SOFTWARE TESTING

Doug Hoff
Centriq Training

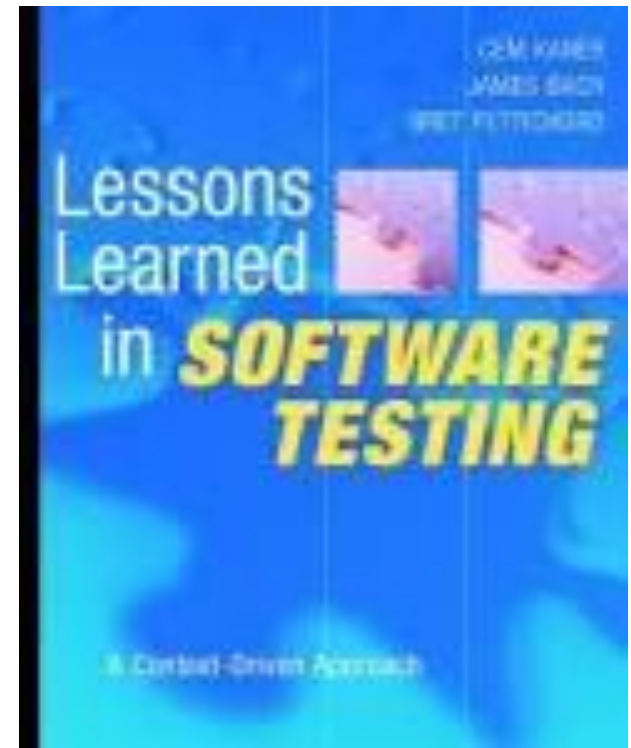# Class materials

- Workbook
  - ***Principles of Software Testing***
    by Doug Hoff, 2013
- Book
  - ***Lessons Learned in Software Testing***
    by Cem Kaner, James Bach, and Bret
    Pettichord, John Wiley & Sons. 2002

History of Bugs

World's Biggest Data Breaches

Selected losses greater than 30,000 records

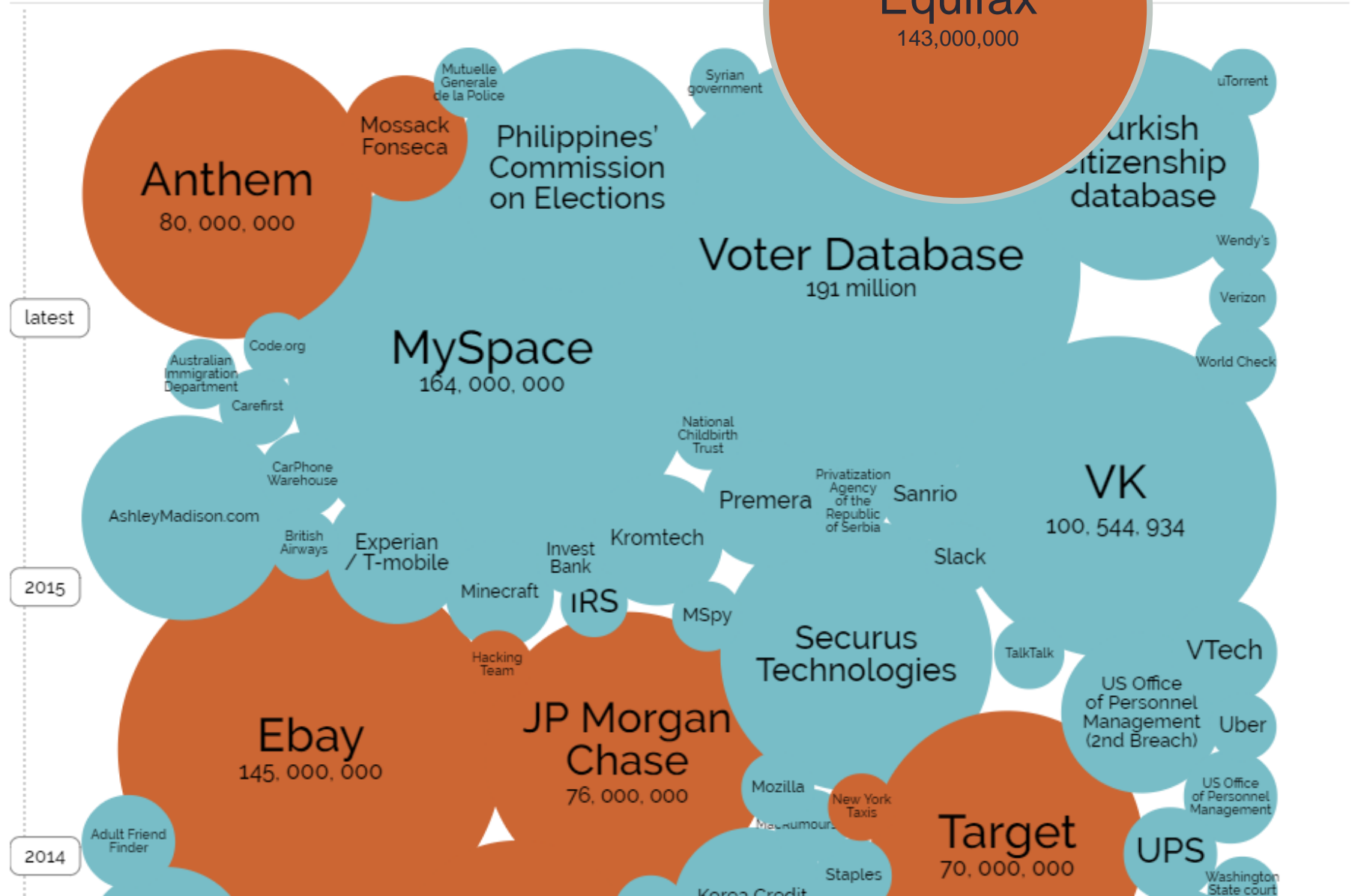(updated 11th July 2016)

interesting story

Equifax
143,000,000

Mutuelle Generale de la Police

Mossack Fonseca

Philippines' Commission on Elections

Syrian government

...urkish ...itizenship database

uTorrent

Anthem
80, 000, 000

Wendy's

latest

Code.org

Voter Database
191 million

Verizon

Australian Immigration Department

MySpace
164, 000, 000

World Check

Carefirst

CarPhone Warehouse

National Childbirth Trust

VK
100, 544, 934

AshleyMadison.com

Premera

Privatization Agency of the Republic of Serbia

Sanrio

British Airways

Experian / T-mobile

Invest Bank

Kromtech

Slack

Minecraft

2015

IRS

MSpy

Hacking Team

Securus Technologies

TalkTalk

VTech

US Office of Personnel Management (2nd Breach)

Uber

Ebay
145, 000, 000

JP Morgan Chase
76, 000, 000

Mozilla

New York Taxis

US Office of Personnel Management

Adult Friend Finder

...acRumour...

Target
70, 000, 000

UPS

2014

Staples

Korea Credit

Washington State court

Root cause of major bugs

- … the most notorious bugs in the history of software development were all unit bugs.
Dr Boris Beizer

Testing slows velocity?

- Codebases without tests manifest themselves in teams that are stressed and overworked due to an ever-increasing workload and firefighting. Velocity goes down over time. Meanwhile, I've never known a team with thorough test coverage that delivered *slower* than a team without automated tests. In fact I've observed teams that had no tests and crunched constantly, added tests and became predictable and successful, then removed the tests after idiotic leadership decisions to artificially increase velocity, and watched their velocity drop way down once the testing infrastructure, and especially culture, fell into disrepair.
- -- Rob Galanakis
-

How the software industry tries to prevent bugs

- Better tools
  - Visual Studio
  - Eclipse

- Better languages
  - Java is better than C
  - Microsoft builds better languages in general
    - Created TypeScript to improve on JavaScript
    - Created C# to improve on Java

Bugs

- Mariner I – 1962
  - Destroyed by range officer when flight became erratic
  - Combination of errors
    - Requirements documentation error
    - Hardware failure

- Therac-25 medical linear accelerator 1985-1987
  - Massive overdoses of radiation
  - **Cause**: race condition in software

AT&T bug - 1990

- Failure of 1/3 of US telephones
- **Cause**: Maintenance code release did not have regression testing done

Patriot Missile Failure – 1991

- System failed to intercept missile killing 29 US soldiers during Gulf War
- **Cause**: time calculation errors

The DCS bug – 1991

- Failure of 1/3 of US telephones (again)
- Cause: Code typo

The Intel Pentium chip - 1994

- Incorrect floating point division which would cause Excel to calculate wrong about once a month
- **Cause**: unit tests were not run for all table values

