Chapter 6. **Artifacts and Repositories**

What are the items that are produced and who uses them?

In this chapter, I detail the things that are created by the project team and who uses them.

Ultimately, the result of the Quality with Agile through Pictures process is a product release, which includes a software build and supporting documentation. Everything else that is produced is used to assist people with achieving a quality product release.

In this chapter, I describe the artifacts produced during the process in terms of their purpose. I identify who is responsible for their content and what is their contribution towards a quality product. Artifacts and repositories are listed in the order used in the process.

6.1 **Business Need**

Business needs are provided by the customer and captured as epics in the product backlog. The product owner is responsible for identifying business needs. The business analyst analyzes the business need to derive accurate user stories. The business need is the source for use cases that model the functionality of the product.

A business need takes the form of textual notes or spoken word (or any form that the customer provides).

Understanding and accurately capturing the business need is essential for a quality product. If the final release does not meet the business need then quality is lost in all artifacts created between eliciting requirements and deploying the release.

6.2 **Epic**

Although a common term on Scrum projects, the epic is not defined in the Scrum glossary. The epic type user story evolved from user stories that were too big to complete in a single sprint. These epics are broken down into sprint-sized user stories. However, the development team wants to keep a record of the original business need, so the epic is retained and user stories become children of the epic.

Figure 11 is the basis for an epic template that you could adopt and modify to fit your situation.

Feature Name	
<Description>	
For	<business role>
who wants	<something of benefit to my job>
so that	<the reason why this will improve my job>
unlike	<existing situation this will improved>
Additional information	
<Stakeholder>	<Additional information related to the user story that may assist with implementation>

Figure 11 – Example Epic Template

The epic user story is the responsibility of the product owner. When using software tools to capture and manage your backlogs, it may be the business analyst that actually creates the epic. However, the product owner approves and prioritizes epics.

In the Quality with Agile through Pictures process, an epic captures information about a business need, even if this business need can be developed in a single sprint. Epics are not developed or pulled into sprints. The epic is always broken down into 1 or more user stories and these user stories are developed in sprints.

The reason for this structure is to retain a clear separation between the business process and the system requirements. (Most of the time, the system requirements will only partially satisfy the business need.)

Figure 12 shows the epic life cycle.

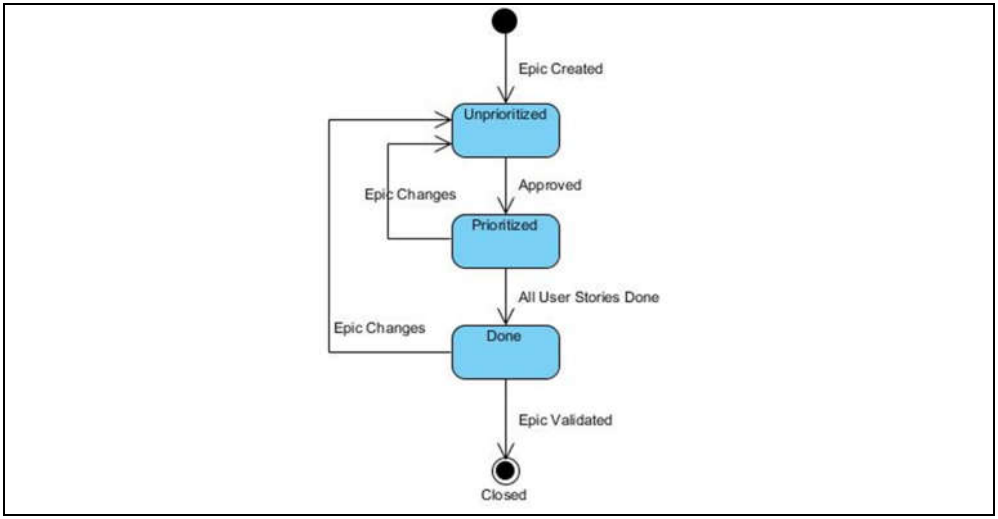

Figure 12- Epic Life Cycle

When an epic is created, it is unprioritized and has no associated user stories. In this state information about the epic content is being elicited from the stakeholders. Once approved (by the product owner), the epic and its user stories are prioritized in the product backlog and the business analyst can start working on the epic's child user stories.

