

Generic Fault Process (GFP)

Generic Fault Process (GFP)

James Edgell
Version 1.0
Updated 3/21/08

Generic Fault Process (GFP)

1.0	Generic Fault Process (GFP) Overview.....	3
2.0	Roles & Responsibilities.....	3
3.0	Fault state flows	4
3.1	New faults	4
3.2	Fault assignment (Fault Initiator, FTS Administrator, Engineering Manager) ..	5
3.3	Fault processing	6
3.3.1.	The five possible outcomes of the Fault Processing phase.....	6
3.3.2.	Fault Resolution Steps (Software Engineer).....	6
3.4	Fault Verification and Regression testing (Component Tester)	7
3.5	Fault Verification and Regression testing (Integration Tester).....	7
3.6	Sanity testing (Release Manager)	8
3.7	User Acceptance Testing (UAT) (Release Manager)	8
3.8	Fault closure (Software Engineer)	8
3.9	Deployment.....	8
3.10	Fault State Flow Chart Diagram	9
3.11	Fault prioritization	10
4.0	Termination verses monitoring a fault.....	10
Appendix A – Field Descriptions		12
Appendix B – Fault Status Values and Authorized Changes		15

1.0 Generic Fault Process (GFP) Overview

The intention of the Generic Fault Process is to track, measure, and analyze fault data. Each fault should be closely monitored to assure that forward momentum is maintained until closure. By following the steps outlined in this document, all faults will be handled in a consistent manner to effectively process faults.

2.0 Roles & Responsibilities

Role	Description
<i>FTS Administrator</i>	The person who monitors the FTS for <i>New</i> faults that are not currently assigned and assigns them.
<i>Fault Initiator</i>	The person who enters a fault into a Fault Tracking System (FTS)
<i>Engineering Manager</i>	The person who monitors the FTS for <i>New</i> faults that are not currently assigned to an available <i>Software Engineers</i> and assigns them. This person also performs an auditing role and will reject FTS records that violate the process outlined in this document.
<i>Software Engineer</i>	The person responsible for resolving the Fault and updating the FTS record.
<i>Component Test Manager</i>	The person who monitors the FTS for <i>New</i> faults that are not currently assigned to an available <i>Component Tester</i> and assigns them. This person also performs an auditing role and will reject FTS records that violate the process outlined in this document.
<i>Component Tester</i>	The person who performs initial Testing and Verification Testing on an individual module or component. In many cases this person also performs the role of <i>Fault Initiator</i> .
<i>System Integration Test Manager</i>	The person who monitors the FTS for <i>New</i> faults that are not currently assigned to an available <i>System Integration Tester</i> and assigns them. This person also performs an auditing role and will reject FTS records that violate the process outlined in this document.
<i>System Integration Tester</i>	The person who performs Testing and Verification on a completely integrated product. In many cases this person also performs the role of <i>Fault Initiator</i> .
<i>Release Manager</i>	The person the interfaces with customers to collect customer found defects and prepare and coordinate new releases.

Generic Fault Process (GFP)

NOTE: If your organization is small, you can combine most of these roles into a single person. For example; Engineering Manager and Software Engineer can be combined into a single role and that person is responsible for all tasks associated with both roles.

3.0 Fault state flows

This section outlines the different states that a fault passes through when processed by the system. It is the responsibility of each person to set the states correctly when processing a fault. Use the descriptions and the Fault Statechart Diagram to better understand the entire process.

3.1 New faults

Any person can enter a fault into a Fault Tracking System (FTS). The person who enters the fault into FTS becomes the “*Fault Initiator*” and assumes all responsibilities as outlined for this role.

Example 1: If a customer service representative is informed by a customer that there is a problem with the system, it is the customer service representative’s responsibility to gather the appropriate information and enter it into the FTS; In this case the *customer service representative* is assuming the role of *Fault Initiator*.

Example 2: If a software engineer discovers a fault, that software engineer needs to enter the fault into the FTS. The fault must be entered, even if that engineer is responsible for resolving the fault. In this case the *Software Engineer* becomes the *Fault Initiator* and assumes the responsibilities for both roles.

