McKenna Principals

Software Defects and SW Reliability Assessment

Kristine Hejna

Maybe you don't know

- A lot of good work has been done on Software Defect Metrics.
- Everyone makes mistakes
- This should be of interest to quality people interested in assessing software quality across the lifecycle, software engineers interested in improving product quality, and reliability engineers who have not seen how reliability is assessed in software.
  This is not about testing.

Starting Points

- Software Development Life Cycle(SDLC) means waterfall or agile, iterative, evolutionary, or ?
- A defect is anything that will cause you to change something you thought you had finished

By caring about defect metrics you understand that you will have to record and analyze information that doesn't add value to your product. It may reduce the cost of the value-added steps, or reduce other non-value-added work.

All SW Dev has a lifecycle

- Ideally –
  - Requirements – learn what the user needs, and what that means in a computer system
  - Design – choose the method for meeting the needs that is sufficient and cost-effective
  - Code (or Implement) - create the sequenced commands and instructions to the computer system to implement the design
  - Test – check that
    - Your code implements the design
    - Your product satisfies the user need
  - Deliver and Support

Dimensions of Software Quality

- **Accessibility:** The degree to which software can be used comfortably by a wide variety of people, including those who require assistive technologies like screen magnifiers or voice recognition.
- **Compatibility:** The suitability for use in different environments like different Operating Systems, Browsers, etc.
- **Concurrency:** The ability of software to service multiple requests to the same resources at the same time.
- **Efficiency:** The ability of software to perform well or achieve a result without wasted energy, resources, effort, time or money.
- **Functionality**: The ability of software to carry out the functions as specified or desired.
- **Installability:** The ability of software to be installed in a specified environment.
- **Localizability:** The ability of software to be used in different languages, time zones etc.
- **Maintainability:** The ease with which software can be modified (adding or enhancing features, fixing bugs, etc)
- **Performance:** The speed at which software performs under a particular load.
- **Portability:** The ability of software to be transferred easily from one location to another.
- **Reliability:** The ability of software to perform a required function under stated conditions for stated period of time without any errors.
- **Scalability:** The measure of software's ability to increase or decrease in performance in response to changes in software's processing demands.
- **Security:** The extent of protection against unauthorized access, invasion of privacy, theft, loss of data, etc.
- **Testability:** The ability of software to be easily tested.
- **Usability:** The degree of software's ease of use.