Ariane 5 - 1996

- European rocket and four satellites ($500 million) blew up.
- Project cost $7 billion
- **Cause**: system test cancelled due to budget reasons
- Bug: converting a double to a short
    - 64 bit double = -1,085,102,592,571,150,096
      1111 0000 1111 0000 1111 0000 1111 0000
      1111 0000 1111 0000 1111 0000 1111 0000
    - 16 bit short = -61,680
      1111 0000 1111 0000
- **Solution**: unit test

Mars Climate Orbiter - 1999

- Satellite crashed into Mars
- **Cause**: System - requirements error
- **Solution**: system test

Northeast blackout - 2003

- 55 million people lost power causing 11 deaths.
- **Trigger**: System- race condition (thread deadlock)
- **Cause**: combination
  - System-– thread analysis incomplete
  - Business – failed alarm system, backup failed
  - Business - lack of investigation of an incident report
- **Solution**: system test

Southwest blackout - 2011

- 7 million people lost power
- **Cause**: System- analysis incomplete - incorrect failure modeling
- **Trigger**: Business - hardware process not followed

2016

- Yahoo! Data breaches
  - https://en.wikipedia.org/wiki/Yahoo!_data_breaches
- Jeep Cherokee entertainment system hack
  - https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=hacker%20jeep%20software
- DDOS attack on East Coast DNS
  - https://www.wired.com/2016/10/internet-outage-ddos-dns-dyn/

What is a race condition? (thread communication)

Account holder 1

Enters bank
Asks for $100
Checks account balance ($200)
Receives $100 (balance = $100)
Leaves bank

Enters bank again
Asks for $100 more
Checks account balance ($100)
Receives $100 (overdrawn)
Leaves bank

Account holder 2

Enters another bank
Asks for $100
Delay
Checks account balance ($100)
Delay
Receives $100 (balance = $0)
Leaves bank

Another race condition - thread resource contention

Cook 1
making cookies

rolls the mixing machine over to the dry goods to get the flour

passes Cook 2 but doesn't notice

can't find flour but decides to wait for it to return
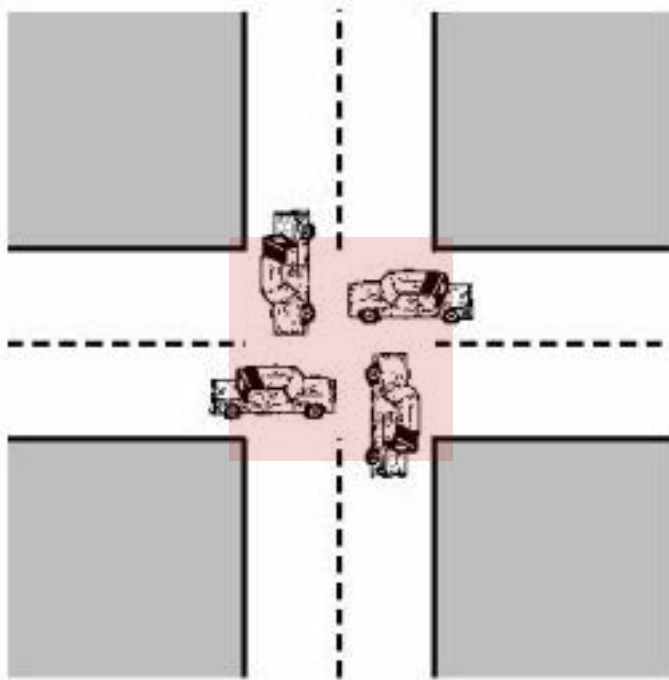
Cook 2
also making cookies

rolls the flour bin over to the prep area where the mixer is

passes Cook 1 but doesn't notice

can't find mixer but decides to wait for it to return

A deadlock

Another race condition - thread resource contention
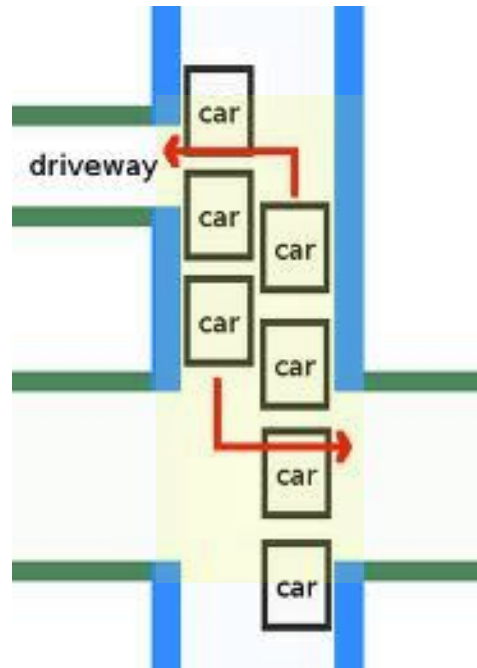
Race condition solutions

- Ask programmers where threads are used
- Ask programmers to provide delays in threads while testing
  - Delays cause more probable thread contention

Exercise - Northeast blackout

- What was the root cause of the system failure?
- Who is at fault here: the designer, the programmer, the tester, or the management? If you had one person to fire as a scapegoat, who would you choose?
- What was the risk in terms of effect and probability that was the result of a design choice to allow the possibility of a race condition?
- Was management right in assigning more weight to preventing hackers than ensuring quality? Why?

QUALITY ASSURANCE

Provide evidence of the fitness for use of the total software product

Activities

- software testing
    - verify and validate application lifecycle activities
    - test execution
- quality control
    - observe whether requirements and standards are being met
    - inspections and reviews
- software configuration management
    - labeling, tracking, and controlling changes in software
    - Version control

Attitude

- **destructive** - dirty
  - you are looking for problems

- **constructive** - clean
  - you are trying to build in quality in the long run

Software quality assurance plan

- an outline of quality measures to ensure quality levels

- covers
  - reviews and audits
  - standards and compliance
  - inspections, follow-ups, and approvals
  - SCM summary if covered elsewhere
  - software and hardware used

- umbrella document that can contain
  - software test strategy
  - software test plan

ISO 9000

- used for total quality management programs (TQM)
- the fundamentals of quality management systems including eight management principles
- History
  - US DOD standard - 1959
  - NATO standard - 1969
  - British standards - 1974 - 1979
  - ISO proposed - 1979
  - ISO published - 1987
- ISO 9001
  - requirements that organizations wishing to meet the standard have to fulfill

Deming's 14 quality principles

- Dr. Edward Deming - US govt. statistician
- Deming Prize - Japanese quality improvement award
- 85% of the cost of quality is under management's control and is management's responsibility
- The Deming Institute - https://www.deming.org

Deming's 14 quality principles

- 1 Create constancy of purpose
  - create plans
- 2 Adopt the new philosophy
  - rethink
- 3 Cease dependence on mass inspection
  - test at every step
- 4 End the practice of awarding business on price tag alone
  - make relationships

Deming's 14 quality principles

- 5 Improve the system of production and service constantly and forever

  - continually improve

- 6 Institute training and retraining

  - keep learning

- 7 Institute leadership

  - managers should initiate action

- 8 Drive out fear

  - are inspections used in a negative way?

- 9 Break down barriers between staff areas

  - encourage teamwork

Deming's 14 quality principles

- 10 Eliminate slogans, exhortations, and targets for the workforce
  - Workers can't use slogans to do a better job

Deming's 14 quality principles

- 11 Eliminate numerical goals
  - no quotas, no shame

Deming's 14 quality principles

- 12 Remove barriers to pride of workmanship
  - delegate real authority
- 13 Institute a vigorous program of education and retraining
- 14 Take action to accomplish the transformations

BASICS OF TESTING

Bugs

- **Failure** = Expectations not met
- **Fault, defect, bug** = software failure before deployment
- **Failure, escape** = user expectations of software after deployment

- ITIL incident – a reduction in service
- ITIL problem – a cause related to a set of incidents

Bugs
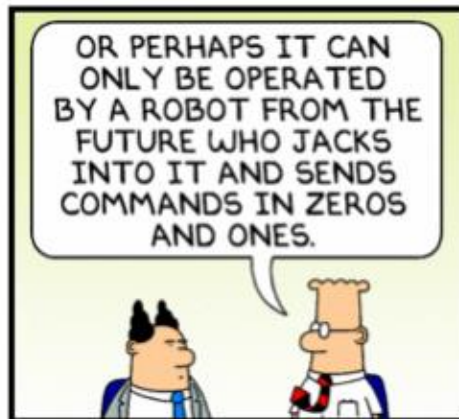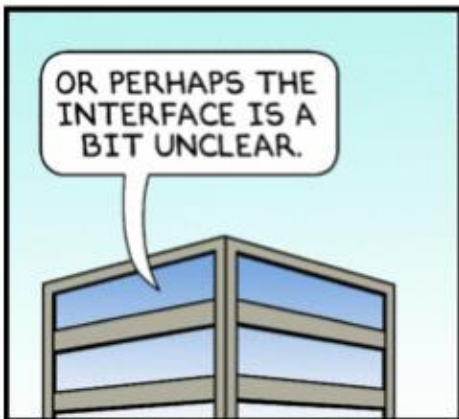
Exercise - bugs

- You buy a printer…
- Is this a bug (i.e. a quality problem) with the printer? Explain your answer in terms of reasonable customer expectations of quality.

