

CS1632: Defects

Wonsun Ahn

Defects and Enhancements

Importance of understanding implicit requirements

Defects, Defined

- When *observed behavior* \neq *expected behavior*
- How can we know *expected behavior*?
 - One word: *Requirements*

A Defect Must Lead to Defective Behavior

```
// Requirement: Code shall always print "cat"
// Is there a defect in this code?
int k = 4;
if (k > 100) {
    System.out.println("centipede");
} else {
    System.out.println("cat");
}
```

- It's not OK to have ugly code even if it does not impact behavior
- But it is still not a defect if it does not cause defective behavior

Defects vs Enhancements

- Main job of software QA team is to find and report *defects*
- But a QA team is also expected to find and suggest *enhancements*
- What's in common between *defects* and *enhancements*?
 - Both involve modifications to software that can improve software quality
- What's the difference?
 - *Defect*: A violation of requirements
 - *Enhancement*: A proposed improvement to existing requirements

Differentiating Defects vs Enhancements

- Differentiating is important: often has legal implications
 - *Defect*: Developer must often pay customer for any damages
 - *Enhancement*: Customer may pay developer for the added improvement
- Differentiating sounds easy enough!
 - If software violates pre-existing requirements → defect
 - If software doesn't violate pre-existing requirements → enhancement
- But sometimes differentiating the two is surprisingly hard
 - Mainly due to *implicit requirements*

Explicit and Implicit Requirements

1. *Explicit requirement*

- A requirement that is documented on the Software Requirements Specification (SRS)
- Includes both functional and non-functional requirements (quality attributes)

2. *Implicit requirement*

- A requirement not documented in the SRS but is still expected in the application domain
 - E.g., Databases should never store passwords unencrypted
 - E.g., Flight software should never have a single point of failure
- Even if software does not violate SRS, if it violates implicit requirements
→ Still a defect!

Case 1: Is this a Defect?

- Observed behavior: Program loses data on system power loss.
 - Suppose SRS didn't specify behavior on power loss explicitly.
- Is this a defect?
 - Depends on whether there is an implicit requirement that was violated.
- If application domain is a database: **defect**
 - Implicit requirement: no data loss shall happen in any circumstance.
- If application domain is a game of solitaire: **not a defect**
 - No expectation that game will be saved on a power loss.

Case 2: Is this a Defect?

- Observed behavior: Program becomes unresponsive for 10 seconds.
 - Suppose SRS didn't specify performance expectations explicitly.
- Is this a defect?
- If application domain is a real-time game: **defect**
 - Implicit requirement: a real-time game must be responsive at all times.
- If application domain is a batch file copy tool : **not a defect**
 - No expectation that app will be fully responsive while the copy is happening.
- ☛ the answer depends in large part on the application domain!

Understand Implicit Requirements

- You need to understand implicit requirements that come with domain
 - You may need to do some research on prior literature on the subject matter
 - You may need to talk to a subject matter expert (SME) if you don't understand
 - Sometimes, the best SME is your customer
- Communication!
- Communication!
- Communication!

Reporting Defects

How to report defects?

Varies based on company/project, but there are some common items that go into a bug report.

A Typical Bug Report Template

- SUMMARY
- DESCRIPTION
- REPRODUCTION STEPS
- EXPECTED BEHAVIOR
- OBSERVED BEHAVIOR
- IMPACT
- SEVERITY
- PRIORITY

Summary - *succinct description of problem*

- A one sentence description of bug
- Examples:
 - Number of widgets in cart not refreshed when removing 2 widgets
 - If time zone is changed during execution, idle tasks never wake up
 - CPU pegs at 100% after the addition of two nodes to the list
 - Title does not display after clicking "Next"
 - Page title is "Alll Entries", should be "All Entries"

DESCRIPTION - *details of problem*

- A detailed description of everything the tester discovered
- Examples:
 - *Summary*: Number of widgets not refreshed when removing 2 widgets
 - *Description*: If 2 widgets are removed at once from the shopping cart, the number of widgets is not changed from the initial value.
Removing 3, 4, and 5 widgets resulted in the same defective behavior.
The value is updated correctly if the widgets are removed one at a time.
- Be careful not to overgeneralize
 - Describing the contours of the issue accurately helps developer

REPRODUCTION STEPS

- *Preconditions + Steps to Reproduce Defect*

- First, list *preconditions* (if there are any)
 - If defect found by test case, identical to test case preconditions
 - If not, should have the same level of detail
- Next, enumerate *steps* required to reproduce defect
 - Again, will look very similar to test case *execution steps*
- It's usually better to err on the side of over-specifying
 - If developer cannot reproduce the defect, it cannot be fixed

REPRODUCTION STEPS

- BAD: Put some things in the shopping cart. Take a couple things out.
- GOOD:
 - Precondition: Shopping cart is empty.
 - 1. Add 3 widgets to shopping cart one by one.
 - 2. Remove 2 widgets from shopping cart at once.

