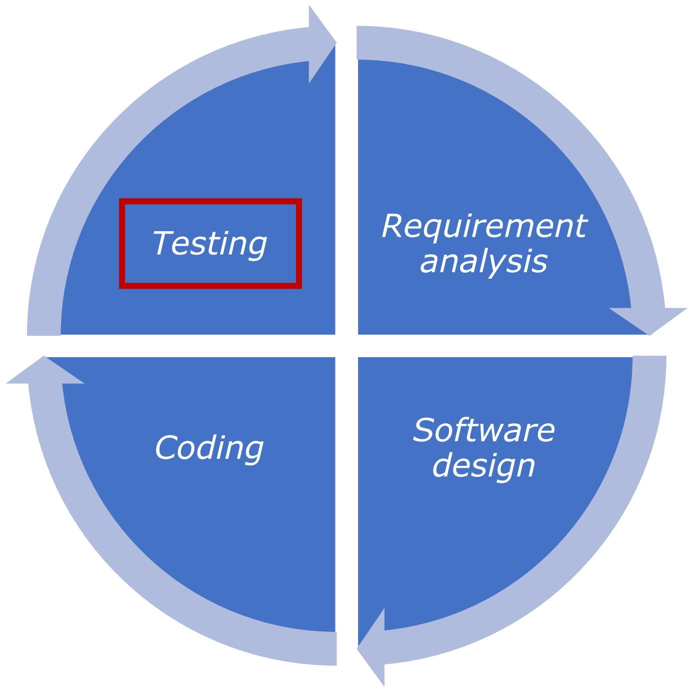
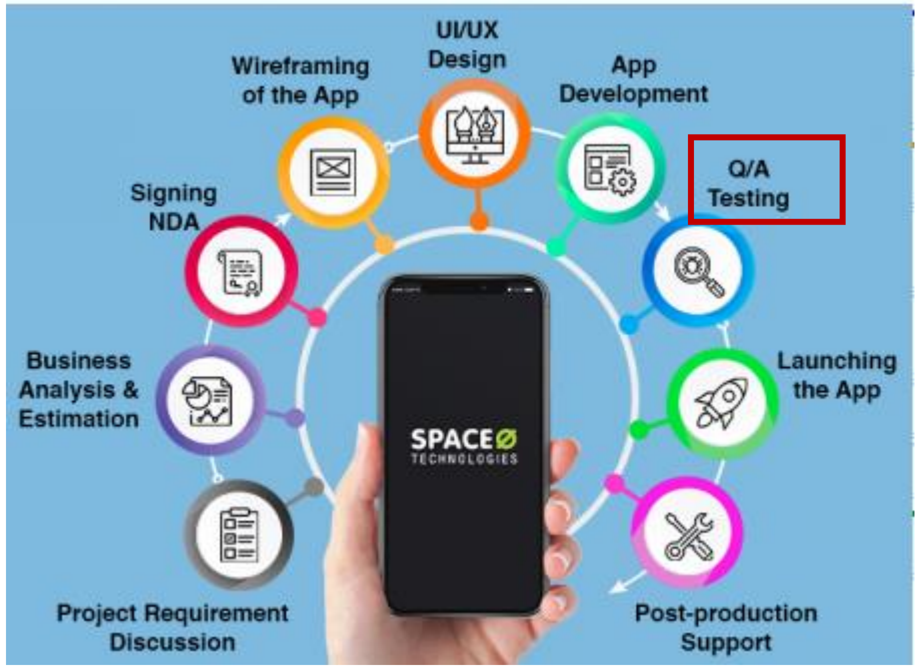
Last Session:
Software Testing