Example 3: If a tester discovers a new fault while testing, the tester becomes the *Fault Initiator* and assumes the responsibilities for that role.

The *Fault Initiator* must populate the data for the following fields:

Field Name	Values
Assigned to	If you already know who the person that this fault is assigned to, choose the name and set the <i>Status</i> field to Assigned .
Category	<ul style="list-style-type: none">• Fault• Enhancement – This is marker for any enhancement that was misidentified as a defect.
Consistency	<ul style="list-style-type: none">• Consistent – the fault can be reproduced at will.• Intermittent – if this is selected, Monitor Termination should also be set.
Customer	The customer that is or will experience the fault.
Description	A full description of the issue identified in the Title field.
Found Release	The software version number that the fault was found in.
Impact	Level 1-5 Level 1: the fault will result in extreme customer dissatisfaction. Level 2: the fault will probably cause customer dissatisfaction. These two impact levels must be fixed prior to release. Level 3: the fault may cause customer dissatisfaction and should be fixed as soon as possible. Level 4: the fault will probably be noticed by customers and should be fixed in the next convenient release.

Generic Fault Process (GFP)

Field Name	Values
	Level 5: the fault is unlikely to be noticed by customers or to cause customer dissatisfaction.
Monitor Termination	This field is used to set an agreed date that this fault will be terminated on if it is not reproducible.
Phase Found	<p>If the fault is known to be in a released version of software, this field is set to <i>Release</i> regardless of where it was found.</p> <ul style="list-style-type: none"> • <i>Requirements</i> – Found when defining the requirements or a requirements FTR. • <i>Design</i> – Found when reviewing a system design or a design FTR. • <i>Implementation</i> – The fault was identified during implementation (coding) • <i>Unit Test</i> – The fault was found by the engineer during a unit test phase. • <i>Component Test</i> – The fault was found by a <i>Component Tester</i> while testing at a component level. • <i>Integration Test</i> – The fault was found by an <i>Integration Tester</i> while testing the interaction between components such as end-to-end testing or a completely integrated product. • <i>Staging</i> – The fault was found while testing on a staging system • <i>Release</i> – the fault was found in software that has been released to one or more customers. If the discovered fault is in software that has been released to a customer; this field must be set to 'Release'. If the fault is discovered in pre-released software, but know also to be in a released version, this field must be set to 'Release'.
Project	Identify the correct project that the software is covered under.
Severity	<ul style="list-style-type: none"> • Severity Level A: (also classified as critical) this level is meant to represent faults that result in an unusable system, sub-system, product, or critical features from the customer's perspective. • Severity Level B: (also classified as major) this level is meant to represent faults that limit a customer's normal use of the system, sub-system, product, or major non-critical features from a customer's perspective. Typical examples would include: • Severity Level C: (also classified as annoying) this level is meant to represent faults that annoy the customer, but don't preclude full use of the system, sub-system, product, or critical features from a customer's perspective. Typical examples would include: • Severity Level D: (also classified as cosmetic) this level represents faults that the customer is unlikely to notice (cosmetic). Typical examples would include:
Steps to reproduce	Enter detailed step-by-step instructions on how to reproduce the fault.
Title	A short description of the fault.
Workaround	Any known steps to workaround the fault or steps that can minimize the impact of the fault.

3.2 Fault assignment (Fault Initiator, FTS Administrator, Engineering Manager)

Once a fault is entered into the Fault Tracking System (FTS), the *Fault Initiator* can assign it directly to an *Engineering Manager* that is responsible for that component or module and set the *Status* to **Assigned**.

If the *Fault Initiator* does not know who the correct *Engineering Manager* is, they should assign it to the *FTS Administrator* and set the status to **New**. In this case, it is the responsibility of the *FTS Administrator* to monitor the database and assign all **New** faults to an available *Engineering Manager*.

