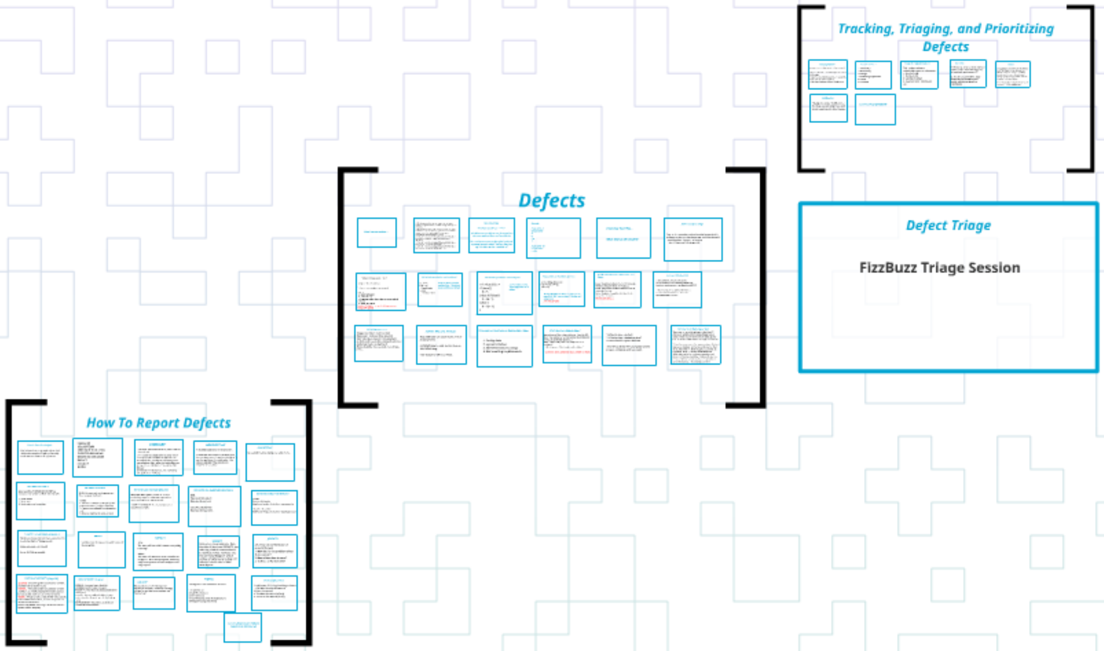
Lecture 5 - Defect Reporting



Lecture 5 - Defect Reporting

Tracking, Triaging, and Prioritizing Defects

Defects

Defect Triage

FizzBuzz Triage Session

How To Report Defects

Defects

First, some review....

1. The function shall accept one argument, a positive integer. If argument is not a positive integer, do not print anything.
2. The function shall print the numbers from 1 to n, where n is the argument to the function, separated by newlines appropriate to the system, with the exceptions enumerated in requirements 3, 4, and 5.
3. If the current number being printed is evenly divisible by 3 and 5, instead of printing the number, print "FizzBuzz".
4. If the current number being printed is evenly divisible by 3, but not evenly divisible by 5, print "Fizz".
5. If the current number being printed is evenly divisible by 5, but not evenly divisible by 3, print "Buzz".

Split into groups.

Develop a test plan for FizzBuzz.

We will execute one of your test plans against a reference implementation and note defects.

For conciseness, you can just give input and expected output values. You can also just say "the nth number should be x."

Example:

Input value: 4
Output value:
1
2
Fizz
4

Input value: 12
Output value:
... Fizz

Executing Test Plan...

What defects did we find?

What is a defect, really?

Bug, n.: An unwanted and unintended property of a program or piece of hardware, esp. one that causes it to malfunction. Antonym of feature.
-Eric S. Raymond, *The Jargon File*

Where do bugs come from?

1. Gaps or mistakes in code

2. Gaps or ambiguity in requirements

3. Other
  a. Compiler broken
  b. Bad hardware
  c. Improper safety checks during context switch in kernel
  d. Mistakes in input

Number 3 is by far the smallest of your worries. Focus on #1 and #2.

Not all coding mistakes result in defects!

```
int k = 7;
if (k > 3) {
  // good code
} else {
  // DEFECT HERE
{
```

Only the good code will execute here. This is bad code, but it's not a defect.