Karim Lekadir, PhD

Universitat de Barcelona
Dpt. Matemàtiques & Informàtica

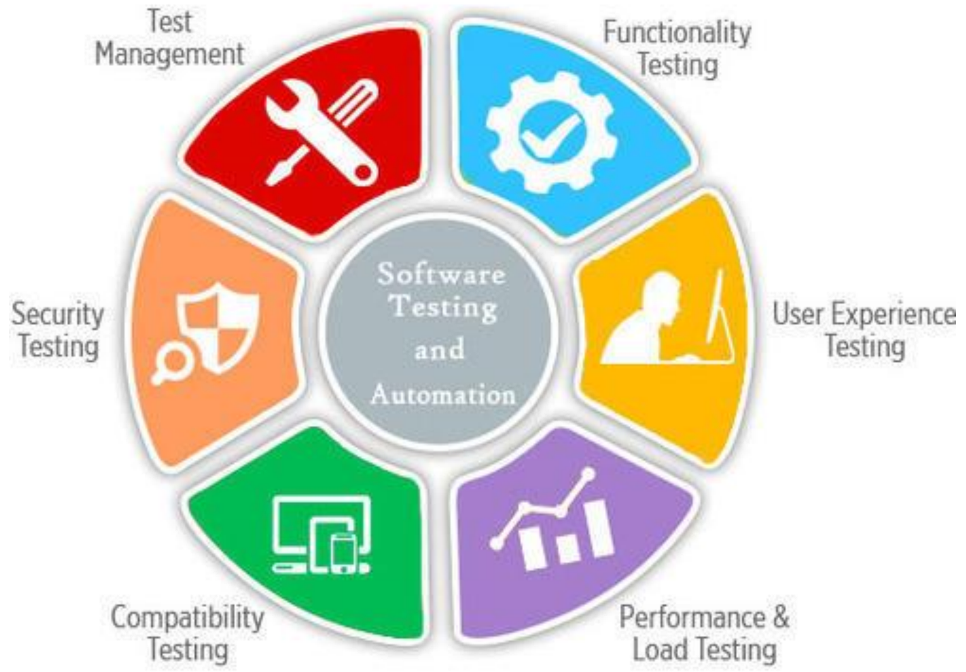
Development Cycle





Motivation

- Testing is a central part of software development to maximise quality and performance.
- It is important to plan the software testing procedures as soon as software development begins
- Good software companies have testing teams / experts



Real Examples

Search International edition

The Guardian

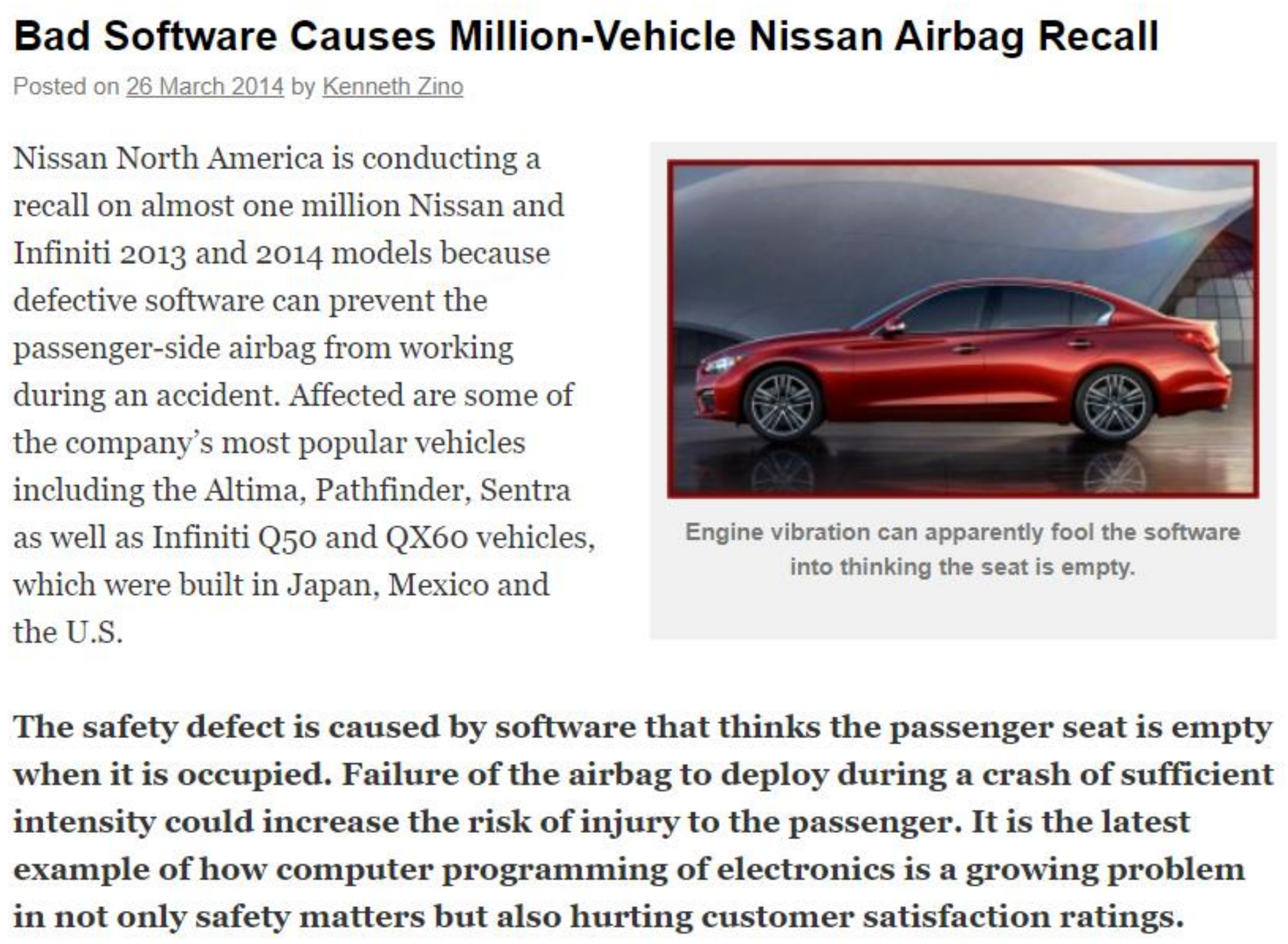
Amazon sellers hit by nightmare before Christmas as glitch cuts prices to 1p