3.3 *Fault processing*

3.3.1. The five possible outcomes of the Fault Processing phase

- **Duplicate** – The problem has already been identified in FTS by another user. In this case the Status field is set to **Duplicate** and the original *FTS ID Number* should be recorded in the *Original Fault ID* field. Do not set the original record to **Duplicate**, only set successive records to **Duplicate Status**.
- **Deferred** – Management has decided not to pursue this fault fix in the current release. Only Management should make this decision.
- **Terminated** – It was determined that the fault entered is not a fault; sometimes know as “by design”.
- **Monitor** – The fault is not reproducible but suspected to be a real fault. The *Software Engineer* will place the fault in a *Monitor* state. (At this point, it is recommended to add additional application logging to capture the root-cause in the event that the fault was reproduced.) There should be an agreement between the *Software Engineer* and the *Fault Initiator* on how long monitoring should take place before terminating the fault. Once the fault monitor time has expired the fault should be terminated. It can always be resurrected by changing the status back to *Assigned* if the fault continues to be reported. Reactivating the fault will assure that all historical data is preserved, preventing duplication of effort in the root-causal analysis process.
- **Resolved** – A successful root-causal analysis was performed, a fix was identified, and a new build with that fix is ready for testing. All faults need to be root-caused, if a fault fix is submitted and the fault is found not to be resolved (known as a “bounced fix”) then the root-cause identified is know to have been false.

3.3.2. Fault Resolution Steps (Software Engineer)

There are several steps that must be performed in order to properly resolve a fault. These steps and data fields are the sole responsibility of the *Software Engineer*.

1. **Take Ownership** – The Software Engineer that will own this fault and take responsibility for its resolution. The Software Engineer must assign their name to the *Software Engineer* field.
2. **Reproduce the fault** - Only faults that can be reproduced can enter the root-causal analysis phase. If a fault can not be reasonably reproduced then it is placed into *Monitor* state. With the fault reproduced or placed into *Monitor* state, update he following fields:
 - **Fault Type**
 - **Progress History**
 - **Monitor Termination (if placed in monitor state)**
3. **Perform a root-causal analysis** - A root-causal analysis is simply identifying the exact cause of the fault. Armed with a root-cause, the Software Engineer should be able to reproduce the fault at will and demonstrate the exact effect using a step-traceable debugger. With a root-cause in hand, update the following FTS fields (***you can not proceed to the next step unless the root-cause is in hand and the following fields are updated in the FTS***):

Generic Fault Process (GFP)

- *Root-Cause*
 - *Estimated Hours*
 - *Estimated Date*
 - *Preventative Measure*
 - *Progress History*
4. **Identify the resolution and implement it** – Knowing the root-cause should greatly simplify the fault fix that needs to be generated. With the resolution completed, update the following FTS fields:
 - *Resolution*
 - *Set Status to “Resolved”*
 - *Actual Hours*
 - *Actual Date*
 - *Progress History*
 5. **Prepare a test build for the test team** – Create the binaries and release notes containing all the FTS numbers and FTS Titles that are fixed in the build and deploy to the component test team. Update the following FTS fields.
 - *Progress History*
 - *Assigned* – this field should be set to the *Component Test Manager* to be assigned or directly to the person who is performing the verification testing.

3.4 **Fault Verification and Regression testing (Component Tester)**

- **Verification Testing** - Fault testing should start with verification of all the resolutions identified in the FTS system for the release to be tested. For every successful fix that was submitted, the *Status* field should be set to **Component Verified**. For every fault that was submitted as resolved that was not fixed, the status should be set to **Bounced**. If any new faults are encountered during verification testing, these faults should be entered as **New**. If there are new or bounced defects found in the build being tested the release should not proceed to regression testing because a new build will be required.
- **Regression Testing** - After all FTS faults are verified first, testing should then proceed with full regression testing. If any new faults are encountered during regression testing, these faults should be entered as **New** and the release should not proceed to *Integration Testing*. After a successful Regression Test with no new faults, the *Tester* should assign all FTS faults to an *Integration Test Manager* for *Integration Testing* with a status set to **Component Verified**.