User stories that are ready-for-dev are pulled into sprints. Development of the epic may span many sprints. Only when all of the child user stories are done can the epic be considered done.

If the scope epic of the epic changes for any reason (i.e. more work is needed), then the epic will no longer be prioritized and the epic (and all of its child user stories that have not been pulled into a sprint) will need to be reviewed and re-prioritized.

Although an epic is considered to be done, it can return to an unprioritized state. This may be the result of a defect user story being created by quality assurance during testing. The defect user story is associated to the epic and now the epic has a child user story that is not done.

Only when all associated user stories have been validated and the business need that is described by the epic is delivered in a release, can the epic be closed.

- **Done may be a confusing name for a state that can be reverted to a not done state. I chose this name because it matches the done state of the child user stories. Feel free to use a different name for this state if done causes confusion.**

The epic user story contributes to product quality by maintaining a record of the reason for the product requirements.

6.3 **Product Backlog**

The product backlog is a repository for work items. The product owner is responsible for the product backlog. The business analyst assists with maintenance of the product backlog. The product backlog contains items that were derived from features, defects, architectural changes, research initiatives or any work identified by a stakeholder. These items are in the form of epics or user stories.

Figure 13 shows the structure of a product backlog. Backlog items can be in various states. (See user story and epic definitions for item states.)

Program Backlog	
Prioritized Epics	**Prioritized User Stories**
feature A	user story
…	
	Unprioritized User Stories
feature B	user story
Unprioritized Features	
feature C	user story
…	

Figure 13 – Product Backlog Structure

Epics are shown in a separate list from user stories. This is because it is the user stories that are developed, not epics. Epics are decomposed into child user stories. The user stories in the right column may or may not be assigned to a parent epic.

A new epic is initially unprioritized. As epics are reviewed, they are assigned a priority.

- **Priority is indicated by the position in the list of items, with higher priority items shown above lower priority items.**

If an epic cannot be approved for development, it will remain unprioritized.

Prioritized epics are decomposed into child user stories. Child user stories are initially unprioritized and must be reviewed and prioritized before being considered for assignment to a sprint.

Unprioritized epics may also have child user stories, but those user stories will also be unprioritized.

User stories may also be created independently of an epic. These will initially be unprioritized and likewise, must be reviewed and prioritized before they can be assigned to a sprint.

- **Unprioritized items may be sorted by the order in which they are expected to be reviewed. In my experience, there is no meaning to the order of many items in the product backlog – unprioritized items are often placed randomly towards the bottom of the backlog.**

Both epics and user stories may be removed from the product backlog during backlog grooming, whether prioritized or unprioritized. If an epic is removed then it is assumed that all associated child user stories are also removed (unless they have already been taken into a sprint).

- **See Scrum guidelines for removing user stories from a sprint.**

I find it really helps to manage your product backlog if there is clear separation between prioritized and unprioritized backlog items.

- **If the tool does not support separators between backlog items, I create a 'dummy' user story and use that as a separator between prioritized and unprioritized items.**

A well-organized product backlog contributes to product quality by ensuring that requirements are not missed and features are prioritized correctly.

6.4 **Use Case**

A use case captures a process (either manual or automated). It is used to scope work in terms of the actors that use or gain benefit from the process described in the use case.

A use case may be detailed in terms of a sequence of steps that are performed by a person or system and are within the scope of the use case. Events cause the use case to transition through its steps. An event ends the current step and starts a subsequent step in the process. Every use case has 1 start state and 1 or more stop states.

Use cases come in 2 flavors: business or system. Both types of use case have a single primary actor and any number of secondary actors.

Use case steps may be documented textually or using activity diagrams. (I prefer to use both as a crosscheck to make sure that steps are not omitted. I diagram the activity first and then write out the steps as in a textual format. If both formats are consistent then I am reasonably certain that I have captured all the known steps in the business process.)