What is testing?

- Testing is not debugging

- A test is a sample examination
  - of a single test object
  - under controlled condition
  - comparing against expectations
  - reducing major risk

- Purposes
  - to find failures
  - to measure quality
  - to provide confidence

What is testing?

- The process of demonstrating correct implementation of requirements.
- Models of software
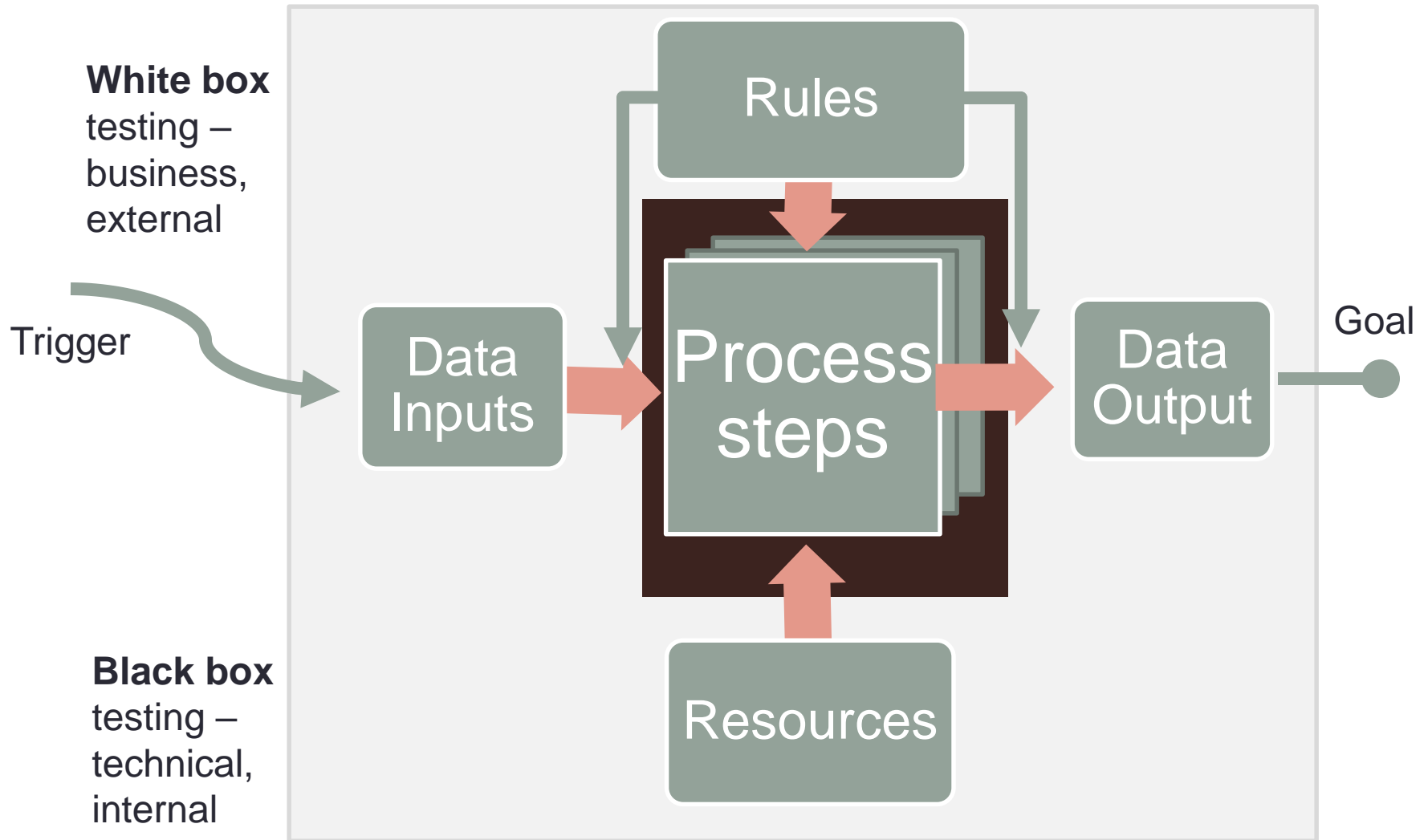- Document reviews

Requirements should specify / test

- Process
  - Business logic – functional
  - System qualities – non-functional
    - Capacity, Security, Maintenance, Availablility …
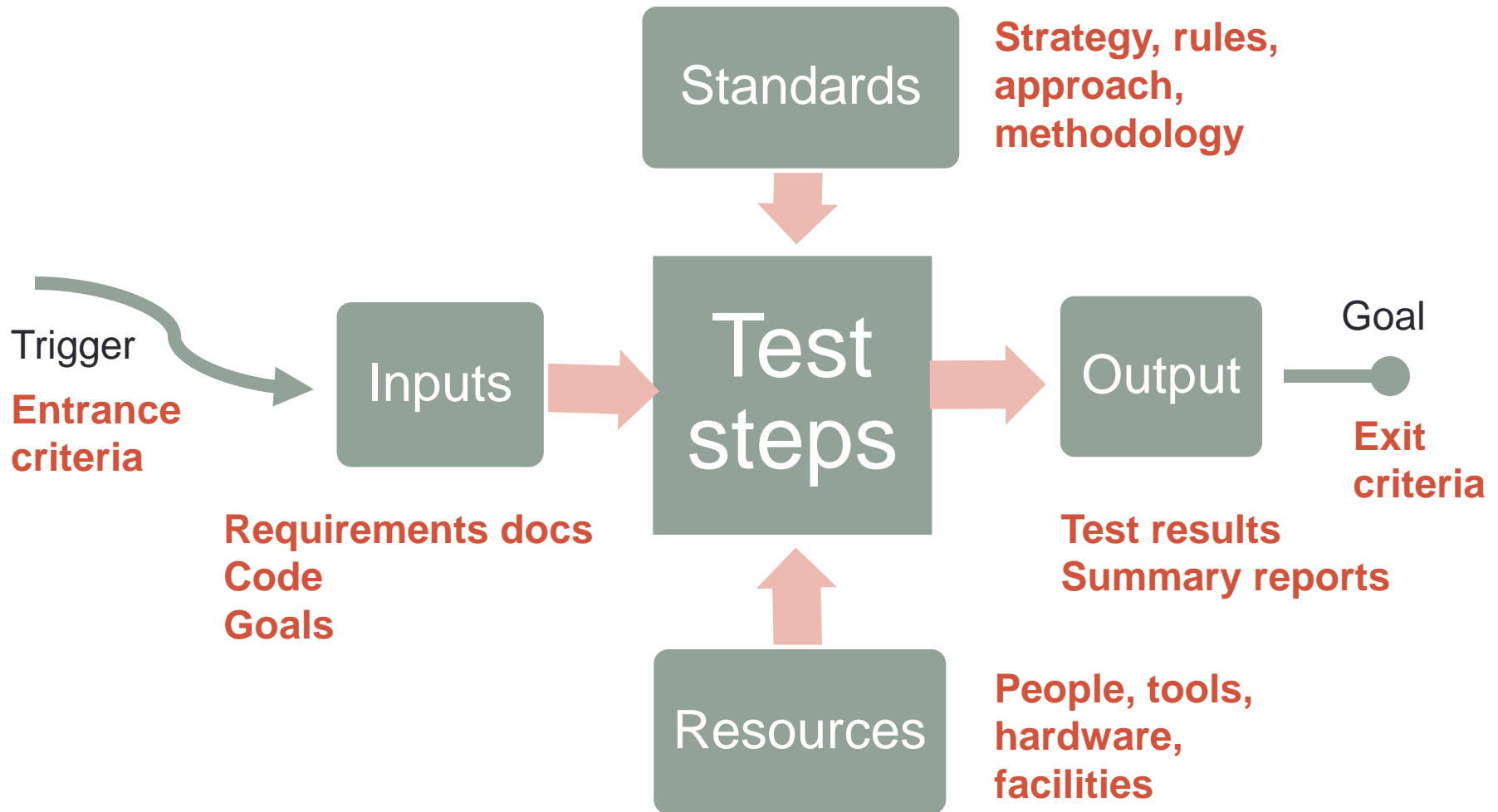   - Rules
- Data
  - Entities
  - Relationships
  - Rules

