Section 1 Tracking Fixes and Enhancements

1.1 Introduction

The Environmental Screening Tool (EST) development team uses the Bugzilla software to report and track bugs and requests for enhancements. This section provides an overview of how it is used in the EST development work flow. More detailed information about Bugzilla and its user document are found in the Bugzilla Guide available on the Bugzilla website at: http://www.bugzilla.org/

The ETDM Bugzilla page is found at: https://dev.fla-etat.org/bugzilla

A link to the ETDM Bugzilla page is also available on the EST dev site, in the left menu under Administration, Developer Tools. Two products are currently tracked on the ETDM Bugzilla page:

1. ETDM-EST – the secure EST website
2. ETDM Public Access Site – the customized version of the EST available to the public through the FDOT CEMO website

1.2 Definitions

The following definitions are used when reporting or updating a record in the Bugzilla software. These definitions are copied here for easy reference from the on-line help at https://dev.fla-etat.org/bugzilla/page.cgi?id=fields.html, with minor clarification as it relates to the EST.

Status and Resolution

The Status and Resolution fields used together define and track the life cycle of a bug. The following table shows the combinations of Status and Resolution as used in the EST work flow.

STATUS	RESOLUTION
The **status** field indicates the general health of a bug. Only certain status transitions are allowed.	The **resolution** field indicates what happened to this bug.
NEW This bug has recently been added to the assignee's list of bugs and must be processed. Bugs in this state may be accepted, and become **ASSIGNED**, passed on to someone else, and remain **NEW**, or resolved and marked **RESOLVED**. **ASSIGNED** This bug is not yet resolved, but is assigned to the proper person. From here bugs can be given to another person and become **NEW**, or resolved and become **RESOLVED**. **REOPENED** This bug was once resolved, but the resolution was deemed incorrect. For example, a bug is **REOPENED** when more information shows up and the bug is now reproducible. From here bugs are either marked **ASSIGNED** or **RESOLVED**.	No resolution yet. All bugs which are in one of these "open" states have the resolution set to blank. All other bugs will be marked with one of the following resolutions.

UNCONFIRMED *(Not Used In EST)* This bug has recently been added to the database. Nobody has validated that this bug is true. Users who have the "canconfirm" permission set may confirm this bug, changing its state to NEW. Or, it may be directly resolved and marked RESOLVED.	
RESOLVED A resolution has been taken, and it is awaiting verification by QA. From here bugs are either re-opened and become **REOPENED**, are marked **VERIFIED**, or are closed for good and marked **CLOSED**. **VERIFIED** QA has looked at the bug and the resolution and agrees that the appropriate resolution has been taken. Bugs remain in this state until moved to production, at which point they become **CLOSED**. **CLOSED** The bug is considered dead, the resolution is correct. Any zombie bugs who choose to walk the earth again must do so by becoming **REOPENED**.	**FIXED** A fix for this bug is checked onto DEV and tested. **INVALID** The problem described is not a bug. (Usually for user error.) **WONTFIX** The problem described is a valid bug which will never be fixed. **(Do not use this without prior authorization from FDOT.)** **DUPLICATE** The problem is a duplicate of an existing bug. Marking a bug duplicate requires the bug# of the duplicating bug and will at least put that bug number in the description field. **WORKSFORME** All attempts at reproducing this bug were futile, and reading the code produces no clues as to why the described behavior would occur. If more information appears later, the bug can be reopened. **MOVED (Not used in EST)** The problem was specific to a related product whose bugs are tracked in another bug database. The bug has been moved to that database.

Severity

This field describes the impact of a bug.

Blocker	Blocks development and/or testing work
Critical	crashes, loss of data, severe memory leak
Major	major loss of function
Minor	minor loss of function, or other problem where easy workaround is present
Trivial	cosmetic problem like misspelled words or misaligned text

Enhancement Request for enhancement

Priority

This field describes the importance and order in which a bug should be fixed. This field is utilized by the programmers/engineers to prioritize their work to be done. The available priorities range from **P1** (most important) to **P5** (least important.)