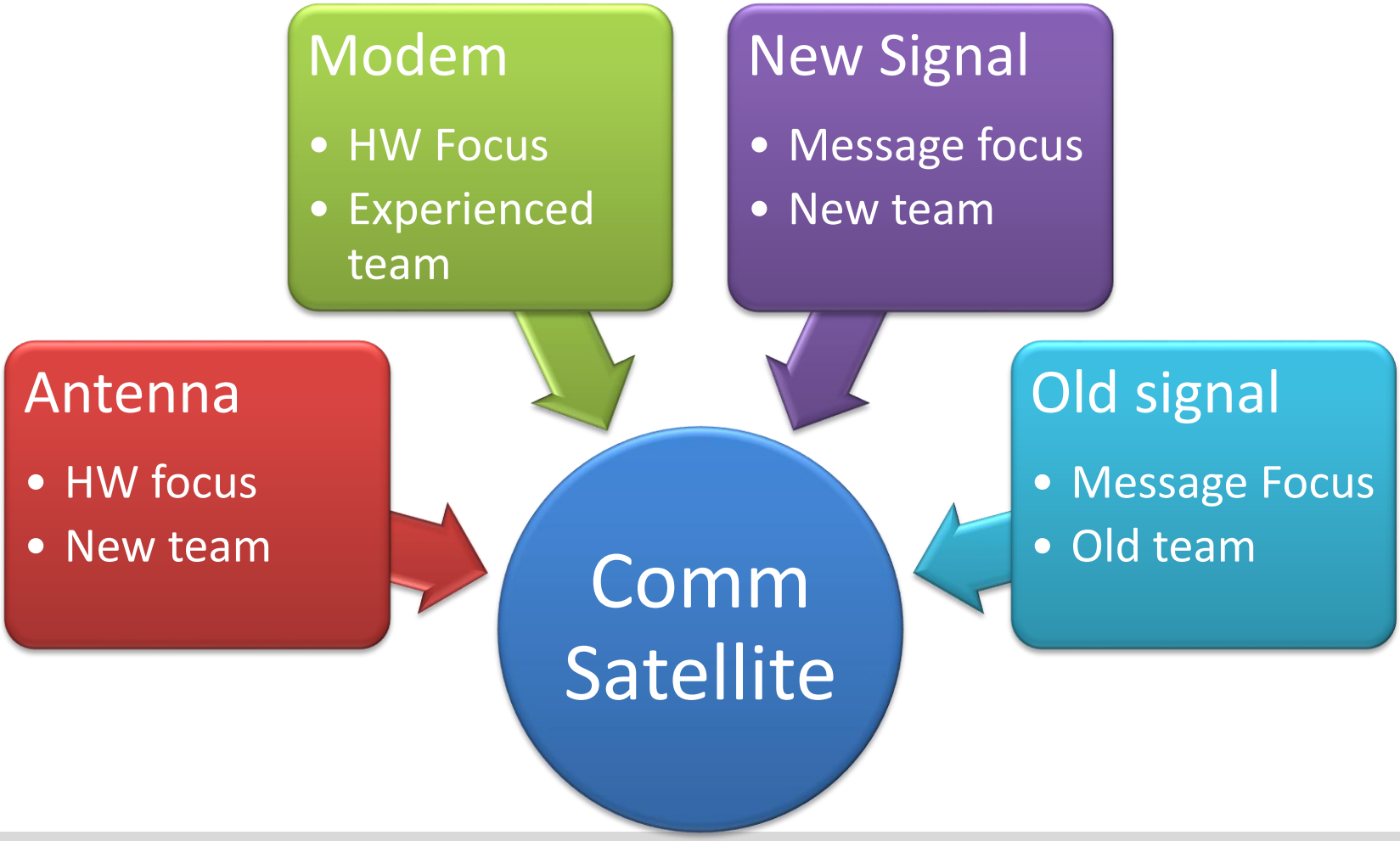
Track the Defects

- **Record all instances** that you or others discover that will make you change the requirements, the design, the code, the test, the documentation, or any other part of the product you had assumed didn't need more work.
- **Use a tool**, it makes it much easier.
- **Track the date**, and who found it. You might find one person is a really picky reviewer.
- **Record the lifecycle phase** when the defect occurred (did you miss a requirement?), and when it was found (and the customer won't accept sans serif fonts at the final turnover test.)
- **Track the cause**. With practice and a little thought about how you go about things, you will find that you have a limited set of causes. This is where Orthogonal Defect Classification comes in.

