can be of different types. In this paper, we will use the requirements model and definitions from Wiegers [Wie2009] as shown in Figure 15.

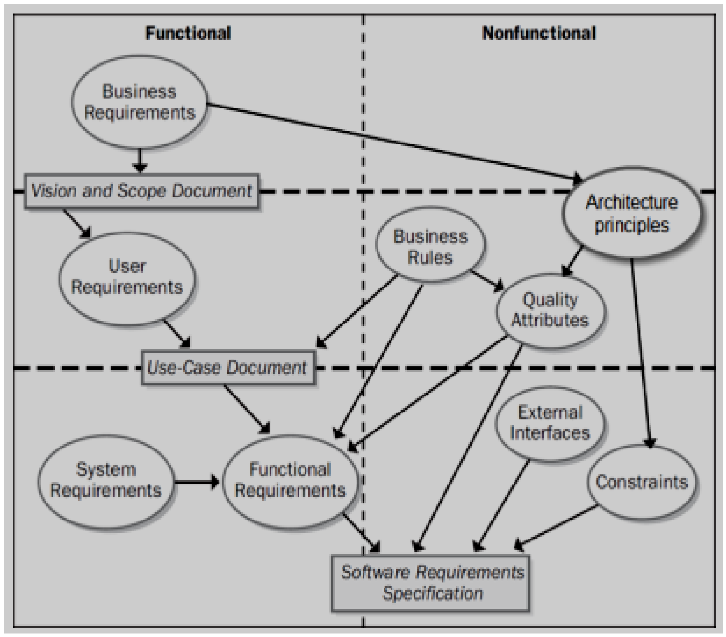


Figure 15: Requirements Model with Levels

Business Requirements
Reflect the top level goals of the product

User Requirements
Transactions and scenarios for user roles

Business Rules
Compliance with standards and regulations

Quality Attributes
Performance, reliability, usability etc.
Span across functional requirements

Functional Requirements
Reflect the capabilities of the product

System Requirements
Technical needs imposed by system design

External Interfaces
Needs to communicate with other systems

Constraints
Imposed restrictions like cost limitations

The product’s Software Requirements Specification (SRS) represents the product backlog and contains the level of requirements that are directly consumed by the development team. As seen from the model in Figure 15, the backlog is more than a breakdown of the user or business goals; it will typically include needs influenced by the solution design and deployment environment.

b. Requirements Engineering

The process of defining and managing the requirements is called Requirements Engineering. The following sections give a brief introduction to the basic parts that traditionally makes up the field of requirements engineering – namely requirements elicitation, analysis, and management.

b.1. Requirements Elicitation

Requirements elicitation is the part of requirements engineering that deals with identifying user roles and system functionality. Literature on requirements elicitation [HH2008, Mas2010] presents variations and/or subsets of the following techniques to extract requirements:

Interviews – normally done with stakeholders and users starting with a defined set of questions but with the possibility to expand and elaborate the discussion to further explore the needs.

Questionnaires – have a defined set of questions and typically reach out to a larger group of individuals than those targeted by the interviews.

Workshops – bring together a small number of users and stakeholders to brainstorm on functionality and ideas for the product.

Analysis of existing systems – if the new product is replacing or will compete with existing systems a lot can be learned from studying the existing system and its documentation.

Observation – of users performing tasks either manually or with an existing system.