Not all coding mistakes result in defects!

```
// k should be 7
if (true) {
  k = 7;
} else if (false) {
  k = (8 - 1);
} else {
  k = (6 + 1)
}
```

k is 7 no matter what. This is ugly code, not a defect.

A defect must be observable by a user.

```
foo = addTogether(a, b);
if (a == 5 && b == 5) {
  foo = 10;
}
```

Assume addTogether does not work for the case (5,5). This is not a defect! The user will never know.
It is still bad code.

If it does not match the requirement, it's a defect.

A user may have expected an error message when a string is entered for FizzBuzz.
A user may not have cared whether "fizz" is capitalized or not.
A user may want negative numbers to be treated as positives.
They're still defects.

Defects vs Enhancements

If the software does not meet the requirements, or is unstable (which is an implied requirement), then there's a DEFECT.

If the user wants to ADD or MODIFY a requirement/feature/etc, that's an ENHANCEMENT request.

Not all bugs are severe

1. Images are sized 1 pixel too small
2. Delays are 1 ns longer than required
3. Upon shutdown, typo in final statement
4. Seldom-used feature does not work correctly
5. Background color is slightly off
6. There should be three periods in an ellipsis, not two..

Software WILL ship with bugs

Hopefully, you can catch many of them ahead of time.

A KNOWN bug is much better than an UNKNOWN bug.

Your customer will thank you.

I'd recommend you focus on finding bigger bugs.

1. Faulty data
2. System crashes
3. Extreme resource usage
4. Not meeting requirements

What about ambiguous bugs?

Agent Johnson: This is Agent Johnson from the FBI. Be on the lookout for a 1936 maroon Stutz Bearcat!
(A 1936 Stutz Bearcat goes by...)
Chief Wiggum: Ehh, that was really more of a burgundy.
  -The Simpsons, "The Trouble with Trillions"

Communication, communication, communication.

1. What makes a defect?
2. What makes a defect serious?
3. How should I report defects?

Answers to these will vary based on the project, company, and test team.

Side Note on the Etymology of "Bug"

There was an actual bug found in the Mark II computer, which Admiral Grace Hopper wrote about. However, the term is much older, going back at least to 19th-century telegraph operators.

"It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise — this thing gives out and [it is] then that "Bugs" — as such little faults and difficulties are called — show themselves and months of intense watching, study and labor are requisite before commercial success or failure is certainly reached." -Thomas Edison, 1878

First, some review....

1. The function shall accept one argument, a positive integer. If argument is not a positive integer, do not print anything.
2. The function shall print the numbers from 1 to n, where n is the argument to the function, separated by newlines appropriate to the system, with the exceptions enumerated in requirements 3, 4, and 5.
3. If the current number being printed is evenly divisible by 3 and 5, instead of printing the number, print "FizzBuzz".
4. If the current number being printed is evenly divisible by 3, but not evenly divisible by 5, print "Fizz".
5. If the current number being printed is evenly divisible by 5, but not evenly divisible by 3, print "Buzz".

Split into groups.

Develop a test plan for FizzBuzz.

We will execute one of your test plans against a reference implementation and note defects.

For conciseness, you can just give input and expected output values. You can also just say "the nth number should be x."

Example:

Input value: 4
Output value:
1
2
Fizz
4

Input value: 12
Output value:
... Fizz

Executing Test Plan…

What defects did we find?

What is a defect, really?

Bug, n.: An unwanted and unintended property of a program or piece of hardware, esp. one that causes it to malfunction. Antonym of feature.

-Eric S. Raymond, *The Jargon File*

Where do bugs come from?

1. Gaps or mistakes in code

2. Gaps or ambiguity in requirements

3. Other
   a. Compiler broken
   b. Bad hardware
   c. Improper safety checks during context switch in kernel
   d. Mistakes in input

Number 3 is by far the smallest of your worries. Focus on #1 and #2.

Not all coding mistakes result in defects!

```
int k = 7;
if (k > 3) {
  // good code
} else {
   // DEFECT HERE
{
```

Only the good code will execute here. This is bad code, but it's not a defect.

Not all coding mistakes result in defects!

```
// k should be 7
if (true) {
  k = 7;
} else if (false) {
  k = (8 - 1);
} else {
  k = (6 + 1)
}
```

k is 7 no matter what. This is ugly code, not a defect.