REPRODUCTION STEPS

- Example given in Mozilla Firefox web browser project:
https://developer.mozilla.org/en-US/docs/Mozilla/QA/Bug_writing_guidelines#Writing_precise_steps_to_reproduce
- BAD: Open Gmail in another window
- GOOD:
(Any preconditions. E.g. settings in Firefox configuration relevant to defect)
 1. Start Firefox by clicking on the desktop icon
 2. Press Cmd+N (or Ctrl+N for Windows users) to open a new browser window
 3. Paste <https://mail.google.com/> in the address bar and press Enter

EXPECTED AND OBSERVED BEHAVIOR

- *EXPECTED BEHAVIOR*: What you expected according to requirements.
 - Why is it important that this is part of the defect report?
 - ☛ Describing expectations tells why observed behavior is deemed defective
 - If defect found through a test case, may be identical to *postconditions*
- *OBSERVED BEHAVIOR*: What you **ACTUALLY** saw.
 - May be only chance dev sees observed behavior, if bug is not reproduced
 - Be as precise as possible
 - You may even consider attaching a **screenshot** of what you saw

Screenshots are a no-no for Expected Behavior

- Suppose you had the following in a defect report.
 - Expected Behavior: **Result is: 1** is displayed.
 - Observed Behavior: **Value is: 100** is displayed.
- Can you figure out what the defect is?
 - It could be that the numerical value is 100 instead of 1.
 - It could be that the background is red instead of blue.
 - It could be that the wording is “Value is” instead of “Result is”.

Screenshots are a no-no for Expected Behavior

- The following report makes it crystal clear what the defect is.
 - Expected Behavior: The value 1 is displayed in white letters on blue background.
 - Observed Behavior: **Value is: 100** is displayed.
- What are the defects?
 - That the numerical value is 100 instead of 1. ✓
 - That the background is red instead of blue. ✓
 - *Not* that the wording begins with “Value is”. ✗

IMPACT – impact to various stakeholders

- BAD: Everyone will hate this because everything is wrong!
- GOOD: An incorrect number of widgets in the shopping cart will lead **customers** to purchase more than they want. This will lead to an avalanche of customer returns adding pressure to **customer service**.

SEVERITY – how severe is the problem?

- Severity is a combination of several factors:
 1. How bad is the problem when it does occur?
 2. How often does it occur?
 3. Is there a workaround?

LEVELS OF SEVERITY (Bugzilla)

- CRITICAL
- MAJOR
- NORMAL
- MINOR
- TRIVIAL

PRIORITY – ordering of defect resolution

- *Priority*: ordering in which defects should be worked on first
- Typically, a higher severity bug will be given higher priority
 - But not always; other considerations may take precedence

Tracking Defects

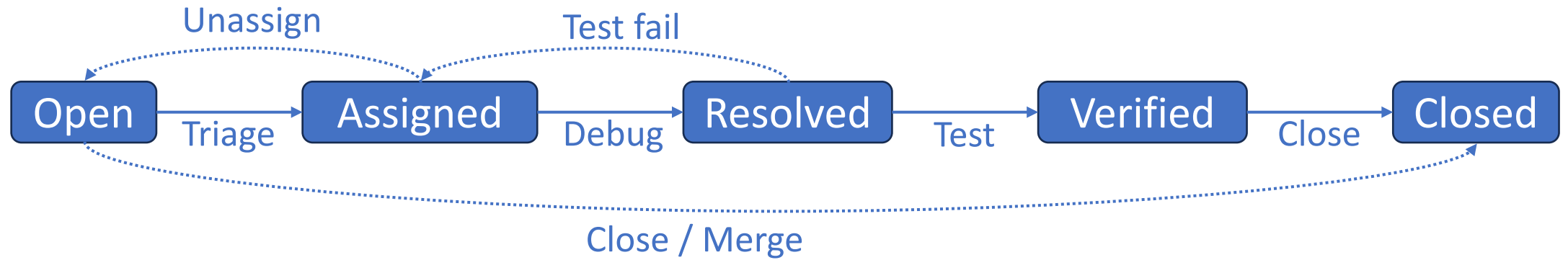
Tracking Defects

- Once defects are reported they need to be tracked
 - To make sure that they are fixed in a timely manner
 - To verify the fix corrects the defect and doesn't cause regression
- Must be done in a systematic way
 - Often hundreds of bugs at various stages of resolution
 - Often done with the help of a *bug tracking system*