P1 – Highest priority, an error in the program that prevents a user from performing their job

P2 – Critical problem that makes using the application difficult to use, but hasn't been reported by a user

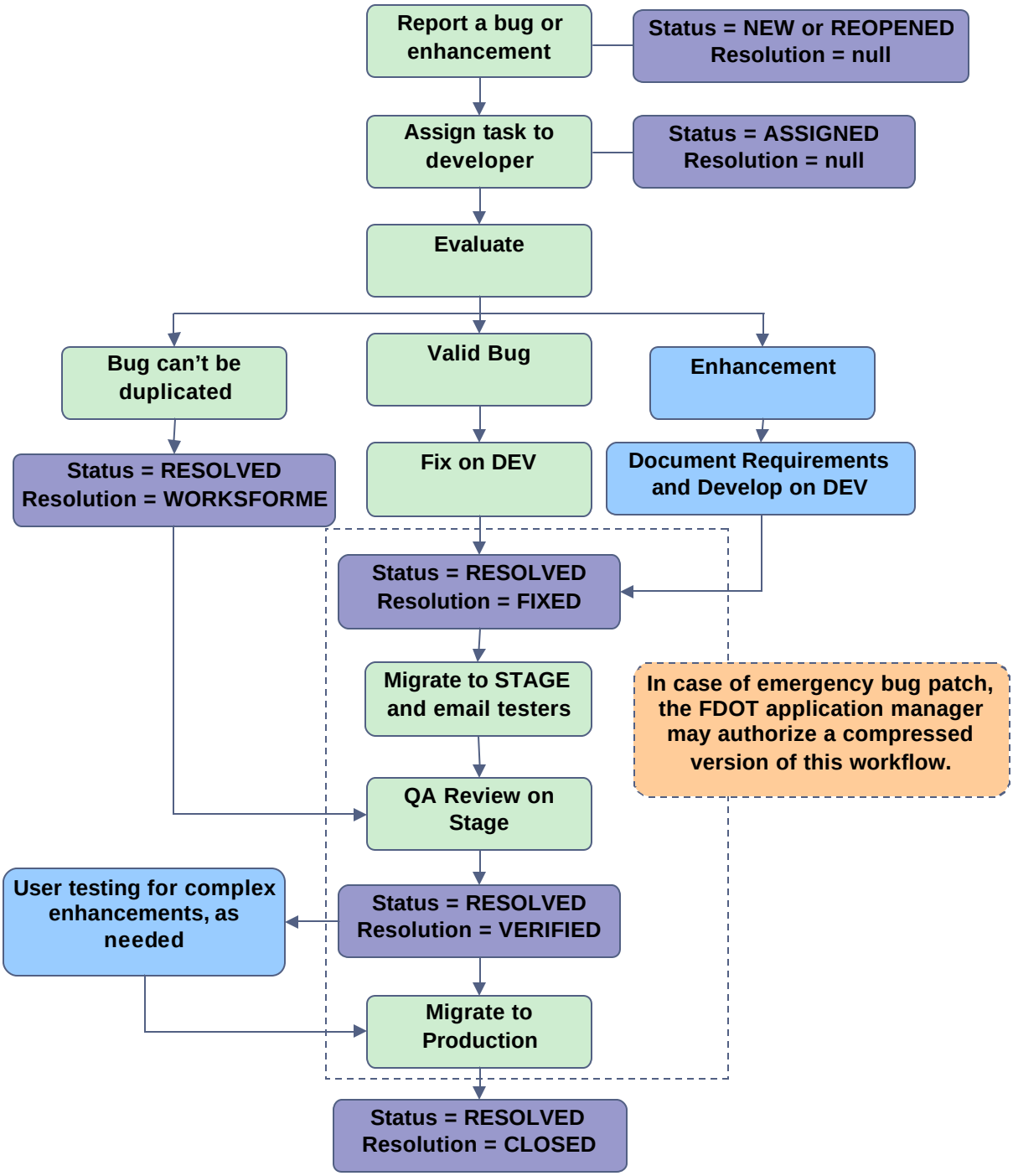
P3 – Minor problem with the functionality

P4 – Minor problem related to formatting or compliance with other GUI standard

P5 – Least important, usually an enhancement for which work has not begun

1.3 Work Flow

The following flow chart illustrates how Bugzilla fits in the EST development life cycle. Tasks are shown in green, Bugzilla updates of status and resolution are shown in blueberry. Tasks shown in dark cyan are only performed for enhancements.



Anyone on the development team may report a bug. The person who receives an assignment may re-assign the bug to another person after discussing the assignment with that person.

Section 2 Version Control and Code Deployment

2.1 Introduction

The EST development team requires the ability to concurrently develop fixes and enhancements to the EST, while at the same time:

- Eliminating conflicts with changes made by other members of the team
- Recovering previous revisions, for example previous stable releases
- Allowing quick deployment any revision to one of the EST development, staging, and production web servers

Some of the software used to support these requirements are Apache ant, Subversion and TortoiseSVN.

Apache Ant is a Java build tool that is responsible for executing a series of interdependent tasks for building the EST web application, for example updating the web site files from the team's latest revisions, compiling the Java files, and setting permissions on the web site files. More information about Apache Ant can be found at http://ant.apache.org.

