SENG 275

SOFTWARE TESTING

DR. NAVNEET KAUR POPLI

DEPT. OF ELECTRICAL AND COMPUTER ENGINEERING

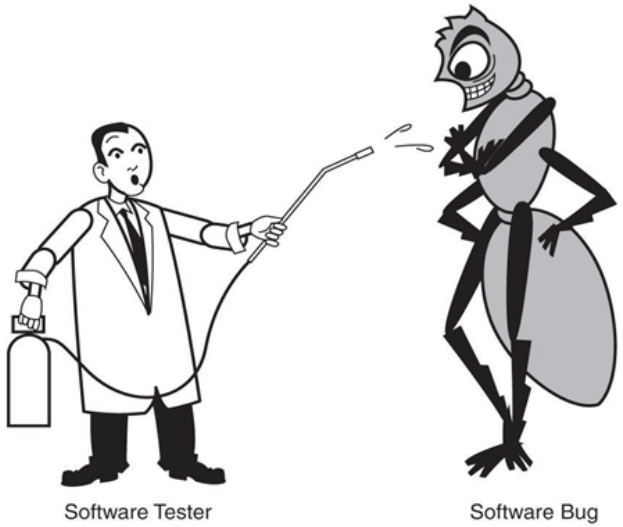
University of Victoria

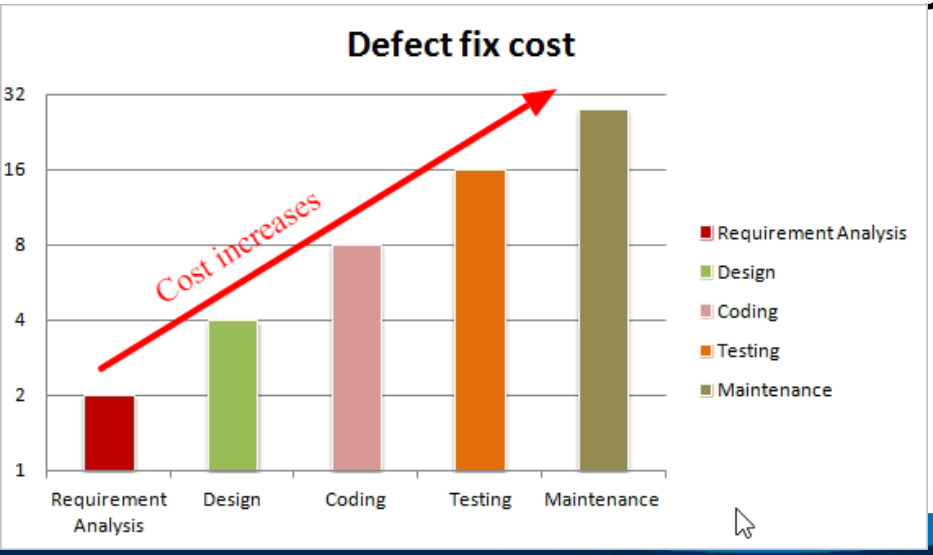
WHY TESTING IS SO DIFFICULT? PRINCIPLES OF SOFTWARE TESTING

University of Victoria

7 principles of software testing

1. Testing shows presence of defects, not their absence
2. Exhaustive testing is not possible
3. Early testing saves time and money
4. Defects cluster together
5. Beware of the Pesticide paradox
6. Testing is context dependent
7. Absence of errors is a fallacy



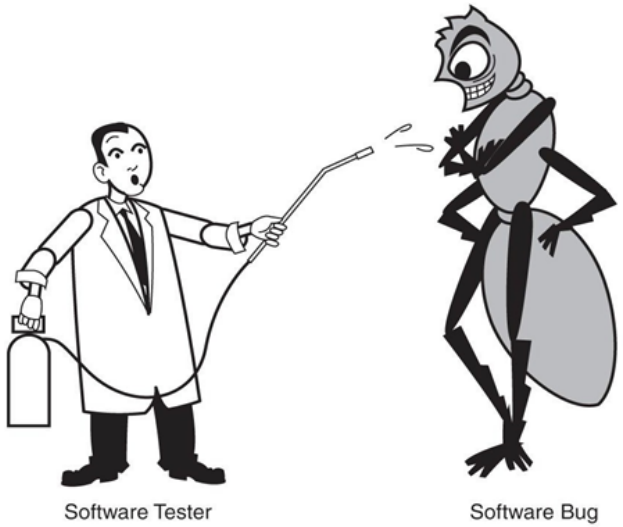


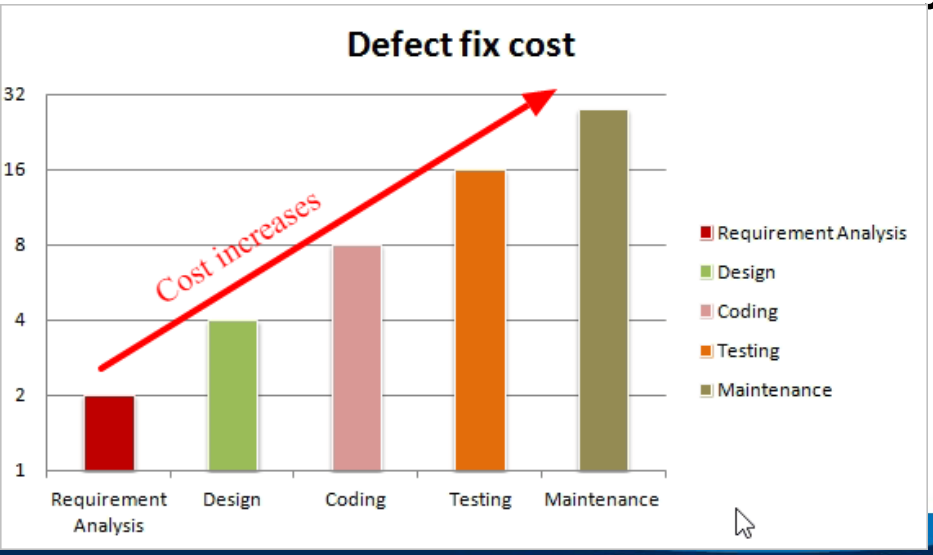
University of Victoria

https://www.boxuk.com/insight/the-seven-principles-of-testing/

7 principles of software testing

1. Testing shows presence of defects, not their absence
2. Exhaustive testing is not possible
3. Early testing saves time and money
4. Defects cluster together
5. Beware of the Pesticide paradox
6. Testing is context dependent
7. Absence of errors is a fallacy





University of Victoria

https://www.boxuk.com/insight/the-seven-principles-of-testing/

1. Testing shows presence of defects, not their absence

- Testing helps greatly reduce the number of **undiscovered defects** hiding in software but finding and resolving these issues is not itself proof that the software or system is **100% issue-free**.
- This concept should always be accepted by teams, and effort should be made to **manage client expectations**.
- Having a **comprehensive test strategy** that includes thorough **test plans, reports and statistics** along with testing release plans can all help with this; reassuring clients as to testing progress and providing confidence that the right areas are being tested.

University of Victoria

Monitoring

- Additionally, **ongoing monitoring and testing after systems have gone into production is vital**.
- Thinking forward to potential issues that could arise is another good way to help mitigate against any future problems, for example considering **load testing** if a site is launching a new marketing campaign, so you can be confident the software will withstand any anticipated larger volumes of traffic.

