**DTE Release Agile Team Best Practice Guidelines**

**Version #1.3**

|  |  |
| --- | --- |
| **Prepared By:** | Jim Wensink / Heather Lewitt / Chris Dorsch / Jamie Koo / Jason Snyder / Casey Whitcraft |
| **Date Prepared:** | xxxxx 2015 |

TABLE OF CONTENTS

Contents

[1 Introduction 7](#_Toc429041977)

[2 Release Team Cycle 9](#_Toc429041978)

[2.1 Planning and Prioritization 9](#_Toc429041979)

[2.2 Release Backlog Creation 9](#_Toc429041980)

[2.2.1 Unplanned Urgent Release Backlog Creation 10](#_Toc429041981)

[2.3 Release Backlog Refinement 10](#_Toc429041982)

[2.3.1 Unplanned Urgent Release Backlog Refinement 10](#_Toc429041983)

[2.4 Scrum Sprints 10](#_Toc429041984)

[2.5 Go-Live Regression / Business Acceptance Testing 10](#_Toc429041985)

[2.6 Go-Live Prep, Production Go-Live Weekend, Warranty 10](#_Toc429041986)

[2.7 Production Go-Lives During a Release Cycle 10](#_Toc429041987)

[2.8 Quarterly Release Calendar 11](#_Toc429041988)

[3 Release Scrum Team Structure 12](#_Toc429041989)

[4 Release Scrum Team Roles and Responsibility Guidelines 12](#_Toc429041990)

[5 DTE Agile Process Best Practices 12](#_Toc429041991)

[5.1 Guideline for User Story Refinement 12](#_Toc429041992)

[5.1.1 Guideline for User Story Decomposition 13](#_Toc429041993)

[5.1.2 Guideline for User Story Size 13](#_Toc429041994)

[5.1.3 Guideline for User Story Refinement prior to Sprint Planning 14](#_Toc429041995)

[5.1.4 Guideline for User Story Refinement During a Sprint 14](#_Toc429041996)

[5.2 Guideline for Definition of DONE 14](#_Toc429041997)

[5.3 Guideline for Product Owner ACCEPTANCE of User Story 14](#_Toc429041998)

[6 User Story Template Definition 14](#_Toc429041999)

[6.1 Epic State Transition 15](#_Toc429042000)

[6.2 User Story State Transitions 15](#_Toc429042001)

[6.3 Task State Transitions 16](#_Toc429042002)

[6.4 Test Case State Transitions 17](#_Toc429042003)

[6.5 Epic – Product Backlog Prioritization 17](#_Toc429042004)

[6.6 Epic – Indicating Go-Live commitment in Release Backlog 17](#_Toc429042005)

[6.7 User Story Priority – Release Backlog 18](#_Toc429042006)

[6.8 Task Priority 19](#_Toc429042007)

[6.9 Tracking Effort Completed and Effort ToDo 19](#_Toc429042008)

[6.10 BA Standard Task Template 19](#_Toc429042009)

[6.10.1 BA Support /Review 19](#_Toc429042010)

[6.11 QA Standard Task Template 19](#_Toc429042011)

[6.11.1 QA – Test Design Pre-Sprint 19](#_Toc429042012)

[6.11.2 QA – Test Design 19](#_Toc429042013)

[6.11.3 QA - Execution 20](#_Toc429042014)

[6.12 Product Owner Standard Task Template 20](#_Toc429042015)

[6.12.1 Product Owner Review & Acceptance Task Template 20](#_Toc429042016)

[7 User Story Guidelines for Requirements Documentation 20](#_Toc429042017)

[7.1 Requirements Documentation at Epic Level 20](#_Toc429042018)

[7.2 Requirements Documentation at User Story Level 20](#_Toc429042019)

[7.3 Mechanism to Track Requirements Approval 20](#_Toc429042020)

[8 User Story Guidelines for Application Development Documentation 20](#_Toc429042021)

[8.1 Application Design Documentation at Epic Level 20](#_Toc429042022)

[8.2 Application Design Documentation at User Story Level 21](#_Toc429042023)

[8.3 Application Design Documentation at Task Level 21](#_Toc429042024)

[8.4 Mechanism to Track Application Design Approvals 21](#_Toc429042025)

[8.5 Mechanism for estimating Testing Support 21](#_Toc429042026)

[8.6 Mechanism for handling the failure of a test case execution 21](#_Toc429042027)

[8.6.1 Tracking Correcting a Test Failure in an Original Development Task 22](#_Toc429042028)

[8.6.2 Tracking Correcting a Test Failure in a Testing Support Task 22](#_Toc429042029)

[9 User Story Guidelines for Rules Design Documentation 22](#_Toc429042030)

[9.1 Rules Design Documentation at Epic Level 22](#_Toc429042031)

[9.2 Rules Design Documentation at User Story Level 22](#_Toc429042032)

[10 User Story Guidelines for QA Test Documentation 22](#_Toc429042033)

[10.1 QA Test Case Documentation at User Story Level 23](#_Toc429042034)

[10.2 QA Test Case Documentation at FIT User Story Level 24](#_Toc429042035)

[10.3 Mechanism For Test Result Sign-off 25](#_Toc429042036)

[10.4 Mechanism for handling the failure of a test case execution 25](#_Toc429042037)

[11 Defect Tracking 25](#_Toc429042038)

[11.1 Defect Tracking During a Sprint 25](#_Toc429042039)

[11.1.1 Deferring defects from a sprint 26](#_Toc429042040)

[11.1.2 Creating Bug Work Items for defects resolved in a sprint 26](#_Toc429042041)

[11.2 Defect Tracking During UAT 26](#_Toc429042042)

[11.3 Defect Tracking During Regression 27](#_Toc429042043)

[11.4 Defect Tracking During Warranty 27](#_Toc429042044)

[11.5 Processing Defects when closing the Release Backlog 28](#_Toc429042045)

[11.6 TFS Template for Bug Work Items 28](#_Toc429042046)

[11.6.1 Defect State and Reason 28](#_Toc429042047)

[11.6.2 Defect Severity 29](#_Toc429042048)

[11.6.3 Bug Annotation 29](#_Toc429042049)

[11.6.4 Defect Prioritization for Sprint Planning 29](#_Toc429042050)

[12 Code Management and Test Region Management during a Sprint 30](#_Toc429042051)

[12.1 Merging Code Streams 31](#_Toc429042052)

[12.2 Merging Hotfixes or Defects Fixed during QA Regression / UAT / Warranty 31](#_Toc429042053)

[12.3 Coordinating Multiple Scrum Teams Within the Release Team 31](#_Toc429042054)

[12.4 Coordinating Code Management and IT-QA Region Management with Project Scrum Teams 32](#_Toc429042055)

[13 Continuous Integration 32](#_Toc429042056)

[13.1 Automated Build 32](#_Toc429042057)

[13.2 Build System Trigger of Automated Test 32](#_Toc429042058)

[13.3 Automated Test Case Coverage 32](#_Toc429042059)

[14 Sprint Review Guidelines 32](#_Toc429042060)

[15 Release Deployment Testing 33](#_Toc429042061)

[15.1 Release Regression Testing 33](#_Toc429042062)

[15.2 Business/Ops User Acceptance Testing 33](#_Toc429042063)

[16 TFS Project Dashboard 33](#_Toc429042064)

|  |
| --- |
| revision History |
| **Date Revised** | **Document Version** | **Description of Change** | **Author** |
| 3/16/2015 | 1.0 | Initial Version |  |
|  | 1.1 | Updates from discussion at Q2 Planning meeting in Santa Ana. | Jim Wensink and team |
| June | 1.2 | Updated to Guideline changes associated with TFS migration.Reworked timeline, replacing sprint 4 with sprint 0.Added section on defect tracking | Jim Wensink |
|  | 1.3 | Added info on Automation Status field of Test Case Work Item, and updated QA information with TFS Screen shotsAdded info on Bug Work Item Title format, Template usage, and annotationAdded info on when a PPM should be represented by multiple EpicsNoted the need to add discussion on indicating Epic Priority in Product BacklogAdded information on Continuous IntegrationAdded information on TFS Project Dashboard | Jim WensinkCasey Whitcraft |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| **APPROVAL / SIGN-OFF HISTORY** |
| **Date** | **Document Version** | **Name** | **Title** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

#  Introduction

The DTE Release Team utilization of Agile Principles has evolved from a Hybrid development approach in 2014 Q4 and 2015 Q1, to a development approach that utilizes more aspects of Agile Principles as team members are trained on Agile.

