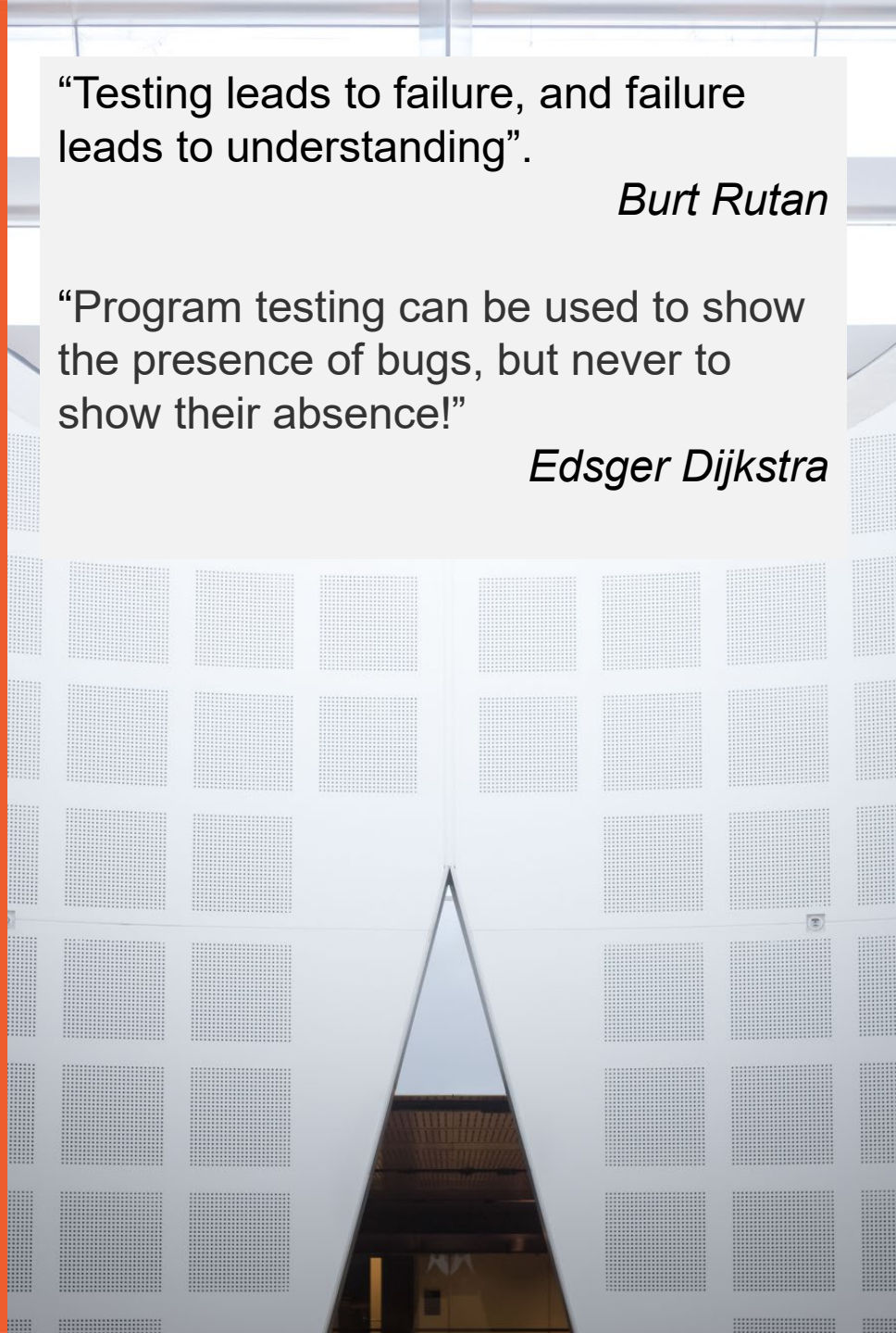
INFO5990: Professional Practice in IT

Week 7: Test Management

School of Computer Science

THE UNIVERSITY OF SYDNEY



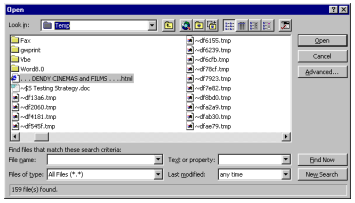
Software Testing

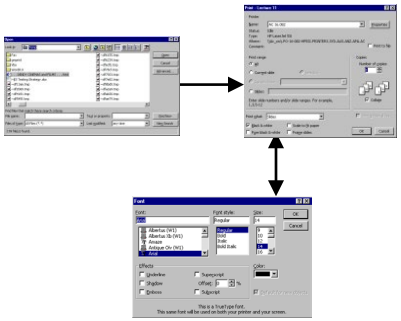
- A systematic process of evaluating software to:
  - Confirm it meets specified requirements (Verification).
  - Ensure it satisfies user needs (Validation).
  - Identify defects, bugs, or issues.
  - Assure quality and mitigate risks.
  - Evaluate performance, security, and usability.

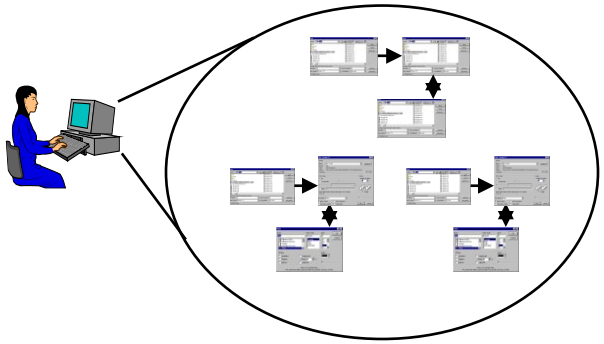
Testing Phases

Requirements Phase:	• **Requirements Review:** Testing begins by reviewing and validating the software requirements to ensure they are clear, complete, and achievable.
Planning Phase:	• **Test Planning:** Detailed test plans are created, outlining testing strategies, objectives, resources, and schedules.
Design Phase:	• **Test Design:** Test cases and test scenarios are designed based on the software design and requirements.
Development Phase:	• **Unit Testing:** Developers conduct unit testing to evaluate individual code units (e.g., functions or modules).
Integration Phase:	• **Integration Testing:** Testing focuses on interactions between integrated components or modules.
System Phase:	• **System Testing:** The entire software system is tested to ensure it functions correctly as a whole.
Acceptance Phase:	• **User Acceptance Testing (UAT):** End-users test the software to verify it meets their needs and expectations.