University of Victoria

Samsung Galaxy Note 7- defect after release

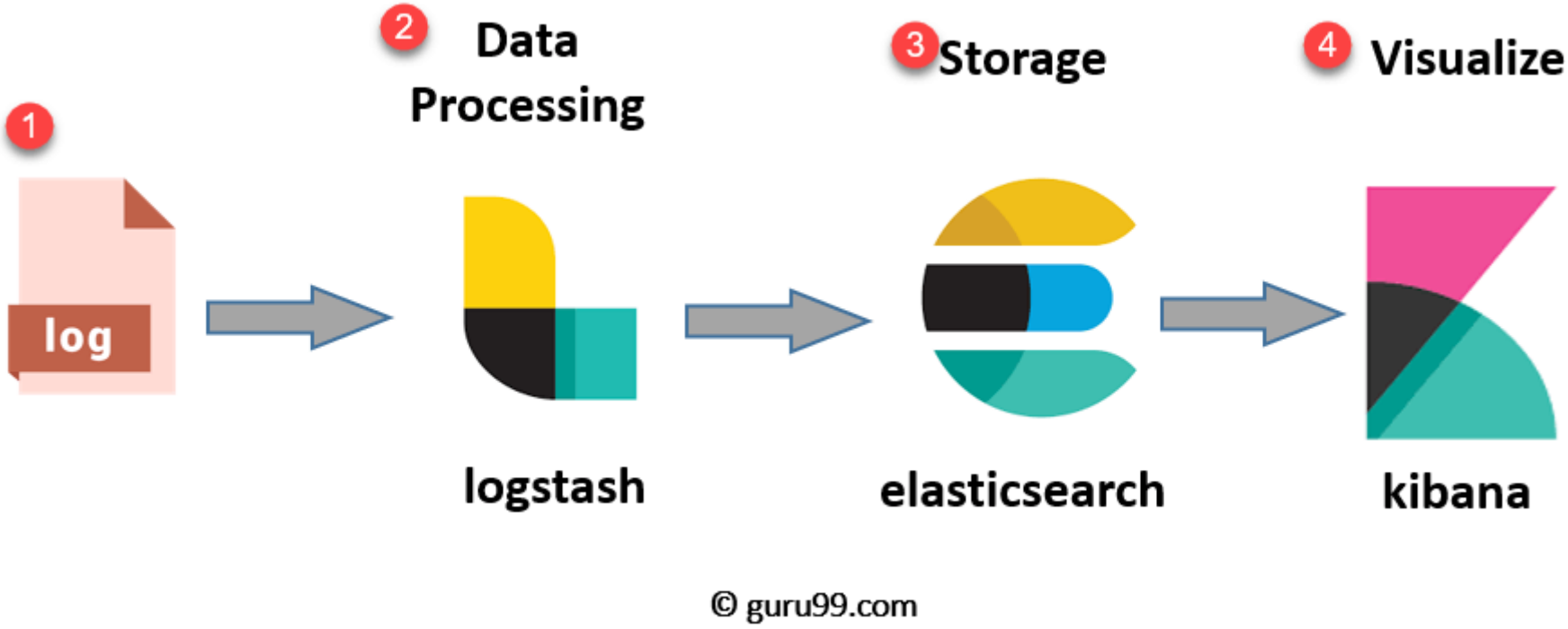
University of Victoria

Samsung Galaxy Note 7- defect after release

- Samsung Galaxy Note 7 had to be removed from the market in October 2016 just 2 months after its launch because the device caught fire by itself, both at rest and in use.
- An initial investigation identified faulty batteries from one of its suppliers as the root cause

University of Victoria

Infrastructure Monitoring using ELK Stack



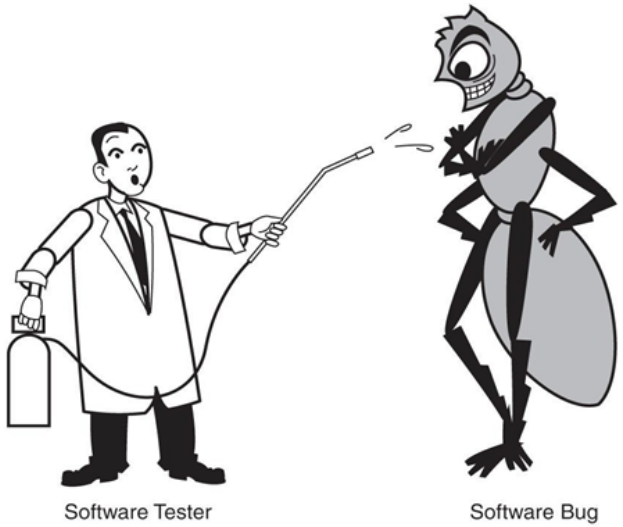
University of Victoria

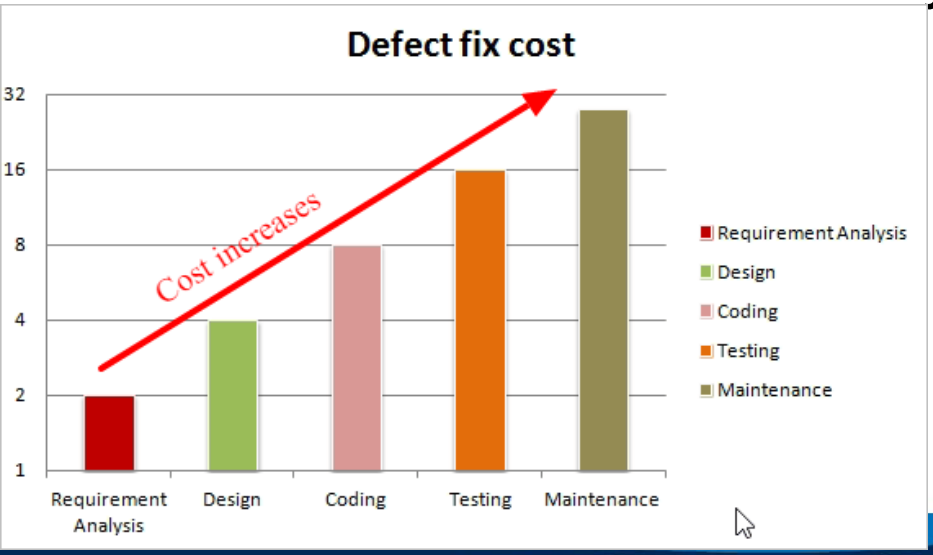
Infrastructure Monitoring using ELK Stack

- The ELK Stack is a collection of three open-source products — Elasticsearch, Logstash, and Kibana. ELK stack provides centralized logging in order to **identify problems with servers or applications.**
- **Logs:** Server logs that need to be analyzed are identified
- **Logstash:** Collect logs and events data.
- **ElasticSearch:** The transformed data from Logstash is Stored, Searched, and indexed in a NoSQL database.
- **Kibana:** Kibana uses Elasticsearch DB to Visualize data.

7 principles of software testing

1. Testing shows presence of defects, not their absence
2. Exhaustive testing is not possible
3. Early testing saves time and money
4. Defects cluster together
5. Beware of the Pesticide paradox
6. Testing is context dependent
7. Absence of errors is a fallacy





University of Victoria

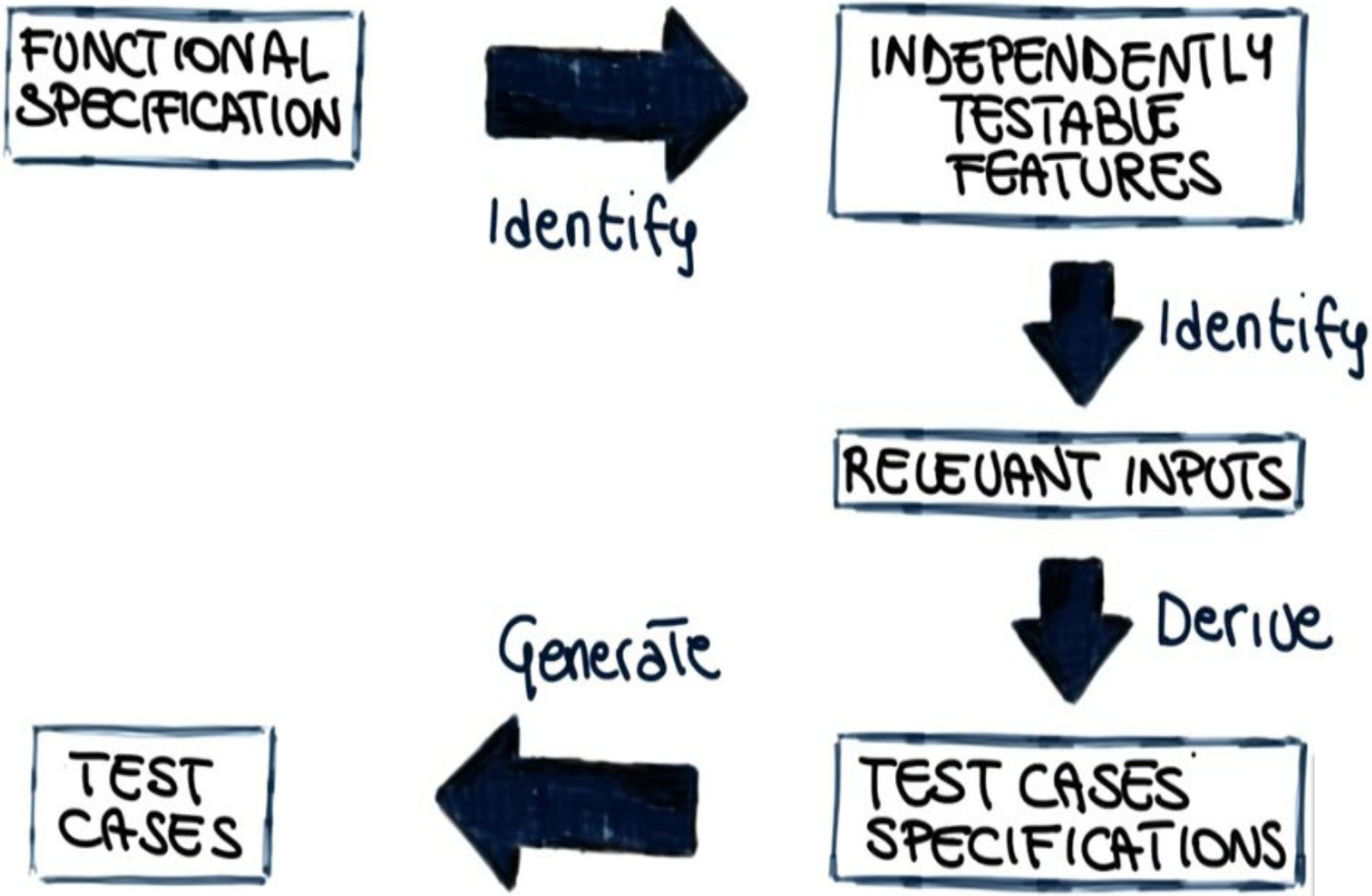
https://www.boxuk.com/insight/the-seven-principles-of-testing/

