# Bug Management

## Software Testing

Lector: Aleksandar Karamfilov
Skype: sanders_kar
Email: alex@pragmatic.bg
Facebook: http://facebook.com/a.karamfilov


PRAGMATIC

www.pragmatic.bg

# Agenda

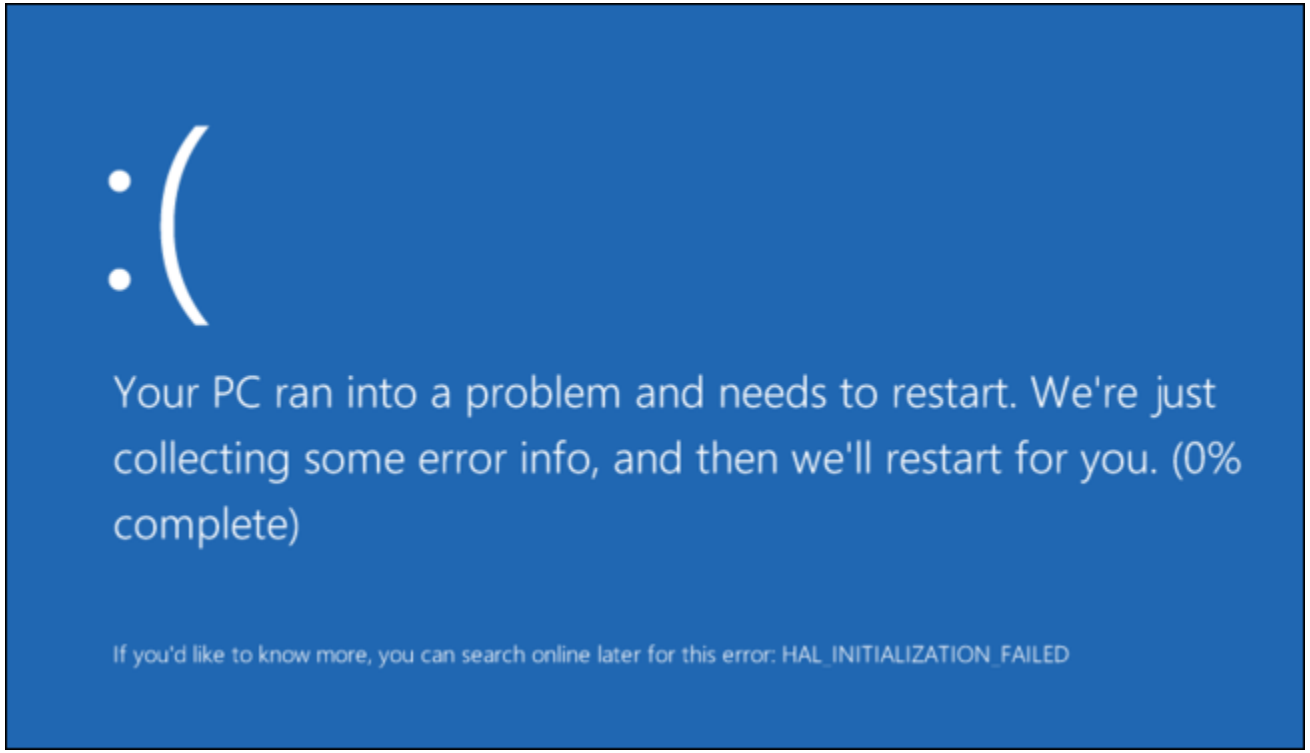
- Software bugs
- Bug tracking systems and workflow
- Bug triage
- Why write bug reports?
- Anatomy of a bug report
- Bug report procedure
- Mistakes
- Using the BTS data
- How no to use data
- Bug bars

# What is a bug?

“A software bug is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result or to behave in unintended ways”

# What is a failure?

“It is the inability of a system or component to perform required function according to its specification”.