A defect must be observable by a user.

```
foo = addTogether(a, b);
if (a == 5 && b == 5) {
  foo = 10;
}
```

Assume addTogether does not work for the case (5,5). This is not a defect! The user will never know.

It is still bad code.

If it does not match the requirement, it's a defect.

A user may have expected an error message when a string is entered for FizzBuzz.
A user may not have cared whether "fizz" is capitalized or not.
A user may want negative numbers to be treated as positives.
They're still defects.

Defects vs Enhancements

If the software does not meet the requirements, or is unstable (which is an implied requirement), then there's a DEFECT.

If the user wants to ADD or MODIFY a requirement/feature/etc, that's an ENHANCEMENT request.

Not all bugs are severe

1. Images are sized 1 pixel too small
2. Delays are 1 ns longer than required
3. Upon shutdown, typo in final statement
4. Seldom-used feature does not work correctly
5. Background color is slightly off
6. There should be three periods in an ellipsis, not two..

Software WILL ship with bugs

Hopefully, you can catch many of them ahead of time.

A KNOWN bug is much better than an UNKNOWN bug.

Your customer will thank you.

I'd recommend you focus on finding bigger bugs.

1. Faulty data
2. System crashes
3. Extreme resource usage
4. Not meeting requirements

What about ambiguous bugs?

Agent Johnson: This is Agent Johnson from the FBI. Be on the lookout for a 1936 maroon Stutz Bearcat!
(A 1936 Stutz Bearcat goes by...)
Chief Wiggum: Ehh, that was really more of a burgundy.
-The Simpsons, "The Trouble with Trillions"

Communication, communication, communication.

1. What makes a defect?
2. What makes a defect serious?
3. How should I report defects?

Answers to these will vary based on the project, company, and test team.

Side Note on the Etymology of "Bug"

There was an actual bug found in the Mark II computer, which Admiral Grace Hopper wrote about. However, the term is much older, going back at least to 19th-century telegraph operators.

"It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise — this thing gives out and [it is] then that "Bugs" — as such little faults and difficulties are called — show themselves and months of intense watching, study and labor are requisite before commercial success or failure is certainly reached." -Thomas Edison, 1878

How To Report Defects

How to Describe a Defect

Varies based on company/project, but this is the template I have personally used over the course of my career.

SUMMARY
DESCRIPTION
REPRODUCTION STEPS
EXPECTED BEHAVIOR
OBSERVED BEHAVIOR
IMPACT
SEVERITY
NOTES

SUMMARY

A succinct (one-sentence or so) description of the problem.

1. Title does not display after clicking "Next"
2. CPU pegs after addition of any two cells
3. Total number of widgets in shopping cart not refreshed after removal of more than one
4. Page title is "Alll Entries", should be "All Entries"
5. If timezone is changed during execution, idle tasks never wake up

DESCRIPTION

A detailed explanation of the problem.

If more than one widget is removed from the shopping cart, the number of widgets is not changed from the initial value. This value is updated if the widgets are removed one at a time.

DESCRIPTION

Be careful not to overgeneralize here.

REPRODUCTION STEPS

Specify an EXACT SEQUENCE OF STEPS to reproduce the problem. Make sure you give:

1. Exact values
2. Exact steps
3. Exact manner of execution

REPRODUCTION STEPS

BAD: Put some stuff in shopping cart. Take a couple stuffs out.

GOOD:
1. Add three widgets to shopping cart
2. Note number of widgets listed is 3
3. Remove two widgets from shopping cart
3. Observe number of widgets listed

EXPECTED AND OBSERVED BEHAVIOR

EXPECTED BEHAVIOR: This should note, as precisely as possible, what you expected to see according to the requirements.

OBSERVED BEHAVIOR: This should note what you ACTUALLY saw.

EXPECTED VS OBSERVED BEHAVIOR

BAD:
Expected Behavior:
Number is correct.

Observed Behavior:
Number is incorrect.

EXPECTED VS OBSERVED BEHAVIOR

GOOD:
Expected Behavior:
Number of widgets listed in shopping cart is 1.

Observed Behavior:
Number of widgets listed in shopping cart is 3.