2. Exhaustive testing is possible but not practical

- It is almost impossible to test EVERYTHING – all **combinations of inputs and preconditions** – and you could also argue that attempting to do so is not an efficient use of time and budget. Though this is possible, but it is not practical.
- However, one of the skills of testing is **assessing risks and planning your tests around these** – you can then cover vast areas, while making sure you are testing the most important functions.
- With **careful planning and assessment**, your **test coverage** can remain excellent and enable that necessary confidence in your software, without requiring that you test every single line of code.

University of Victoria

A SYSTEMATIC FUNCTIONAL-TESTING APPROACH



Identifying testable features

Q)

int printSum(int a, int b)

How many independently testable features do we have here:

- 1
- 2
- 3
- >3

Identifying testable features

Q)

int printSum(int a, int b)

How many independently testable features do we have here:

- 1
- 2
- 3
- >3

Identify Relevant inputs for these features- Test Data Selection- how long will the testing take?

Exhaustive testing:

Q1) What is the **input domain** of this function?

Q2) How **long** would it take to exhaustively test the function int printSum(int a, int b) ?

Identify Relevant inputs for these features- Test Data Selection- how long will the testing take?

Q1) What is the **input domain** of this function?

A1) int a – number of possible inputs= 2^{32}

int b – number of possible inputs= 2^{32}

Total number of inputs= = 2^{32} X 2^{32} = 2^{64} = $1.8X10^{19} \cong 10^{19}$

Q2) How **long** would it take to exhaustively test the function printSum(int a, int b) ?

A2) Total number of tests we need to run to cover the whole domain=10^{19}

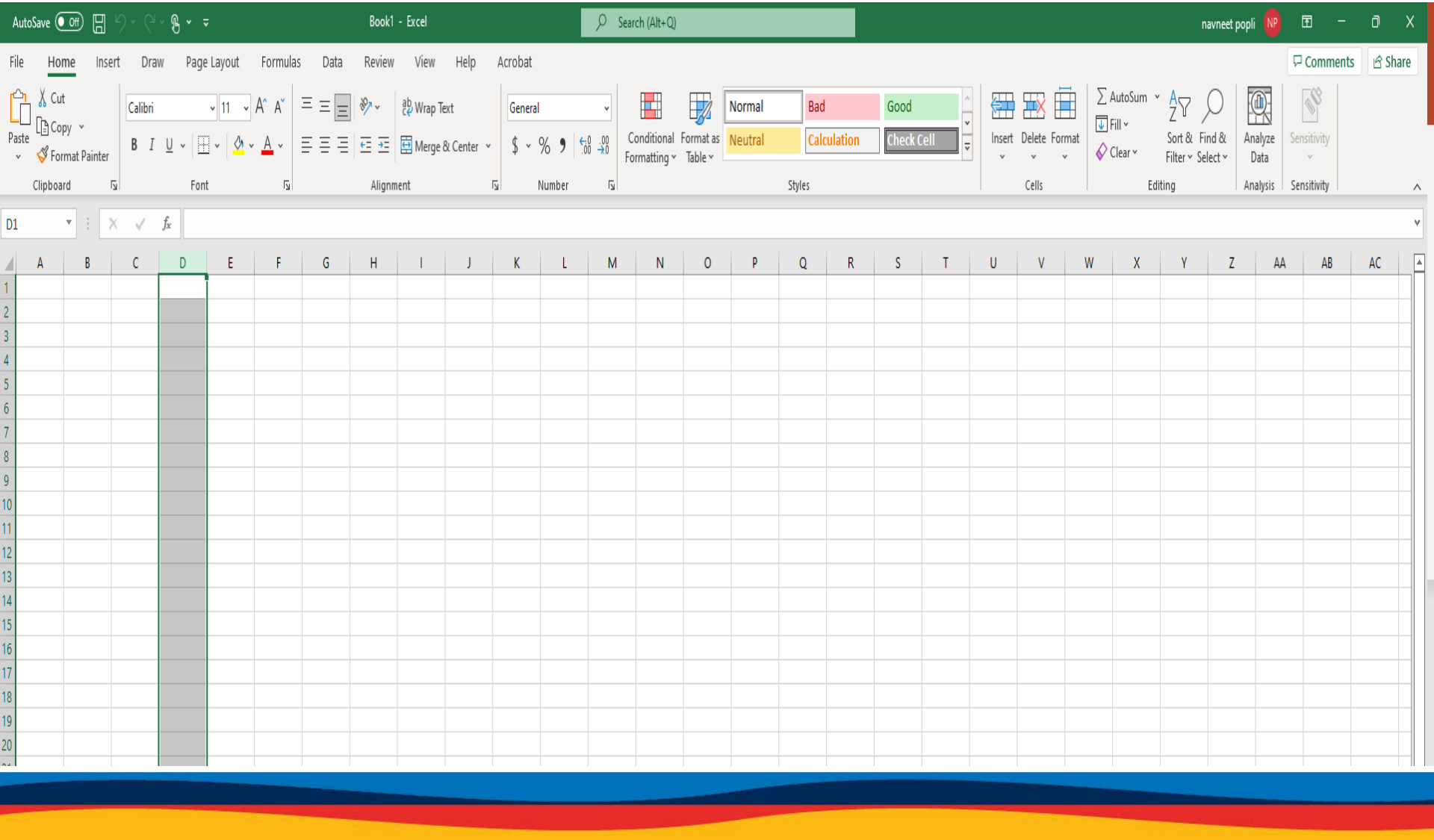
If we are able to run 1 test per nanosecond, we can run 10^9 tests/sec

10^{10} seconds overall

Around 600 years

University of Victoria

Consider a spreadsheet- how many testable features are there?

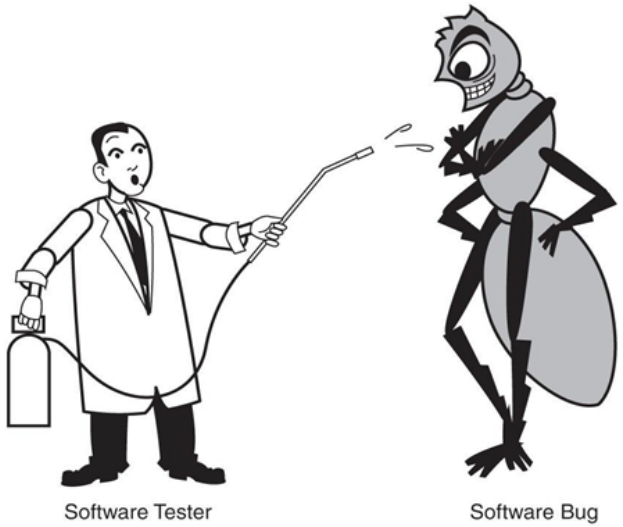


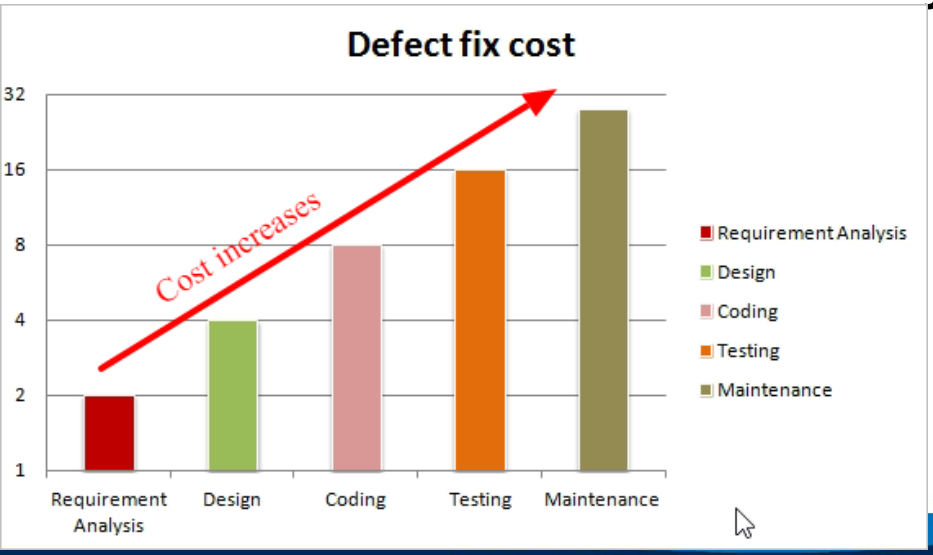
Spreadsheet testable features

- Cell merge
- Pie chart creation
- File open feature
- Wrap text feature…….