Release Phase:	• **Deployment Testing:** Testing is performed to ensure a smooth deployment of the software in the production environment.
Maintenance Phase:	• **Regression Testing:** Tests are rerun to verify that new changes do not introduce new defects.

Testing during development







Component Test

- To ensure that each component behaves ‘correctly’.
- Uses white-box testing to check each program function fully.

Integration Test

- To test interaction between related components.
- Focuses on interfaces between components.

System Test

- To ensure that the user requirements have been met.
- Focuses on usual business processes, and normal workflow.

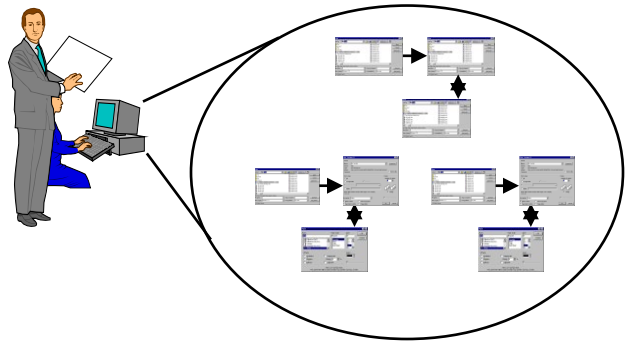
Top-Down vs Bottom-Up

Aspect	Top-Down Integration Testing	Bottom-Up Integration Testing
Testing Direction	Starts from higher-level modules and goes downward	Starts from lower-level modules and goes upward
Initial Components	Real higher-level components; stubs or drivers for lower-level components	Real lower-level components; stubs or drivers for higher-level components
Dependencies	May require stubs or drivers for lower-level components	May require stubs or drivers for higher-level components
Advantages	Early validation of system functionality; supports high-level design	Early detection of critical component issues; supports parallel development
Drawbacks	Dependencies on incomplete lower-level components; delayed system testing	May require extensive use of stubs and drivers; complex coordination