The hybrid approach attempted to transition the Executive Sponsor Release Prioritization aspect of the ASP IT Portfolio Management Process and the traditional DTE waterfall Release Development processs into a hybrid development approach that utilized Agile scrum iterative principles during IT-DEV and IT-QA per the following diagram:

Go Live Activities

**Milestones:**

PPM Item UAT Test Complete – 3/20

Q4 IT Modified Agile Release Delivery –

**Milestones:**

Epic User Story (PPM Item) Dev and QA Complete – 3/6

Q4 DTE Planning and Requirements –

**Milestones:**

 Requirements (BRD or Epic User Story) Complete – 10/31

 Planning Est. for Release Lock – 11/21

Portfolio Mgmt Prioritization

IT-QA Regression

Iteration 1 IT-Dev / IT-QA

BA – Epic User Story Generation and Review

BA - BRD Generation and Review

Iteration 2 IT-Dev / IT-QA

UAT Test Cycle

DEV - BRD -> User Stories 10/31

Product Deployment

Iteration 3 IT-Dev / IT-QA

Dev – High Level Design 11/21 or 11/26

Iteration 4 IT-Dev / IT-QA

 **Release Lock ->**

 **Week of 11/24**

**BRD** – When a BRD has been generated, it is the Document of Record pertaining to Requirements and scope.

This document will be the basis for DTE Release Team Agile Best Practice Guidelines going forward until the ASP Agile Standard Practices are in place. At that time the DTE Release Team will align with the ASP Agile Standard Practices.

The following diagram illustrates the DTE Release Agile Process going forward:

DTE Planning and Requirements

Go Live Activities

IT-QA Regression (MODEL)

UAT Test Cycle (MODEL)

Product

GO-LIVE

GO – Live Deployments

To MODEL at sprint completion based on Release Calendar

Sprint x Planning

* Based on Scrum capacity accept User Stories into Sprint Backlog

Sprint Backlog:

* User Stories refined during sprint if changes to User Story content identified by scrum team and Product Owner

Execute Sprint

* Daily Stand-ups
* Detailed Design (if applicable)
* Coding / Rules Changes
* QA testing of Changes
* Test Result Review to DONE

Sprint Review

* Demo Epic / User Story that are DONE to Business
* Business / Product Owner ACCEPTANCE

 Sprint Execution Cycles

 Intake

Portfolio Mgmt / Exec Sponsors Prioritization of Epics

Portfolio Mgmt Unplanned Urgent Prioritization

BA – Epic Initial User Story / Acceptance Criteria Creation

* Decomposition of Epics into business stories
* Sizing of stories
* User Story Description
* Acceptance Criteria
* Business review / approval
* Decomposition for sizing

Scrum Team User Story Refinement

* High Level Design
* Initial Pass Code and Rules Dev Tasks
* Initial Pass Test Case Tasks
* Further Decomposition for sizing

Release Backlog - User Stories Refined:

* Execution needed to reach DONE is understood
* Effort needed to reach DONE is understood
* User Stories meet size guidelines

# Release Team Cycle

Beginning in 2015 Q3, the DTE Release Cycle is made up of the following activities:

* Planning and Prioritization – 2 week duration
* Release Backlog Creation – 2 week duration
* Release Backlog Refinement – 3 week duration
* Scrum Sprints – 12 week duration (Four 3 week sprints)
	+ Sprint 0 – initial sprint of Release Cycle that has 50% allocation toward User Story refinement, and 50% allocation toward User Story execution of carry-over Epics from previous quarter and Unplanned Urgent Epicts
	+ Sprint 1, 2, 3 – allocated timeframe to execute User Stories in Release Backlog
* Go-Live Regression / Business Acceptance Testing – 12 days
* Go-Live Prep – 3 days

## Planning and Prioritization

Planning and Prioritization consists of selecting the Release Item requests, and the Production Support Item requests to be prioritized to be deployed to Production during a Release Cycle.

The Release Item requests are prioritized by the Portfolio Manager and Executive Sponsors based on the ASP IT Portfolio Management process. Committed Release Items is based on 75% of the team capacity for the Release Cycle. The remaining team capacity is reserved for Unplanned Urgent requests and scope adjustments expected with an Agile based process.

The Production Support Item requests are prioritized by the BA Production Support Lead / Product Owner based on Portfolio Management capacity allocations to Production Support for the Release Cycle.

## Release Backlog Creation

A Release Backlog is created by the scrum team from the Release Item requests and Production Support Item requests prioritized as part of the Planning and Prioritization activity. The Release Item requests are Epics created by the DTE BSA team as part of the Portfolio Intake Process. Backlog creation includes initial User Story decomposition of the Epic request, authoring of the User Story description, and in collaboration with business stakeholders the representation of business requirements as Acceptance Criteria for those User Stories.

Having the Intake Process create an Epic for the Release Item request is new to the DTE Intake process as of March 2015. During a transitional period, the scrum team will find Release Item requests prioritized into a Release Cycle that do not have Epics, but that only have a Scope document attached to the Release Item PPM. For those Release Items, part of the Release Backlog Creation will be the incorporation of the scope document information into an Epic.

Release Backlog Creation for a Release Item Epic is complete when business stakeholders have reviewed and approved the User Story(ies) description, and Acceptance Criteria for that Release Item Epic request.

The expectation is that the Release Backlog Creation for all Release Item Epic requests prioritized into the Quarterly Release will be completed during the five week duration allocated for Release Backlog Creation and Release Backlog Refinement.

### Unplanned Urgent Release Backlog Creation

As Unplanned Urgent requests are prioritized by executive sponsors into a Release Cycle, the scrum team will follow the process of User Story creation for those requests.

## Release Backlog Refinement

Release backlog User Stories are further refined by scrum team members:

* adding high level design
* initial IT-Dev and Rules development tasks
* Initial Test tasks, Test Cases and
* possibly further decomposing the User Stories to be “right sized”.

The expectation is that 70% - 80% of the Release Backlog Refinement will be completed during Sprint 0.

The remaining Release Backlog Refinement will be completed as a background activity by the scrum team thru the early sprints of the Quarterly sprinting cycle.

### Unplanned Urgent Release Backlog Refinement

As Unplanned Urgent request are prioritized by executive sponsors into a Release Cycle, and User Story creation is completed for those requests, those User Stories will be added to the list to be refined as a background activity by the scrum team during sprint execution.