Small businesses count cost of error in RepricerExpress software that resulted in thousands of items going for a song

▲ Boxes of goods at an Amazon warehouse. Photograph: Graeme Robertson for the Guardian

There were Christmas shopping bargains galore on Amazon's website over the weekend ... for about an hour. Because of a technical glitch, the prices of thousands of items crashed to 1p - giving eagle-eyed customers a pre-Christmas treat while leaving scores of small family-owned businesses nursing heavy losses, with some warning they could enter the new year facing closure.

Real Examples



Bad Software Causes Million-Vehicle Nissan Airbag Recall

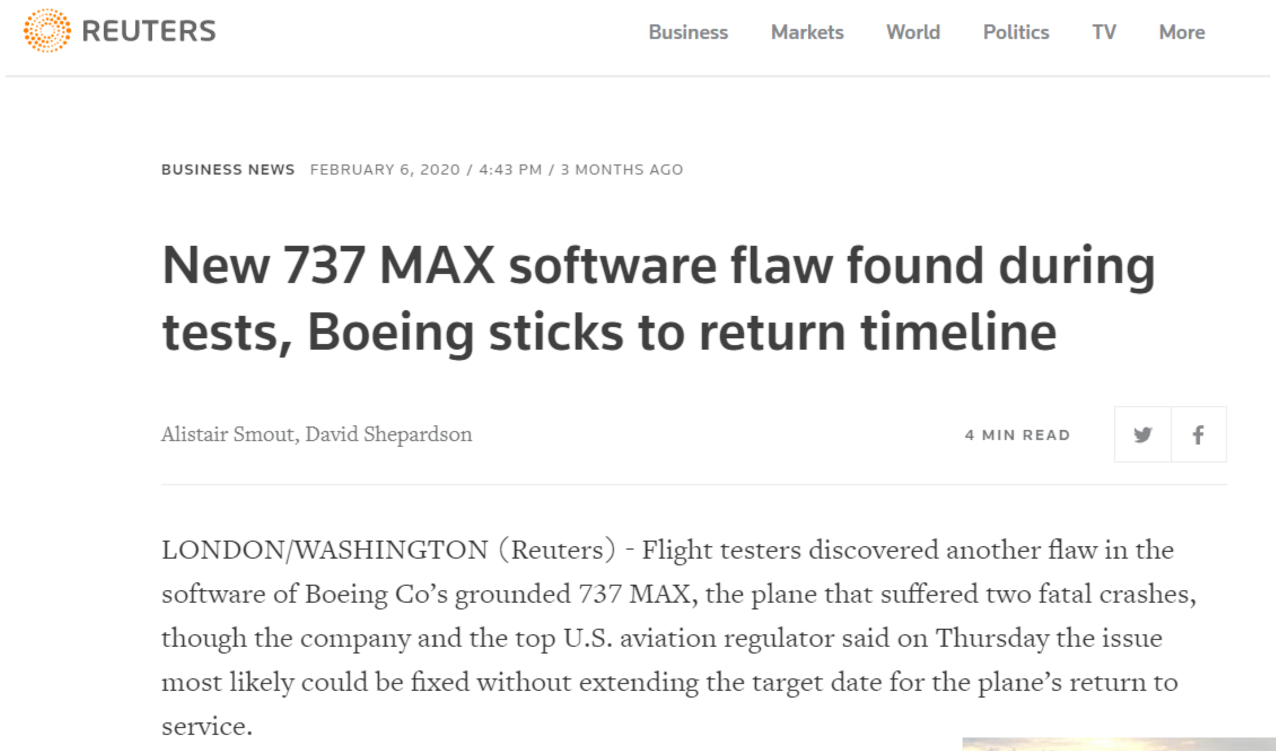
Posted on 26 March 2014 by Kenneth Zino

Nissan North America is conducting a recall on almost one million Nissan and Infiniti 2013 and 2014 models because defective software can prevent the passenger-side airbag from working during an accident. Affected are some of the company's most popular vehicles including the Altima, Pathfinder, Sentra as well as Infiniti Q50 and QX60 vehicles, which were built in Japan, Mexico and the U.S.

Engine vibration can apparently fool the software into thinking the seat is empty.

The safety defect is caused by software that thinks the passenger seat is empty when it is occupied. Failure of the airbag to deploy during a crash of sufficient intensity could increase the risk of injury to the passenger. It is the latest example of how computer programming of electronics is a growing problem in not only safety matters but also hurting customer satisfaction ratings.

Real Examples



REUTERS

Business Markets World Politics TV More

BUSINESS NEWS FEBRUARY 6, 2020 / 4:43 PM / 3 MONTHS AGO

New 737 MAX software flaw found during tests, Boeing sticks to return timeline

Alistair Smout, David Shepardson

4 MIN READ