Functional vs Non-functional Requirments

Aspect	Functional Requirements	Non-Functional Requirements
Definition	Specify what the system should do and its features	Specify how the system should perform and its qualities
Focus	Features, actions, behaviors	Performance, security, usability, scalability, etc.
Example	User login, shopping cart functionality	Response time, encryption, user interface, concurrency
What they address	Core system functionality	System performance, user experience, technical aspects
User expectation	Defines the visible and usable functionality	Defines the system's quality attributes and constraints
Development impact	Guides feature design and development	Guides technical implementation and system design
Testing approach	Form the basis for functional testing	Form the basis for non-functional testing
Importance in design	Specifies features and use cases	Specifies performance goals, constraints, and limits

Testing during deployment



Performance Test

- To test system performance under maximum expected load.
- Simulates key processes under maximum load.

Soak Test and Stress Test

- To ensure that system is stable over extended period.
- Load increased until system fails. Checks effects of over-load

Acceptance Test

- Compares system functionality against agreed-on user requirements
- Carried out by client using scenarios, supervised by developer

Example -1

Question 1: Match the Testing Type to the Scenario

- 1. Running a test suite to check if a specific function within a module works as expected.
- 2. Verifying that different modules in the software application can communicate and interact seamlessly.
- 3. Testing the entire software application to ensure that all functional and non-functional requirements are met.
- 4. Evaluating how quickly the software responds to user actions under different levels of load.
- 5. Subjecting the software to extreme usage conditions to identify its breaking points and limitations.
- 6. Having end-users validate whether the software meets their needs and performs as they expect.

Example -1

Question 1: Match the Testing Type to the Scenario

- 1. Running a test suite to check if a specific function within a module works as expected. - Unit Testing
- 2. Verifying that different modules in the software application can communicate and interact seamlessly. - Integration Testing
- 3. Testing the entire software application to ensure that all functional and non-functional requirements are met. - System Testing
- 4. Evaluating how quickly the software responds to user actions under different levels of load. - Performance Testing
- 5. Subjecting the software to extreme usage conditions to identify its breaking points and limitations. - Stress Testing
- 6. Having end-users validate whether the software meets their needs and performs as they expect. - User Acceptance Testing

Example -2

Question 2: Match the Scenario to the Testing Type

- 1. Checking if the login button on a website redirects users to the correct dashboard.
- 2. Testing how well different components interact when a user submits a form and the data is processed.
- 3. Verifying that the software works correctly on different browsers and devices.
- 4. Measuring the response time of a web application when 1000 concurrent users access it.
- 5. Stressing a financial software with an exceptionally high volume of transactions to see when it fails.
- 6. Asking real users to try out a new mobile app and provide feedback on their experience.

Example -2

Question 2: Match the Scenario to the Testing Type

- 1. Checking if the login button on a website redirects users to the correct dashboard. - Unit Testing
- 2. Testing how well different components interact when a user submits a form and the data is processed. - Integration Testing
- 3. Verifying that the software works correctly on different browsers and devices. - System Testing
- 4. Measuring the response time of a web application when 1000 concurrent users access it. - Performance Testing
- 5. Stressing a financial software with an exceptionally high volume of transactions to see when it fails. - Stress Testing
- 6. Asking real users to try out a new mobile app and provide feedback on their experience. - User Acceptance Testing

Example -3

Question 3: Match the Testing Type to the Scenario

- 1. Verifying that each individual class in the codebase functions correctly and as intended.
- 2. Testing how well the payment gateway module interacts with the customer account module.
- 3. Ensuring that the software's search functionality, login process, and data retrieval work seamlessly together.
- 4. Analyzing how quickly an e-commerce platform loads pages and processes transactions during a flash sale.
- 5. Continuously subjecting the system to a sustained workload over an extended period to assess its stability and reliability.
- 6. Inviting a group of representative customers to test a new software release and provide feedback.