7 principles of software testing

1. Testing shows presence of defects, not their absence
2. Exhaustive testing is not possible
3. Early testing saves time and money
4. Defects cluster together
5. Beware of the Pesticide paradox
6. Testing is context dependent
7. Absence of errors is a fallacy

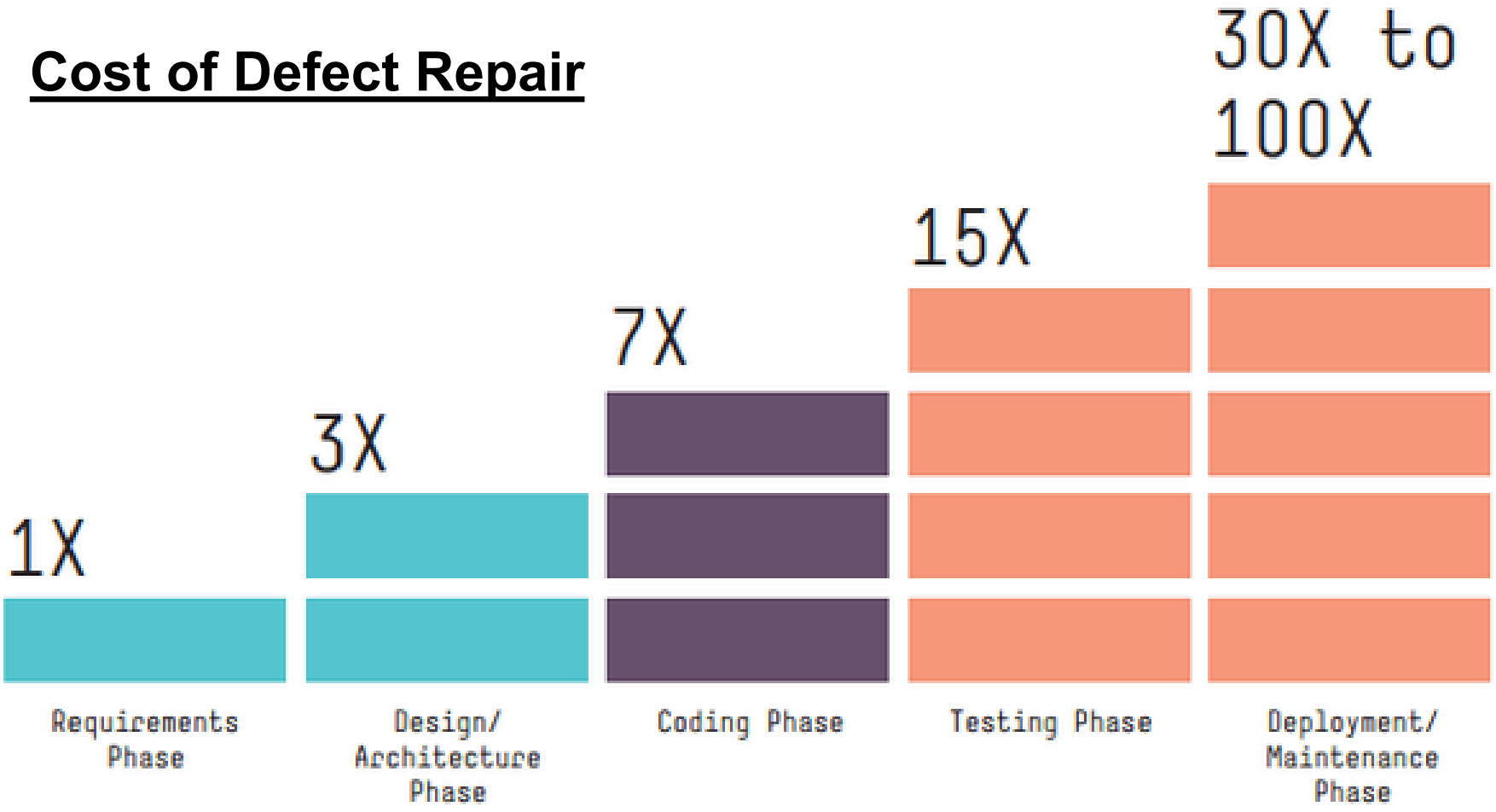




University of Victoria

https://www.boxuk.com/insight/the-seven-principles-of-testing/

3. Early testing saves time and money

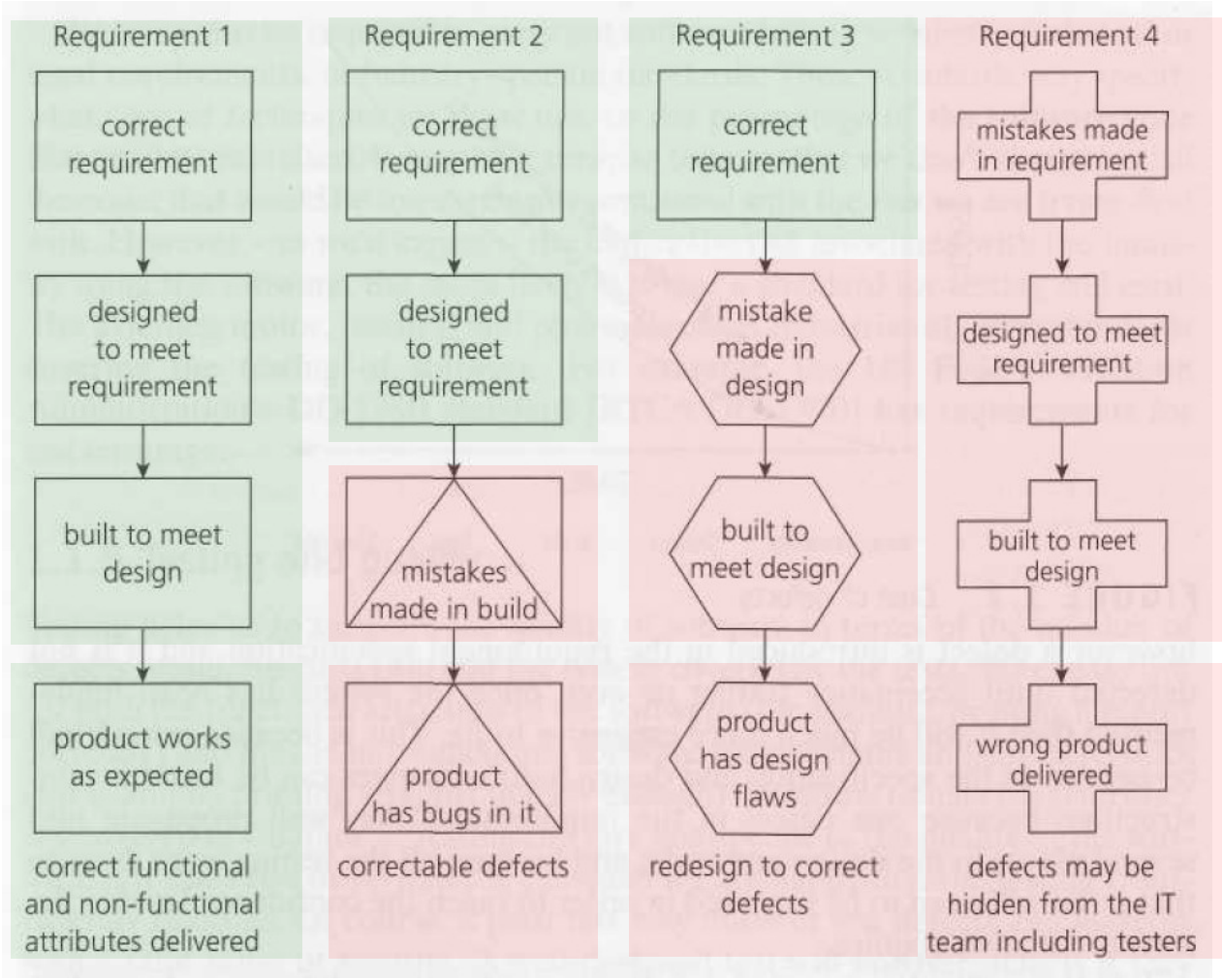


3. Early testing saves time and money

- Testing early is fundamentally important in the software lifecycle. This could even mean testing **requirements before coding has started**, for example – amending issues at this stage is a lot easier and **cheaper** than doing so right at the end of the product's lifecycle, by which time whole areas of functionality might need to be re-written, leading to overruns and missed deadlines.
- Involving testing early is also a fundamental **Agile principle**, which sees testing as an activity **throughout, rather than a phase** (which in a traditional waterfall approach would be at the end) because it enables quick and timely **continuous feedback loops**.
- When a team encounters hurdles or impediments, early feedback is one of the best ways to overcome these, and testers are essential for this. Consider the tester as the 'information provider' – a valuable role to play.
- Essentially, testing early can even help you prevent defects in the first place!

University of Victoria

Faults can be introduced at any moment in the software development process



Finding faults in different development phases may require different types of testing

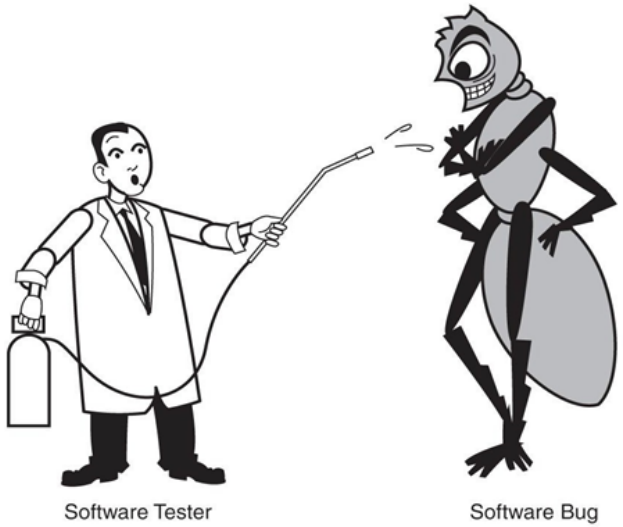
University of Victoria

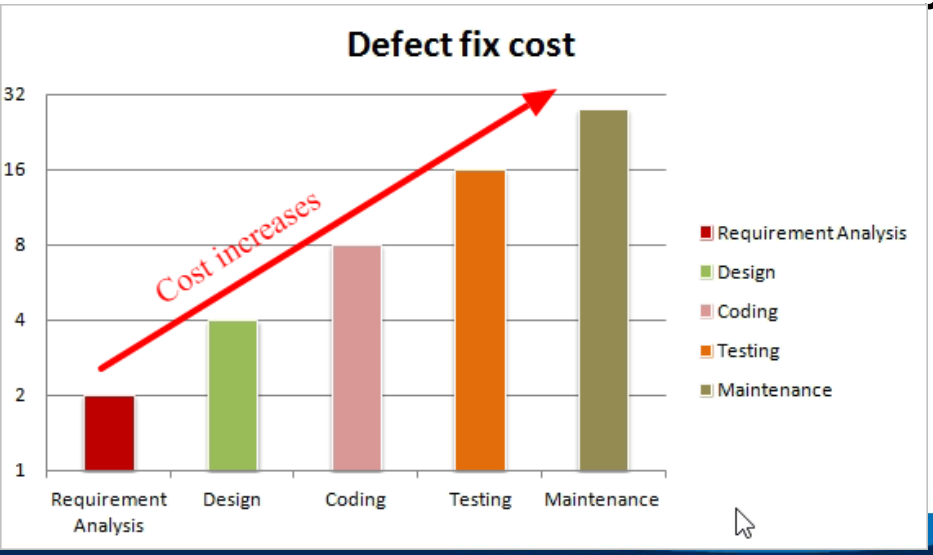
Therefore, Start testing as early as possible

- To let tests guide design
- To get feedback as early as possible
- To find bugs when they are cheapest to fix
- To find bugs when have caused least damage