EXPECTED VS OBSERVED BEHAVIOR

What you saw versus what you expected to see is the CRUX of a bug report.

Make sure you get it right!

Be as PRECISE as possible.

IMPACT

Describe how this would impact a user of the program.

IMPACT

BAD:
The user will hate this because everything is wrong!

GOOD:
The user will see an incorrect number of widgets in their shopping cart, meaning they could purchase fewer widgets than they expect.

SEVERITY

How severe the problem is. Note that this differs from PRIORITY, the ordering of which defects should be worked on first. However, the two are not orthogonal; *ceteris paribus*, a higher-severity bug will take precedence over a lower-severity one.

SEVERITY

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)

BLOCKER - Renders system unusable or unsafe, and there is no workaround.
CRITICAL - Renders system unusable or unsafe and there is a workaround, OR renders system barely usable and there is no workaround.
MAJOR - A very serious problem with the system, but a workaround exists, or a serious problem with no workaround.
(commonly called "red bugs" since these levels appear red in Bugzilla)

LEVELS OF SEVERITY (Bugzilla)

NORMAL - A normal bug. Would be considered a small issue to a user.
MINOR - A minor bug would be considered an annoyance.
TRIVIAL - The user will probably not even notice it unless they are specifically looking for it.
ENHANCEMENT - Not a defect at all, but a request for enhancement.

VARIABILITY

The specific criteria will change from company to company, but the terminology will tend to be pretty commonplace, e.g. "blocker bug."

NOTES

Usually more technical and detailed.

1. Stack traces
2. Log file excerpts
3. Environment
4. Anything that may be helpful to a developer fixing this defect

Tracking Information

Usually part of the bug-tracking software.
1. Software Version discovered
2. Date discovered
3. Test Run discovered (if any)
4. Test Case discovered (if any)

Let's Try Putting Our FizzBuzz Defects Into This Format

How to Describe a Defect

Varies based on company/project, but this is the template I have personally used over the course of my career.

SUMMARY

DESCRIPTION

REPRODUCTION STEPS

EXPECTED BEHAVIOR

OBSERVED BEHAVIOR

IMPACT

SEVERITY

NOTES

SUMMARY

A succinct (one-sentence or so) description of the problem.

1. Title does not display after clicking "Next"
2. CPU pegs after addition of any two cells
3. Total number of widgets in shopping cart not refreshed after removal of more than one
4. Page title is "Alll Entries", should be "All Entries"
5. If timezone is changed during execution, idle tasks never wake up

DESCRIPTION

A detailed explanation of the problem.

If more than one widget is removed from the shopping cart, the number of widgets is not changed from the initial value. This value is updated if the widgets are removed one at a time.

DESCRIPTION

Be careful not to overgeneralize here.

REPRODUCTION STEPS

Specify an EXACT SEQUENCE OF STEPS to reproduce the problem. Make sure you give:

1. Exact values
2. Exact steps
3. Exact manner of execution

REPRODUCTION STEPS

BAD: Put some stuff in shopping cart. Take a couple stuffs out.

GOOD:

1. Add three widgets to shopping cart
2. Note number of widgets listed is 3
3. Remove two widgets from shopping cart
3. Observe number of widgets listed

EXPECTED AND OBSERVED BEHAVIOR

EXPECTED BEHAVIOR: This should note, as precisely as possible, what you expected to see according to the requirements.

OBSERVED BEHAVIOR: This should note what you ACTUALLY saw.

EXPECTED VS OBSERVED BEHAVIOR

BAD:
Expected Behavior:
Number is correct.

Observed Behavior:
Number is incorrect.

EXPECTED VS OBSERVED BEHAVIOR

GOOD:

Expected Behavior:

Number of widgets listed in shopping cart is 1.

Observed Behavior:

Number of widgets listed in shopping cart is 3.

EXPECTED VS OBSERVED BEHAVIOR

What you saw versus what you expected to see is the CRUX of a bug report.

Make sure you get it right!

Be as PRECISE as possible.

IMPACT

Describe how this would impact a user of the program.

IMPACT

BAD:
The user will hate this because everything is wrong!

GOOD:
The user will see an incorrect number of widgets in their shopping cart, meaning they could purchase fewer widgets than they expect.