LONDON/WASHINGTON (Reuters) - Flight testers discovered another flaw in the software of Boeing Co's grounded 737 MAX, the plane that suffered two fatal crashes, though the company and the top U.S. aviation regulator said on Thursday the issue most likely could be fixed without extending the target date for the plane's return to service.

Types of Software Testing

- More than 100 types, for example:

✓ Acceptance, Accessibility, Agile, Ad-hoc, Alpha, API, Automated, Beta, Benchmark, Black-box, Code-driven, Compatibility, Comparison, Component, Configuration, Compliance, Error-Handling, End-to-end, Endurance, Functional, Gray Box, Integration, Install/uninstall, Localization, Non-functional, Negative, Operational, Performance, Regression, Recovery, Requirements, Security, Scenario, Scalability, Stability, Storage, Stress, System, Upgrade, Unit, User Interface, Volume, White-box, Workflow

Types of Software Testing

- More than 100 types, for example:

✓ **Acceptance**, **Accessibility**, Agile, Ad-hoc, Alpha, API, Automated, Beta, Benchmark, Black-box, Code-driven, **Compatibility**, Comparison, Component, Configuration, Compliance, Error-Handling, End-to-end, Endurance, Functional, Gray Box, **Integration**, Install/uninstall, Localization, Non-functional, Negative, Operational, **Performance**, Regression, Recovery, Requirements, Security, Scenario, **Scalability**, Stability, Storage, Stress, System, Upgrade, Unit, User Interface, Volume, White-box, Workflow

Types of Software Testing

- More than 100 types, for example:

✓ Acceptance, Accessibility, Agile, Ad-hoc, Alpha, API, Automated, Beta, Benchmark, **Black-box**, Code-driven, Compatibility, Comparison, Component, Configuration, Compliance, Error-Handling, End-to-end, Endurance, Functional, Gray Box, Integration, Install/uninstall, Localization, Non-functional, Negative, Operational, Performance, Regression, Recovery, Requirements, Security, Scenario, Scalability, Stability, Storage, Stress, System, Upgrade, Unit, User Interface, Volume, **White-box**, Workflow