7 principles of software testing

1. Testing shows presence of defects, not their absence
2. Exhaustive testing is not possible
3. Early testing saves time and money
4. Defects cluster together
5. Beware of the Pesticide paradox
6. Testing is context dependent
7. Absence of errors is a fallacy



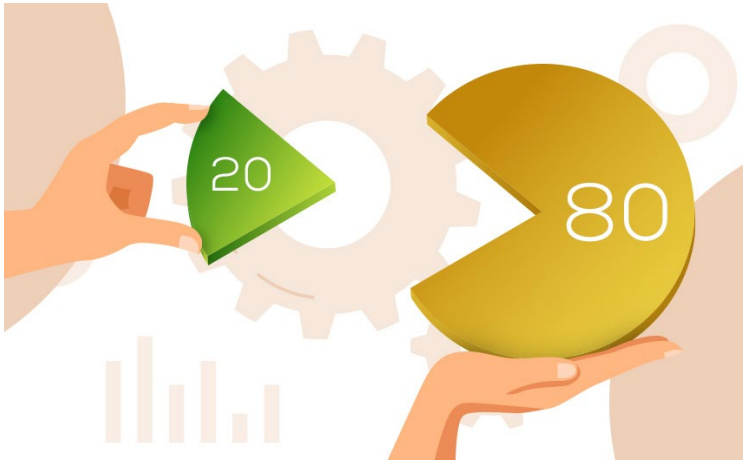


University of Victoria

https://www.boxuk.com/insight/the-seven-principles-of-testing/

4. Defects cluster together

- This is the idea that certain components or modules of software usually contain the greatest number of issues or are responsible for most operational failures.
- Testing therefore, should be **focused on these areas** (proportionally to the expected – and later observed – defect density of these areas). The **Pareto principle of 80:20** can be applied – **80 percent of defects are due to 20 percent of code!**
- This is particularly the case with large and complex systems, but defect density can vary for a range of reasons.



University of Victoria

4. Defects cluster together

- Issues are not evenly distributed throughout the whole system, and the more **complicated a component**, or the more **third-party dependencies** there are, the more likely it is that there will be defects.
- **Inheriting legacy code** and **developing new features** in certain components that are undergoing **frequent changes** and are therefore **more volatile**, can also cause defect clustering.
- Knowing this could prove to be very valuable for your testing; if we find one defect in a particular module/area there is a strong chance of discovering many more there.
- Identifying **the more complex components, or areas that have more dependencies or are changing the most**, for example, can help you **concentrate your testing on these crucial risk areas.**

University of Victoria

Facts that can contribute to this principle

- Legacy code
- Different teams touching the same part of code
- Features that may suffer many changes
- Fickle 3rd-party integration