The process model

White box testing – business, external

Trigger

Black box testing – technical, internal

Rules

Data Inputs

Process steps

Data Output

Goal

Resources

The testing process model

A data model

Measurement

- Numerical constraint
- Set constraint

Units (datatypes)

- Time bounded
- Physical
- Binary
- Complex

Testing terms

- **test** - the process of reducing risk of failed expectations
- **test object** - one unit of scope of testing
- **test management** - the project management pieces focused on testing
- **test process** - the project management test plan or complete overview of managing the testing
- **test case** - how to carry out testing on one test object
  - **test run / test suite** - a set of related test cases run at one time
  - **test scenario** - sequentially linked and dependent test cases

No complex software system is bug free

- You always have less than perfect
  - reliable systems can be less than perfect
- The absence of bugs does not guarantee a correct system

Validation & verification

- **Validation**
  - a quality gate

- Analysis
- Design
- Development
- Validation test

- **Verification**
  - A check, review, feedback

- Analysis
  - Verification
- Design
  - Verification
- Development
  - Verification
- Validation test

Separation of concerns

- programmers should not test their own programs
- programming groups should not test their own programs

Testing Management

Management Issues - Resources

- More code = more testing = more resources needed

Management Issues - Cost

- Approximately 25% of budget

- Estimate based on
  - similar projects
  - similar team skill levels
  - history of testing

- Total cost includes
  - prevention - preparation is biggest payback
  - inspection - meeting standards
  - internal failure (fault) fixing - before delivery
  - external failure (bug) - after delivery is the most damaging

Management Issues - Scope

- Testing is never complete
  - input combination possibilities are too large
  - logic path combination possibilities are too large
  - user interface issues are too large
- Programmers code by trial and error
  - they catch 99% of their errors
  - in 1990, one to three bugs per 100 line of code was normal
- Language improvement
  - languages like Java and C# have tried to eliminate programmer errors
  - languages like JavaScript and PHP have not considered that problem as much

Context Driven Testing
The Seven Basic Principles (Cem Kaner)

1. The value of any practice depends on its context.
2. There are good practices in context, but there are no best practices.
3. People, working together, are the most important part of any project's context.
4. Projects unfold over time in ways that are often not predictable.
5. The product is a solution. If the problem isn't solved, the product doesn't work.
6. Good software testing is a challenging intellectual process.
7. Only through judgment and skill, exercised cooperatively throughout the entire project, are we able to do the right things at the right times to effectively test our products.

Technical reviews

- a good quality control technique
- more productive than automated testing
- reduces defects at first possible time before the code is executable
- structured walkthrough
  - reviewer presents a description of the software and the others give feedback
- inspection
  - formal process with a go / no-go decision and exit criteria

Technical review roles

- review leader / facilitator

- author / producer

- scribe / recorder

- reviewer
  - no more than six

Technical review steps

- Plan the review
- Schedule the review
  - soon after a producer has completed the software and before any work dependent on that software.
- Develop the agenda
- Hold the review
- Create a review report
  - the output of the review is the report and can be written up in any format as long as everyone has their needs addressed. Management wants a summary, who was there and what they thought with dates for completion of action items. The user just wants to know how well the project is going. The developer wants the action items which will track to errors, possible problems, and recoding that should be done.
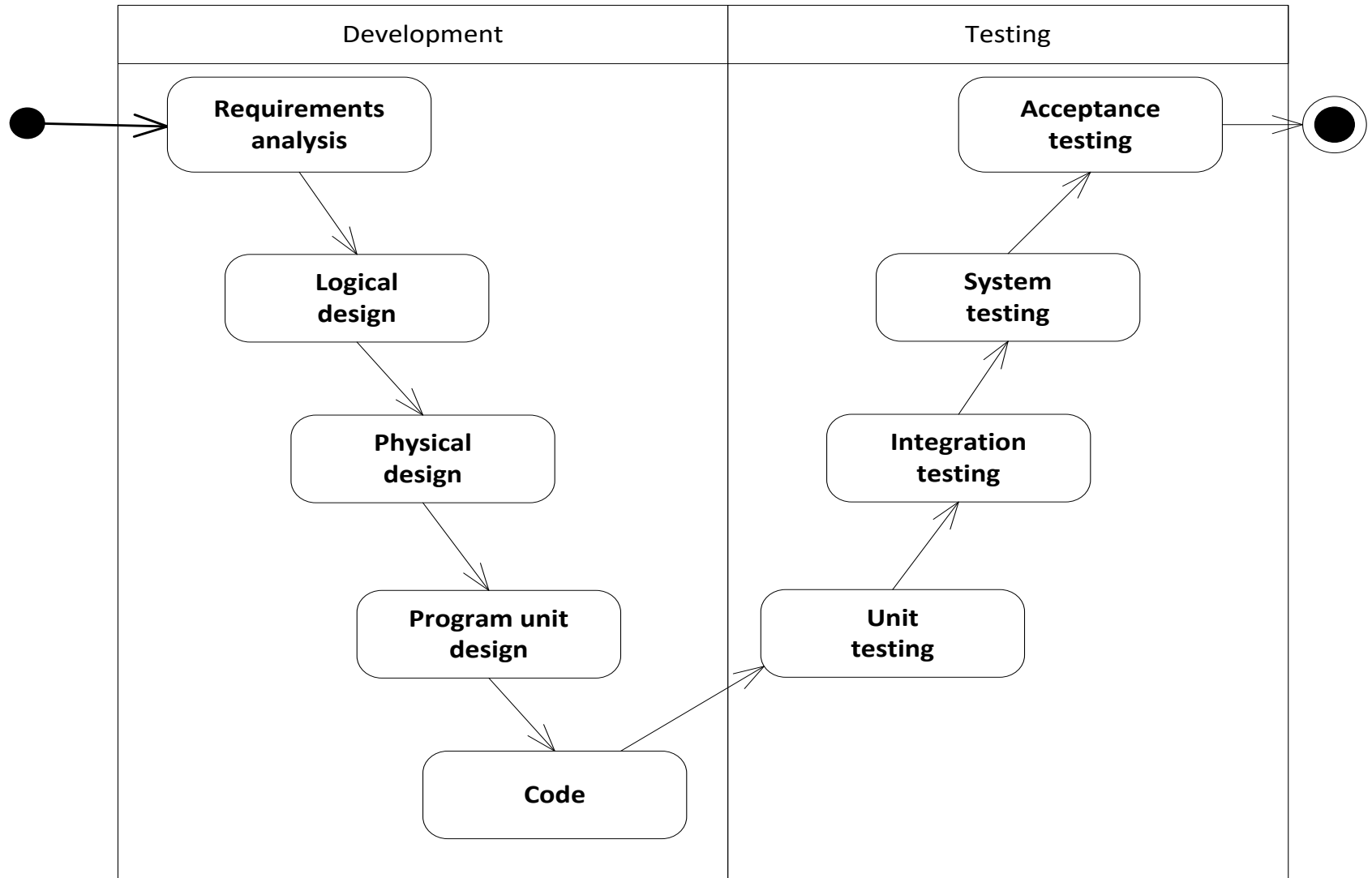
TEST PHASES

Software life cycle testing tasks

- **Project phases**

- Analysis
- Design

- Development

  - Test facilities
  - Test plan
  - Test plan detail
    - test cases
    - test data
    - test scripts
  - Test tools