Example -3

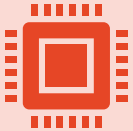
Question 3: Match the Testing Type to the Scenario

- 1. Verifying that each individual class in the codebase functions correctly and as intended. - Unit Testing
- 2. Testing how well the payment gateway module interacts with the customer account module. - Integration Testing
- 3. Ensuring that the software's search functionality, login process, and data retrieval work seamlessly together. - System Testing
- 4. Analyzing how quickly an e-commerce platform loads pages and processes transactions during a flash sale. - Performance Testing
- 5. Pushing an email server with a significantly high number of concurrent email requests to assess its performance limits. - Soak Testing
- 6. Inviting a group of representative customers to test a new software release and provide feedback. - User Acceptance Testing

Load test

Modelling the expected usage of a software program by simulating multiple users accessing the program concurrently.

Important for multi-user systems, often using client/server model, such as web servers.



Examples

a word processor or graphics editor could be tested on an extremely large document;

a financial package could required to generate a report based on many years' data.

A spreadsheet could be tested with maximum columns and rows

Soak Testing

Testing with a significant load extended over a significant period of time, to discover how the system behaves under sustained use

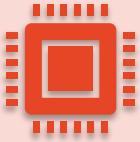
Examples

A system may behave as expected when tested for 1 hour, but fail when it is tested for 3 hours.

Can expose problems such as memory leaks or stack overflows.

Also rounding, accumulation, or compounding errors, which can cause the system to fail or behave unexpectedly after some time.

Stress Testing



Subjecting a system to unreasonable load, while denying it the resources needed to process that load, (RAM, disc, mips, interrupts, etc).

Examples:

Stressing the system to breaking point to determine whether the breakdown is potentially harmful.

Emphasis is on robustness, availability, and error handling, especially when under a load that is heavier than normally expected.

Will it perform 'acceptably' in all situations?

Particularly important for "mission critical" software, such as medical, space, defence applications

It is desirable for the system to 'degrade gracefully'.

Load VS Stress Vs Soak

Test Type	Purpose	Test Scenario	Example Question
Load Test	Assess performance under expected load conditions	Simulate concurrent users or transactions within expected limits	Can the system handle 1000 users concurrently?
Soak Test	Evaluate system stability over time	Sustain load over an extended period (hours/days)	How does the system perform after 72 hours of continuous usage?
Stress Test	Identify breaking points and recovery behavior	Push system beyond its limits, observe failures and recovery	What happens when the system receives 10 times the normal traffic?

Why is it hard?

- Ariane 4 rocket

European Space Agency (ESA).

- Ariane 5 rocket

Same software. So, what went wrong?
Fitting 64-bit Inertial Reference System (velocity and altitude) to 16-bit

Nectar Card Fiasco : Tuesday, 17 September 2002

http://news.bbc.co.uk/1/hi/business/2268797.stm

- Customer loyalty scheme, set up jointly by Sainsbury's, BP, Barclaycard and Debenhams: intended to rival 'Air Miles' loyalty scheme.
  - Offered a reward of 100 points bonus if you registered via internet.

    500 points would get you a 'Big Mac'!

- As the deadline,17 September, approached millions, hoping to sign up online, found they were locked out.
- They tried, again and again! UNTIL …

The system collapsed!

- Approaching the deadline the website was getting 10,000 hits an hour!
- The sponsors were forced to pull tens of millions of pounds worth of advertising, hundreds of TV spots and press ads.
- *"The online operation was simply taken by surprise by the demand"*. Ian Barber, Barclaycard

  "All I can think of here is that marketing has not been properly communicating with IT. To send this volume of letters out driving people to the website and not have the capacity in place is a serious flaw".
  Andrew Didcott, Internet expert)

What do you think went wrong?

- Was it a problem of testing?
  - The functionality of the system had been tested, but not under load
  - Performance testing had not anticipated the level of user load
  - Nobody knew that the system would fail completely under pressure

- Was it *really* a problem of testing?
  - Why wasn't it tested at the right level?
    - Should it have been tested with 10,000 hits/hour?
    - But what if it then had 50,000 hits/hour? 500,000 hits/hour?
  - Test management….