3.5 **Fault Verification and Regression testing (Integration Tester)**

- **Verification Testing** - of all the resolutions identified in the FTS system for the release to be tested. For every successful fix that was submitted, the *Status* field should be set to **System Verified**. For every fault that was submitted as *Component Validated* that was not fixed, the status should be set to **Bounced**. If any new faults are encountered during verification testing, these faults should be

Generic Fault Process (GFP)

entered as *New*. If there are new or bounced defects found in the build being tested the release should not proceed to regression testing because a new build will be required.

- **Regression Testing** - After all FTS faults are verified first, testing should then proceed with full regression testing. If any new faults are encountered during regression testing, these faults should be entered as *New* and the release should not proceed. After a successful Regression Test with no new faults, the *System Tester* should assign all FTS faults to a *Release Manager* for *Sanity Testing* with a status set to *System Verified*.

3.6 **Sanity testing (Release Manager)**

Sanity testing is performed by the *Release Manager*. The *Release Manager* that has intimate knowledge of the customer's unique needs and is the last stop check before releasing any software updates to the customer. The *Release Manager* should not solely depend on the test teams to catch all Faults. At this stage a *Release Manager* can do one of three things with a Fault, they can **Validate**, **Terminate**, or **Bounce** the resolution. If the fault passes testing, the *Release Manager* should set the *Status* field to **Validated** and reassign it to the *Software Engineer* for closure after the fix has been deployed to production. If the fault is not fixed, the *Release Manager* should reassign it to the original *Software Engineer*, update the **Progress History** as to why they believe it *Bounced*, and set the *Status* to **Bounced**.

3.7 **User Acceptance Testing (UAT) (Release Manager)**

Candidate production releases should be deployed for *User Acceptance Testing (UAT)*. Only after the customer has confirmed that the release is acceptable, the *Release Manager* will inform Engineering via an e-mail that the release can be deployed to production.

3.8 **Fault closure (Software Engineer)**

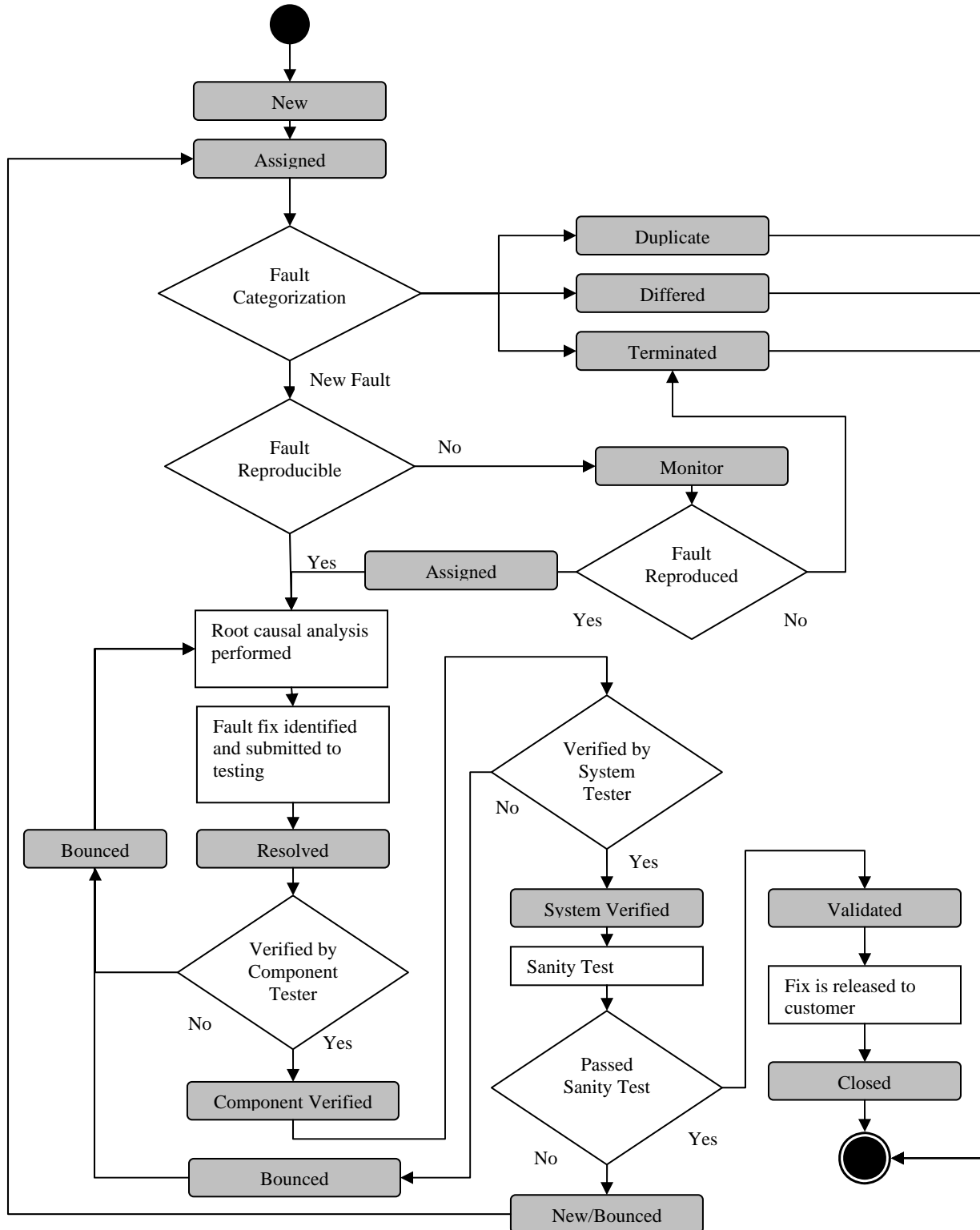
After receiving the *User Acceptance Test (UAT)* approval e-mail from a *Release Manager*, the *Software Engineer* will set the faults for this release to **Closed**.