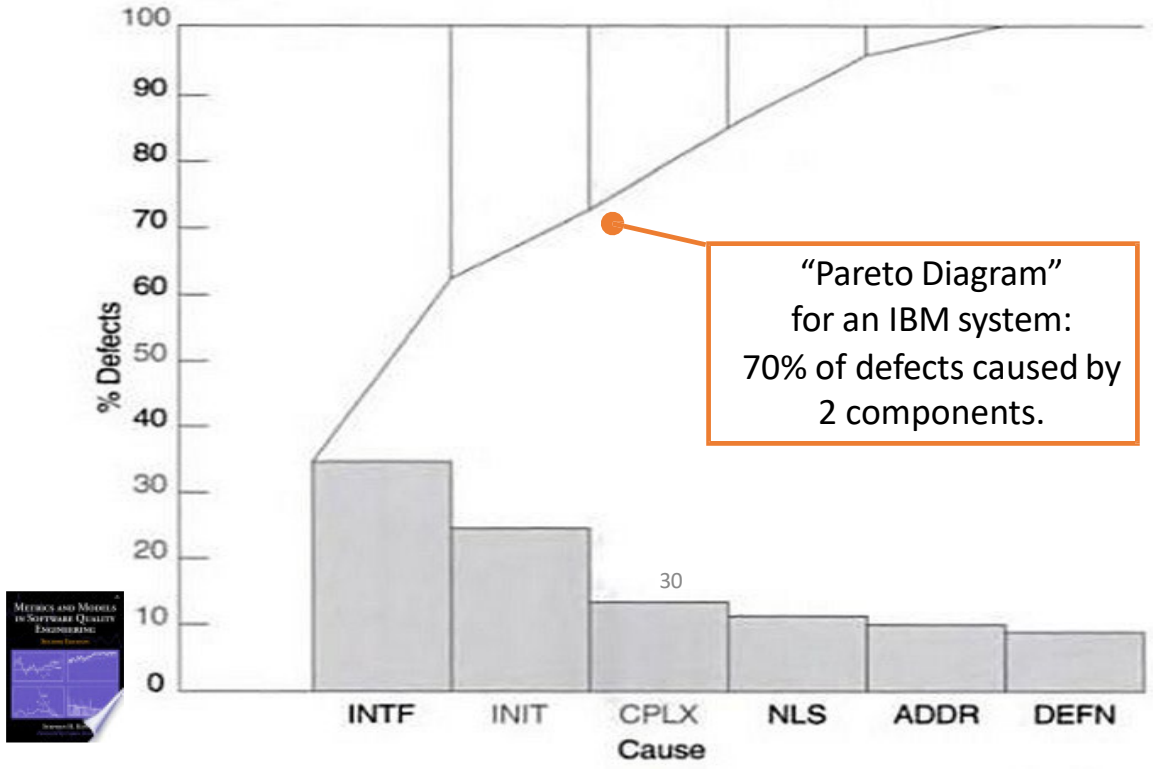
HOTSPOTS

4. Defects cluster together

- "Hot" components requiring frequent change, bad habits, poor developers, tricky logic, business uncertainty, innovative, size, ….
- Pareto Principle / Law of vital few:
  - *80% of effects come from 20% of causes*
- *Use to focus test effort*

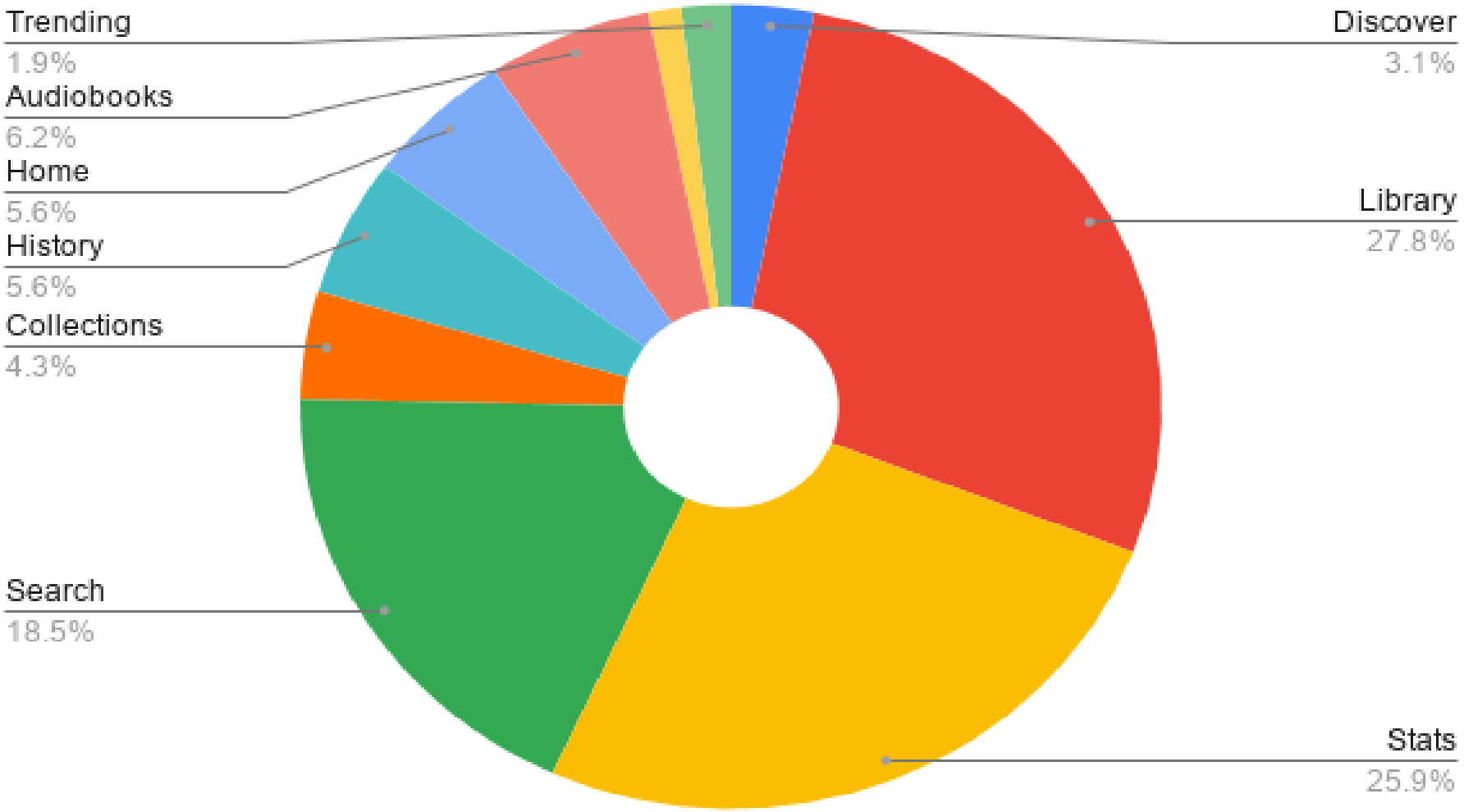
If you find a bug, keep on searching in its 'neighborhood'

IBM Rochester product
NLS: translation-related national languages
INTF: Interface defects
ADDR: addresses
DEFN: data definition
CPLX: complex logical problems
INIT: data initialization



"Pareto Diagram" for an IBM system: 70% of defects caused by 2 components.

Example of defect clustering *(Defect distribution by Module in a books related application)*

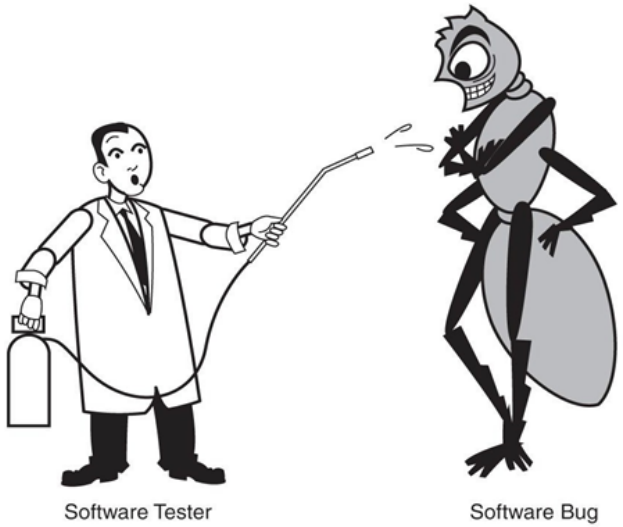


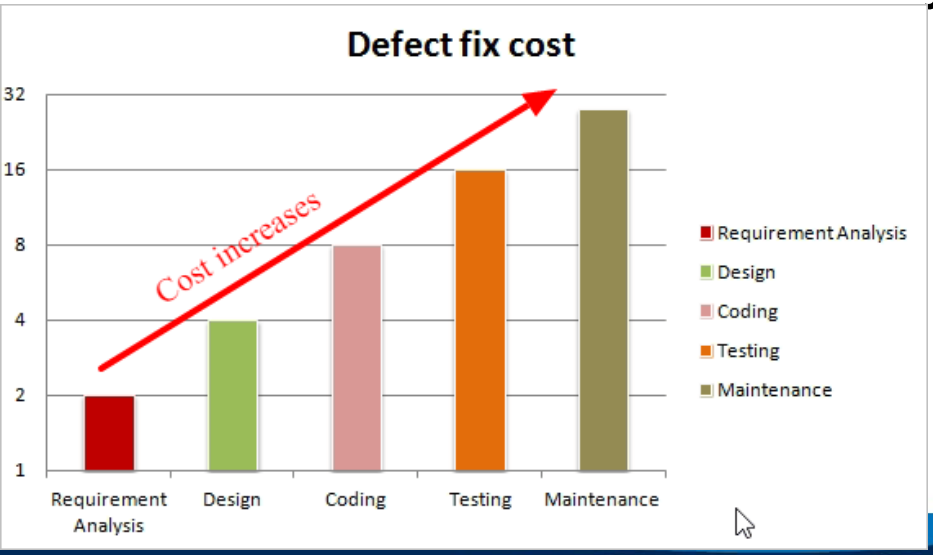
Example of defect clustering (Defect distribution by Module)

- In the above chart, you can clearly visualize the defect clusters. Focusing on Library, Stats and Search module will help to largely reduce the number of issues.
- Once you start cleaning your defect clusters, the number of defects starts dropping, and eventually, that module will not be a threat.
- **Defect clusters change over time**.
- You need to constantly identify new defect clusters and work on eradicating them.