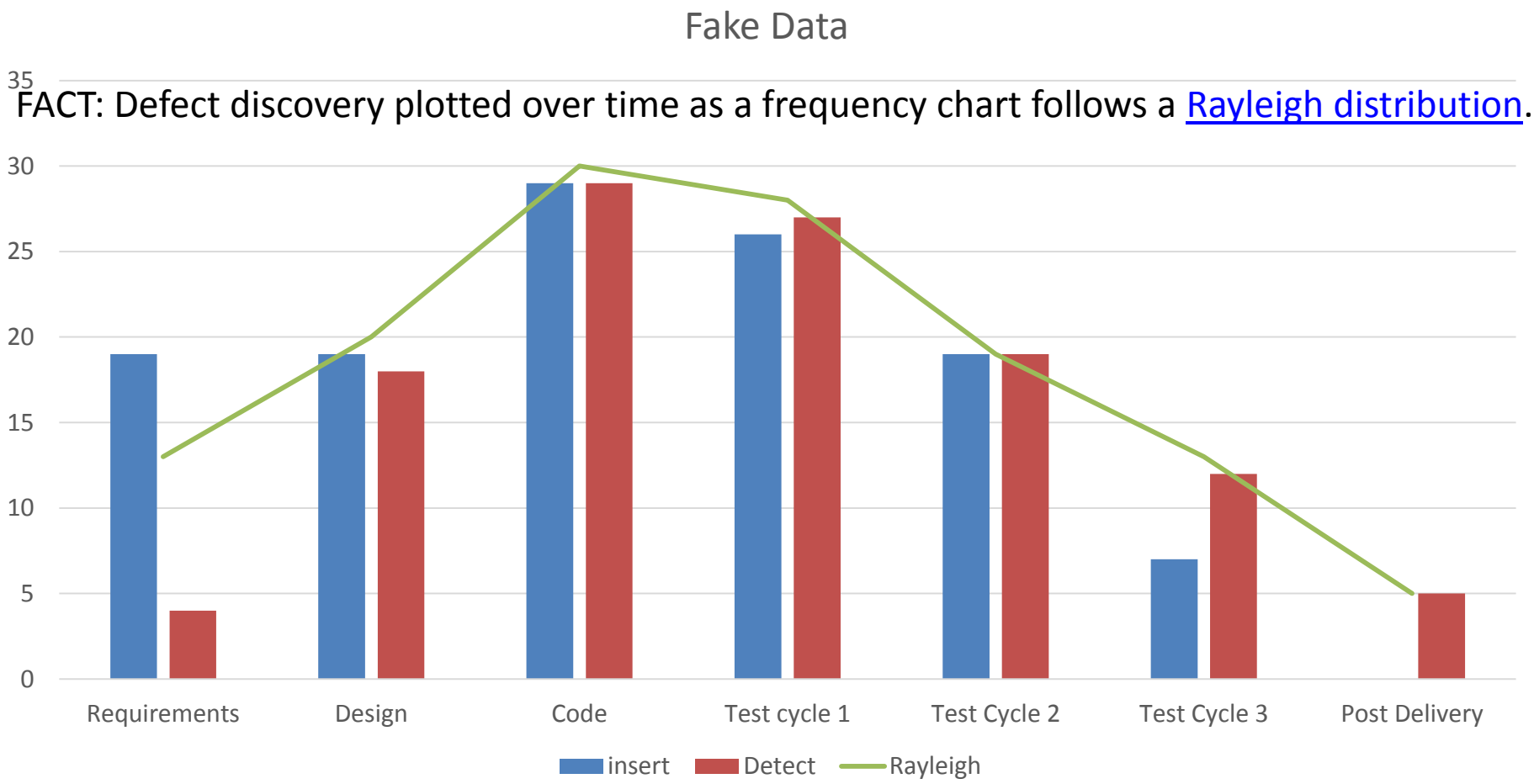
Analyze the Defects

- Why collect this info unless you can learn something?
  - One thing you learn is each development is different
    - Different team
    - Different environment
    - Different product
    - Different types and numbers of defects
  - SW not truly “repeatable process”
    - Enough similarities to benefit from analysis