3.9 **Deployment**

This is a good point for an approval board to review the release prior to deployment in production.

3.10 Fault State Flow Chart Diagram

To view the Statechart, you must have Microsoft Word in *Print Layout* mode. To do this, select from the menu bar View/Print Layout.



3.11 Fault prioritization

Faults are prioritized by *Impact*. Any fault that has an *Impact* of '1' or '2' is termed above-the-line. Any fault that has an *Impact* of '3', '4' or '5' is termed below-the-line. Above the line faults are resolved first with faults of *Impact* level '1' addressed before *Impact* Level '2'. After the above-the-line faults are resolved, the focus should switch to below-the-line faults using the same ranked order.

Severity is closely related to *Impact*, Normally *Severity* A through B carries an *Impact* of 1 through 2 and *Severity* C through D carries an *Impact* of 3 through 5.

It is the *Fault Initiators* responsibility to set both *Severity* and *Impact* levels. If a Software Engineer does not agree with these levels, both the Fault Initiator and the Software Engineer must negotiate a level that is appropriate using the guidelines set in Appendix A of this document. If neither side can agree on the levels, Management will make a final assignment to these fields.

4.0 Termination verses monitoring a fault

Faults that can not be reasonably reproduced by a *Software Engineer*, the engineer can request the *Fault Initiator* to move the fault to *Terminate*. If the *Fault Initiator* feels compelled that the fault is real and not a mistake, the *Software Engineer* can propose that the fault be placed in a Monitor state with an agreed *Termination Date*. If the fault is not reported from the field or reproduced during in-house testing by the agreed *Termination Date*, The *Software Engineer* should once again request permission to *Terminate* the fault.

Generic Fault Process (GFP)

Glossary:

Term	Definition
Fault	A feature or requirement that is not working as described in the requirements.
Enhancement	A feature that was not in the original requirements. Any additional features that were not in the previous official release are considered enhancements.