7 principles of software testing

1. Testing shows presence of defects, not their absence
2. Exhaustive testing is not possible
3. Early testing saves time and money
4. Defects cluster together
5. Beware of the Pesticide paradox
6. Testing is context dependent
7. Absence of errors is a fallacy





University of Victoria

https://www.boxuk.com/insight/the-seven-principles-of-testing/

5. Beware of the Pesticide paradox-keep your tests relevant

- This is based on the theory that when you use pesticide repeatedly on crops, insects will eventually build up an **immunity**, rendering it ineffective.
- Similarly with testing, if the **same tests** are run continuously then – while they might confirm the software is working – eventually they will **fail to find new issues**.
- It is important to keep **reviewing your tests and modifying or adding to your scenarios** to help prevent the pesticide paradox from occurring – maybe using varying methods of testing techniques, methods and approaches in parallel.

University of Victoria

Example-update your test cases with each cycle and add new cases to the old set.

- Let's say you are testing an application. You have written a set of test cases.
- Now you run **one cycle of testing**. You find few bugs and report them to the development team. Development team fixes the bugs and reverts to you with the updated code.
- You again **execute the same set of test cases**. This time you find that few of the bug were still not fixed, and you report that back to the development team. They work on it and send an update to you.
- Once again, you execute the **same set of test cases and don't find any bugs**.
- Now in a **new release some changes were made in the application**. **You run the same set of test cases and they all pass.**
- But what you miss here is the **new bugs that may have introduced when the fix and new changes were applied**.
- **The old set of test cases are incapable of identifying these new bugs.**
- This is called Pesticide Paradox. To avoid this, you need to update your test cases with each cycle and add new cases to the old set.

University of Victoria

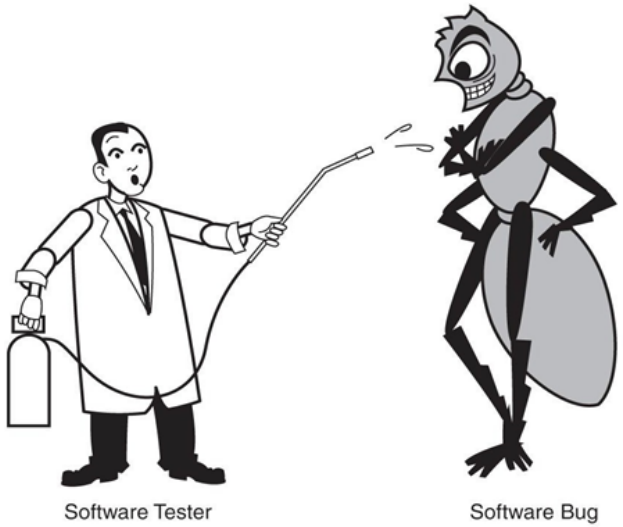
Overcome this “Pesticide Paradox

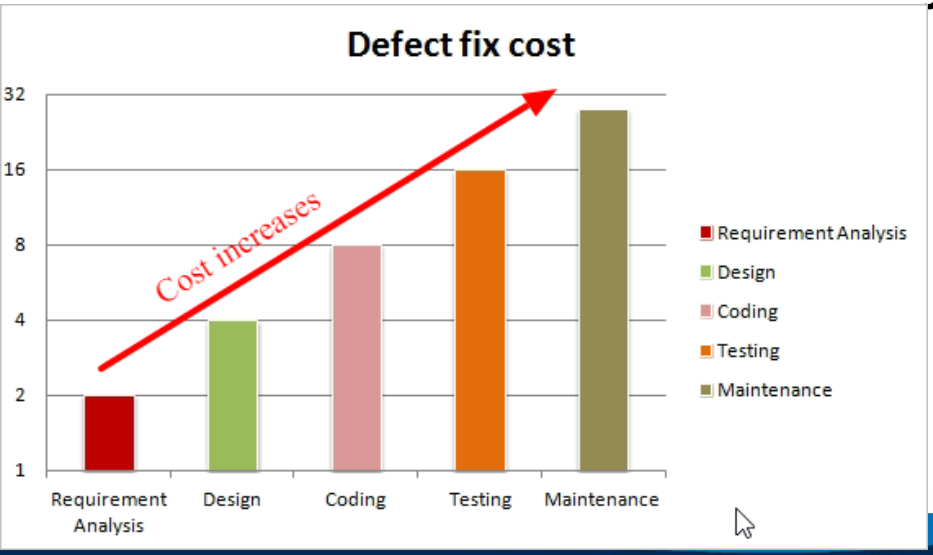
- **Regularly review** and, if necessary, **correct the tests already existent.**
- Refresh yourself with a pause and certainly **new ideas** will appear on your mind to create new tests; every time you have new ideas, write them down no matter where you are; it has already happened to me, having a task on a day and, at home, appeared in my mind new ideas in order to deal with a problem;
- **Request opinion to another tester**, even if they are less experienced tester than you, to review your tests or ideas. He/she can give you new ideas or only talk about something that will wake and refresh your brain;
- The opposite is also valid: if a tester requests your opinion do not refuse it: you will learn together;
- In case you are a tester dedicated to a specific area of a product, dare you to **explore other areas** in order to **discover new bugs**; you will **know more about the product too**.

University of Victoria

7 principles of software testing

1. Testing shows presence of defects, not their absence
2. Exhaustive testing is not possible
3. Early testing saves time and money
4. Defects cluster together
5. Beware of the Pesticide paradox
6. Testing is context dependent
7. Absence of errors is a fallacy





University of Victoria

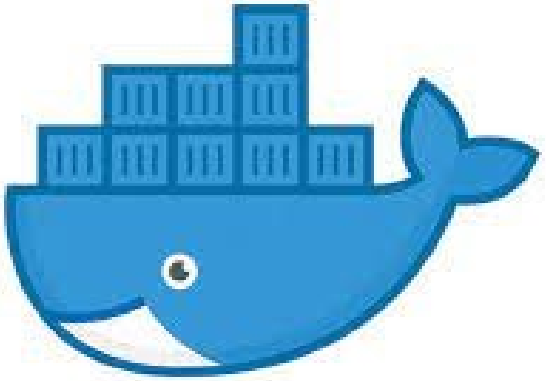
https://www.boxuk.com/insight/the-seven-principles-of-testing/

6. Testing is context dependent

- Testing is ALL about the context.
- The methods and types of testing carried out can completely depend on the context of the software or systems – for example, an **e-commerce website** can require different types of testing and approaches to an **API application**, or a **database reporting application.**
- What you are testing will always affect your approach.