Examples of textual and graphical formats are shown in section 7.4.1.2.

6.4.1 Business Use Case

A business use case describes a process in terms of the benefit that it gives to its primary actor. The business use case scopes the work described by epic user stories. (Ideally, a business use case would correspond to a single epic. This is a guideline for scoping work. It is not a universal rule and some epics may not even require a business use case.)

The primary actor is responsible for the success of the use case. Secondary actors assist with completing the business process.

See Appendix A - for the business use case notation.

6.4.2 System Use Case

A system use case describes a process that is performed by a system as it interacts with its primary and secondary actors. The system use case details how that work is automated by a system.

The primary actor initiates the system use case. Secondary actors are external roles or systems that interact with the system use case in order to allow it to complete its functionality.

See Appendix A - for the system use case notation.

Use Case Summary

Both types of use case can be modeled using activity or sequence diagrams. The notation for activity and sequence diagrams is described in Appendix A - .

Use cases add to the quality of the product by formalizing epics and user stories such that they can be modeled. A modeled use case can be analyzed and inconsistencies and ambiguities identified.

6.5 **User Story**

A user story captures a system feature that can be implemented in 1 sprint.

- **Defects, spikes, risks and action items are all considered types of user story. It is assumed that all (tracked) work that is performed by the development team is the result of a user story. I.e. there is no separate action item list for the development team.**

The user story often takes format that identifies who gains benefit (As a ..), what it is they want (I want ..), the benefit that is received (so that ..).

Figure 14 is an example of a template that I have used for user stories.

Parent	Parent Epic
Overview	<title of the user story>
<description of the user story>	
User Story	
As a I want so that	<beneficiary> <what the beneficiary wants> <why they want it>
Acceptance Criteria	
Given when then	
<links to supporting information>	

Figure 14 – User Story Template

The user story template contains a link to an epic user story, if applicable. The title helps to identify the user story. (If useful, you might want to make the title unique.) A textual description is included, followed by the user story formal description. Additionally, the user story may include several acceptance criteria and links to supporting information, such as screenshots and documentation.

The user story is the responsibility of the product owner, in that the product owner approves and prioritizes user stories. However, since user stories are created for the most part by the business analyst, development team or quality assurance, the product owner may not be aware of the content of a user story while it is unprioritized. In many cases, approval is based on recommendations from technical roles.

- **The product owner does not need to be aware of the details of a user story, unlike an epic - which the product owner should always have full knowledge.**

Figure 15 shows the life cycle of a user story.