# Where do we find them?

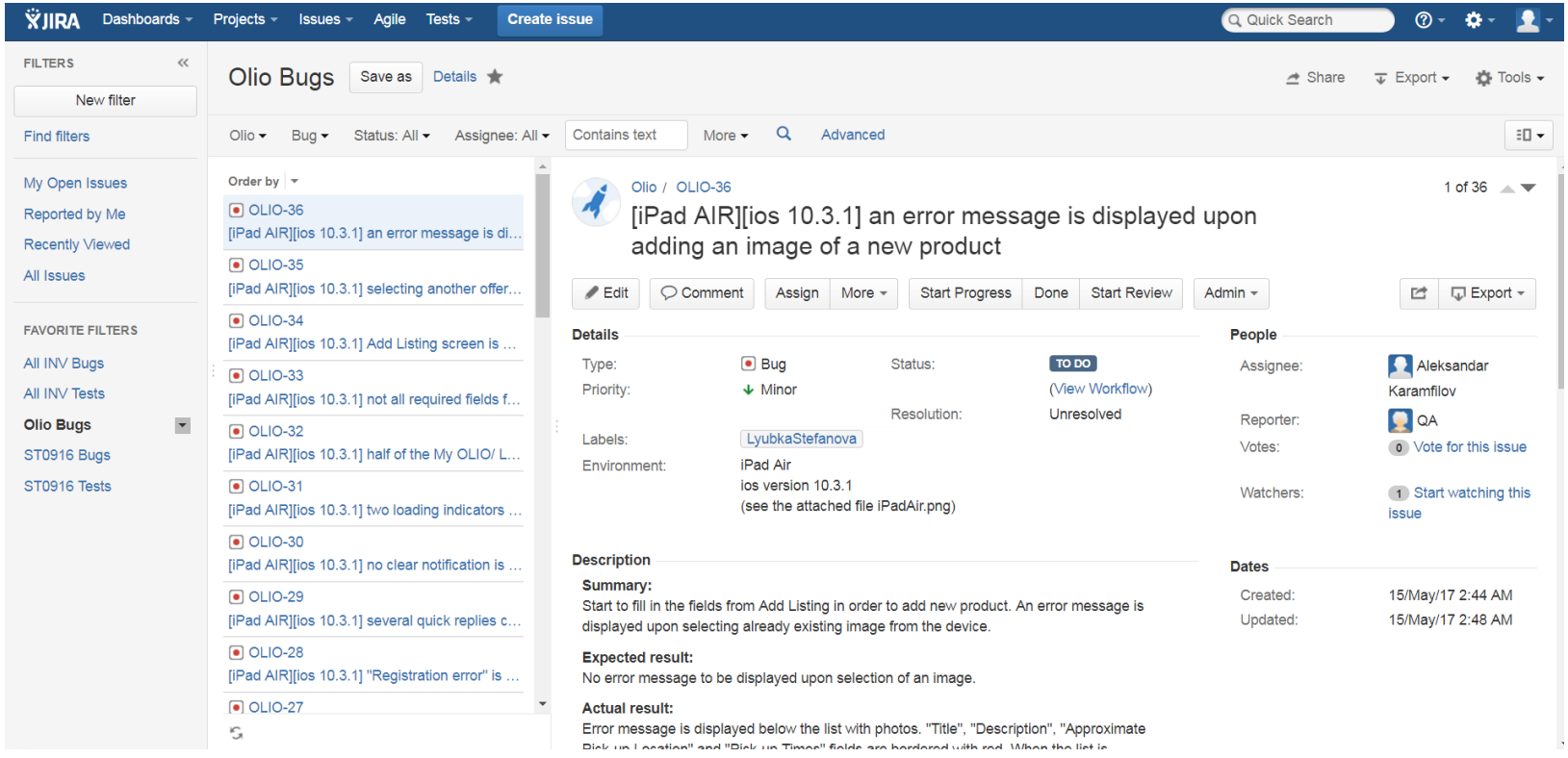
We find failures when we execute tests against System Under Test(SUT).

# Bug Tracking System

How bug tracking systems work?