University of Victoria

Container



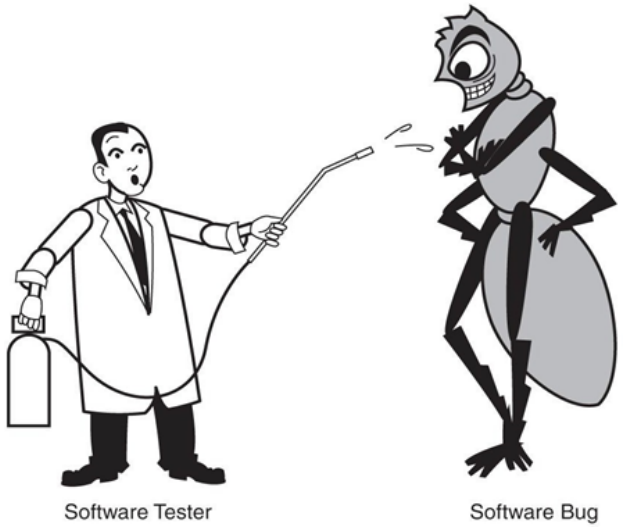
University of Victoria

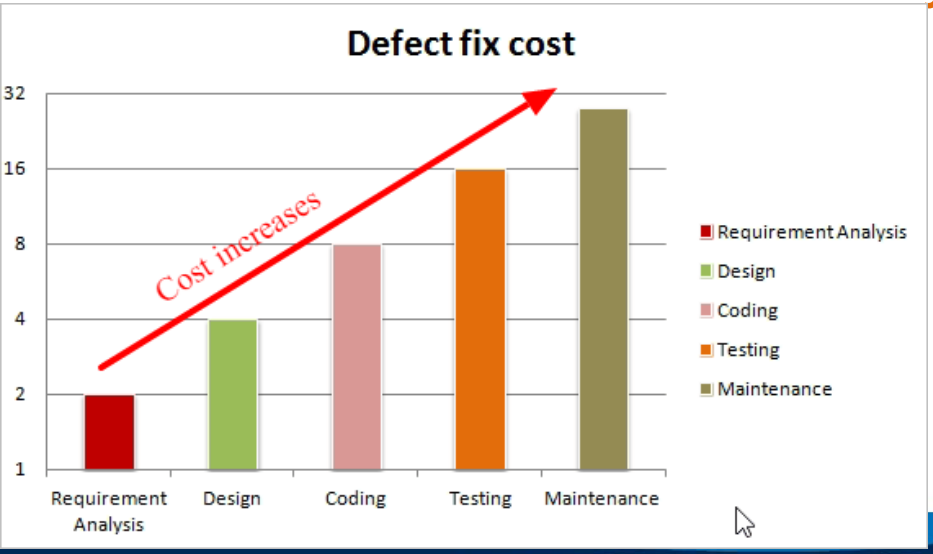
Testing is context dependent

- For example, a critical system software, such as medical or transports, needs more and different testing than an e-commerce website or a videogame. A **critical system** software will require **risk-based testing**, be demanding with **industry regulators** and specific test design techniques.
- On the other hand, an **e-commerce website** needs to go through rigorous **functional and performance testing**; for example, it should not have errors on calculations on **order's placement** and should load and work quickly on a **promotional campaign.**

7 principles of software testing

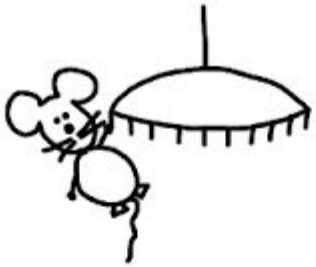
1. Testing shows presence of defects, not their absence
2. Exhaustive testing is not possible
3. Early testing saves time and money
4. Defects cluster together
5. Beware of the Pesticide paradox
6. Testing is context dependent
7. Absence of errors is a fallacy



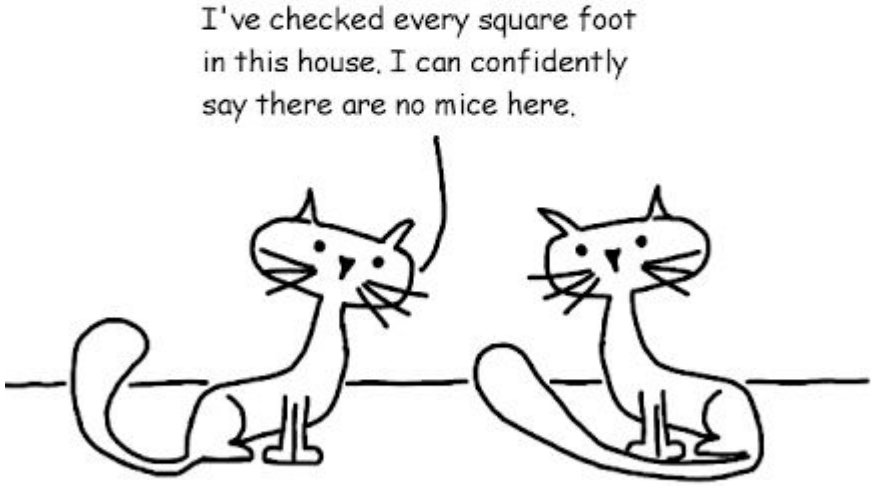


University of Victoria

https://www.boxuk.com/insight/the-seven-principles-of-testing/



7. Absence of errors is a fallacy



Absence of proof is not proof of absence.

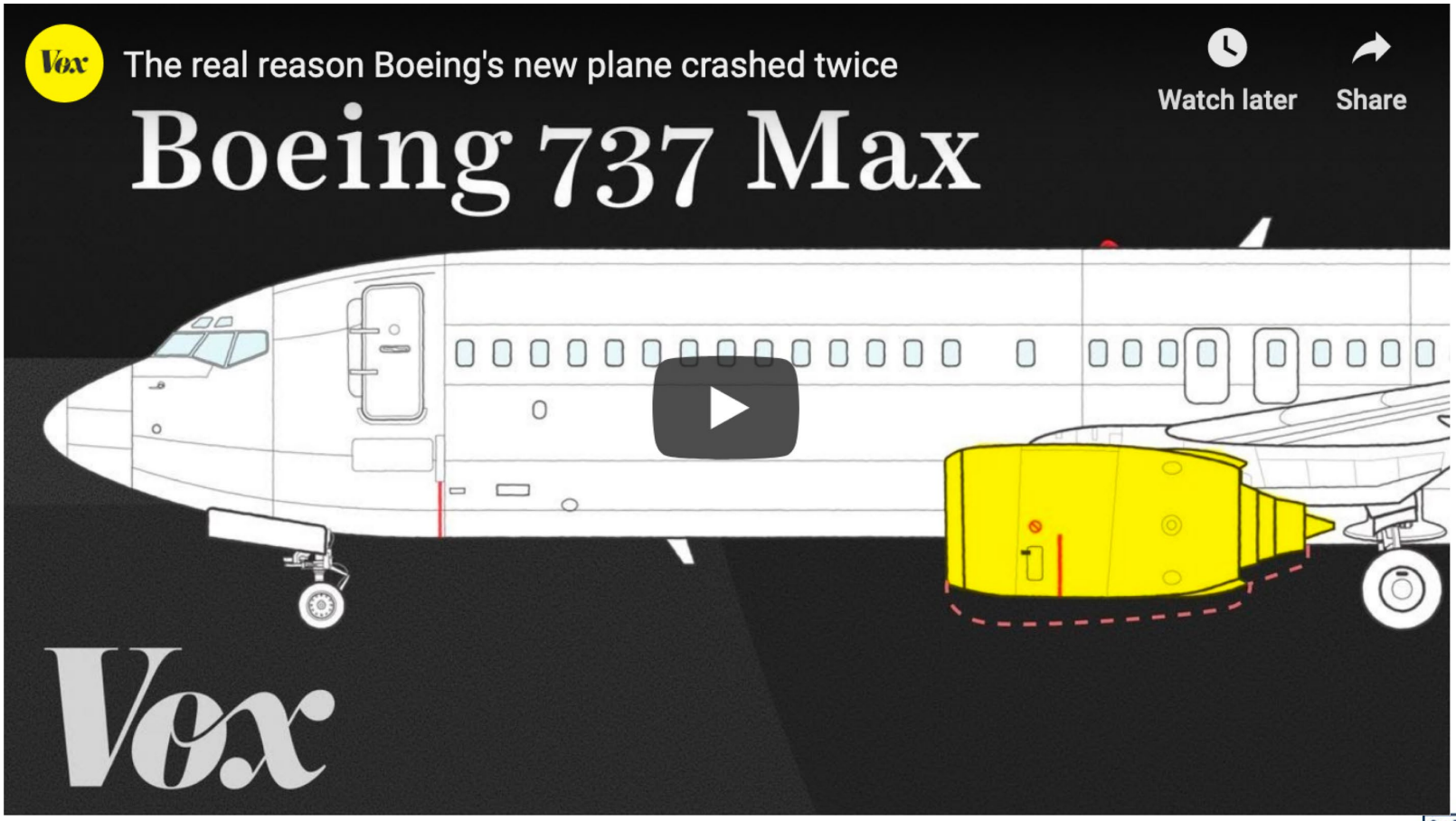
– William Cowper

University of Victoria

7. Absence of errors is a fallacy-"Building the software right versus building the right software."

- If your software or system is unusable (or **is not according to the user's requirements**) then it does not matter how many defects are found and fixed – it is still unusable.
- So, in this sense, it is irrelevant how issue- or error-free your system is; if the usability is so poor users are **unable to navigate,** or/and **it does not match business requirements** then it has failed, despite having few bugs.
- **For Example,** suppose the application is related to an e-commerce site and the requirements against "Shopping Cart or Shopping Basket" functionality which is wrongly interpreted and tested.
- Here, even finding more defects does not help to move the application into the next phase or in the production environment.

University of Victoria



https://www.youtube.com/watch?v=H2tuKiiznsY

University of Victoria