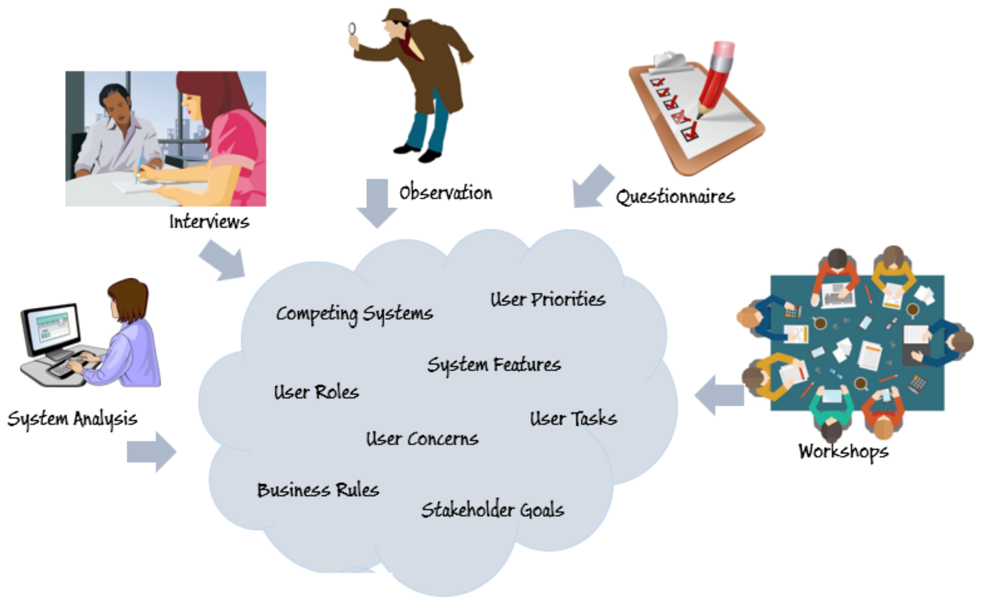


Figure 16: Requirements Elicitation Activities

One would normally apply a combination of requirements elicitation techniques as shown in Figure 16. This is an iterative process where techniques are repeated throughout development to elaborate and refine the results. The requirements gathering team would typically repeat techniques at any time they need more information, for example doing additional interviews to gain deeper insights or to reach out to a new or larger set of users.

b.2. Requirements Analysis

The requirements elicitation activities are outward facing – extracting knowledge from customers, users and other sources. Processing the "raw data" from elicitation activities to produce clear and concise information on a level ready for implementation is rarely straightforward, but requires analysis, structuring, and refining of the elicitation results. The first iteration of elicitation activities will provide initial input on users and user roles, user tasks and expectations of the product, concerns and needs that may turn into quality requirements and business rules, and input from the business side including a product vision and high-level goals. As the system is taking shape, additional elicitation will provide feedback to help refine, correct, and improve the detailed requirements. The following activities may be part of the analysis effort:

Scoping – on the top level (scope 0) the system boundary is defined clarifying the core responsibilities of the system (what the system is and what it is not). The scope is broken down into major functional blocks (often called themes) that represent the main business requirements, and further into user requirements, and then functional requirements as shown in Figure 17.

Personas – creating "virtual people" to fill the user roles helps to personalize the users and better understand their various dimensions with respect to work tasks, experience level, preferences, etc.

Visual Models – there are a number of models used to explore and understand requirements. One very common model is the Use Case Model with a stick-man diagram, which shows as a high-level view of how users interact with the system [Coc2001]. In addition to a Use Case Diagram, individual use cases describe transactions of a system initiated by a user or another system role. A use case template, which is used to write use case descriptions, will normally contain a success scenario, alternative and problem scenarios, pre-conditions and post-conditions of the transaction, a primary actor who initiates the transaction, a trigger, applicable business rules, etc. Another visual model is the Business Process diagram [Hva2014, BC2012]. Models help us to see dependencies and holistic sub-collections of requirements that, when combined, will add a higher level of business value to the product.

Storytelling – provides narratives that are realistic scenarios describing users (personas) actively using the system to achieve their goals that help us better understand transactions and operational workflows. As a technique, storytelling goes back to the early days of mankind and is possibly the strongest tool we have for creating understanding and for remembering the essential parts of a larger body of information. A story or tale creates structure and anchors goals and reasons for certain functionality to be part of the product. There are important, but subtle differences between a tale and a workflow model. A tale is intended to tell a story and helps develop empathy and understanding for the frustrations and daily life of a user interacting with the system. Depending on the scope of a tale, it could include multiple workflows. A workflow, on the other hand represents all possible paths through a set of tasks to accomplish a business objective. As such, it includes all options/alternatives, while a narrative tale follows a specific path through one or more workflows. Tales give a perspective on how the system is understood and used by a specific user; while a workflow provides a more comprehensive view of potential paths through system functionality.

Story Maps – group user stories into sequences that fulfill a business goal or transaction. This is a powerful technique and documented by Jeff Patton [Pat2014]. User stories are short statements often in the form: "As a <role> I can <action> so that <goal>," along with associated acceptance criteria and defined testing. Many agile teams work from backlogs of user stories, which are fundamental to development work practices like Kanban [Coh2004, Amb2014].

Prototypes and pilots – limited implementations can give users early experience to validate requirements as well as providing additions and changes to already defined requirements.