Types of Software Testing

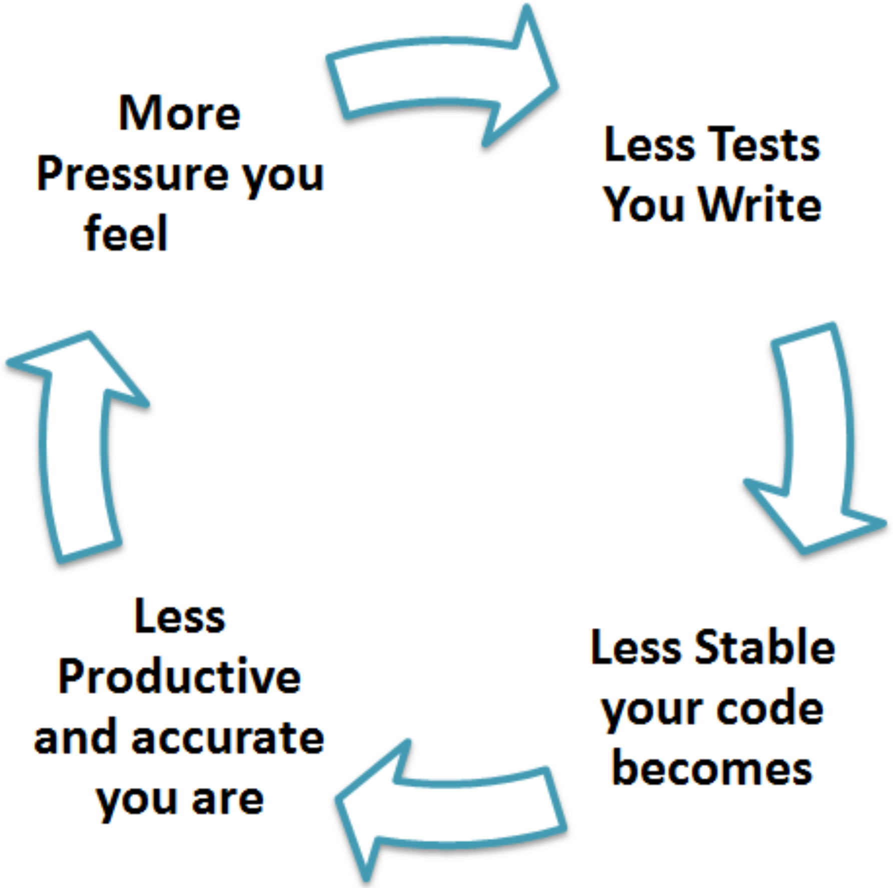
- White-box testing: Inside of the software, i.e. structure, design and code:
  - ✓ Unit testing
  - ✓ Integration testing
- Black-box testing: End-user perspective, i.e. expected results.
- Functional testing: Functions of the software, i.e. usability.
- Non-functional testing: Non-functional aspects such as interoperability, scalability, load/volume, etc.

Unit Testing

- Test each function, class and component of the code separately and specifically
- When: During development/coding
- Typical error: Doing minimal unit testing to save time. This leads to more defects and higher system testing
- Best approach: Write code for automated unit testing, e.g. by verifying the code for a range of cases (e.g. values including extreme values), then flag failed test cases.
- Re-run the test functions every time you modify the unit’s implementation.

Unit Testing

Avoid vicious cycles during testing


More Pressure you feel
Less Tests You Write
Less Stable your code becomes
Less Productive and accurate you are

Unit Testing

- Example 1: HeartApp
- Write code to test the function that reads blood pressure from a device
- Test cases:
  - ✓ Range of values for blood pressure
  - ✓ Test zero, negative and very high/low values
  - ✓ Test empty values (users did not use properly the device)
  - ✓ Test EU vs. American devices (ask manufacturers for potential errors to verify)

Unit Testing

- Example 2: HeartApp
- Write code to test the function that estimates risk
- Test cases:
  - ✓ Range of values for blood pressure, weight, heart rate, etc
  - ✓ Test zero, negative and very high/low values
  - ✓ Test missing values
  - ✓ Test errors in the input (e.g. age = 162 instead of 62)
  - ✓ Test extreme output values (very high or zero probability)

Integration Testing

- Important as different units are developed and tested by different team members
- Tests (1) the joint functioning of the units and (2) communication / interfaces between modules

✓ Method 1: <u>Big bang</u> approach, i.e. all modules at once

✓ Method 2: <u>Incremental</u>, i.e. test two modules together, than three, than four, etc.