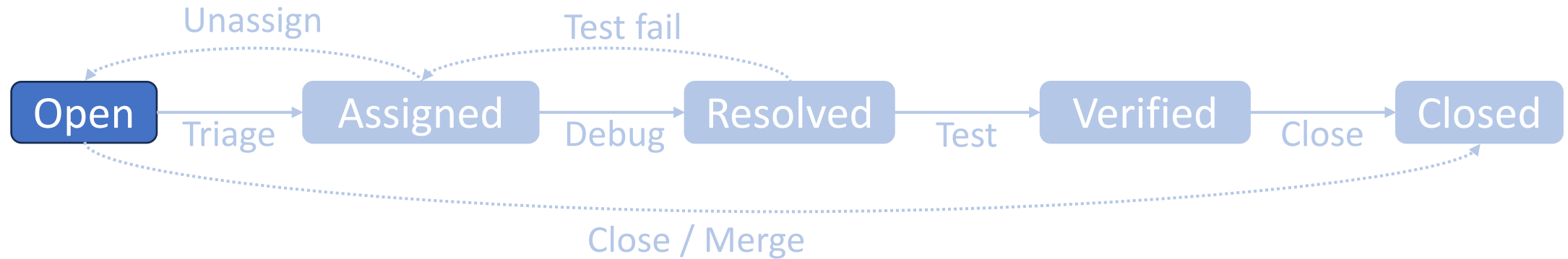
Tracking Defects

- In order to track, defects should have the following info:
 - Identifier: Usually numbered, not named
 - Source: Associated test case, if applicable
 - Version of software found
 - Version of software fixed, if applicable

Lifecycle of a defect

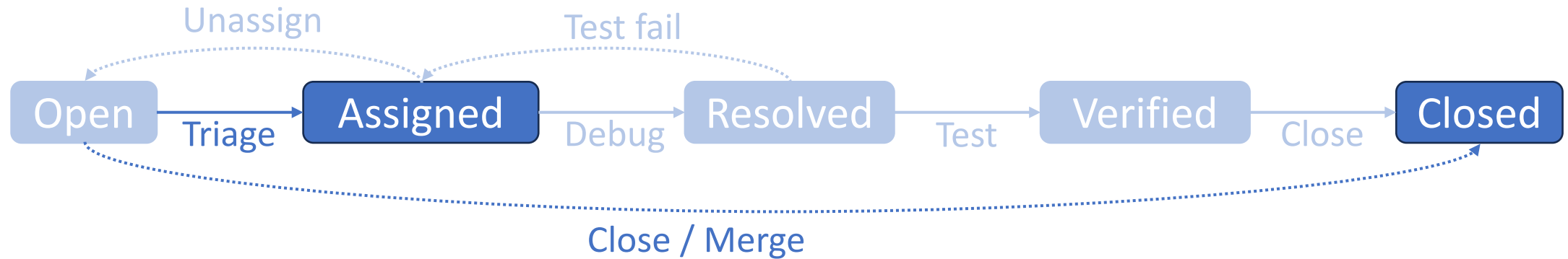


Lifecycle of a defect



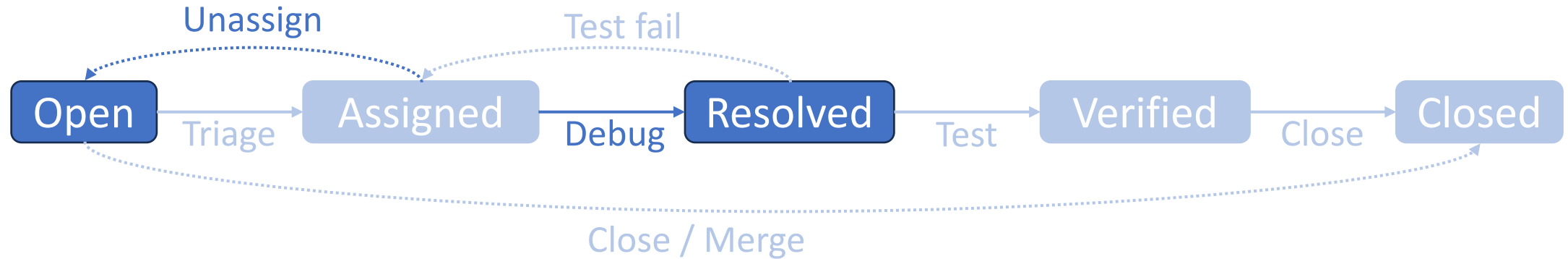
- Defect ticket is first entered into the defect tracking system
 - Either by software QA team or end-user