Subversion is an open-source version control system used by the EST development team to manage concurrent development of the EST source code. A Subversion repository is a database of revisions to the EST source code. The repository is accessible to the entire development team at the password-protected address http://dev.fla-etat.org/svnweb. Subversion can be used from the command-line or from other Subversion clients to interact with the repository, including such actions as:

- Accessing or downloading any previous revision of the source code
- Keeping the local working copy of source code in synch with repository
- Detecting and assisting with resolving conflicts between concurrently edited source code files

Subversion commands can be executed from the command line, or from Subverision clients such as TortoiseSVN and TortoiseMerge. See http://tortoisesvn.tigris.org for details on installation and usage of these clients.

2.2 Work Flow

The usage of Subversion by the EST development team follows an iterative development cycle, outlined in steps 1-6 below. Unless otherwise noted, the steps are executed by an individual programmer. Note that all Subversion (svn) commands shown can be substituted with equivalent steps in TortoiseSVN. In the case of emergency bug fixes, an abbreviated version of this work flow may be authorized by the FDOT Application Manager.

1) On a periodic basis, deploy latest development release candidate to remote development server for testing. This step is executed at least once per cycle, and then repeated D minus 1 times after each iteration of step 2, where D > 1
    a. Tag development release candidate D.
        i. svn copy file://var/www/fla-etat/svn/trunk file://var/www/fla-etat/svn/tags/P.0.D
    b. Deploy release candidate P.0.D to remote development server using Apache ant. The working copy of the development server should correspond to the trunk, so that a switch between tags won't be needed as this step is repeated
    c. Test changes on remote development server, as necessary

d. Update Bugzilla, setting status and resolution as documented in Section 1

e. QA/QC by internal review team.

2) Develop latest fixes and enhancements. This step may be repeated indefinitely within a single development cycle.

a. At the beginning of this step, the trunk is a copy of tag P.0.1, which represents the output of the last step in the cycle (step 6)

b. Checkout, switch, or update working copy to trunk

i. svn checkout file://var/www/fla-etat/svn/branches/trunk .

ii. svn switch file://var/www/fla-etat/svn/branches/trunk .

iii. svn update .

c. Make changes to working copy files, as necessary per task requirements

i. Use any text editor or suitable Integrated Development Environment (IDE) of choice, for example vim (Vi IMproved, a programmers text editor), IntelliJ IDEA, or Eclipse

d. Test changes on local development server, as necessary per task requirements

e. Update working copy, detecting out-of-date local files and conflicting changes

i. svn update .

f. Resolve conflicts, if any, using a merge program such as diff, TortoiseMerge, WinMerge, or a suitable IDE.

g. Commit changes in working copy to trunk

i. svn commit . –message "<programmer comment on changes>"

h. Update Bugzilla, setting status and resolution as documented in Section 1

3) On a periodic basis, create and deploy the latest stable branch to remote staging server for testing and bug fixes. This step is executed at least once per cycle, and then repeated S minus1 times after each iteration of step 4, where S > 1

a. Copy latest development release candidate P.0.D to stable branch

i. svn copy file://var/www/fla-etat/svn/tag/P.0.D file://var/www/fla-etat/svn/branches/stable

b. Tag stage release candidate P.S.0.

i. svn copy file://var/www/fla-etat/svn/branches/stable file://var/www/fla-etat/svn/tags/P.S.0

c. Deploy release candidate P.S.0 to stage using Apache ant. The working copy of the staging server should correspond to the stable branch, so that a switch between tags won't be needed as this following steps are repeated

d. Test changes on remote staging server, as necessary

e. Notify testers via e-mail to begin internal review

f. QA/QC by internal review team

g. Update Bugzilla, setting status and resolution as documented in Section 1

4) Fix bugs on stable branch. Repeat until appointed bug fix time has elapsed.

a. Checkout, switch, or update working copy to stable branch

i. svn checkout file://var/www/fla-etat/svn/branches/stable .

ii. svn switch file://var/www/fla-etat/svn/branches/stable .

iii. svn update .

b. Make changes to working copy files, as necessary per assigned bugs

i. Use any text editor or suitable Integrated Development Environment (IDE) of choice, for example vim (Vi IMproved, a programmers text editor), IntelliJ IDEA, or Eclipse

c. Test changes on local development server, as necessary

d. Update working copy, detecting out-of-date local files and conflicting changes

i. svn update .

e. Resolve conflicts, if any, using a merge program such as diff, TortoiseMerge, or WinMerge

f. Commit changes in working copy to stable branch

i. svn commit . –message "<programmer comment on changes>"

g. Update Bugzilla, setting status and resolution as documented in Section 1

5) On a periodic basis, roll up latest stable branch to production for release. This step is executed at least once per cycle, and then repeated R minus1 times after each iteration of step 6, where R > 1

a. Copy latest stable release candidate P.S.0 to tagged release R.0.0, where R = P+1

i. svn copy file://var/www/fla-etat/svn/tags/P.S.0 file://var/www/fla-etat/svn/tags/R.0.0

b. Deploy release candidate R.0.0 to remote production server

i. Use ant switch –Dtag=R.0.0

c. Update Bugzilla, setting status and resolution as documented in Section 1

6) Prepare for next development iteration.