Appendix A – Field Descriptions

Field Name	Values	Responsibility
Open date	The date that this fault was entered.	Automatic
Status	Please set this field to <i>Assigned</i> if you selected a name in the <i>Assigned to</i> field. For unassigned records, leave this field to <i>New</i> . Values and authorized changes are listed in Appendix B of this document.	Current Fault Owner
Category	<ul style="list-style-type: none"> Fault - a feature that is not working correctly or a missing feature listed in a requirement. Enhancement - a feature that has not been implemented and was not listed in a formal requirement. 	Fault Initiator
Consistency	Set this field to reflect a <i>Consistent</i> or <i>Intermitted</i> occurrence of the fault.	Fault Initiator
Customer	The customer that is or will experience the fault.	Fault Initiator
Description	A full description of the fault identified in the Title field.	Fault Initiator
Found Release	The software version number that the fault was found in.	Fault Initiator
Impact	<ul style="list-style-type: none"> Level 1-5 Level 1: the fault will result in extreme customer dissatisfaction. Level 2: the fault will probably cause customer dissatisfaction. These two impact levels must be fixed prior to release. Level 3: the fault may cause customer dissatisfaction and should be fixed as soon as possible. Level 4: the fault will probably be noticed by customers and should be fixed in the next convenient release. Level 5: the fault is unlikely to be noticed by customers or to cause customer dissatisfaction. 	Fault Initiator
Monitor Termination	This field is used to set an agreed date that this fault will be terminated on if it is not reproducible.	Fault Initiator
Project	Identify the correct project that the software is covered under.	Fault Initiator
Severity	<p>This field is very important in identifying fault priority. The value should be set based on the extensive guidelines listed below:</p> <ul style="list-style-type: none"> Severity Level A: (also classified as critical) this level is meant to represent faults that result in an unusable system, sub-system, product, or critical features from the customer’s perspective. <ul style="list-style-type: none"> Faults that have a potential for personal harm to a user. Faults that eliminate most functionality of a critical feature or subsystem. Faults that prevent the primary functionality of the communications system (talk-group call in a voice system, data in a data system, interconnect in a primarily interconnect voice system, etc.). Faults that cause irreparable loss of system data (infrastructure configuration management, or accounting management). Fault that keep the system from meeting published standards / performance specifications, including regulatory and safety standards. Faults that allow breach of system / product security. Faults that prevent Abanco or a customer from maintaining the system, where such maintenance would have prevented a Severity A fault. (i.e., the maintenance is part of a work-around) Faults that default secure communications to clear communications. High severity faults where there are no acceptable work-arounds. Severity Level B: (also classified as major) This level is meant to 	Fault Initiator

Generic Fault Process (GFP)

Field Name	Values	Responsibility
	<p>represent faults that limit a customer's normal use of the system, sub-system, product, or major non-critical features from a customer's perspective. Typical examples would include:</p> <ul style="list-style-type: none"> ○ Faults that significantly inconvenience customer communications. ○ Faults that cause loss of most of the functionality of a non-critical feature or subsystem, and there are no work-arounds. ○ Severity "A" faults that have a short-term work-around with a commitment and viable plan for resolution. <ul style="list-style-type: none"> ● Severity Level C: (also classified as annoying) this level is meant to represent faults that annoy the customer, but don't preclude full use of the system, sub-system, product, or critical features from a customer's perspective. Typical examples would include: <ul style="list-style-type: none"> ○ Faults that don't affect the operation of the product/system, however are visible to the user. These faults tend to be annoying for the customer. ○ Faults resulting in minor functions or features being inoperable, unsupported, or unreliable. ● Severity Level D: (also classified as cosmetic) this level represents faults that the customer is unlikely to notice (cosmetic). Typical examples would include: <ul style="list-style-type: none"> ○ Faults resulting in minor functions or features being unsupported or unreliable in ways the customer will not notice. ○ Non-critical user interface / documentation errors. ○ 3. Faults that have no impact in how the customer perceives the system to work. 	
Steps to reproduce	Enter detailed step-by-step instructions on how to reproduce the issue.	Fault Initiator
Title	A short description of the fault.	Fault Initiator
Phase Found	<p>If the fault is known to be in a released version of software, this field is set to <i>Release</i> regardless of where it was found.</p> <ul style="list-style-type: none"> ● <i>Requirements</i> – Found when defining the requirements or a requirements FTR. ● <i>Design</i> – Found when reviewing a system design or a design FTR. ● <i>Implementation</i> – The fault was identified during development (coding) ● <i>Unit Test</i> – The fault was found by the engineer during a unit test phase. ● <i>Component Test</i> – The fault was found by a tester while testing at a component level. ● <i>Integration Test</i> – The fault was found by a tester while testing the interaction between components such as end-to-end testing. ● <i>Staging</i> – The fault was found while testing on a staging system ● <i>Release</i> – The fault was found in software that has been released to one or more customers. If the discovered fault is in software that has been released to a customer; this field must be set to 'Release'. If the fault is discovered in pre-released software, but know also to be in a released version, this field must be set to 'Release'. 	Fault Initiator – for released software Software Engineer - for pre-released software.
Assigned to	If you already know who the person that this fault is assigned to, please select the name and set the <i>Status</i> field to <i>Assigned</i> .	Fault Initiator or FTS Admin
Workaround	Any known steps to workaround the issue or steps that can minimize the impact of the fault.	Fault Initiator or Software Engineer
Actual Date	After the work is completed on the fault, enter the actual date that this issue was resolved	Software Engineer
Actual Hours	After the work is completed on the fault, enter the actual hours that it took to get this issue resolved.	Software Engineer
Add related issue	If this fault was a duplicate of a fault already in the system, place the	Software Engineer