- The incremental approach facilitates detection of errors, but does not allow for early prototyping

Integration Testing

- Example: HeartApp
- Write code to test the function that estimates risk, while reading data from medical devices
- Test cases:
  - ✓ Multiple devices and metric systems
  - ✓ Retrospective data (e.g. 100 cases from hospital)
  - ✓ Retrospective data + noise + outliers
  - ✓ Simulated data and groups (normal / abnormal)
  - ✓ Different combination of extreme values, exceptions, etc.
  - ✓ Verify outputs (e.g. high values)

Performance Testing

- Test whether (or to which extent) the software achieves the expected results (depending on the goal of the software)
- This often requires a dedicated test study with retrospective (existing) and prospective (new) data
- The process includes the definition of procedures, including the **data**, **users** and **testers**
- Performance testing will require performance **metrics, indicators & criteria**

Performance Testing

- Example: Heart App

✓ Data retrospective: 200 cases from Vall d'Hebron Hospital (100 with heart attack, 100 without)

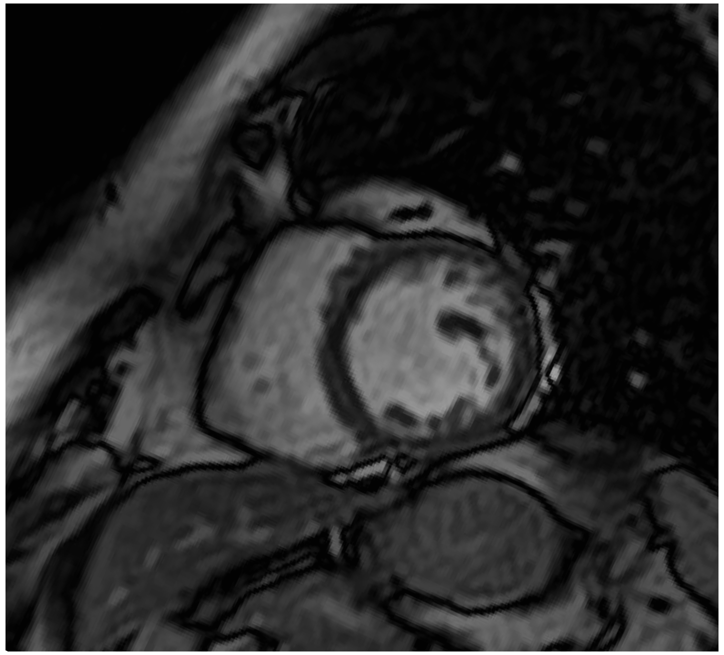
✓ Data prospective: 100 new individuals

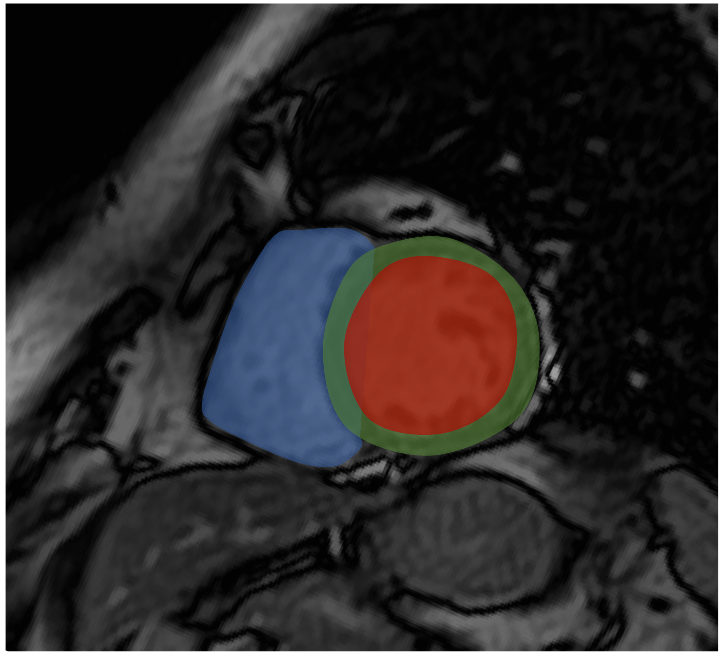
✓ Metrics: Accuracy of risk prediction (number of correct predictions / total number of cases)

✓ **Acceptance criteria**: Prediction accuracy > 80 %

Performance Testing

- Example 2: Software to delineate medical images





Performance Testing

- Example 2: Software to delineate medical images
- Study:
  - ✓ 10 cardiologists
  - ✓ Record time to delineate manually
  - ✓ Record time to delineate using automatic method (e.g. deep learning)
  - ✓ **Acceptance criteria**: Reduces time by 50% at least

Performance Testing

- Example 3: Software to compress files
- Study:
  - ✓ Different types of files (text, images, videos, etc)
  - ✓ Metric 1: Reduction in disk space
  - ✓ Metric 2: Time taken to compress the file
  - ✓ Metric 3: Time taken to decompress the file

Usability/Acceptance Testing

- Even if the software is performant, it is important to test usability and acceptance by the end-users:

✓ Do they like the aesthetics and design?