Lifecycle of a defect



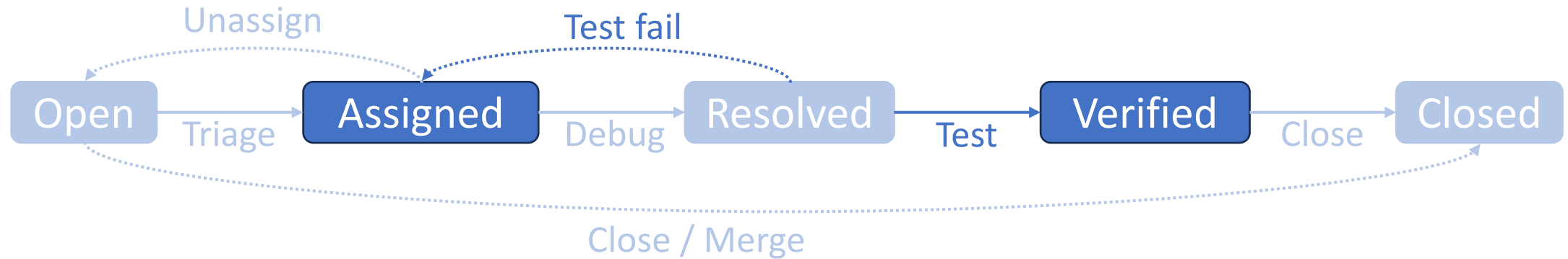
- **Triage** (a.k.a. Defect Review): stakeholders meet to determine
 1. If the defect is not a valid defect (if not, then Close)
 2. If the defect is a duplicate (if so, then Merge with previous ticket)
 3. Final severity and priority
 4. Assignment of defect to a particular developer

Lifecycle of a defect



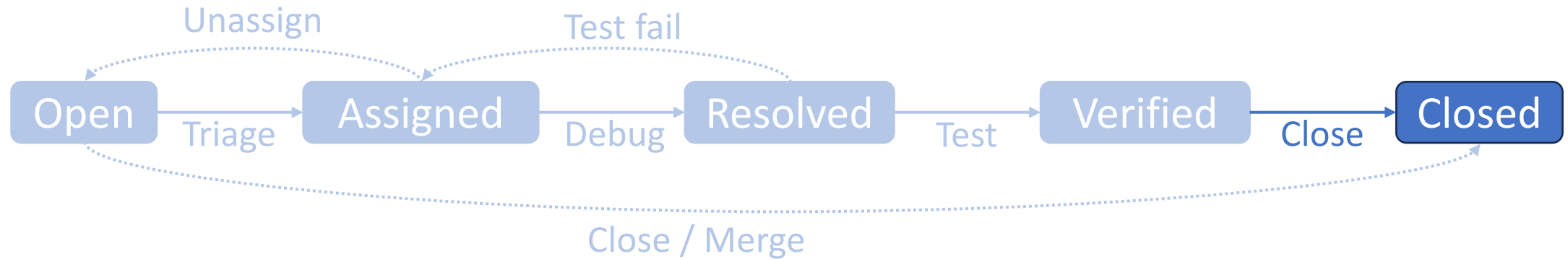
- Assigned developer debugs the defect
- If not correct developer, unassign and reassign to different developer

Lifecycle of a defect



- Software QA team performs various tests to verify fix
 - Regression test, Performance test, Security test, etc.
- If any test fails, reassign ticket back to developer

Lifecycle of a defect



- Close the defect ticket after verification is done
- Closed ticket remains in the system for future reference
- (Optionally) add test case for defect in regression test suite

Make the defect lifecycle transparent to users

- It is normal for non-trivial software to ship with defects
- Defects should be advertised, not hidden
 - Gives users confidence that defects are getting addressed
 - Allows users to work around defects
 - Allows users to participate in defect resolution

Now Please Read Textbook Chapter 9