Another testing story

- A global grocery and general merchandising retailer
  - Grocery market leader in the UK
  - 702 stores
  - 30% market share
  - Stores in 14 countries
- In 2002 new online shopping system was to be rolled out
- Management demanded ‘no glitches’

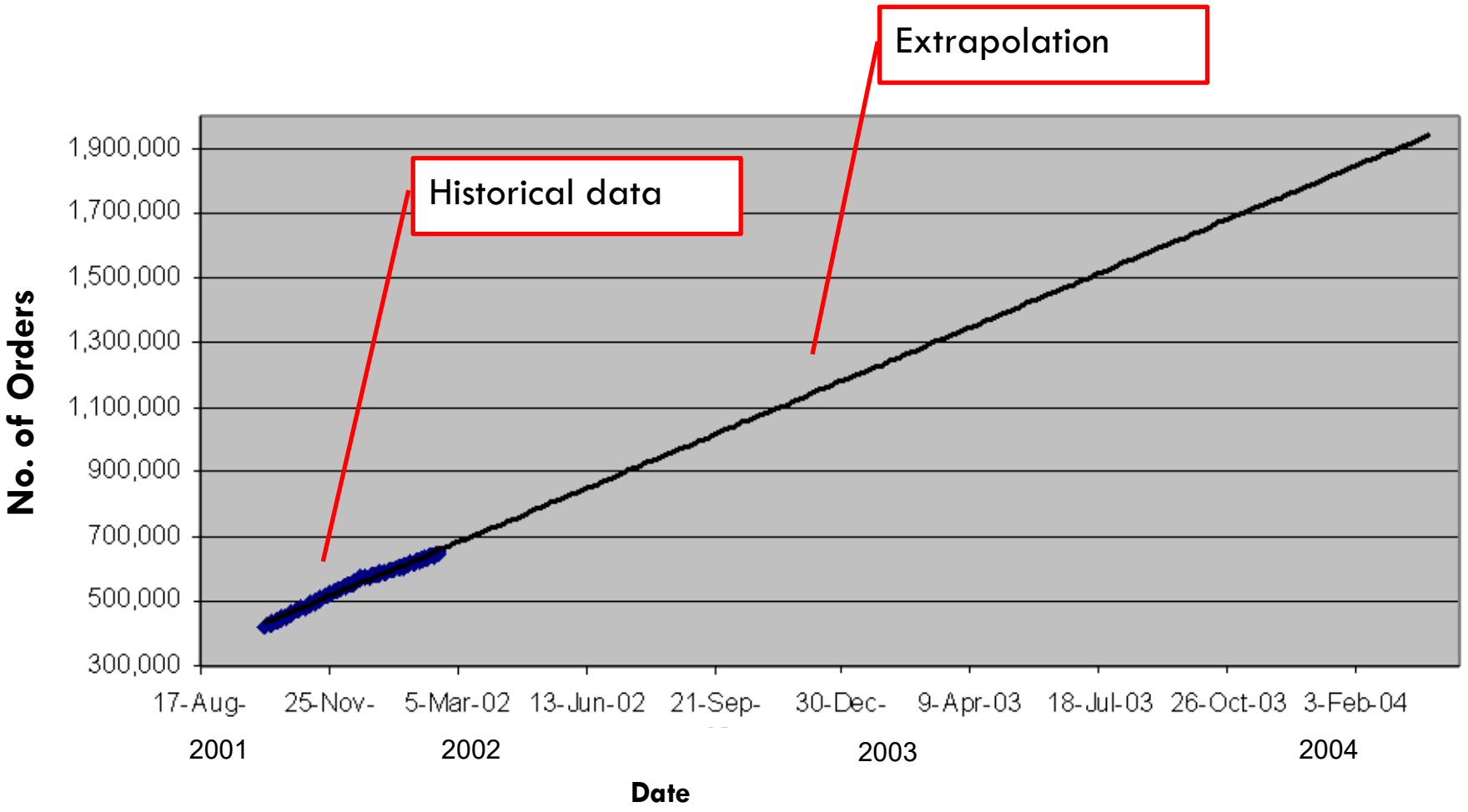
The test plan

- Total budget for testing, £1 million
- Testing to be carried out off line (so as not to interfere with live system)
  - Capacity model: to simulate user load two years into future
  - Usage model: typical mix of tasks
  - Test database: full-sized database, since size affects performance
  - To include soak and stress testing