## Scrum Sprints

User Story execution during a scrum sprint included the following tasks:

* Detailed Design of application code and Rules (if applicable)
* Development changes of application code, Rules, Configurations, and Settings
* QA Test of development changes

## Go-Live Regression / Business Acceptance Testing

At predefined sprint completions, the code base will be moved to MODEL for final Production Go-Live preparation. This consists of a 12 day period where QA completes a System Regression test cycle, and the business completes a Business Acceptance Test cycle for the Item requests available for Go-Live.

## Go-Live Prep, Production Go-Live Weekend, Warranty

The final step before Production Go-Live, is the Go-Live prep where final Change Controls are generated and approved. Following the Go-Live Weekend deployment is a BA/business Deployment Validation and 2 week warranty period.

## Production Go-Lives During a Release Cycle

Release Calendar will have two planned Production Go-Live weekends during a quarterly cycle, targeted at the end of sprint 1 and sprint 3. These Go-Live weekends will be planned to align with SFL Go-Live weekends and Tracking System Go-Live weekends.

Go-Live weekend at the end of sprint 1 will contain:

* Requests prioritized due to business needs (i.e. client commitment or compliance issue) by Executive Sponsors to deploy to production at the sprint 1 Go-Live
* Requests prioritized into the Release Cycle that the Product Owner and scrum team feel can be completed during sprint 1 and can be targeted to deploy to production at the sprint 1 Go-Live.

Go-Live weekend at the end of sprint 3 will contain:

* Requests prioritized into the Release Cycle that were deployed to production at the sprint 1 Go-Live. These are the requests that were completed during sprint 2 and 3 of the Quarterly Cycle.

 

## Quarterly Release Calendar

The DTE 2015 Quarterly Release Calendar follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Milestone** | **Quarter 1 (Q1)****March 28** | **Milestone** | **Quarter 2 (Q2)****June 13** | **Quarter 3 (Q3)****Aug 1 and Sept 12** | **Quarter 4 (Q4)****Oct 24 and Dec 5** |
| Top priorities due from business, RRs due from PMs | September 1 | Top priorities due from business, RRs due from PMs | November 10 | May 8 | Jul 31 |
| **QUARTERLY PLANNING &PRIORITIZATION** | **September 8** | **QUARTERLY PLANNING &PRIORITIZATION** | **November 24** | **May 15** | **Aug 7** |
| BRD’s, BRD review / approval, or BA Initial User Story | Sept 15 – Oct 31 | User stories, acceptance criteria and baseline estimates due | Feb 27 | Jun 19 | Jul 10 | Sept 11 | Oct 2 |
| Design spec. or User Story Refinement Baseline estimates due | Nov 3 – Nov 21 |
| **LOCK DOWN** | **November 24** | **PRIORIZATION TOUCH BASE** | **February 25** | **June 19** | **September 11** |
| Dev, unit test, QA | Dec 8 – Feb. 27 | Two Sprint Cycles  | Mar. 2– May 15  | May 18 – June 19(Q3S0)+June 22 – July 10 (Q3S1) | Jul 13– July 31 (Q3S2)+ Aug 3 – Aug 21 (Q3S3) | Aug 24 – Sept 11(Q4S0) + Sept 14 – Oct 2(Q4S1) | Oct 5 – Oct 23(Q4S2)+Oct 26 – Nov 13(Q4S3) |
| UAT | Mar 2 – Mar 20 | Regression and business acceptance | May 18 – June 5 | July 13 – July 28 | Aug 24 – Sep 8 | Oct 5 – Oct 20 | Nov 16 – Dec 1 |
| Go Live Prep **Code Freeze** | Mar 23 – Mar 27 | Code freeze | June 8 – June 12 | July 29 – July 31 | Sept 9 – Sept 11 | Oct 21 – Oct 23 | Dec 2 – Dec 4 |
| **GO LIVE** | **March 28** | **GO LIVE** | **June 13** | **August 1** | **September 12** | **October 24** | **December 5** |
| Release warranty period | Mar 28 – Apr 10 | Warranty period | June 15 - 26 | Aug 3 – Aug 14  | Sept 13 - 25 | Oct 25 - Nov 13 | Dec 6 - 18 |
| Final release report and release closed | May 1 | Release Retrospective | July 17 | October 16 | January 8 |

# Release Scrum Team Structure

As of Q2 2015, the DTE Release Scrum Team will consist of:

|  |
| --- |
| DTE Release Scrum Team Members |
| Product Owner | Chris Dorsch |
| Scrum Master | Jim Wensink |
| BA | Q2 2015Liz CrispJana HallIsrael Carter | Q3 2015Sai ErrabelliKim DavisSteve Zanders |
| IT-DEV | Paul RodriguezDoug SipfleChang Li |
| IT-QA | Sheila BillinsVivian VoCasey Whitcraft (part-time) |
| Rules Architect | Reade Young |
| Rules Analyst | Susan Williams |

# Release Scrum Team Roles and Responsibility Guidelines

The Release Scrum team wants to promote the agile principle that team members collaborate across functional disciplines, but recognizes that the different team members bring specific expertise to the team. Based on that, the following attached file contains the Roles and Responsibility Guidelines that the individual functional role will bring to the Release Team:

 

# DTE Agile Process Best Practices

## Guideline for User Story Refinement

Each Release Item (PPM Request) will be represented in the DTE Product Backlog as an Epic. The norm is for there to be a one-to-one relationship between PPM request and Epic. When the execution of a Release Item results in dividing the Go-Live deployment of a PPM request into multiple Go-Live deployments, an Epic will be created to represent each Go-Live. User Stories associated with the PPM request will be children of the Epic representing when the User Story is Go-Live deployed. (Future consideration – use the Feature Work Item to represent different phase Go-Lives, therefore maintaining the one-to-one relationship between PPM request and Epic )

During a Release Cycle, the Epic is used for each Release Item as a business facing artifact of the Agile Principle that aligns with a prioritized committed request to Executive Sponsors for that Release Cycle. It also allows for a consistant location to attach artifacts that are representative of the comprehensive business request. The Epic is created as part of the DTE Intake Process.

User Story Refinement will then be the process of associating User Stories to that Epic.

The Title of the Epic should be the following format:

“<PPM #> - <PPM Short Description>”

When multiple Epics are created for a Release Item, the Epic name convention should add an indication of the Go-Live deployment:

“<PPM #> - <PPM Short Description> – Go-Live <mon/day/yr>”

The Title of User Stories associated with an Epic should be the following format:

“<PPM #> - <User Story Number> <Short Description of User Story>

Production Support Requests prioritized into a Release Cycle, will be represented in the Release Backlog as a User Story. The Title of the User Story should be the following format:

“Prod Support <PPM #> - <PPM Short Description>”

### Guideline for User Story Decomposition

As part of User Story Creation and Refinement the Epic (Release Request) may be decomposed into more than one User Story. User Story decomposition should be driven by User Story size or to separate functionality associate with the Epic into functionality that is required to satisfy the business request, and functionality that is optional. User Stories that are optional would be beneficial but are not required to satisfy the business request, and therefore will be prioritized LOW / MEDIUM in the Release backlog.

Decomposed User Stories should follow the INVEST principle:

A User Story Must Be:

* Independent – Independent from other User Stories
* Negotiable – Details must be negotiated
* Valuable – Valuable to the customer
* Estimable – Estimable by the Scrum team
* Small – Small in implementation effort
* Testable – Testable to allow confirmation

### Guideline for User Story Size

An Epic should be represented as a single User Story when the team feels that the full content of the Epic can be executed by the scrum team in 7-8 working days.

An Epic should be decomposed into multiple User Stories when the team feels that the full content of the Epic cannot be executed by the scrum team in 7-8 working days. A decomposed User Story should at most represent complexity that the team feels can be executed by the scrum team in 5 or less working days.