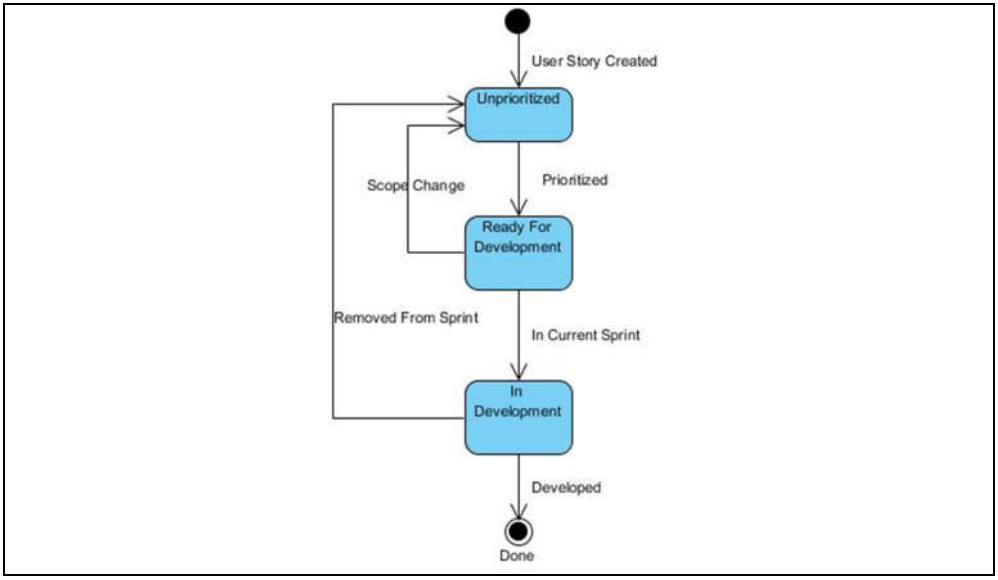

Figure 15 - User Story Life Cycle

- **This user story lifecycle is as simple as it gets. In reality, user stories will often take several more states than shown.**

User stories are created by any stakeholder with an interest in the success of the project. New user stories are unprioritized.

The business analyst will analyze the user story to ensure that it is accurate and relevant. Quality assurance reviews the user story to make sure that it is clear

and unambiguous. Approval for development is given by the product owner, and the user story is prioritized in the product backlog as ready-for-dev.

- **I assume that it does not make sense to prioritize a user story whose parent epic is not prioritized. If you really want to prioritize a child user story of an epic that has not been approved for development, I recommend that it is disconnected as a child of the epic. When the epic is prioritized to the child user story may be re-connected.**

If the scope of the user story changes, it should become unprioritized until it has been approved again for development.

While in the ready-for-dev state and the user story has been prioritized, it may be pulled into a sprint backlog. On the start day of that sprint, the user story is now in development. At anytime during the sprint, the user story may be declared done by the development team. If the sprint finishes before the user story is complete (or the user story is removed from the sprint before it is done), the user story is returned to the product backlog in the unapproved state.

- **See the Sprint Backlog section 6.9 for user story states while in development.**

Even though the user story is in a done state, it will still be needed by quality assurance during testing. The state of the user story will not change as a result of testing.

- **Unlike an epic, once a user story is declared as done by the development team, it cannot be undone.**

If a defect is found that is related to a done user story, a new user story is created. If the done user story is a child of an epic, this new user story will become an unprioritized child of that same epic.

If a user story is too large to be developed in a single sprint, this will be determined at the time of review. Prior to being reviewed the user story is unprioritized. When it is reviewed, it is converted to an epic and prioritized. User stories may now be created as children of the epic.

The user story contributes to product quality by encouraging consistent, complete and unambiguous requirements.

6.6 **Acceptance Criteria**

Acceptance criteria are derived from the system use case activity diagrams. They describe functional steps that a user of the product experiences when using the product.

The business analyst is responsible for creating acceptance criteria.

Acceptance criteria are used to assist with creating test cases to validate the product. Once a set of test cases for the feature have been defined the acceptance criteria do not need to be maintained.

Acceptance criteria are often documented textually, in the user story. Developers can use them to verify the quality of their code. In this manner, the acceptance criteria follow the same life cycle as the user story.

Acceptance criteria ensure quality by making sure that test coverage is complete (nothing remains untested).

6.7 **Requirement**

The term requirement refers to any part of the model that is used to capture the behavior of the system. This includes, use cases, activity diagram components, classes, actors and traceability between those components.

The business analyst is responsible for maintaining requirements.

Requirements are the driving force behind the user stories and test cases. Without these, the quality of the product may be severely compromised.

6.8 **Model**

The project model captures both business and product information. It is partitioned into a business view, a solution view and an architectural view. The business view captures business processes (as use cases), business artifacts and business activities. The solution view includes a use case view containing system use cases and their activity, and a logical model view containing information about data used by the product. The architectural view captures the hardware and software into which the solution is deployed.

Figure 16 captures this structure with packages that represent views within the model.