Test data and test scenarios

Estimating target load

Extrapolating historical data: Oct 2001 – Apr 2002



Determining target load

- Historical data Oct 2001 – Apr 2002
- Trend extrapolated to Feb 2004 gave target 48,655 orders per week
- Corporate plan specified 341,642 orders for final 4 weeks of year = 85, 410 pwk
- +25% for Christmas rush
- +10% for contingency
- Target load = 117,439 orders per week

Another example

- Australian census 2016

- What went wrong?

"On 9 August 2016, at about 7.30pm, the census website was hit by distributed denial-of-service attacks. The site was flooded with traffic in an attempt to overload it and shut it down. Along with a hardware failure, and a false report that data was at risk, there was widespread concern about the security of the census. Just after 8pm the ABS took the website offline for 40 hours.

In his memoir, A Bigger Picture, the former prime minister Malcolm Turnbull described the event as a "humiliating debacle for a government that was promoting innovation, agility and the promise of the digital era".

Initially, the ABS and IT contractor IBM stated there had been a "massive cyberhack" and implied it had been the work of a foreign state. But as Turnbull noted, the DDoS attacks were "quite modest in scale" and a result of a failure on IBM's part to deliver on its contractual obligations around DDoS protection.

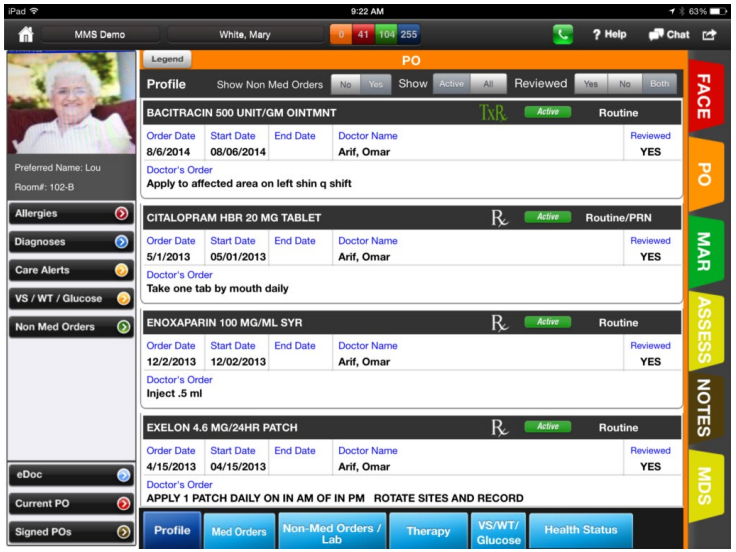
The Guardian Newspaper, 11/7/2021
https://www.theguardian.com/australia-news/2021/jul/11/censusfail-hangs-over-australian-bureau-of-statistics-as-it-prepares-for-2021-survey

- The Tesco online shopping system performed without a hitch since it went live in May 2002
- In terms of revenue, Tesco is one of the 10 biggest retailers in the world.

Another case study: Mobile MedSoft QA

THE CLIENT

"Mobile MedSoft is a leading software developer in the Healthcare market with mobile apps and cloud computing technologies that reduce office and paperwork to allow practitioners and nurses to focus on patients. Mobile MedSoft's software suite and services are increasingly employed in the pharmaceutical and long-term care market as transitions to mobile healthcare continue to rise. Chosen by Microsoft as a mission-critical partner, Mobile Medsoft needed a responsive software testing partner".



Another case study: Mobile MedSoft QA

THE QA CHALLENGES

- **Application Complexity**: Mobile MedSoft's iPad app is very complex due to its multitude of inherent user scenarios in various contextual environments.