Integrated Development and Testing System Testing
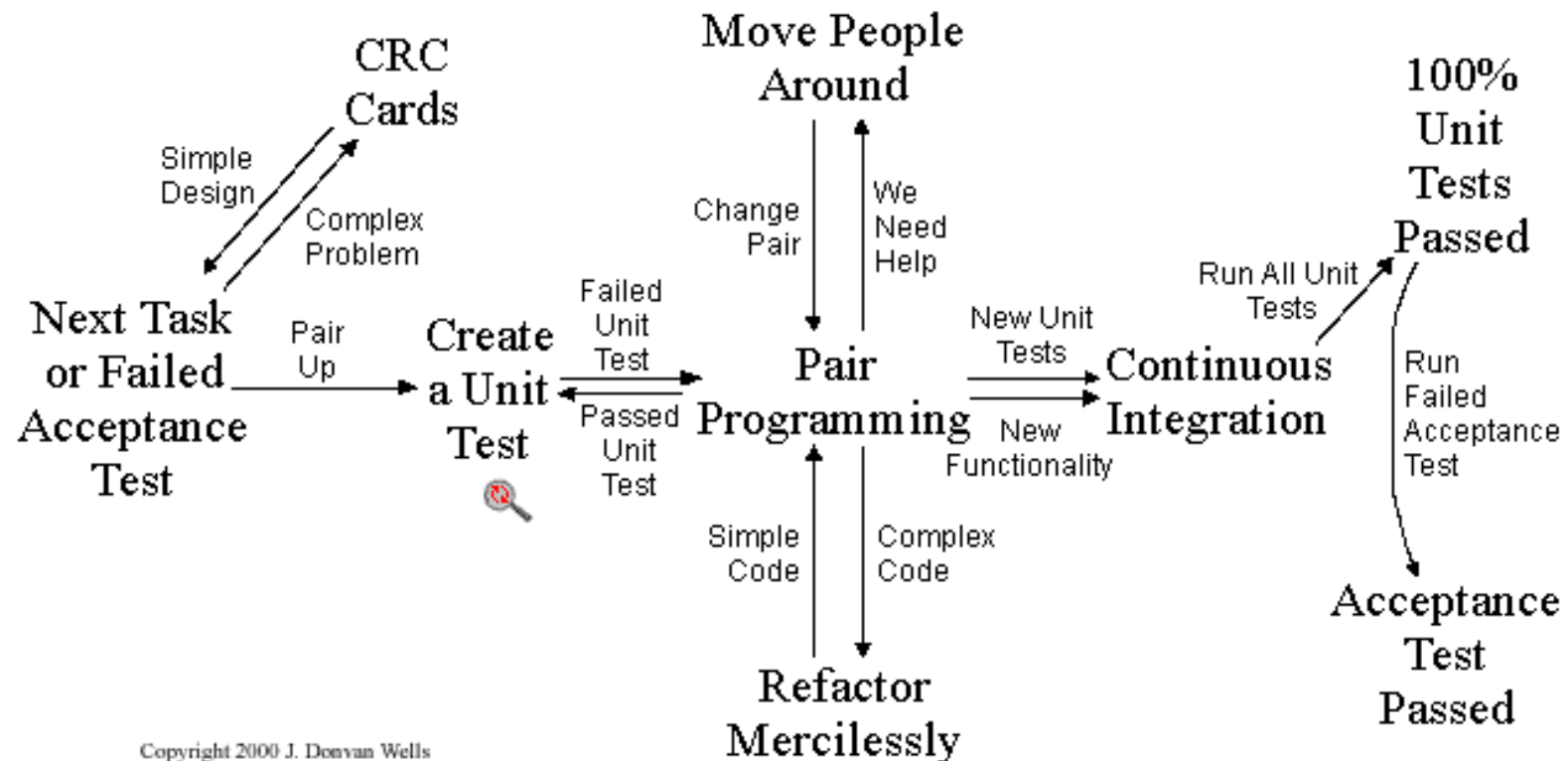
Software testing granularity - V model

Unit testing

- Lowest level of testing
  - done by programmers
  - often automated
  - focuses on logic errors

Unit testing - Extreme Programming

- uses an xUnit framework
- tests are created **before** and for all code units

Integration testing

- sometimes called physical design

- two or more units

- focuses on architectural components
  - components are groups of code units with minimal interfaces (low coupling not requiring changes when one thing changes) and similar functionality (high cohesion)

- reviews for standards adherence

- interface testing

Integration testing - steps

- Identify unit interfaces
- Reconcile interfaces for completeness
- Create integration test conditions
- Evaluate the completeness of the integration test conditions
  - records, files, searches
  - matches, merges, attributes
  - stress
  - control

System test

- also called logical design
- combines all of the workflow, data and business rules
- Types
  - Load Testing
  - Performance Testing
  - Stress & Hot Spot Testing
  - Spike & Bounce Testing
  - Reliability Testing
  - Configuration Testing
  - Acceptance Testing

Acceptance testing

- User acceptance test (UAT)
- last type of testing, app goes live afterwards
- more customer involvement = less importance of UAT
- Use checklists for
  - quality control
  - methodology checks
  - requirements tracking through to test

TEST PLAN

Intro

- Sets expectations (OLA/SLA) between QA and developers
- Tests, test results
- Roles, responsibilities
- Processes, reports, schedule
- Two types
  - a comprehensive plan including a test strategy (smaller projects)
  - overviews of each detailed test plan

Test plan development

- continually updated - a "living" document
- all functional requirements are covered
  - all test cases track through code to requirements

Test plan - structure

- Prepare an introduction
- Define high-level functional requirements
  - User story
  - Use case
- Identify test types
- Identify test entrance/exit criteria
  - When you find no showstopper bugs in 2 days?
  - When you find no major bugs in one week?

Test plan - Define deliverables

- Test design
- Change requests
- Metrics
- Test case
- Test log summary
- Interim test report
- System summary report
- Defect report

Test plan - process

- Organize test team
- Establish test environment
- Define dependencies
- Create test schedule
- Select test tools
- Establish defect recording and tracking procedures
- Establish change request procedures

Test plan - process

- Establish version control procedures
- Define configuration build procedures
- Define project issue resolution procedures
- Establish reporting procedures
- Define approval procedures

IEEE 829 test plan

- Test plan identifier – a unique name or ID
- Introduction
- Test items
- Features to be tested
- Features not to be tested
- Approach – the strategies for specific tests
- Item pass/fail criteria

IEEE 829 test plan

- Suspension criteria and resumption requirements – map to entry and exit criteria
- Test deliverables
- Testing tasks
- Environmental needs
- Responsibilities
- Staffing and training needs
- Schedule
- Risks and contingencies
- Approvals

Test plan by project phase

- see table in workbook

Test strategy

- A static document
- Part of the project plan and usually written by the PM
- Basic steps
  - Start with awareness
    - Understand the goals , business needs
  - First, test broadly
    - Do exploratory testing
  - Then, challenge the system
    - Pick data boundaries
    - Pick priority processes

Exercise - test strategy

- The instructor will give you a test object.
- Discuss as a class the best way to fill in the test strategy template
  - Do not do the entrance / exit criteria
  - Test project management is an additional checklist

Implicit project testing criteria

- the program will not change significantly
- there is enough time to do effective testing and fix defects

Exercise - entrance / exit criteria

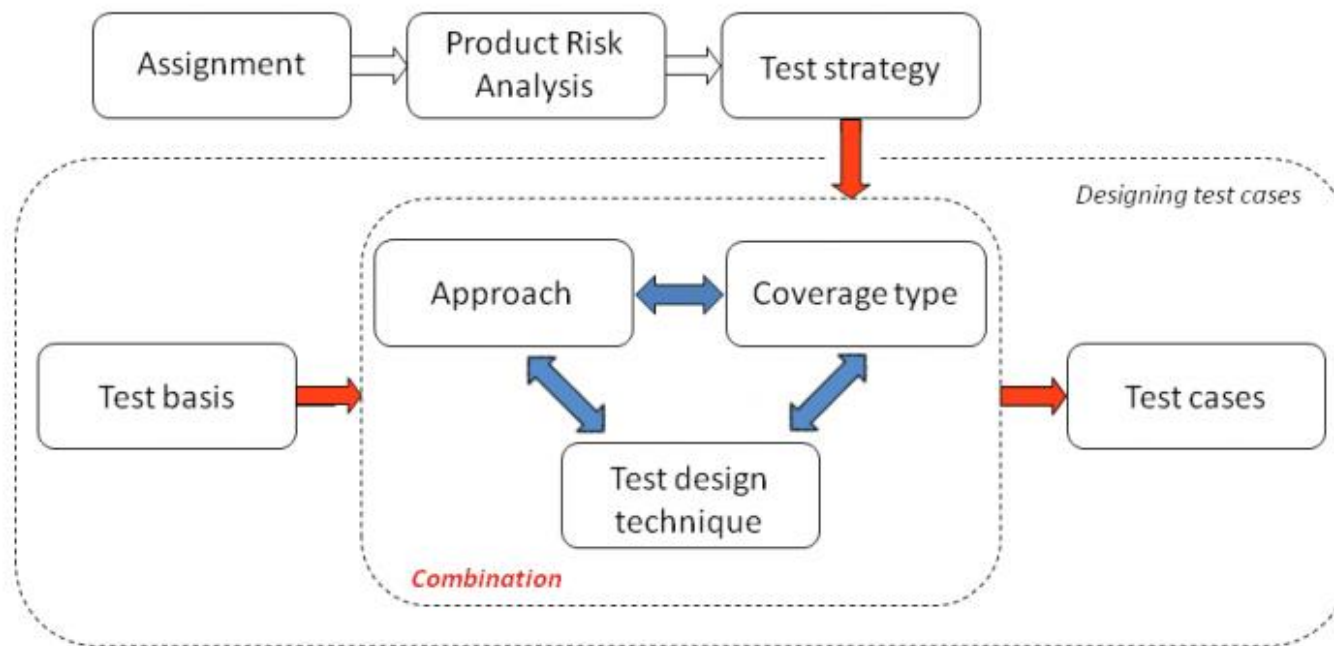- Rank the entrance and exit criteria
- Add others if you want