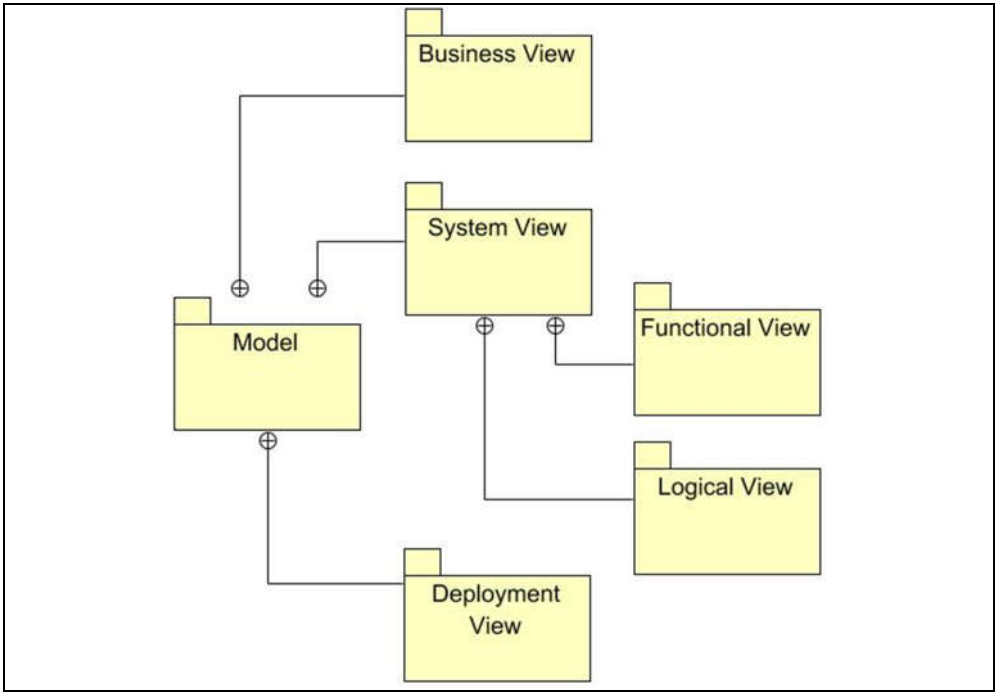

Figure 16 - Model Structure Diagram

It is not necessary to create these models views in any particular order. It is not even necessary to create some views. Benefit is gained from each view as a standalone item.

The model is the responsibility of the business analyst. It retains an abstract view of the business needs, product and architecture of the product. The business analyst uses the model to address questions about the latest build and uses it as a foundation for analyzing new product requirements. Quality assurance uses the model to generate test cases.

All elements in the model are linked within and across packages. This allows a traceability view to be generated within the model.

- **I use a separate package to contain all actors in the model, because they can be shared by all views of the model.**

The model contributes to the quality of the product by keeping a record of its functionality and data and dependencies between those components.

- **It also provides traceability that may prove useful for auditing type exercises.**

6.8.1 Business View

The business view includes business use cases and their mapping to the system view.

A business use case captures a process that is derived from a business need. The business use case is broken into business actions that are distributed across the solution space (or performed manually).

The business analyst is responsible for the business view. The business view keeps a record of business processes that are implemented by the product. It can also be used to create test cases for user acceptance testing. (User acceptance testing is out of scope for this book.)

6.8.2 Functional View

The functional view includes system use cases and their mapping to data and user interfaces.

A system use case captures behavior that the product exhibits. This behavior is broken into system actions. Actions are used to organize functionality into user stories. System use case actions are also used to identify data and to map functionality to user interfaces.

The business analyst is responsible for maintaining a functional view of the product.

The functional view records the functionality of the product as viewed from an external observer. (That observer may be a user or a system interface.) It is used to generate acceptance criteria used by developers and testers. The functional view allows the business analyst to address questions about (as-is and to-be) functionality supported by the product. The functional view allows the business analyst to easily identify product functionality that is impacted as a result of a new user story.

6.8.3 Logical View

The logical view captures the data used by the system. It shows dependencies and relationships between that data.

The business analyst is responsible for maintaining a logical view of the product.

The logical view records the externally observable data used by the product. (That observer may be a user or a system interface.) It is used to assist with detailing a user interface or to develop a system interface to the system. Testers use the data specified by the logical view to validate user and system interfaces. The logical view allows the business analyst to identify data that is impacted as the result of a new user story.