### Guideline for User Story Refinement prior to Sprint Planning

As part of Release Backlog User Story Refinement consists of:User

* Decomposing Epics into User Stories that meet guidelines of User Story Size.
* Decomposing Epics into User Stories to isolate optional functionality of the request.
* Authoring and getting Business approval of User Story Description and Acceptance Criteria

The goal here is to follow the Agile Principle to emphasize working software over up front formal, detailed requirements. So the requirements represented in the User Story / Acceptance Criteria should be at a level that the scrum team and business stakeholders feel are complete enough that the scrum team will interpret them successfully into demonstratable software that meets the business request.

* Authoring initial pass of Code Development and Rules Development Tasks

This includes making a task estimate

* Authoring intial pass of QA Test Tasks, and QA Test Cases

This included making test execution task estimate

* Attaching any high level design documentation generated as part the Authoring of Code Development and Rules Development tasks

### Guideline for User Story Refinement During a Sprint

When a User Story is accepted into a Sprint Backlog at a Sprint Planning session the expectation is that the User Story Refinement has reached a point of completion that the scrum team can make the judgement:

* what execution is needed to reach DONE for the User Story
* the effort associated with the User Story execution is understood
* the User Story size conforms to sizing guidelines.

During the execution of tasks for a User Story, refinement to the User Story may be identified (changes / clarifications to Acceptance Criteria, changes / clarifications to existing development tasks or test cases, additional development tasks or test cases). As part of scrum, when these refinements are identified, the team should collaborate with the Product Owner on the needed changes, make the agreed upon refinements to the User Story, and raise the issue to the Scrum Master or Product Owner if any of the refinements put at risk executing the User Story to DONE during the sprint.

## Guideline for Definition of DONE

When all DEV, RULES, and QA tasks are Complete, User Story test results will be reviewed by the BSA who will declare DONE after verifying that test results demonstrate comprehensive coverage of Acceptance Criteria.

## Guideline for Product Owner ACCEPTANCE of User Story

<To Be Defined>

# User Story Template Definition

## Epic State Transition

|  |
| --- |
| **When Epic In Release Team Product Backlog** |
| **New** | Epic Created in TFS |
| **Active** | Business Need Information authored by Intake BAs (Scope information). Preliminary estimates being gathered |
| **Defined** | Estimates available. Intake Complete. Epic ready for prioritization |
| **Removed** | Epic has been cancelled by Product Owner |

|  |
| --- |
| **When Epic In Release Team Release Backlog** |
| **Active** | Scrum BAs adding detailed business need to Epic, one or more User Stories still being refined |
| **Defined** | Epic approved and all User Story refinement complete for sprint planning |
| **Delivered** | All User Stories ACCEPTED at Sprint Reviews |
| **Removed** | User Story has been cancelled by Product Owner |

## User Story State Transitions

|  |
| --- |
| **When User Story In Release Team Product Backlog** |
| **New** | Initial User Story being authored by Intake BAs |

|  |
| --- |
| **When User Story In Release Team Release Backlog**  |
| **New** | User Story / Acceptance Criteria being authored by Scrum BAs |
| **Approved** | User Story / Acceptance Criteria ready for Task Refinement, it has been reviewed with business stakeholders |
| **In Analysis** | User Story Task Refinement in progress |
| **Ready** | User Story refined to point that it is ready for sprint planning |
| **Removed** | User Story has been cancelled by Product Owner |

|  |
| --- |
| **When in User Story in Release Team Sprint Backlog** |
| **Committed** | User Story not being worked |
| **In Progress** | User Story actively being worked by IT-DEV or RULES |
| **In Progress** | All  IT-DEV and RULES tasks ready for test, but User Story not being tested |
| **In Progress** | QA test in progress |
| **In Progress** | All QA tests PASSED, but BA review not complete |
| **DONE** | BA reviewed test results and claimed DONE, User Story targeted for sprint review  (In Template – status indicates PO accepted) |
| **DONE** | PO and stakeholders accepted DONE status at sprint review  (Not in Template) |
| **Removed** | User Story has been cancelled by Product Owner |

## Task State Transitions

|  |
| --- |
| **When Task In Release Team Release Backlog**  |
| **New** | Task being authored or has been authored by Scrum Team |
| **Removed** | Task has been cancelled by Scrum Team |

|  |
| --- |
| **When Task In Release Team Sprint Backlog**  |
| **New** | Execution of task has not started |
| **Active** | Task being executed |
| **Closed** | Task execution complete |
| **Removed** | Task has been cancelled by Scrum Team |

## Test Case State Transitions

|  |
| --- |
| **When Test Case In Release Team Release Backlog**  |
| **Design** | Test Case being authored or has been authored by Scrum Team |
| **Removed** | Test has been cancelled by Scrum Team |

|  |
| --- |
| **When Test Case In Release Team Sprint Backlog**  |
| **Design** | Test Case has not Executed with PASS Condition |
| **Ready** | Test Case design is complete and available for test execution |
| **Closed** | Test execution of test case complete |
| **Removed** | Task has been cancelled by Scrum Team |

## Epic – Product Backlog Prioritization

<To Be Added>

## Epic – Indicating Go-Live commitment in Release Backlog

Per the ASP Portfolio Management Prioritization process, based on Release Team capacity and executive sponsor product backlog prioritization, a list of Epics are committed to GO-LIVE during a Release Cycle. That list of Epics and their associated User Stories are moved from the Product Backlog to the Release Release backlog for the specified Release Cycle.

The timing of the Go-Live within the Release Cycle of a specific Epic is further negotiated by Portfolio Management with Executive Sponsors as follows:

1. Go-Live Executive Sponsor commitment no later than the sprint 3 GO-LIVE. This is the default commitment
2. Go-Live Executive Sponsor commitment at the sprint 1 GO-LIVE. This is an exception commitment based on client requirement or compliance.
3. Go-Live Executive Sponsor commitment Off-Cycle. This is preferably a rare exception, as (1) and (2) provide Go-Live intervals of 6 weeks.

Once the above Go-Live commitments are negotiated with executive sponsors, any changes to those Go-Live commitments can only be made with acceptance by the executive sponsor.

With the sprint 1 GO-LIVE, one additional classification of Epic Go-Live commitment is needed internal to the team to identify the Epics that the Product Owner and Scrum Team feel can be completed in time for the sprint 1 GO-LIVE. When Epics are targeted for the sprint 1 GO-LIVE by the Product Owner and Scrum Team, that commitment can be changed internally by the Product Owner and Scrum Team due to sprint dynamics.

Based on the above, there is the need in the Release Backlog to have the following indication for an Epics Go-Live commitment:

* Executive Sponsor Commitment to Deploy Sprint 1 Go-Live or an Off-Cycle Go-Live
* Product Owner and Scrum Team Target to Deploy Sprint 1 Go-Live
* Executive Sponsor Commitment to Deploy Sprint 3 Go-Live
* Executive Sponsor Agreement to defer from the Release Cycle Go-Lives

In TFS, the Priority Field of the Epic will be used to specify these for Go-Live Statuses:

* Priority “1” - Executive Sponsor Commitment to Deploy Sprint 1 Go-Live or an Off-Cycle Go-Live
* Priority “2” - Product Owner and Scrum Team Target to Deploy Sprint 1 Go-Live
* Priority “3” - Executive Sponsor Commitment to Deploy Sprint 3 Go-Live
* Priority “4” – Epics that contain technical User stories in a Release Backlog – User Stories that need to complete during the Release Cycle, but do not represent business requests (i.e. Code Build or Code Stream Merge User Stories)

Executive Sponsor initially establishes an Epic as being a “Priority 1” or “Priority 3” commitment.