- **Environment Complexity**: difficulty in keeping pace with regression testing across the many iOS updates and versions, as well as the multitude of devices. Many of Mobile MedSoft's clients use each of the various types of iPads with different resolutions and operating systems.

- **Agile Development:** With so many possible use cases with various steps and permutations, as well as utilizing Agile development with 1-2 week release cycles, Mobile MedSoft was becoming overwhelmed with their manual regression testing demands and was struggling to maintain their high software quality standards.

Another case study: Mobile MedSoft QA

THE XBOSOFT SOLUTION

XBOSoft developed and implemented an overall mobile test strategy built around an automation platform. The partnership enabled Mobile MedSoft to benefit from a wide range of software testing solutions:

- **Full Regression and Test Management:** XBOSoft performs rapid regression on Mobile MedSoft's iOS application MedTablet.

- **Automated Testing:** XBOSoft continues to push Automated Mobile Functional Testing across Mobile MedSoft's Healthcare mobile apps. Mobile MedSoft chose XBOSoft as their software testing partner in 2009 for their deep domain experience and technical expertise in automated functional testing on the mobile platform.

- **Usability Testing:** XBOSoft is continually providing input beyond functional defects where the application can provide better UX and user satisfaction. We continuously make recommendations for improved navigation and user flows that increase simplicity and task completion rates while reducing user errors.

- **Performance Testing:** XBOSoft understands that Performance is a key part of the user experience on mobile, the testing requirements from Mobile MedSoft have not only grown due to it's increasing user base but also in sophisticated feature demands. XBOSoft provides load and performance testing to make sure the application continues to improve.

Another case study: Mobile MedSoft QA

THE XBOSOFT SOLUTION

- **Agile Testing Collaboration**: With Agile development and compressed time box development cycles of 1 to 2 weeks, pressures mount on regression testing as functionality continues to increase. XBO's collaboration model in working with Agile development teams has a record of proven success using communication systems and methods that simulate working together in the same office.

RESULTS

- With tests executed and results reported with every software build via their test automation suites, Mobile MedSoft has been able to steadily optimize and improve the quality of it's iPad app as it consistently implements new features and technology.

- Ultimately this has resulted in relieving manual testing efforts by 50% while simultaneously enabling more exploratory testing for new features.

- Not only has Mobile MedSoft been able to reduce manual effort, cadence has also been increased as new builds are released 100 percent faster with increased velocity and improved developer/QA processes.