TEST CASE DESIGN

What tests should be run

Design process

Implementing the test strategy

Characteristics of a good test

- It has a reasonable probability of catching an error
- It is not redundant
- It's the best of breed
- It is neither too simple nor too complex
- It makes program failures obvious

Test groupings

- By object - data base, GUI, documents, bug report
- By role- developer, UAT, security
- By technique - orthogonal, boundary value, random
- By objective or test type - load, stress, scenario
- By scope granularity - integrated, system
- By extent - partial regression
- By time – before too much, repeated, phase, delayed
- By entity type – validation, usability, cardinality
- By business rule: process rules, data rules

Description of test type

- Not just single style of regression or unit
- Multiple styles based on strategy
  - Patient module rule regression
  - Risk based data validation stress
  - Partial regression system interface
  - Boundary GUI validation
  - Etc.

Review and approve the design

- Schedule and prepare for the review
- Obtain approvals

Role / visibility techniques

- Mostly associated with testing by role
- Black box
  - Business roles
  - functional testing
  - customer focused
- White / transparent box
  - Technical roles
  - logic testing
  - automated
- Grey box
 - testers who know something about the design

Static vs. dynamic

- Static testing
  - syntax check
  - walkthroughs of code
  - inspections
  - document evaluations
- Dynamic testing
  - debugging with tools
  - any technique with breakpoints

Informal techniques

- **Free form testing** – Ad hoc or brainstorming using intuition to define test cases based on risk or technical knowledge, etc.
- Guerilla testing
- Smoke test – more about coverage
- Exploratory testing
- Entrance criteria testing

Exercise – web site exploratory test

1. http://kansascity.com/
2. Chrome F12
3. Console tab or
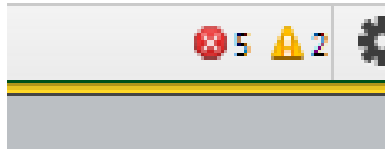
- What are the possible tests you could run from this information?
- Can you find any other site with errors or warnings?

Coverage techniques - dev

- **Condition/rule coverage testing** – Verifying each condition in a decision takes on all possible outcomes at least once. (completeness)

- **Branch/process coverage testing** – Verifying each branch has true and false outcomes at least once. (lean use cases)

- **Statement coverage testing** – Testing every statement in a program so that it executes at least once (unreachability)

Integration techniques – dev

- **Bottom-up testing** – Integrating modules or programs starting from the most fine-grained code module.
- **Top down testing** – Integrating modules or programs starting at the largest granularity of modules.
- **Sandwich testing** – Integrating modules or programs from the top and the bottom simultaneously

Project

Use case level
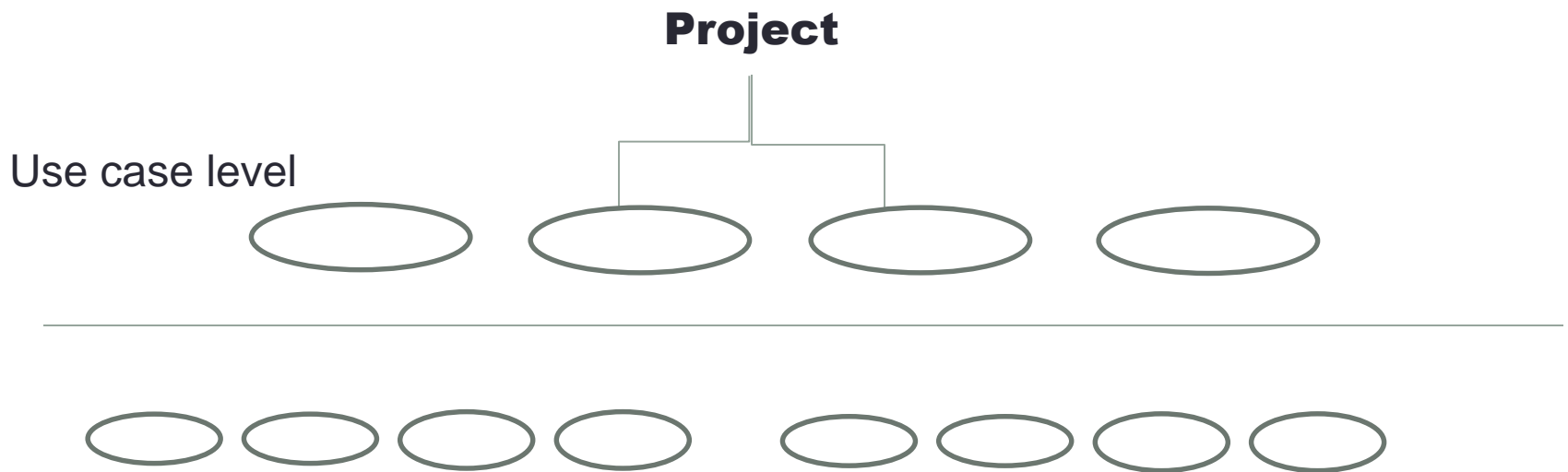
Formal techniques

- **Equivalence partitioning** – Each input condition/facet/category is partitioned into two or more groups/classes. Test cases are generated from representative valid and invalid classes.
  - **Range testing** – Testing where each input identifies a range that should produce the same system behavior. Same as equivalence partitioning.
- **Boundary value testing** – Test cases generated from valid & invalid boundary values of equivalence classes.
  - **Example**: Facet = numeric value
  - Given a set of valid values between 10 and 50, the boundary values of interest to test are
    valid: 10 and 50 (smallest and largest), and
    invalid: 9 (largest integer that is too small) and 51 (smallest integer that is too big).

Exercise – equivalence classes

- What valid/invalid classes of value formats govern how Google calculates an addition formula from the query box?
  - **Whole number digits**
    - 1 + 1
  - **Format - English word equivalent**:
    - One + one
  - Etc.
- Other types of groupings of tests can include
  - Operators
  - Units and unit conversions
  - Length
  - Variables

Formal techniques

- **Orthogonal array testing** – A systematic, statistical way of testing **pair-wise** interactions. An orthogonal array has the balancing property that, for each pair of columns, all parameter-level combinations occur an equal number of times.
  - useful for integration testing of software components
  - useful for testing combinations of configurable options

Formal techniques

- **Six Sigma** - Six Sigma is a rigorous and disciplined methodology that uses data and statistical analysis to measure and improve a company's operational performance by identifying and eliminating "defects" in manufacturing and service-related processes.
  - http://www.isixsigma.com/
- **Statistical profile testing** – Test profile generation using statistical techniques defining transaction paths, conditions, functions, and data tables

Data techniques

- **CRUD testing** – Building a Creation, Read, Update and Delete (CRUD) matrix to test all of those functions.

- **Positive and negative testing** – Testing all of the input values with positive and negative values

- **Random testing** – Testing using a random selection from a specific set of input values where any value is as likely as another.

Data techniques

- **State transition testing** – Testing what causes data to go from one state/type to the next.

- **Database testing** – Testing access, security, and data integrity of data table entries

Rule techniques

- **Decision tables** – Tables showing decision criteria on the axes and the respective actions in the cells.
  - Workflow - process
  - Data

Interface techniques

- **Basis path testing** – Identifying tests based on flow and paths of a program or system
  - Web routing, navigation
- **Cause effect testing** – Mapping multiple simultaneous inputs which may affect others to identify their conditions to test.
- **Thread testing** – Testing by combining units of code into threads of functionality which together accomplish a function or a set of functions.