Executive Sponsor agreement required to change an Epic from a “Priority 1”

Product Owner and Scrum team can change an Epic from a “Priority 3” to a “Priority 2”, or from a “Priority 2” to a “Priority 3”.

## User Story Priority – Release Backlog

The Product Owner needs to be continually prioritizing the User Stories in the Release Backlog to give guidance on User Stories to focus refinement for sprint planning, and for guidance during sprint planning. Prior to each sprint planning meeting, the Product Owner will update the User Story priorities in the Release Backlog to the following four buckets:

* HIGH – must be added to the next sprint backlog
* MEDIUM – Tier 2 priority to consider to be added to the next sprint backlog
* LOW – Tier 3 priority to consider to be added to the next sprint backlog
* BLOCKED – not a candidate to be added to the next sprint backlog (BLOCKING issue should be documented in BLOCKED field of User Story)

In TFS, the “Stack Rank” field will be used to represent User Story Priority:

* Stack Rank of “1” - HIGH – must be added to the next sprint backlog
* Stack Rank of “2” - MEDIUM – Tier 2 priority to consider to be added to the next sprint backlog
* Stack Rank of “3” - LOW – Tier 3 priority to consider to be added to the next sprint backlog
* Stack Rank of “4” - BLOCKED – not a candidate to be added to the next sprint backlog (BLOCKING issue should be documented in BLOCKED field of User Story)

## Task Priority

When working tasks from a sprint backlog, the task priorities should be consistent with the priorities of the User Story that the task is associated with (work tasks associated with User Stories of PRIORITY HIGH first,…..).

In TFS, since there is no automated way to have the “Stack Rank” propogated from a User Story to a Task, and there is concern about manually propagating the priorities from User Stories to Tasks, the task work item will not explicitly have a priority setting.

A query will be setup in TFS, “Sprint x backlog – Me” that will enable a scrum team member to see the User Stories in a sprint backlog, listed in priority order, and the tasks assigned to the scrum team member for each User Story. This should facilitate having the scrum team member select tasks to work associated with the highest priority User Story.

## Tracking Effort Completed and Effort ToDo

During a Sprint, prior to each daily stand-up, a task owner needs to update the following fields for User Story tasks that are actively being executed:

* Remaining Work – update with the estimated hours remain to complete execution of the task
* Completed Work - Use this field for the hours that you have completed on a task.

## BA Standard Task Template

### BA Support /Review

**Estimate:** Depends on size of User Story

**Purpose:** All support tasks the BA will encounter throughout the Sprint. ITQA/Dev Support, answering requirements questions, reviewing test results, etc.

**Owner:** BA only

## QA Standard Task Template

### QA – Test Design Pre-Sprint

**Estimate:** Hours to be spent on Test Case design during User Story Refinement prior to accepting a User Story into a sprint

**Purpose:** Test Case Design needs to be completed prior to sprint planning to a point that the team can understand the testing complexities needed to make the judgement at sprint planning if the User Story can be accommodated in the sprint.

**Owner:** QA team member

As this is a task that is not executed during the sprint, it will not be moved into the sprint backlog when the User Story is moved into a sprint backlog.

### QA – Test Design

**Estimate:** Hours to be spent on completing Test Case design during a sprint

**Purpose:** Completion of test case design during the sprint.

**Owner:** QA team member

### QA - Execution

**Estimate:** hours to be spent setting up and executing Test Cases

**Purpose:** Execution of Test Cases.

**Owner:** QA team member

## Product Owner Standard Task Template

### Product Owner Review & Acceptance Task Template

**Estimate:** 30 – 60 Mins. depending on size of User Story

**Purpose:** Product Owner to review IT-QA test results and Accept User Story into sprint review.

**Owner:** Product Owner

# User Story Guidelines for Requirements Documentation

## Requirements Documentation at Epic Level

As part of the DTE Intake process, a BA will document the business need at the Epic level in TFS.

During transition, for Release Request that did not have an Epic created during Intake, the BA will create the Epci during Release Backlog User Story creation by attaching the PPM Scope Document to the Epic in TFS

The BA will attach a process map to the Epic to outline the business process flow to reach the business need when it is felt the complexity of the request justifies the documentation. As a guideline, an Epic that is decomposed into four or more User Stories should be considered as complex enough to justify a process map.

## Requirements Documentation at User Story Level

As part of the Release Backlog Refinement prior to sprint planning, the BA will document the Business objective of the User Story in the User Story description in the following User Story template format:

*“As a <Role/Actor>, I would like to <Action>, so that I can <Objective / User Goal>”*

The expected results of the user story that meet the Business objective are documented in the Acceptance Criteria by the BA. The Acceptance Criteria are the basis for the test cases of the User Story.

## Mechanism to Track Requirements Approval

User Story(ies) description and Acceptance Criteria associated with an Epic are reviewed and approved by business stakeholders before proceeding to other User Story refinement activities (i.e. Development and Test Task authoring).

# User Story Guidelines for Application Development Documentation

## Application Design Documentation at Epic Level

Each Release Item (PPM Request) prioritized during Release Cycle Planning will be represented in the Release Backlog as an Epic. An Epic is represented as one or more User Stories per the guidelines for User Story sizing in Section 5.1.

High level design will be done at the Epic level if required as part of the pre-sprint refinement activity. A high level design document will be produced if its need is immediate and significant, or if the complexities of the changes indicate the need for a high level design document. The scrum team will make that decision as part of the refinement process.

Some guidelines that would indicate the need for a high level design include:

* Magnitude of the changes require the effort to be decomposed into four or more user stories
* The request requires changes to an interface to a cross-system Assurant or vendor application
* The request requires changes to the overall application system architecture

## Application Design Documentation at User Story Level

In the pre-sprint refinement phase, the team identifies the user stories which are complex and may need detail design documentation. Generally, a user story composed of three or more tasks may be a candidate for a design document. It’s necessary to keep the document very light which may include high level architecture modeling and wireframes. This is to identify the major areas of technical risk early in the project without taking on the risk of over modeling.

If design doesn’t get completed before the sprint planning meeting but the complexity justifies detail design, it is possible to consider the user story for the sprint by including a task for design that requires design document.

## Application Design Documentation at Task Level

Tasks will be used to break down user stories to be delivered into manageable units to be worked by the development team. Tasks typically range in size from 1 hour to 2 days. Tasks larger than this guideline should be broken down further to allow the team to incrementally complete the work and show progress. Developer will describe what he/she intends to accomplish in each task.

## Mechanism to Track Application Design Approvals

When a design document is created for a complex story, it will be reviewed with the team. Unless anyone objects to what’s presented at the end of the review meeting, it would be considered “approved”.

## Mechanism for estimating Testing Support

When refining a User Story, a developer needs to consider estimating time needed to support the testing activities of the User Story. This actual testing support (i.e. supporting test case authoring, test setup, collaborating on test execution) and any anticipated effort to resolve issues from test execution failures.

The developer should consider either accounting for testing support estimates in the estimates of the individual development task, or generalizing this effort into a separate “testing support task” in the User Story.

## Mechanism for handling the failure of a test case execution

When a testing task fails, the tracking of effort to correct the test failure condition can be tracked one of two ways:

* If the scrum team can identify the development task that is associated with the failed test, tracking of the effort to correct the test failure condition can be accomplished in the original development Task.
* If the scrum team cannot identify the development task that is associated with the failed test, a separate development testing support task should be used to track the effort to correct the test failure.