# Bug Tracking System

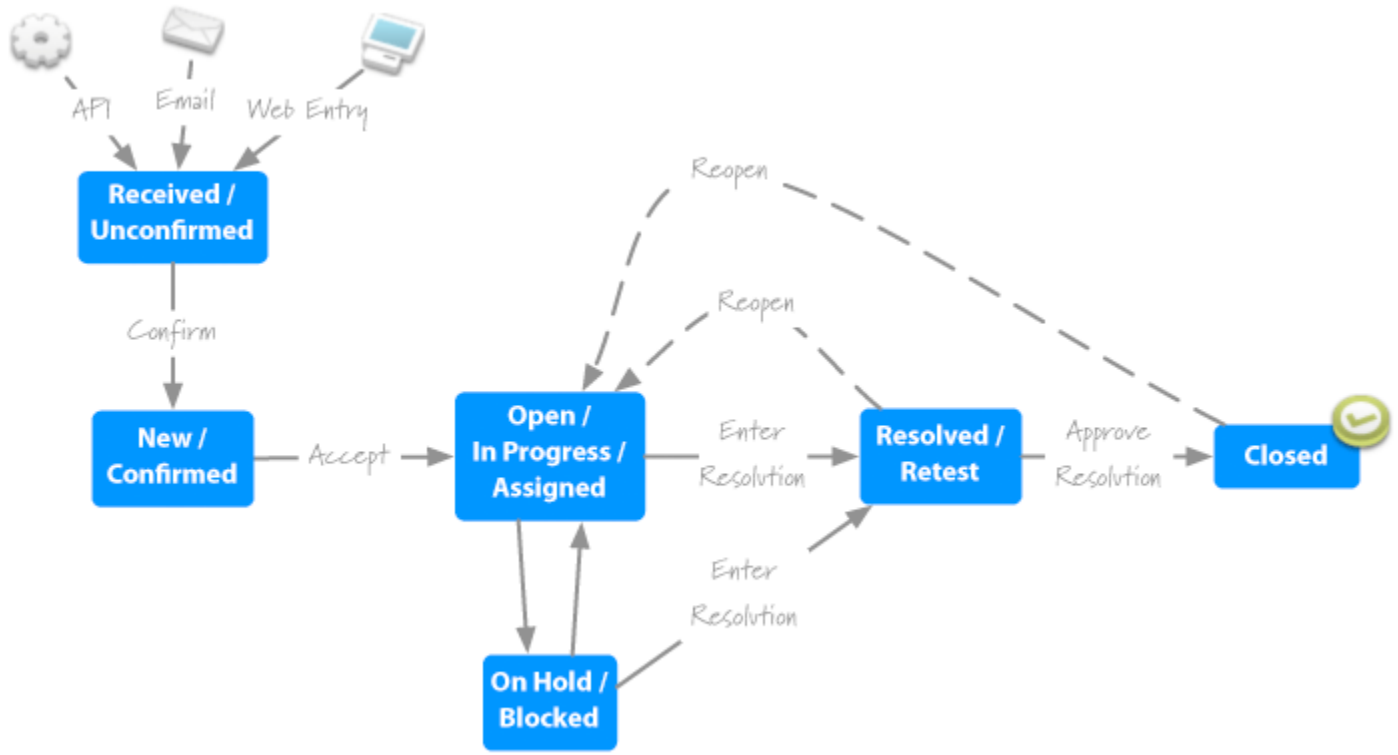
"A bug tracking system or defect tracking system is a software application that keeps track of reported software bugs in software development projects."



# Attributes of BTS

- Ease of use
- Configurability
- Reliability
- Notifications
- Interoperability
- External user access

# Bug Workflow



# Bug Triage

**Bug triage** is a process where issues are screened and prioritized. **Triage** should help ensure we appropriately manage all reported issues - **bugs** as well as improvements and feature requests.

# Bug Reports

How to report bugs properly

# Why write a bug report?

Bug reports are accurate, long-living records of software failures and their resolutions. On many teams, it might seem much easier to take care of bugs in e-mail or hallway conversations, but there are significant advantages to record all bugs in BTS.
One very important reason is the ability to later analyze data for root causes of the bugs and defect removal efficiency (DRE).

# Anatomy of a bug report

- Title
- Description
- Status
- Version number
- Feature area
- Steps to reproduce
- Assignment
- Severity
- Environment
- Customer impact

# Title

The bug title is perhaps the most important(or most used) bit of information in the bug report.
Casual users of the system scan titles to get an idea of the types of bugs in the product or in particular area. Title is also the most searched field in the bug database and is a quick way to find similar bugs.

# Title examples

- Program crash – **too short**
- When running many instances of the program at the same time, a crash occurs in one of the dialogs – **both wordy and vague**
- Program crash in Settings dialog box under low-memory conditions – **specific, accurate and tells enough of the story to understand the bug from the report**

# Description

The description answers all of the questions not obvious from the title. It includes a summary of the bug, customer impact information, and **expected** results versus **actual** results.

**Summary:**
Start to fill in the fields from Add Listing in order to add new product. An error message is displayed upon selecting already existing image from the device.