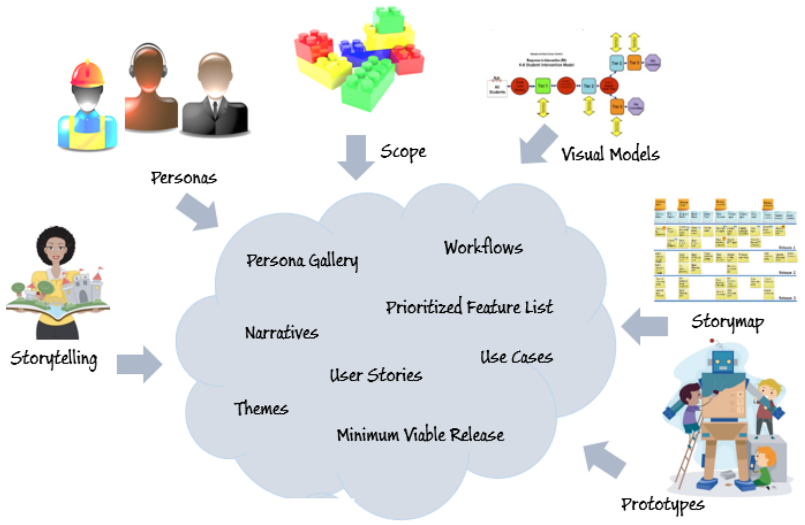


Figure 17: Requirements Analysis Activities

When planning a software product there are three fundamental questions that must be answered: Who will be using the system, what actions will these users want or need to do (including what they want to see), and in any system with some autonomous actions, what does the system need to do? In technical terms, you answer these questions as you define the user roles, the user tasks, and the system functions. This can be an overwhelming undertaking for a large system with several different user roles and hundreds of detailed functions. A single representation of the system will rarely be sufficient to represent a complex system. The materials produced during requirements analysis (business requirements, scenarios, visual models, user stories, etc.) should combine to provide a multi-faceted view of a new product and (iteratively) feed the system development activities of architecting, coding, and testing the software.

b.3. Requirements Management

Managing the requirements includes determining how they are documented, validated and approved. It also includes a change process, and the possibility to trace back to deliverables and testing.

Documentation – business requirements are frequently presented in a structured (logical) form in a requirements document, or possibly kept in a requirements management tool. A common structure is at least three layers deep with the top level being the Voice of Customer, the middle level the Business Requirements, and the third level the Software Requirements. Requirements should have an associated priority, and the software requirements level should comply with a quality model for requirements (like SMART: specific, measurable, actionable, realistic, and time-related).

Validation and approval – this is normally done by the stakeholders (customers and users) and is more common on high-formality projects. With an agile approach, the validation/approval (and change process) is better done by close communication between the product owner and the team, continuous integration and frequent feedback on working software.

3 Amigos Meeting – before a backlog item like a user story can be moved into development, representatives for the three roles of business, QA and Development meet to agree on the definition of Done (i.e. acceptance criteria) and Testing (how to test, what tests to develop). This ensures that each backlog item is well defined before implementation.

Change process – this can either be of high formality, with approval workflows for any requirements changes, or as described above, less formal with frequent team/customer collaboration.

Traceability – when using tools to manage the requirements, code, defects, and testing (ALM), these entities are linked to enable reports like burn-down charts, defect trend diagrams and other quality metrics.

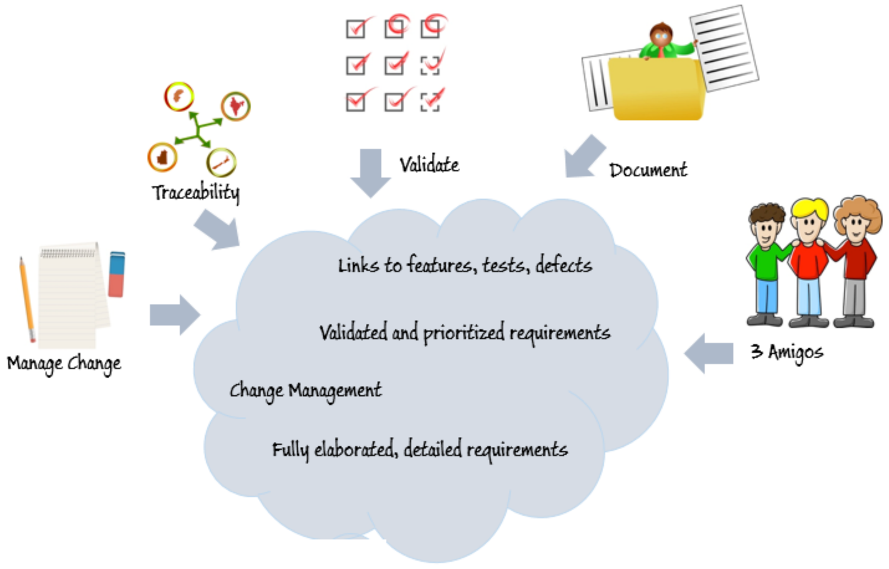

Validate
Document
Traceability
Links to features, tests, defects
Validated and prioritized requirements
Manage Change
3 Amigos
Change Management
Fully elaborated, detailed requirements