### Tracking Correcting a Test Failure in an Original Development Task

* The developer needs to change the development task to “Active”, and adjust the Remaining hours to the estimate of the time expected to fix the test failure.
* The original “Estimate” should not be adjusted.
* Effort to fix the test failure should be added to the “Completed Work” field.
* When the corrective action is included in a build for testing, the task should be changed to “Closed.

### Tracking Correcting a Test Failure in a Testing Support Task

* If necessary, the developer needs to change the testing support task to “Active”, and adjust the To Do hours to the estimate of the time expected to fix the test failure.
* The original “Detail Estimate” should not be adjusted.
* Effort to fix the test failure should be added to the “Completed Work” field.
* When the corrective action is included in a build for testing, if no other testing support is anticipated, the task should be changed to “Closed.

# User Story Guidelines for Rules Design Documentation

## Rules Design Documentation at Epic Level

## Rules Design Documentation at User Story Level

Each User Story that requires rules work will have a task titled “Design and Build- Rules”

The rules task description will include:

* The acceptance criteria that is being addressed by the rules task
* All processes, rules, and queues added for testing purposes
* All setting changes and additions
* Any other additions and alterations made to DTE ITQA for testing purposes
* What client/s that can be tested
* How the build for the user story can be tested to meet acceptance criteria

All design and build documentation will be saved in a single Excel spreadsheet and attached to the task in Version One and shall include the following:

* Screenshots of rules, processes, and queues altered or created for the testing of the user story
* Screenshots of all setting changes and additions made for testing purposes
* Screenshots of any other additions and alterations made to DTE ITQA for testing purposes
* Any testing logs and screenshots of testing performed by Rules Architect or BRA

Rules design and build documentation will not have a formal review approval process, but the expectation is that scrum team members whose tasks rely on Rules will review the information included in VersionOne.

# User Story Guidelines for QA Test Documentation

## QA Test Case Documentation at User Story Level

In TFS, Test Cases for a User Story will be captured as a Test Case Work Item, and should be created as a Link Type “Tested By” relationship to the User Story

Each Test Case will be named as ‘System Test #\_’. (Shown Below)



Each Test Case will have each area filled out, such as Description, Setup (when applicable), Inputs (when applicable), Steps, Expected Results and Actual Results

Each Test Case will have screen shots stored in a word document and attached to the individual test case. Screen shots are not to be in one document as a whole unless there is only one test case. Screen shots also should not be entered at the user story work item. Again, screen shots are be entered at the test case work item.

Example Test Result Document:



The Steps within the testing task are to be detailed so that the developer, BA, RA and Product Owner can all recreate with ease. Testing task steps are to be numbered. (Remember, we want to be able to move these to automation.) See example below.



The Status field of the Test Case Work Item should be used to indicate the results of the latest execution of the Test Case:

* Ready – Test Case has not been executed
* Failed – Last execution of test case failed
* Passed – Last execution of test case passed

User Stories can have a testing task named Test Case Review. This will be an optional meeting with BA, RA and Developer to ensure the testing tasks meet the Acceptance Criteria. Moving this Testing Task to PASSED will be the indication that the Test Cases have been reviewed and concensus reached that they comprehensively cover the Acceptance Criteria.

The Automation Status field of the Test Case Work Item should be used to indicate the following:

* Blank – means this test case is not a candidate for automation
* Automated – successfully added to the automation suite
* Planned – QA Lead/Tester has determined this is a candidate for automation; (currently shown as Not Automated below but JJ requested to be changed)

## QA Test Case Documentation at FIT User Story Level

Any Epic that has been decomposed into multiple User Stories should be evaluated for the possible need for a FIT User Story. This will be a Feature Integration Test cycle of the comprehensive Epic, executing key test cases from all the User Stories of the Epic to guarantee that when all development is complete, the Epic has maintained coverage of the collection of User Story Acceptance Criterias. The Format and Content of this Test Case is the same as the format and content of an individual User Story Test Case Task.

A FIT User Story is needed if in completing the development there was the need for code integration between the coding changes for two or more user stories.

## Mechanism For Test Result Sign-off

All User Stories should have a task named BA Support/Review. Moving this Task to Completed will be the indication that the Test Cases have been reviewed by the BA and confirmation received that the results comprehensively cover the Acceptance Criteria.

When QA Testing has Passed all QA Test Cases for a User Story, an email will be sent by the QA tester to the Business Analyst (cc: Casey) to advise that testing is complete and the QA tester should work with the BA to get confirmation of testing results.

As a best practice, as part of the BA test result review, the test results will be discussed with business stakeholders to receive feedback from business that the Acceptance Criteria were demonstrated by test. This review of test results by business stakeholders as part of the BA test result review process is not required, but desired, for BA sign-off. If business stakeholders are not available to participate in this review of a User Story test results prior to the the end of a sprint, User Story DONE status should be declared based on only the BA test result review.

## Mechanism for handling the failure of a test case execution

When a testing task fails, the QA tester needs to collaborate with the scrum team to confirm that the testing results represent a true test failure, and that the information in the test results is complete enough for effective corrective action to be taken. When this is completed, the following actions should be taken:

* The Title of the Test Case should be changed to “\*\* INCLUDES RETEST”. This will allow for all members to know that this task had a test failure.
* The results of the failed test execution should be attached to the test case work item
* The “Remaining Work” hours of the “QA – Test Execution” task should be increased allow time for retesting.

The results of each execution of the test case should be attached to the test case work item per the guidelines above to document the number of failed attempts of the test case as well as the PASS execution of the test case.

When executing the retest, the “Completed Work” field of the “QA – Test Execution” task should be updated with the time spent executing the retest.

The “Original Estimate” will not increase based on additional work associated with the retest.

# Defect Tracking

## Defect Tracking During a Sprint

During sprint execution, the scrum team will be resolving issues associated with Test Case execution failures as they progress a User Story to DONE. As discussed above, one best practice during sprint execution of a User Story is to capture the results of each test case execution as an attachment to the Test Case. Therefore, when a User Story is “In Progress”, all needed information with respect to issues identified during testing will be internal to the User Story artifacts. Resolving test case execution failures is part of the scrum team commitment to reach User Story Definition of Done during the sprint timebox.

### Deferring defects from a sprint

During sprint execution, at times, the scrum team will collaborate with the Product Owner to defer investigating or resolving an issue raised during User Story testing until after the active sprint, and to claim User Story DONE during that sprint, even with the open issue. In these instances, the issue that has been identified needs to be incorporated into the Release backlog to be prioritized at a later time.

When the scrum team and Product Owner claims a User Story is DONE with unresolved issues existing from test, those issues should be documented in the TFS Project as Bug Work Items in the Release Backlog. The Bug Work Item should be linked as a Child to the Epic parent of the User Story the issue is associated with.

When creating a Bug Work Item for an issue to be deferred from a sprint, the Bug Work Item Title should follow the following formats:

<PPM #> - <Description>

The Test Phase field should be set to “QA”.

### Creating Bug Work Items for defects resolved in a sprint

For internal QA metrics, the QA team may opt to create bug work items for test case execution failures even when the issue is investigated and resolved during the sprint. These QA internal Bug Work Item artifacts should be isolated from Backlog and sprint management by setting the Test Phase field to “Intra-sprint”.

## Defect Tracking During UAT

During UAT phase, all defects identified should be entered to the TFS Project as Bug Work Items in the Release Backlog.