**Expected result:**
No error message to be displayed upon selection of an image.

**Actual result:**
Error message is displayed below the list with photos. "Title", "Description", "Approximate Pick-up Location" and "Pick-up Times" fields are bordered with red. When the list is dismissed, the error message is changed to another error message.

# Status

Status is either “Active”, “Resolved” or “Closed” and reflects the work that needs to be done with the bug.

# Version number

All bugs should include the version of the software where the bug was found. Knowing the exact version where a bug was found is a huge help in reproducing bugs or verifying fixes.

# Feature area

Most teams require that bugs include the area or subarea of the product where the bug is. This is beneficial when examine bugs across the product to determine which areas might have risk or might need additional time to understand and fix larger numbers of bugs.

# Steps to reproduce

Reproduction steps are often included in the description, but some systems separate this visual component of the bug report. Repro steps also must be as concise as possible. Although a bug may have been found in 10 steps, it is important to take the time to see if any of the steps are irrelevant.
Reducing the steps improves the changes of quickly isolation the root cause of the bug.

# Steps to reproduce(example)

**Steps to reproduce:**
Prerequisite - OILIO app to be installed, user to be logged;
1. Open the app
2. Press + (add) button
3. Start filling up the field with photo, select the camera button (upper right corner) => list with options how to select photo is displayed
4. Select "Choose existing" => list with photos is displayed on the upper right side together with an error message "Item details incomplete", partly covered by the list. "Title", "Description", "Approximate Pick-up Location" and "Pick-up Times" fields are bordered with red.
5. Dismiss the photos list by tapping outside of it => the error message is changed to"Error"

# Assignment

Each bug is assigned to exactly one person at a time. It is responsibility of the current bug owner to address any issues(that is, fix the bug) or to reassign the bug.

# Bug Severity

“Severity can be defined as the degree of impact a failure has on the development or operation of a component application being tested.”

# Bug Severity Levels

- Bug causes system crash or data loss
- Bug causes major functionality or other severe problems; product crashes in obscure cases
- Bug causes minor functionality problems, may affect fit and finish.
- Bug contains typos, unclear wording, or error message in low-visibility fields

# Bug Priority

“Priority indicates the importance or urgency of fixing a failure”

# Customer impact

It is valuable to include a customer impact description in the bug report. Customer impact description include how the bug affects the user and how the problem will affect customer scenarios and requirements.

# Environment

It is important to clearly describe the conditions of the test environment and steps necessary to reproduce environment as part of the bug.

- Hardware specification and configuration
- System, component and application versions
- Tools and processes employed
- Related connectivity and data configurations
- Roles, permissions, and other applicable settings

# Resolution

This field is filled when the bug is resolved.

- **Fixed** - the underlying fault was fixed
- **Not reproducible** - cant reproduce same issue
- **Duplicate** – when two bugs describe the same issue
- **By Design** – when system is intended to work that way
- **Postponed** – will be fixed in the future

# Bug report procedure

- Collect error logs or any tracing information
- Reproduce it one more time
- Search for duplicates among reported bugs
- Collect information about testing environment
- Collect snapshot and videos
- Apply models or any artifacts related
- Describe steps or even better – create test
- Report it in Bug Tracking System (BTS)

# Good bug report

- Failure report title should be explicit enough not to read description from BTS.
- Should not duplicate existing failures/bugs
- Should describe exactly one problem
- Should be reproducible, if not report incident
- Should describe real deviation/problem

# Metrics

How to use the data from the Bug Tracking System?

# Using the BTS data

- Total found / total bugs fixed
- Total bugs per language
- Bug find rate over time
- Bug fix rate over time
- Bugs by code area
- Bugs by severity

# Using the BTS data

- How found
- Where found
- When introduced
- Bug reactivation rate
- Average time to resolve
- Average time to close

# How not to use the data

Many managers collect and track bug data for performance management. However alone the gross number of bugs reported metric provides very little valuable information with regard to individual performance. When measuring we should consider:

- Feature complexity
- Developer ability
- Specification completeness
- Timeliness of reporting

# Bug bars

In simplest terms, a bug bar is a limit on the number of bugs that a developer can have assigned to her at any particular time. If the number of bugs assigned to developer goes over the magic number, the developer is expected to stop feature work and fig bugs.
Example: assigned bugs <= 5.

# Questions