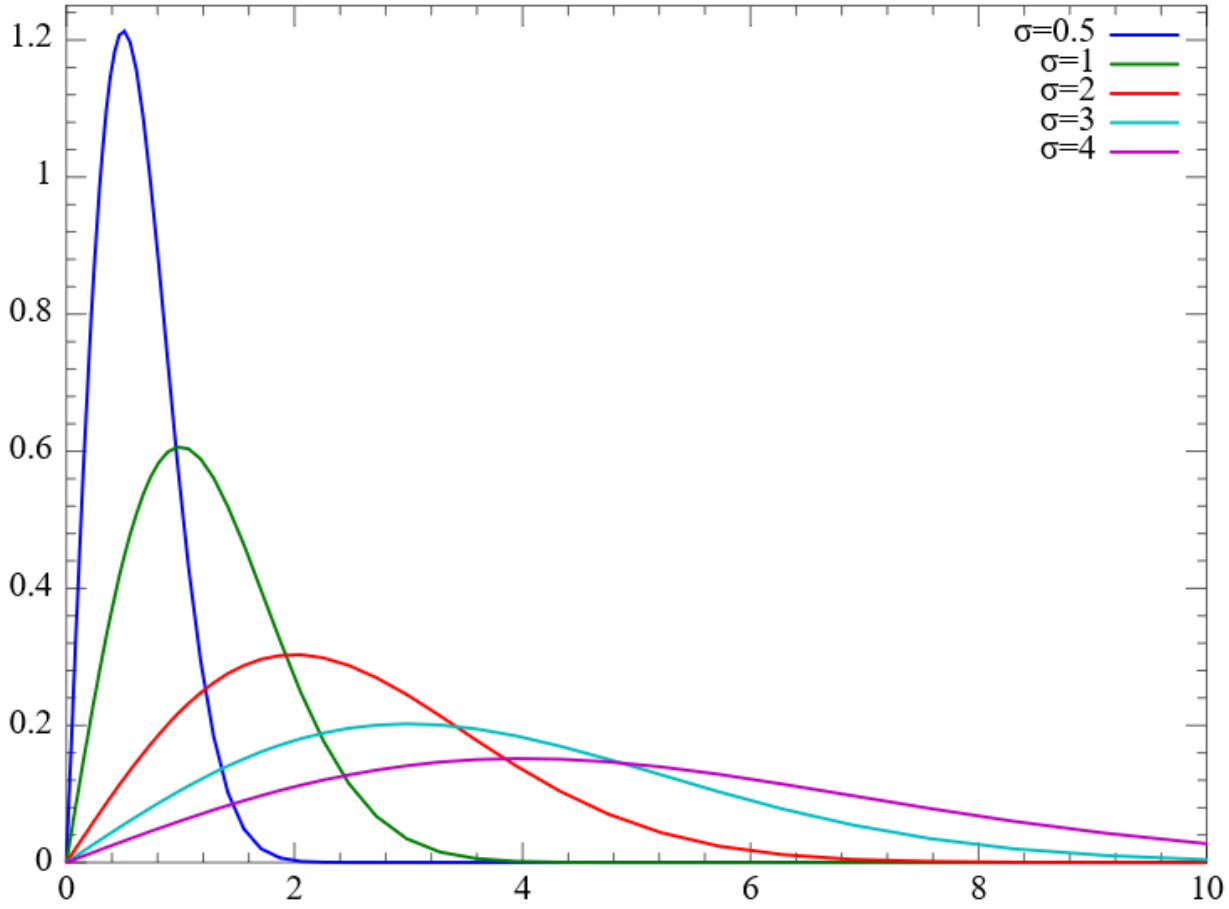
Example – Different defect profiles



Reliability – Reduce defects after delivery

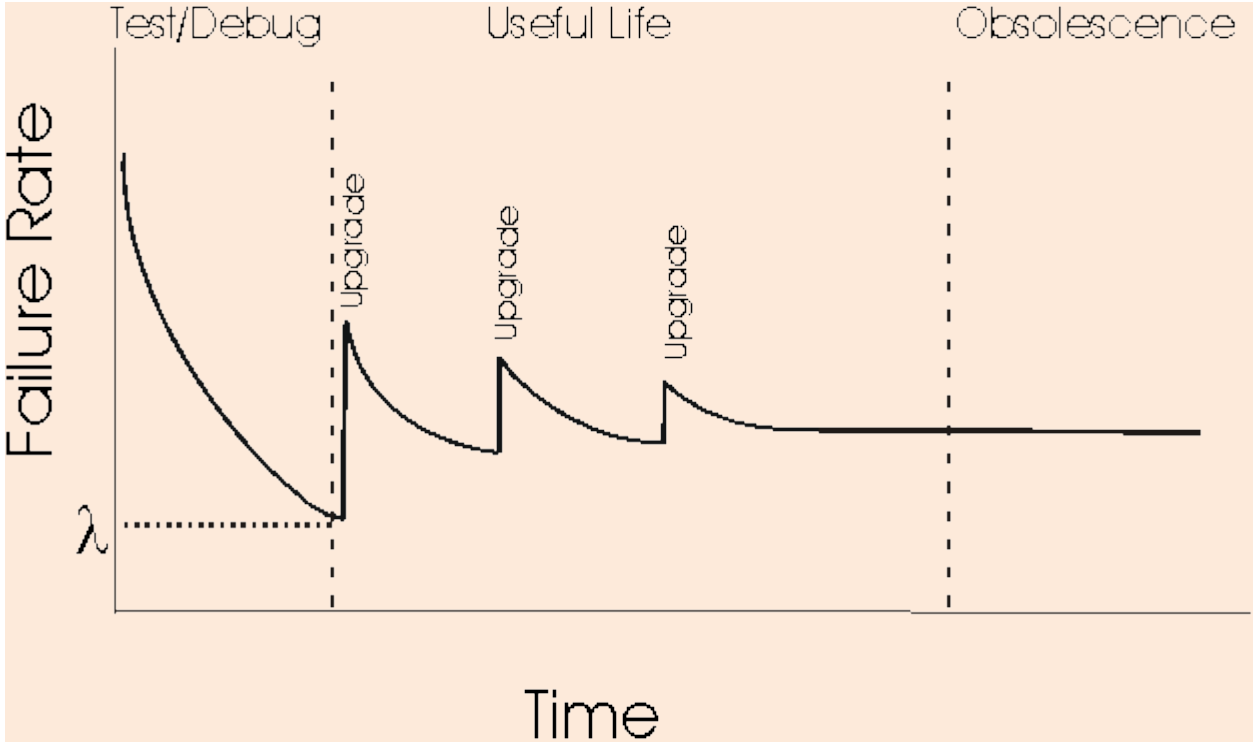


Rayleigh Distributions



- *Ship Shape*
- A target level for defects found after delivery marked on your plot and tells you when you can ship at that defect level.