✓ User-friendliness: Is the software easy to use? Intuitive? Do the users understand it?

✓ If they like it, will they use it, how often and when?

Usability/Acceptance Testing

- Methods: (1) using observers, (2) by using surveys, (3) by recording user reactions (voice recording, face expression, screen activity, etc).

✓ Select the appropriate **users** for the tests

✓ Select the appropriate **testing environment** (e.g. home, university, work, hospital, etc)

✓ Identify **usability indicators**

Usability/Acceptance Testing

- Example: HeartApp - 10 patients or cardiologists in 2 hospitals (Vall d'Hebron + Hospital Clinic)
- Usability indicators:

✓ Average time to obtain risk prediction

✓ Ergonomics (e.g. total number of clicks, number of steps per task, system speed);

✓ Percentage of messages dismissed by the patient;

✓ Number of times user consults help services;

✓ Time to learn to use the system;

Usability/Acceptance Testing

- Example: HeartApp - 10 patients or cardiologists in 2 hospitals (Vall d'Hebron + Hospital Clinic)
- Usability questionnaires:

✓ Level of understanding of software by users (from 0 to 10);

✓ Satisfaction with the visual interfaces (from 0 to 10);

✓ Usefulness of error messages/alerts (from 0 to 10);

✓ Impact on clinician's productivity (low to high);

✓ Level of intention-to-use of the system (e.g. only when needed vs. full use).

Non-Functional Testing

- The software has a great performance, the users like it, <u>BUT</u>:
- ➢ Is it <u>compatible</u> with different IT systems, devices, etc?
- ➢ Is it <u>portable</u> to other systems?
- ➢ Is it <u>interoperable</u> with American/Japanese devices?
- ➢ Can it be <u>installed</u> in all countries? hospitals?
- ➢ Is it safe and <u>secure</u> (can people hack it)?
- ➢ Is it <u>scalable</u> to other domains, markets, areas, users, etc?

Scalability Testing

- Example: HeartApp
- ✓ Is it possible to add new predictors (e.g. voice, face expression)?
- ✓ How easy it is to add a new device, for example to estimate blood coagulation?
- ✓ How much time it takes to train HeartApp to predict stroke, instead of heart attacks?
- ✓ How easy it is to scale the software so it can be used by GPs?

Final Deliverable

- Three tested elements
  1. Code
  2. Functional
  3. Usability
- How did you test each?
- What results did you obtain?
- Were the tests successful?
- What are the identified limitations / weaknesses?

Question 1

- When is it best to perform software testing?

A. After coding is finished

[✓ B.] During and after coding

C. Before coding

D. After software design

Question 2

- How is unit testing best performed?

A. By looking at the code

B. By asking another person to check the code

C. By testing a range of input parameters

[✓ D.] By implementing dedicated unit testing functions

Question 3

- What is integration testing?

A. Testing multiple devices.

B. Testing multiple users.

[✓ C.] Testing all components of the code together.

D. Testing using multiple datasets.

Question 4

- What is functional testing?

A. Testing specific functions in the code

B. Testing the functions of the different users

C. Testing the function of the devices

[✓ D.] Testing the functions of the software

Question 5

- When is it best to plan software testing?

A. After coding is finished

B. During coding

[✓ C.] After requirement gathering

D. Before requirement gathering

Question 6

- Non-functional testing does not include?

A. Scalability

B. Compatibility

[✓ C.] Usability

D. Security

Question 7

- The order in which software testing is performed is:

A. Unit, functional, integration, non-functional

B. Unit, performance, non-functional, integration

[✓ C.] Unit, integration, functional, non-functional

D. Unit, integration, performance, usability

Question 8

- Which is not a method to perform usability testing?

A. Surveying / Questionnaires.

B. Recording behaviours

[✓ C.] Focus groups

D. Observation

Question 9

- Performance testing requires:

A. Performance metrics

B. Data

C. At least data and users

[✓ D.] At least data and metrics