Usability techniques (UX)

- **Exception testing** – Identifying error messages and exception handling processes and conditions that trigger them

- **Mobile testing** – taking a web site to a touch based interface that has limited bandwidth.

- **Accessibility testing** – WAI-ARIA
  - https://www.w3.org/WAI/intro/aria

Review/static techniques

- **Desk checking** – When the developer reviews the code at their desk for accuracy.

- **Inspections** – Formal peer reviews that use checklists, entry criteria, and exit criteria

- **Structured walkthrough** – A technique to conduct a meeting for system error discovery

Analysis testing techniques

- Any analysis tool can be reused to provide for a second opinion and discuss the comparison of results.
- **Interview** – talking to a stakeholder
- **Requirements review** – Rewriting requirements statements so test cases are mapped better.
- **Prototyping** – A general approach to gather requirements from users by demonstrating some part of the system under development using visual means probably not using code that will be in the final system.

Feedback techniques

- **Pareto analysis** – Analysis of defect patterns to identify causes and sources.
- **Prior defect history testing** – Test cases are created or rerun for every defect found in prior tests of the system
  - Test catalog

Customer Complaints
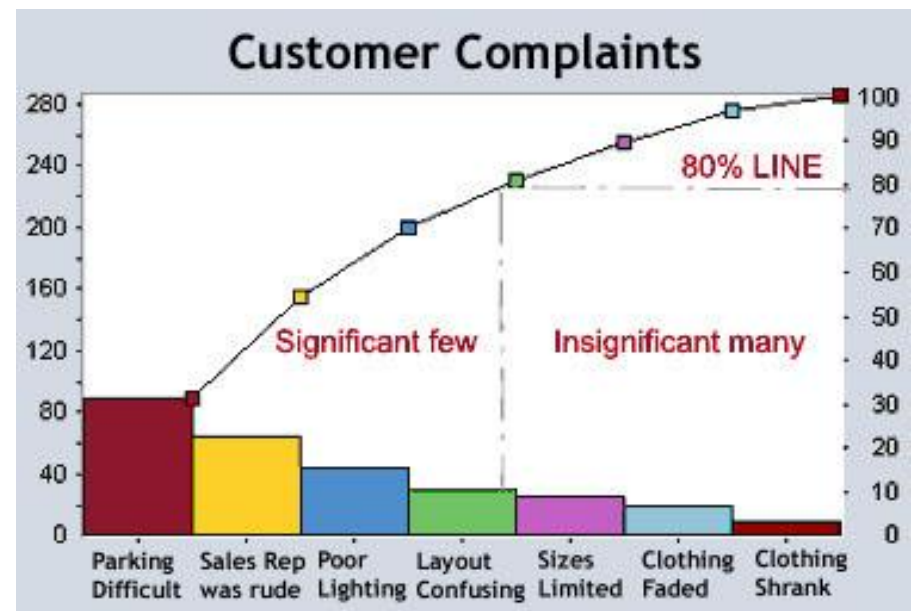
Create a list of common software errors

- A catalog of tests
- ***Testing Computer Software*** by Cem Kaner, Jack Falk, Hung Quoc Nguyen - Appendix A - Common Software Errors
  - UI
  - Error handling
  - Boundary-related errors
  - Calculation errors
  - Initial and later states
  - Control flow errors
  - Errors in handling or interpreting data
  - Race conditions
  - Load conditions
  - Hardware
  - Source, version and ID control
  - Testing errors

Project triggered techniques

- **Risk based testing** – Measuring the degree of business risk in a system to improve testing.

- **Regression testing** – Testing in light of changes made during a development iteration, debugging, maintenance, or a new release.

- **User acceptance testing** – Testing after exiting testing phase by internal tests.

Development techniques

- **Syntax testing** – A data driven technique to test combinations of input syntax
  - 1 + 1
  - One + one
  - One plus one
  - Won pluz one
  - wonpluzone

TEST CASE DOCUMENTATION

A test case template - IEEE 829

- Test case name
- Test ID
- Test suite(s)
- Priority
- Hardware required
- Software required
- Duration, planned
- Effort, planned
- Setup
- Tear down
- **Test steps - use case detail**
- **Execution summary**
  - System Config ID
  - Tester
  - Date completed
  - Duration, actual
  - Effort, actual

Results types

- PASS
- FAIL
- BLOCK
- IP (in progress)

- SKIP
- WARN
- DUPE
- OBS
- IDK, IND

Form - System/software configuration

- Useful for multiple reusable configurations
- Use a code in your test cases

Exercise – system configuration

- Fill out the system/ software configuration form in the workbook for your computer that you are using in this class.

A test case template - data entry

- Instead of Test steps - use case detail
- use screen and field names

Issues – formality

- Informal level- exploratory testing
  - Try finding a bug within the payroll printing module
  - Write charters to give direction to the tester
- Formal level- scripted testing
- In-between
- Guidelines
  - ability to communicate
  - ability to understand
  - culture

Levels of detail

- Level 0
  - No documentation
- Level 1
  - Summary description - several lines
- Level 2
  - Multiple one-line tasks - up to a page
- Level 3
  - Level 2 + description, initial conditions, input, expected results - up to 3 pages
- Level 4 - Highest manual level
- Level 5 - Automated

Exercise

- Run a test case based on a use case. (Yelp)
  - Make sure to review the use case first!

WORKFLOW TESTING

Testing the process

Refine the functional test requirements

- Workflow decomposition
  - The most detailed **business** description possible
  - No design, no automation
- Examples of repeatable **goal-driven** workflows
  - approve customer credit
  - create order
  - order component
  - receive revenue
  - pay bill
  - purchase item

Refine the functional test requirements

- Examples of higher level grouped workflows
  - No specific goal, groups of goals
- **Manage** customer
  - approve credit **for customer**
  - create order **for customer**
- **Manage** warehouse
  - order inventory **for warehouse**
  - run re-order point report **for warehouse**
- **Manage** store
  - pay bill at store by clerk
  - purchase item **at store by clerk**

Refine the functional test requirements

- **Highest level workflows – modules, subsystems**
  - Business requirements, strategic level, business case
- Financial processing
- Inventory processing
- Order processing
- Reporting
- Customer processing

Exercise – feature list

- List the features of your calculator. Write the features as a verb-noun combination such as "Add one number to another number."
  - Write at the most detailed, but business language, level of description possible.

It's not a bug...

FEATURE

SCENARIO BASED TESTING

Use case structure

- Always
  - ID
  - Name – verb/noun from the system perspective
  - Course of events – summary or detail
- Optional
  - Metadata
  - Pre-conditions
  - Alternate paths – errors
  - Alternate paths – included, optional
  - Notes – non-functional usually
  - Rules

Exercise -
Use cases / scenario test cases

- Scenario testing can be extrapolated from use cases.

- What are the basic use cases that would occur on the calculator that use the feature set you extracted?

  - Use verb-noun syntax and make sure that you can put down the calculator at the end of the use case and when you pick it up again start the same or next use case.

- Now, take one of the above use case names and write out a narrative of the "soap opera" on a blank sheet.

  - The script will be a conversation between the user (user presses key) and the calculator (calculator displays number).

Requirements prioritization

- Impact
  - Internal – number of people in org using
  - External – number of people who would buy product

- Urgency / Value
  - Internal – level of hierarchy, VP, CEO, manager
  - External – amount you would pay to buy or replace

Exercise – Use case prioritization

- If you were to prepare a budget for testing the calculator program by priority level, what percentage of your budget would you allocate to each level?
  - Does this make sense?
  - If you had more money including more people, could you do the testing faster?
  - If your budget gets cut by 30%, what testing would you not do? Why?

Exercise – Build a scenario based test case

- Expand a use case from an informal to formal detail.
  - Add the list of tasks that explain what to do as a conversation between the system and the user.
- Build a scenario based test case on the use case narrative. For test options, exaggerate each aspect of it:
  - for each variable, substitute a more extreme value
  - if a scenario can include a repeating element, repeat it lots of times
  - make the environment less hospitable to the case (increase or decrease memory, printer resolution, video resolution, etc.)