Reliability Growth Function



- Gets better with time (duh)
- Follows an S curve (also unsurprising)
- Inverse Rayleigh

Requirements Defects

- Customers can leave out information that they take for granted.
  - "You can crowd the screens with options, I don't care" As long as you make the options far enough apart that I can use a touch i
- You can r
- They can ed with deliverab on, for example)

TERMINOLOGY:

Defects are **inserted** or **injected**. This sounds like you meant it to happen. Don't worry about it. They happen, nobody is perfect.

Design Defects

- When the design will not actually satisfy the requirements, or when it is incomplete, often with respect to off-nominal conditions, or failure to consider constraints.
  - Developers tend to think about achieving the functionality, not about what could go wrong.
  - Engineers tend to think within their specialty, which is why we want “specialty engineering” involved in design review as well as systems engineers
    - Human factors, security, reliability, mechanical

Code Defects

- “Correct” code and lint
- Code that doesn’t do what you think you said
- Insecure code, fragile code, inefficient code
- Code that doesn’t implement the design or
- Code that doesn’t satisfy the requirements
- It is often more valuable to the coder than the project to track the kinds of coding mistakes

Test Defects

- Seems like something that should not happen
  - Happened at my job last week
- People staying up late to finish something take shortcuts
- Could be configuration management -
  - Not the right test config, or data
- Could be testers didn't get the same message from the requirements

Classifying Defects - Cause

- Orthogonal Defect Classification (ODC)
  - developed in late 1980's and 1990's.
  - is known to reduce the time taken to perform root cause analysis by over a factor of 10. The gains come primarily from a different approach to root cause analysis, where the ODC data is generated rapidly (in minutes, as opposed to hours per defect) and analytics used for the cause and effect analysis

Classifying Defect Causes

- You can use ODC, but it can be unsatisfying
- Allow speculation on cause or description of the problem – at least 256 characters
- It might be nice to have a separate field for text and a drop down for faster categorization
- Cause or reason could be the lack of one of the Dimensions of Software Quality
- More work for analysis, can lead to better long term results and reliability analysis.

Classifying Defects - Severity

- **DEFECT SEVERITY – How bad is the problem?**
  - Severity and Priority not the same
  - Severity and Impact are (usually) the same
- **Critical**: Defect affects the critical functionality & prevents further application testing. No workaround.
- **Major**: Major functionality not working but able to test application. It has workaround but not obvious.
- **Minor**: Doesn’t work right, but has a usable workaround
- **Trivial**: Not as good as it could be or should be, but still works.

Classifying Defects
Priority and Probability

- Priority – how soon does it need to be fixed?
  - Sometimes a defect needs to be fixed right away even when it has a workaround.
  - For instance, a feature needed for a demonstration
- Probability – how likely is it to be encountered
  - Like risk, probability assessed on 2 factors
  - (Frequency of feature use) X (noticeability of bug)
    - A spelling error on main page, on an obscure word

More classifications

- Status
  - Open – Closed – Deferred – Cancelled – Rejected
  - I suggest also “assigned”, “in work”
  - Check workflows in tools like Jira
- Once it has been fixed – what component/module had the problem

Reliability Analysis

- Predicting distribution by time and severity of defect detections during post-delivery operation.
  - Usually not a bathtub model for software –
    - It doesn't wear out and get worse
  - Not a Poisson model, with *randomly* encountered objects along the path
  - Follows the same Rayleigh if you continue to update the software (fix the found bugs)
  - Make use of the defect probability assessment

Resources - Online

- Orthogonal Defect Classification
  - http://researcher.watson.ibm.com/researcher/view_group.php?id=480
  - https://en.wikipedia.org/wiki/Orthogonal_Defect_Classification
- Standards
  - http://standards.ieee.org/findstds/standard/software_and_systems_engineering.html

Resources - Online

McKenna Principals

- Software Reliability
  - https://en.wikipedia.org/wiki/Software_reliability_testing - 10K foot level, use its references
  - http://johnmusa.com/
  - http://www.weibull.com/hotwire/issue84/relbasics84.htm
  - http://www.se.rit.edu/~swen-350/slides/SoftwareReliabilityEngineering.pdf
  - http://www.dote.osd.mil/docs/dote-temp-guidebook/Sw_ReliabilityGrowthApproach_RAMS2013_v1a.pdf

Resources - Online

- Defect Classification for testers
  - http://www.requirementdriventesting.com/software-test-methodology-rdtrule6/
  - http://www.softwaretestingstuff.com/2008/05/classification-of-defects-bugs.html
  - http://softwaretestingfundamentals.com/defect/
  - http://www.softwaretestingclass.com/
  - http://www.testingexcellence.com/
    - severity-and-priority-difference/
    - defect-clustering-in-software-testing/