SEVERITY

How severe the problem is. Note that this differs from PRIORITY, the ordering of which defects should be worked on first. However, the two are not orthogonal; *ceteris paribus*, a higher-severity bug will take precedence over a lower-severity one.

SEVERITY

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)

BLOCKER - Renders system unusable or unsafe, and there is no workaround.
CRITICAL - Renders system unusable or unsafe and there is a workaround, OR renders system barely usable and there is no workaround.
MAJOR - A very serious problem with the system, but a workaround exists, or a serious problem with no workaround.
(commonly called "red bugs" since these levels appear red in Bugzilla)

LEVELS OF SEVERITY (Bugzilla)

NORMAL - A normal bug. Would be considered a small issue to a user.
MINOR - A minor bug would be considered an annoyance.
TRIVIAL - The user will probably not even notice it unless they are specifically looking for it.
ENHANCEMENT - Not a defect at all, but a request for enhancement.

VARIABILITY

The specific criteria will change from company to company, but the terminology will tend to be pretty commonplace, e.g. "blocker bug."

NOTES

Usually more technical and detailed.

1. **Stack traces**
2. **Log file excerpts**
3. **Environment**
4. **Anything that may be helpful to a developer fixing this defect**

Tracking Information

Usually part of the bug-tracking software.

1. Software Version discovered
2. Date discovered
3. Test Run discovered (if any)
4. Test Case discovered (if any)

Let's Try Putting Our FizzBuzz Defects Into This Format

Tracking, Triaging, and Prioritizing Defects

Tracking Defects

Defects are usually numbered, not named.

They should have the following information:
1. Identifier
2. Source - associated test case, if applicable
3. Version of software found
4. Version of software fixed, if applicable

Lifecycle of a Defect

1. Discovery
2. Recording
3. Triage
4. Sub-triage (optional)
5. Fixed
6. Verified

Triage (or "Defect Review")

This is where relevant stakeholders meet to determine:
1. Final severity
2. Final priority
3. Validity of defect
4. Need for more information

etc.

Sub-Triage

For very large projects, there may be a "system triage" and sub-triages, say for each functional group or IPT.

At this point, systems-level triage usually does more filtering and sorting, with less emphasis on prioritization.

Fixing

The developer then works on a fix for the bug. This may be an iterative process, with the developer and tester working hand in hand to ensure that the fix is correct and complete, and does not break other parts of the software.

NOTE: This is where automated test suites and unit tests help IMMENSELY!

Verification

Finally, the tester verifies that the bug was actually fixed and is not causing any other issues.

Let's Try Triaging FizzBuzz!

Tracking Defects

Defects are usually numbered, not named.

They should have the following information:

1. **Identifier**
2. **Source - associated test case, if applicable**
3. **Version of software found**
4. **Version of software fixed, if applicable**

Lifecycle of a Defect

1. Discovery
2. Recording
3. Triage
4. Sub-triage (optional)
5. Fixed
6. Verified

Triage (or "Defect Review")

This is where relevant stakeholders meet to determine:

1. Final severity

2. Final priority

3. Validity of defect

4. Need for more information

etc.

Sub-Triage

For very large projects, there may be a "system triage" and sub-triages, say for each functional group or IPT.

At this point, systems-level triage usually does more filtering and sorting, with less emphasis on prioritization.

Fixing

The developer then works on a fix for the bug. This may be an iterative process, with the developer and tester working hand in hand to ensure that the fix is correct and complete, and does not break other parts of the software.

NOTE: This is where automated test suites and unit tests help IMMENSELY!

Verification

Finally, the tester verifies that the bug was actually fixed and is not causing any other issues.

Let's Try Triaging FizzBuzz!

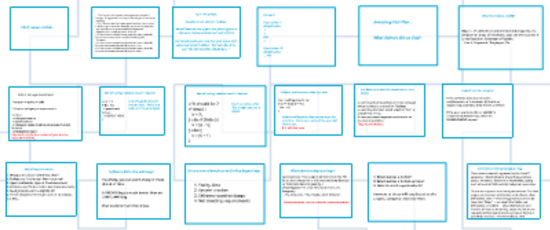
Defect Triage

FizzBuzz Triage Session

Lecture 5 - Defect Reporting

Tracking, Triaging, and Prioritizing Defects

Defects



Defect Triage

FizzBuzz Triage Session

How To Report Defects