a. Merge latest production release R.0.0 with development trunk, storing result in working copy

i. svn merge file://var/www/fla-etat/svn/tags/R.0.0 file://var/www/fla-etat/svn/trunk .

b. Resolve conflicts, if any, using a merge program such as diff, TortoiseMerge, or WinMerge

c. Commit changes in working copy to development trunk

i. svn commit . –message "<programmer comment on changes>"

d. Tag new development trunk release R.0.1

i. svn copy file://var/www/fla-etat/svn/trunk file://var/www/fla-etat/svn/tags/R.0.1

e. Deploy new development trunk release R.0.1 to development server using Apache Ant. The working copy of the development server should correspond to the trunk, so that a switch between tags won't be needed as this following steps are repeated

Section 3 Internal Testing

3.1 Identify Fixes or Enhancements for Testing

1. In Bugzilla, select bugs where Status = RESOLVED and Resolution = FIXED or WORKSFORME. For example, to test on the secure EST site use the following steps in Bugzilla:
   - Select Search, click on the Tab for Advanced Search
   - Status = Resolved
   - Product = ETDM-EST
   - Use the other options to narrow the search, if needed

2. All bugs with Status = RESOLVED and Resolution = FIXED or WORKSFORME are ready for testing on DEV.
3. After an email notification indicates that STAGE deployment is complete, these bugs may be tested on STAGE.

3.2 Selecting Accounts and Projects

1. Test Accounts

Select an account from the list of test accounts. Each account has different privileges and accessibility throughout the ETDM system. When following the testing procedure, be sure to test utilizing various accounts. This allows you to make sure access is being granted properly to whatever is being tested. *Administration >> View Page Permissions* contains information regarding access granted to specific roles.

2. Projects

You can test using already existing projects [on stage and dev] or you can create a new project. Utilize the new search button in the upper right hand corner to search for existing projects.

a. Search: If you already have a project in mind to use, select manual search. If you want to search by using criteria, select "power search" and select the criteria pertinent to what you will be testing. "Power search" will allow you to select projects according to planning organization, status within the ETDM process, phase of the ETDM process, etc. When using the "power search", it is important to make selections to narrow down your search as oppose to leaving all fields at "All." Otherwise, the program will attempt to select all projects in the system which will take an extended amount of time.

b. Create New Project:
   1. Select the *Create New Project* wizard.
   2. Project Name = Name, Test, Date
   3. Planning ID = Any Letter
   4. Planning Organization = May Vary
   5. ETDM Phase = May Vary
   6. County = Within Planning Organization
   7. Beginning Location = A
   8. Ending Location = B
   9. Consistency = Make Any Selection
   10. Purpose and Need = Test
   11. Project Description = Test
   12. Summary of Public Comment = Test
   13. Select and Mode and Alternative Type
   14. FIHS = Y or N
   15. Total Length = Any Number
   16. Total Cost = Any Number
   17. Beginning Location = A
   18. Ending Location = B
   19. Go To *Tools >> Maintain Project Diary >> Add Alternative Description*
       Complete All Fields and Submit
   20. *Wizards >> Update EDTM Project*
       Complete All Fields and Submit

3.3 Testing

1. Test the Page
   a. Enter information in all fields; submit
   b. Enter information in some fields, but not all; submit
   c. Leave all fields blank; submit

d. Enter incorrect values in fields; submit

2. After Testing the Page
   a. Check to see that information has been recorded in any corresponding reports(s).
   b. Check to see that information has been recorded in any corresponding tool(s).
   c. Check to see if information can be edited.

3. Spot Check
   Test to make sure other functionality has not been affected. Do this by checking things such as unrelated tools or reports used throughout the life cycle of a project. Also, check things listed under accounts and help.

3.4 Reporting To Bugzilla

1. Search Bugzilla to check that the bug has not already been reported. You can do this by doing a search and entering relevant keywords. This search should be done with open and closed bugs
2.
3. Indicate your findings in the comments section. If you experience the same bug or something new, be sure to reopen the bug. List the page, any selection criteria used on the page, and the username of the test account. Include all steps you went through to generate that bug. If it is an error, copy and paste the error message into the comments section.
4.
5. If the bug is verified fixed, the QA/QC representative sets the bug status to VERIFIED. Other testers should indicate successful resolution in the comments section.