Figure 18: Requirements Management Activities

c. Other Requirements Topics

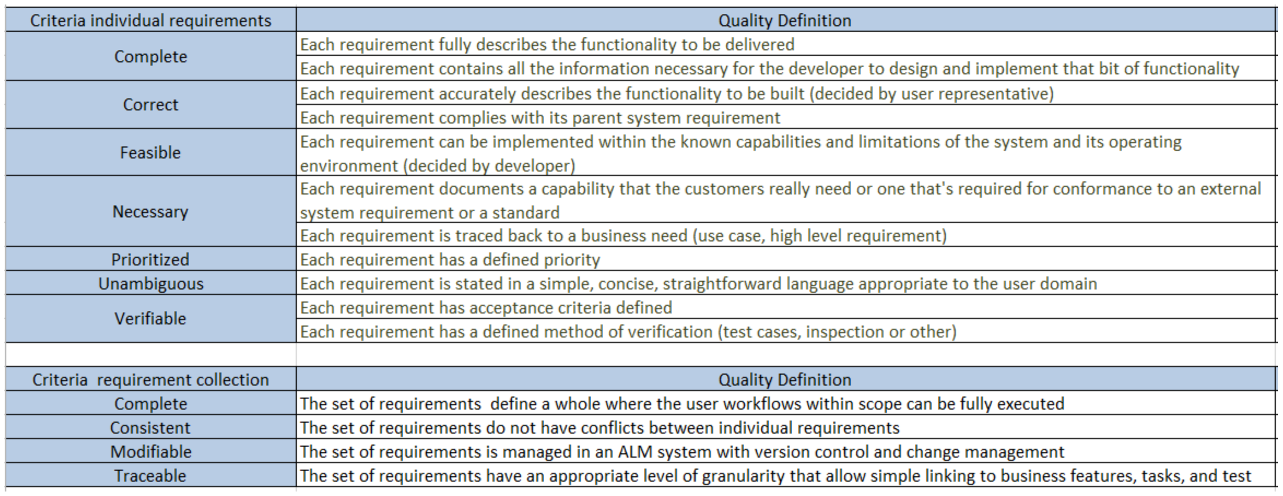
To complete the background on requirements terminology and concepts there are three final areas: system quality, metrics, and traceability.

c.1. Quality Requirements

Quality requirements refer to the requirements that describe what the system needs to be (as opposed to what it can do) and are sometimes referred to as non-functional requirements or the "-ilities" of a system: Reliability, Usability, Availability, etc. We prefer to use the term Quality Requirements rather than non-functional requirements because some of these requirements can be translated into actions and so become part of the system functionality. For example, consider a security requirement to restrict access to authorized users only. This can turn into a requirement to have user access control (login) capabilities.

c.2. Quality of Requirements

The quality aspects of requirements are well explained by Wiegers [Wie2009] and we will adopt his quality criteria definitions as shown in Table 1. In particular, these definition looks at quality both for individual requirements and for requirements collections.



Criteria individual requirements	Quality Definition
Complete	Each requirement fully describes the functionality to be delivered
	Each requirement contains all the information necessary for the developer to design and implement that bit of functionality
Correct	Each requirement accurately describes the functionality to be built (decided by user representative)
	Each requirement complies with its parent system requirement
Feasible	Each requirement can be implemented within the known capabilities and limitations of the system and its operating environment (decided by developer)
Necessary	Each requirement documents a capability that the customers really need or one that's required for conformance to an external system requirement or a standard
	Each requirement is traced back to a business need (use case, high level requirement)
Prioritized	Each requirement has a defined priority
Unambiguous	Each requirement is stated in a simple, concise, straightforward language appropriate to the user domain
Verifiable	Each requirement has acceptance criteria defined
	Each requirement has a defined method of verification (test cases, inspection or other)

Criteria requirement collection	Quality Definition
Complete	The set of requirements define a whole where the user workflows within scope can be fully executed
Consistent	The set of requirements do not have conflicts between individual requirements
Modifiable	The set of requirements is managed in an ALM system with version control and change management
Traceable	The set of requirements have an appropriate level of granularity that allow simple linking to business features, tasks, and test

Table 1: Quality of Requirements

c.3. Metrics

Basic metrics related to requirements lets the development team see how far they are in the development and how well the product is doing in testing. Typical views for an agile team are the burn-down chart, defect trend plots, and the test status view.

c.4. Traceability

You want to link requirements of different levels so you know what user stories fulfill a Business Goal and vice versa. Other items you want to associate with the user story are the tests and defects. This will provide the foundation needed to pull interesting metrics and reports from the backlog system.