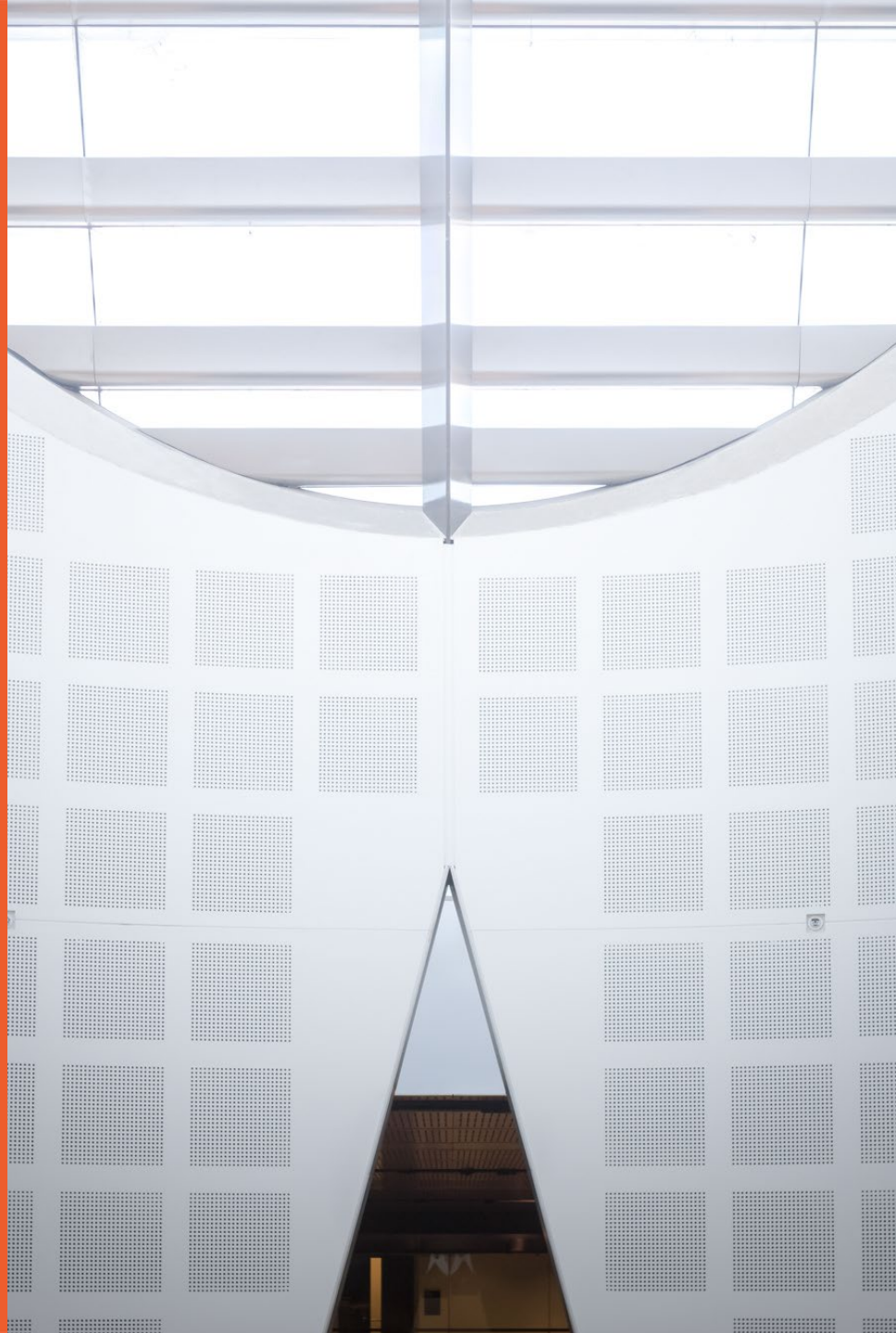
Reminders

- Check Ed regularly!

- Attend tutorials – and engage!
  - (In general, this has been fantastic this semester)
  - Updated approach to participation assessment

INFO5990: Professional Practice in IT

Week 8: Discussion



THE UNIVERSITY OF SYDNEY

Questions

1. What is the difference between audits and testing?
2. What is the biggest challenge in managing testing?
3. How do you know when you have done enough testing?
4. Is it better for developers to test their own code?
5. Who should write the test cases?
6. How do you handle testing in a changing environment?
7. Are the testers to blame when a system fails?
8. When is the right time to establish a testing strategy?

- https://www.kualitee.com/test-management/test-management-2018-top-challenges-opportunities-practices/
- https://techtesters.com/the-expectations-and-challenges-facing-a-modern-day-test-manager/
- https://blog.deviniti.com/atlassian/software-testing-challenges/
- https://www.sealights.io/agile-testing/agile-testing-8-principles-7-challenges-and-how-to-master-them/

What is the biggest challenge in managing testing?

- The foremost challenge in managing testing within software development lies in effectively balancing the need for thorough quality assurance with the constraints of time, especially in the face of changing requirements, scope expansion, and resource limitations. The pressure to meet tight deadlines often compromises testing coverage and overlooks potential defects. Maintaining clear communication and collaboration among development, testing, and other teams is essential, alongside addressing regression testing, test automation, and realistic test data management. Successful management entails adopting best practices, early testing involvement, and resource allocation, ultimately striking a harmonious equilibrium between speed and quality in the testing process.

How do you know when you have done enough testing?

- Knowing when to stop testing in software engineering is guided by specific criteria. This includes reaching the desired benchmarks for code coverage and functionality, observing a decline in the bug discovery rate, addressing high-severity open bugs, and concluding testing periods like beta or alpha testing. These criteria help ensure software stability, functionality, and quality. However, the decision is contextual, balancing project objectives and timelines while acknowledging that perfect testing is challenging.