Generic Fault Process (GFP)

Field Name	Values	Responsibility
	original fault number here before assigning a <i>Status of Duplicate</i>	
Build Label	The Configuration Management Label that contains this fix	Software Engineer
Closed date	The date that this fault was officially closed. This must be set to the date that the customer accepts the resolution. Accuracy is important for fault metrics.	Software Engineer
Fault Type	After a root-cause is identified, the most suitable selection must be chosen.	Software Engineer
Estimated Date	Before the work is started on the fault, enter the date that this issue will be resolved.	Software Engineer
Estimated Hours	Before the work is started on the fault, enter the estimated hours that it will take to get this issue resolved.	Software Engineer
Fix Release	The version number of the software that contains this fix	Software Engineer
Phase Sourced	At what phase did this fault get injected: <ul style="list-style-type: none"> • Requirements – Injected when defining the requirements. • Design – Injected when translating requirements into the design. • Implementation – Injected during implementation (coding) 	Software Engineer
Preventative Measure	Once a root-cause is known for this fault, a preventative measure must be selected from this list prior to setting the fault to resolved	Software Engineer
Progress history	Enter a progress history such as current status updates in the following format with the most current date on top: ###/###/### - History Author Name - Description ###/###/### - History Author Name - Description ###/###/### - History Author Name - Description	Software Engineer
Resolution	A detailed description of what was changed to correct the fault.	Software Engineer
Root-Cause	A detailed description of the root-cause of this fault must be entered by the software engineer prior to setting the status to resolve.	Software Engineer
Software Engineer	The Software Engineer that will own this fault and take responsibility for its resolution will assign their name to this field.	Software Engineer
Written Test Case	This should be checked if there is a written test case that would catch this fault.	Tester
Original Fault ID	In the case where the status is set to Duplicate, this field should contain the ID number for the first recorded instance of this fault.	Software Engineer

Appendix B – Fault Status Values and Authorized Changes

Status	Definition	Who can authorize this status
New	First Entered, unassigned	Anyone
Assigned	Assigned for analysis	Fault Initiator FTS Administrator
Duplicate	Another fault was already entered into the system; the <i>Original Fault ID</i> should be set to the first recorded instance of the original fault.	Software Engineer
Deferred	Resolve in a future release	Management
Terminated	Determined not to be a fault	If it's not a fault the Software Engineer may terminate if in agreement with the Fault Initiator. If it is an actual fault then only Management can authorize a status of Terminated.
Monitor	Fault is currently being monitored	Software Engineer
Resolved	Issue is resolved and waiting for test verification	Software Engineer
Bounced	After testing, the fault was found not to be fixed.	Testers
Component Verified	Verified by a Component Tester	Component Tester
System Verified	Verified by a System Tester	System Tester
Validated	Faults have been validated through Sanity Testing.	Release Manager
Closed	Fault is closed	Software Engineer