When the defect can be associated with a specific Epic developed for the Go-Live being tested, the Bug Work Item should be linked as a Child to the Epic.

When creating a Bug Work Item during UAT, the Bug Work Item Title should follow the one of these formats:

<PPM #> <Description>

or

Unattached – <Description>

The Test Phase field should be set to “UAT”.

UAT defects will be reviewed by the Product Owner and those that are identified as “Must-Fix” before Go-Live will be moved to the active Sprint Backlog and become part of that sprint. When the Product Owner identifies a Bug Work Item as a must fix for a specific Go-Live date, the Title should be edited to the following format:

<PPM #> <Description> - (mm/dd Go-Live)

Where mm/dd represents the month and day of the scheduled Go-Live

These “Must-Fix” defects will be implemented in the Go-Live Release code stream and tested directly in MODEL. A User Story will be added to the Release Backlog to merge these “Must-Fix” changes into the future Release code stream.

## Defect Tracking During Regression

During regression testing, all defects identified should be entered to the TFS Project as Bug Work Items in the Release Backlog.

When the defect can be associated with a specific Epic developed for the Go-Live being tested, the Bug Work Item should be linked as a Child to the Epic.

When creating a Bug Work Item during Regression, the Bug Work Item Title should follow one of the following formats:

<PPM #> - <Description>

or

Unattached –<Description>

The Test Phase field should be set to “Regression”.

Regression defects will be reviewed by the Product Owner and those that are identified as “Must-Fix” before Go-Live will be moved to the active Sprint Backlog and become part of that sprint. When the Product Owner identifies a Bug Work Item as a must fix for a specific Go-Live date, the Title should be edited to the following format:

<PPM #> <Description> - (mm/dd Go-Live)

Where mm/dd represents the month and day of the scheduled Go-Live

These “Must-Fix” defects will be implemented in the Go-Live Release code stream and tested directly in MODEL. A User Story will be added to the Release Backlog to merge these “Must-Fix” changes into the future Release code stream.

## Defect Tracking During Warranty

During the Warranty phase, all defects identified should be entered to the TFS Project as Bug Work Items in the Release Backlog.

When the defect can be associated with a specific Epic developed for the Go-Live being tested, the Bug Work Item should be linked as a Child to the Epic.

When creating a Bug Work Item during Regression, the Bug Work Item Title should follow one of the following formats:

<PPM #> - <Description>

or

Unattached –<Description>

The Test Phase field should be set to “Warranty”.

Warranty defects will be reviewed by the Product Owner and those that are identified as “Must-Fix” as part of Warranty will be moved to the active Sprint Backlog and become part of that sprint. When the Product Owner identifies a Warranty Bug Work Item as a must fix, the Title should be edited to the following format:

<PPM #> <Description> - (mm/dd Warranty)

Where mm/dd represents the month and day of the Go-Live that is under Warranty

These “Must-Fix” defects will be implemented in the Release code stream and tested directly in MODEL. A User Story will be added to the Release Backlog to merge these “Must-Fix” changes into the future Release code stream.

## Processing Defects when closing the Release Backlog

The Release Backlog will be Closed at the end of the warranty period, and all Epics deployed to Production will also be Closed. At the time that the Release Backlog is CLOSED all open Bug Work Items in the Release Backlog will need to be processed in one of the following ways:

* If the Product Owner feels the Open Bug Work Item should be considered for resolution in a future Release Cycle, a Production Support Intake Request should be generated.
* If the Product Owner feels no further action should be taken on the Open Bug Work Item, the Open Bug Work Items should be Closed

## TFS Template for Bug Work Items

When entering a defect as a Bug Work Item, the following fields of the Work Item should be completed:

* Repro Steps – Steps to reproduce the defect, with the unexpected test results included
* System Info – any information pertaining to operational environment (i.e. software version, workstation environment)
* Test Cases – link to test case used to expose the defect
* Attachments – screen shots of unexpected test results if helpful

### Defect State and Reason

The Bug Work Item represents the following states for a defect:

* New – New Defect being triaged for necessary corrective action
* Active – Being worked by IT-DEV, IT-QA, and other testing teams
* Closed – Defect has been fixed and tested
* Removed – Cancelled. Not a defect.

The Reason field should be used in the Closed and Removed states:

**State Closed**

* Reason – Verified

**State Removed**

* Reason – As Designed
* Reason – Cannot Reproduce
* Reason – Duplicate
* Reason - Obsolete

When working a Bug that has been identified in UAT, Regression, or Warranty, the first step is triage and identification of necessary corrective action for the Bug. The BSA leads this effort with support from Operations UAT, IT-DEV, and IT-QA. During this triage, the Bug Work Item should be STATE “New”, and the “Assigned To” field should be a BSA.

Once a UAT/Regression/Warranty Bug Work Item has bee triaged, and corrective action has been documented, the Bug Work Item should be “Assigned To” and IT-DEV resource. The Bug Work Item State should be transitioned to “Active”

When the fix is promoted to MODEL, the developer should indicate this in the Bug Work Item by changing the assigned field to the BA that created the Bug in TFS.

Therefore, for Bug Work Items in the UAT / Regression / Warranty phases:

* When a Bug is assigned to a BSA and in the New State, triage is being performed to establish necessary corrective action
* when a Bug is assigned to a developer or IT-QA resource and in the Active State, a fix is being investigated,
* when a Bug is assigned to a BSA and in the Active State, a fix has been promoted to MODEL and is available for test validation

### Defect Severity

The Bug Work Item has a Severity field to be used to represent Bug Severity:

* 1 – Critical (show-stopper/blocking issue)
* 2 – High (impacts operational functionality)
* 3 – Medium (has an available work around)
* 4 – Low (nice to have)

The scrum team should collaborate to establish the defect severity for Bug Work Items generated during sprint execution.

The BSA should establish the defect severity for Bug Work Items generated during UAT, Regression, or Warranty.

### Bug Annotation

During the execution to resolution of a defect, information such as clarification of root cause and status of correct action should be captured as Comments in the TFS Bug Work Item.

Specifically, when a defect corrective action is promoted to Test, an entry should be added at the top of the “Repro Step” Field that explains the Resolution of the corrective action:

“Resolution - <description of corrective action taken>

### Defect Prioritization for Sprint Planning

Bug Work Items added to a Release Backlog during sprint execution, are considered for acceptance during sprint planning along with User Story Work Items.

The Product Owner needs to be continually prioritizing the Bug Work Items in the Release Backlog along with User Stories in the Release Backlog to give guidance during sprint planning. For consistency, prior to each sprint planning meeting, the Product Owner will update the Bug Work Item priorities in the Release Backlog into the same four buckets used for User Story prioritization. See the section “User Story Priority” for explanation of prioritization and usage of “Stack Rank” field.

# Code Management and Test Region Management during a Sprint

The Epics and User Stories associated with a Release Backlog or Sprint Backlog will have different Go-Live targets per the following:

* Off-Cycle Go-Live Targets based on business prioritizies or commitments
* Epics targeted for Go-Live at the end of sprint 1
* Epics targeted for Go-Live at the end of sprint 3

The IT-DEV team will need to provide Software Configuration Code Branch Management to isolate User Story development into separate code streams based on the Go-Live target dates of the User Stories actively being executed in a sprint.

An IT-QA test region will need to be available for each separate code streams that has User Stories actively being executed in a sprint.