The logical view can specify valid values and other constraints for the data. This provides information for boundary and testing and assists with the user interface design.

6.8.4 Deployment View

The deployment view captures the architecture of the product in terms of its hardware and software components and the connections between those components.

Although the solution architect is responsible for the system architecture, the business analyst may create and maintain architecture diagrams in the deployment view.

The deployment view provides everyone on the development team with a common view of the system architecture. It is primarily used for development and deployment of the product.

6.9 **Sprint Backlog**

The sprint backlog is a repository for user stories that have been assigned to a sprint. The sprint backlog contains only user stories (and child tasks of the user stories). The sprint backlog is a simple prioritized list of user stories. Each user story in the sprint backlog, takes a status of Not started, In Progress or Done.

- **Tasks are out of scope for this book. See Scrum guidelines for more information.**

The sprint backlog may be created and populated anytime prior to the start date of the sprint. The sprint backlog is done the day after the last day of the sprint. Any user stories in the sprint backlog that are not done when the sprint is done are moved to the product backlog as unprioritized items.

- **You may want to setup your process such that any leftover user stories from a sprint are automatically prioritized to the top of the product backlog. Scrum does not dictate where these user stories are placed.**

The sprint backlog contributes to the quality of a product by keeping development focused on functionality that is important to the success of the project.

6.10 **User Interface Design**

A user interface design is a mockup and instructions for navigation, of a screen or part of a screen that is to be built by the development team. The detail in the mockup includes field sizes and location, all colors, images, and sometimes the text that is displayed to the user.

The UI designer is responsible for the UI designs.

A user interface design contributes to product quality by ensuring a consistent look and feel that is compliant with company branding standards. Including text that is appropriate for the user with a well-structured layout gives the product a professional look and feel.

6.11 **Architecture**

The architecture is the foundation on which the product is built. It details the hardware and software components that make up a system and the interfaces between those components.

The architecture is the responsibility of the solution architect. It is captured in the deployment model.

A stable yet scalable architecture is essential for ensuring a quality product.

The architecture contributes to the quality of the product by ensuring that the product is built on a solid foundation that satisfies non-functional requirements.

6.12 **Test Case**

A test case contains a series of steps their preconditions and expected postcondition. The steps describe the actions, data and system responses that users experience when using the product.

The test case is the responsibility of quality assurance.

Test cases are used to validate the accuracy of the product against its requirements. A tester transitions through the steps in the test case, as they are applied to the product build they were written against. Transitioning through the steps should result in the test case postcondition. Any deviations that occur from what is documented in the test case are captured in test results. These test results may generate user stories for the product backlog.

Accurate and complete test cases are important for validating the quality of the product.

6.13 **Incremental Build**

An incremental build is the result of a development sprint. It contains the functionality that was implemented by all user stories that were done during the sprint.

The incremental build is the responsibility of the development team.

An incremental build can be tested as a standalone application.

Incremental development contributes to quality by ensuring that a product build can easily be reverted to a previous working version, when problems arise.

6.14 **Release**

A release is a product build that may be deployed to one or more customers.

The product build is the responsibility of the deployment manager.

A release differs from an incremental build in that the release has been validated that it has sufficient quality to be used by a customer. Releases may vary by customer. The deployed software may be dependent upon the technology used by the customer. The deployment manager ensures that the release is configured correctly for the customer's environment.

A release is a product build whose quality been approved.

6.15 **User Instructions**

User instructions are the documentation describing how the product is used. A set of instructions are written for a specific user.

User instructions are the responsibility of the writer.

Quality does not just apply to the software. Customer facing documentation is also part of a deployment, and quality is easily observed by users of this documentation. If the user manuals are wrong or unclear, the customers could find themselves wasting time by reading the user manuals. The user manual is there to make the user experience more efficient and enjoyable. Them more quality in the documentation, the better the customer experience with the delivered product.