For Example:

**Scenerio (1) –** All User Stories selected at sprint planning belong to Epics that are targeted to be included in the Go-Live at the end of sprint 1

* **Code Stream Management –** All code changes for the sprint are done from the single code branch that is targeted for sprint 1 deploy to business Acceptance testing
* **IT-QA Region Management –** A single IT-QA test region will be used for all User Story execution during the sprint

**Scenerio (2) –** Some User Stories selected at sprint planning belong to Epics that are targeted to be included in the Go-Live at the end of sprint 3, Some User Stories selected at sprint planning belong to Epics that are targeted to be included in an Off-Cycle Release between sprints 1 and 3.

* **Code Stream Management –** Code changes for the sprint will be done from two code streams:

(1) Sprint 3 Code Branch – User Story work associated with Epics targeted to be included in the Go-Live at the end of sprint 3.

(2) Off-Cycle Code Branch based on Sprint 1 Code Stream – User Story work associated with Epics targeted to be included in the Go-Live Off-cycle between sprints 1 and 3.

* **IT-QA Region Management –** Two IT-QA test regions will be used:
1. Sprint testing of User Story work associated with Sprint 3 Code Branch
2. Sprint testing of User Story work associated with Off-cycle Code Branch

At this time, since there are two IT-QA test regions, during a given sprint, User Stories can be actively worked from at most two Code Branches.

## Merging Code Streams

Whenever the Go-Live prioritization requires User Stories to be worked in separate code streams, an activity to merge the code streams after the earlier Go-Live is complete needs to be accounted for.

A User Story needs to be added to the Release Backlog to account for the merging of these code streams. This User Story should contain Test Case Tasks to confirm that the User Stories from the “source” code stream were successfully merged into the “destination” code stream.

That User Story should then be accepted into a sprint at a sprint planning meeting.

Using Scenerio (2) above as an example, a User Story would be created with description “Merge code stream for the Off-Cycle Go-Live between Sprint 1 and Sprint 3 into the Sprint 3 code stream”.

## Merging Hotfixes or Defects Fixed during QA Regression / UAT / Warranty

At times, code fixes will need to be applied directly to a Release code stream for one or more of the following reasons:

* Production Hotfix
* Defects fixed during post-sprint QA Regression
* Defects fixed during post-sprint UAT
* Defects fixed during Go-Live Warranty

These code fixes will result in a build from the relevant Release code stream, and will be QA tested in MODEL or in an IT-QA region isolated from the active Release Scrum IT-QA test region.

These code changes need to be merged into the active Release Scrum code stream at the time judged appropriate by the scrum team. The process followed should be similar to the process for merging code streams:

* A User Story needs to be added to the Release Backlog to account for the merging of these code fixes from the Release code stream into the active Release Scrum code stream. This User Story should contain Test Case Tasks to confirm that the code fixes were successfully merged into the active Release Scrum code stream. Any member of the scrum team can create this User Story, with the best practice guideline being that the IT-QA Lead should take responsibility to support the Product Owner in guaranteeing the User Story is created and refined.
* The User Story should then be accepted into a sprint at a sprint planning meeting. If the timing is such, once refined, this User Story could be added to an “in-flight” sprint  if the scrum team agrees it has capacity to accommodate it.

## Coordinating Multiple Scrum Teams Within the Release Team

During any given quarter that the staffing of the Release Team necessitates the establishment of two or more Release Scrum Teams, the code management and IT-QA region management during a sprint will be managed from a “Scrum of Scrums” collaboration facilitiated by the Release Manager.

The individual scrum teams will share a common Code Management and IT-QA region management strategy during a sprint utilizing common code branches and common IT-QA regions based on the Go-Live target for User Stories accepted into the scrum team sprint backlog per the guidelines described above.

In order to share a common Code Management and IT-QA region management strategy, all Release Scrum Teams will align to a common sprint calendar.

## Coordinating Code Management and IT-QA Region Management with Project Scrum Teams

In order to manage the limited number of IT-QA test regions, it is possible during a Release Cycle to have a DTE Project Scrum Team share a code branch and IT-QA test region with the Release Scrum Team. This would be considered when it has been confirmed that the Project will Go-Live at the same time as a targeted Go-Live for the Release Scrum Team.

In order for a Project Scrum Team to share a common Code branch and IT-QA test region as sprint activities from the Release Scrum Team, the Project Team must meet these criteria:

* Must be following a scrum development process
* Must have a sprint calendar aligned with the Release Scrum Team sprint calendar
* Must have scrum master coordinate scrum activities with the Release Manager in a “Scrum of Scrums”

# Continuous Integration

## Automated Build

The DTE build system is fully automated, with a new DTE package being generated from a code stream in the TFS code repository from a single command.

At this time, developers coordinate the check in of their code for completed dev tasks at the end of the day afer which a build is scheduled and delivered for automated testing upon completion.  The plan is to transition to coordinating these check ins and build to be twice daily – once at noon PT, and once at the end of the day PT.

## Build System Trigger of Automated Test

The automated testing component is designed to poll a drop-off location for any new release that needs to be tested. If it detects a new release and no automated testing is in progress, then it will move the release to its own folder for testing.

The build system at the completion of generating a new DTE package, moves that package to that designated drop-off location. This provides the trigger to the Automated Testing component to execute automation.

## Automated Test Case Coverage

# Sprint Review Guidelines

Sprint review meetings will be coordinated by the Product Owner, with collaboration by all scrum team members.

The Product Owner and BA will initiate discussions with business stakeholders during the sprint to identify key aspects of the Sprint Backlog that the the business stakeholders would like to have included in the sprint review discussion / demo.

BA and Production Owner will coordinate discussions with scrum team toward the end of the sprint to collaborate on planning the review to focus on key aspects business stakeholders want to observe. As part of this planning scrum team will establish if User Stories included in a sprint for a common PPM should be bundled for review discussion, or discussed individually.

BA will take the lead in discussion of User Stories during the review, with support provided by other scrum team members.

# Release Deployment Testing

## Release Regression Testing

## Business/Ops User Acceptance Testing

# TFS Project Dashboard

The DTE TFS project dashboard has been setup to give information regarding the PPM/Epics prioritized into active Scheduled Go-Lives, and information regarding the Release Backlog and active Sprint Backlog:

 (<http://tfs2013.assurant.com:8080/tfs/ASP/ASP.DTE-DraftTracEnterprise/ASP.DTE-DraftTracEnterprise%20Release%20Team> – requires Chrome or IE11 as the web browser),

The example dashboard shown here is based on having the Sept. 12th Go-Live in a UAT cycle, and the Q4 Release Cycle prioritized (Go-Live Oct. 24 and Dec. 5)

Core Sept. 12 Go-Live Business Commitments – This shows the state of the Epics (PPMs) prioritized for the 9/12 Release.

Core Sep 12 Bugs – This shows the status of defects required to be fixed as part of an active UAT cycle

Core Oct 24 Go-Live Business Commitments - This shows the state of the Epics (PPMs) that are Executive Sponsor business commitments to be delivered at the 10/24 Go-Live.

* At this time there are none.

Core Oct 24 Go-Live PO Targets – This shows the state of the Epics (PPMs) that have been targeted by the Product Owner to Go-Live Oct. 24th.

Core Dec. 5 Go-Live Business Commitments -  This shows the state of the Epics (PPMs) that are Executive Sponsor business commitments to be delivered at the 10/24 Go-Live.

Core Release Backlog User Stories – This shows the state of all User Stories prioritized into the Q4 Release Cycle

Core Sprint Backlog User Stories - This shows the state of all User Stories accepted into the CURRENT sprint

Core Sprint Backlog Bugs - This shows the state of all Bug Work Items accepted into the CURRENT sprint