Priority to test estimate

- Requirement priority
  - Low
  - Medium
  - High
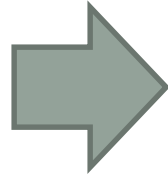
- Risk factors causing bugs
  - Unfamiliarity?
  - Complexity?
    - Low
    - Medium
    - High
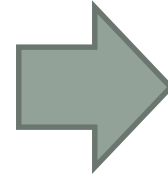
- Test effort
  - Low
  - Medium
  - High

DATA TESTING

CRUD diagram

- CRUD = creation, updating, reading, and deletion
- matrix of roles vs. entity type management

	A	B	C	D	E	F	G	H	I	J	K
1		Entity type									
2	CRUD - process vs. entity	Categories	Subcategories	Products	Product variants	Product related	Specials	Orders	Order items	Order status	Comment
3	Planning	crud	crud								
4	Marketing	r	r	rud	rud	rud	rud	r	r	r	
5	Selling	r	r	r	r	r	r	cru	cru	cru	
6	Compensation						r	r	r	ru	
7	Shipping							r	r	ru	
8	Operations							r	r	ru	
9	Purchasing			crud	crud	crud	crud				
10	Forecasting						r	r	r	r	
11	Receiving			ru	ru	ru	r				
12	Ordering			ru	ru	ru	r	r	r	r	
13	Research	r	r	r	r	r	r	r	r	r	

Data correctness – dictionary

- Usage
- Validation (rules)
  - Range
  - Required
  - Dependencies
- Character sets – sorting, searching
  - Collation – accent aware, case sensitivity
- System defaults – system used, foreign sets
  - CR, CR LF, LF
- Constraints
  - referential

Data interactions

- As a black box tester all fields are assumed to interact with other fields. A grey-box tester can reduce how many tests really need to be performed.

- Given five fields on a form that have one set of valid and one set of invalid data, how many tests need to be run?

- Black box – assume any interaction, all tests
  - (5 fields * 2 options) * (4 fields * 2 options) = 80 tests

- Grey-box - no interactions, one test per field
  - 5 fields * 2 options = 10 tests

- Grey-box - One field dependency on previous field
  - (2 tests * 2 fields) + 3 fields) * 2 options = 14 tests

Exercise – Triangle app tests

- This program accepts as input three integers which it interprets as the lengths of sides of a triangle. It reports whether the triangle is equilateral (three equal sides), isosceles (two equal sides), or scalene (neither equilateral nor isosceles). Write a set of test data to test this program.

Exercise – Test cases

- For the calculator system that you worked with earlier, write a test case in sequential steps of level 2 detail that is for the multiplication chaining feature (2 x 3 =, =, =) using the **test case form** at the end of the book.

- For the calculator system that you worked with earlier, write a test case in sequential steps of level 2 detail that is for the multiplication chaining feature but using the **screen-oriented template** as if you were on the web.

BUSINESS RULE TESTING

Business rule tests

- Analysis
  - Documented – any rule used is defined
  - Testable - rules use values and units
  - Completeness - All branches have been defined
- Test planning
  - Coverage - Tests are defined for each branch of tree (line of table)
- Scope
  - Unit testing – individual rules
  - Integrated tests – web service aggregates, orchestrated

Business rule coverage

- Data
  - Constants – stay constant
  - Values
    - Range
    - Required
    - Dependencies
  - Derived values – check formulas
- Workflow
  - conditional statements

Exercise – Create business rule table

Business Rule: Cost & Pricing type					
Owner: DH					
	Facets/Variables				
Name	What hauling	Fuel	Accessorials	Role	Result - $ per ton
Haul/fuel/Acc 1	New goods	Regular	none	Shipper	1000
Haul/fuel/Acc 2	Old goods	Regular	none	Shipper	800
Haul/fuel/Acc 3	New goods	Diesel	none	Shipper	1050
Haul/fuel/Acc 4	Old goods	Diesel	none	Shipper	850
Haul/fuel/Acc 5	New goods	Regular	Forklift fee	Shipper	1020
Haul/fuel/Acc 6	Old goods	Regular	Forklift fee	Shipper	820
Haul/fuel/Acc 7	New goods	Diesel	Forklift fee	Shipper	1070
Haul/fuel/Acc 8	Old goods	Diesel	Forklift fee	Shipper	870
Haul/fuel/Acc 9	New goods	Regular	Lumper fee	Shipper	1050
Haul/fuel/Acc 10	Old goods	Regular	Lumper fee	Shipper	850
Haul/fuel/Acc 11	New goods	Diesel	Lumper fee	Shipper	1100
Haul/fuel/Acc 12	Old goods	Diesel	Lumper fee	Shipper	900
				Customer	

REGRESSION TESTING

Repeating a test after a change

Using regression testing

- Good models for dependencies don't exist for programmers.

- Best solution for finding dependencies after new code is introduced

- Fairly easy to automate

- After a bug fix:
  - verify the bug has been fixed
  - try to find related bugs with boundary tests
  - run standard battery of tests

The standard battery of tests

- Suites of tests keep building and require more time
- Drop tests that are redundant
  - Equivalence sets, orthogonal array
- Reduce the number of tests on a fixed bug
- Combine test cases
- Automate if possible
- Designate some tests for periodic testing
  - By priority
  - By frequency of use
  - Fixed bugs

ORACLE TESTING

Comparing to another program

What is oracle testing?

- a program that has the right answers and used to provide comparison results to an application under testing.

Exercise

- Do the following before answering the questions.
  - 1. Open the Windows calculator
  - 2. Enter a number that is a perfect square (4, 9, 16, 25, etc.)
  - 3. Get the square root of that number.
  - 4. Subtract the same result to get zero.

Exercise – Oracle testing

1. Is this a bug?
2. Does this meet acceptable standards that the audience using the application would agree with?
3. Would the Windows calculator work as an oracle to test against a calculator program that you wrote?
4. Would the program find any false bugs?
5. Would the program miss any bugs?
6. Would a programmer reject a bug report for this bug as not really a bug?
7. Is the excuse "No one would really do that" really a phrase for "No user I can think of, who I like, would do that on purpose"?
8. Are there any more "bugs" like this in the calculator?

Automated testing

Typical uses

- Load and Stress testing

- Regression
  - tests need to be run at every build of the application

- Multiple data sets, configurations
  - tests are required using multiple data values for the same actions
  - tests require detailed information from system internals such as SQL, GUI attributes, etc.

Constraints

- Large data feeds are difficult
- The cost may be too much (open source tools are getting better)
- The organizational culture isn't ready for testing tools
- No tools test usability
- Not needed if the test is only a one-time test
- May cost too much time to learn, set up, and integrate into the environment if there is time pressure.
- If *ad hoc* testing is normal, a tool will not be useful as much as with formal test design and test cases
- If the tests do not have predictable results, regression tools are useless
- If the system changes code rapidly from one phase to the next, the regression tool will eat up much more time than necessary.

Tool types

- Web site management
- Test management
- Regression testing
- Coverage analysis
- Dynamic testing
- Static testing
- Load testing
- Comparators

Checklist

- Ease of use?
- Training required ?
- Team members experienced?
- Training materials available?
- Will the tool work on the current system?
- Environment upgrade?
- Is the GUI easy?
- Do users make errors with it easily?
- Is the tool physically capable of testing your system?
- Can the tool handle full project testing?

Crash resistant?
Does the tool meet other customers needs?
Were those customers similar to you in needs?
How well did the tool perform for the other customers?
Advanced enough?
Have you tried a demo version yet?
Price reasonable?
Does the tool meet methodology standards?

Jolt Awards 2014

- Finalists
  - **Parasoft Development Testing Platform**
  - **Zeenyx AscentialTest 6.4**
  - **DevExpress TestCafé**
- Productivity Award
  - **SmartBear TestComplete 10.1**
  - **StresStimulus v3.5**
- **Jolt Award**
  - **SmartBear SoapUI Pro 5.0**

WEB TESTING

Web page testing ideas

- **Navigation**
  - **Internal links**
  - **External links**

- **Constraints**
  - **Forms**

- **Syntax**
  - **GET / POST**

Browser lifecycle

- Browser requests URL from server
  - Server responds with one page.
- Browser requests first dependent link on page. (CSS, JS, Gif, PNG, JPG…)
  - Server returns files.
- Browser continues to request links.
- Browser calculates page layout
- Browser executes JavaScript as it received.
- Browser will render page when it can.
  - Blue line – DOM (all html) loaded and can be rendered, no CSS, JS, images…, can be blocked
  - Red line - all content has been loaded

Browser dev tools – Network tab

GET vs POST

- A GET request will show form data collected and sent to requested URL in the query string.

www.bing.com/search?q=GET+vs+POST&go=Submit&qs=n&form=QBLH

- A POST request will hide the form data.

Request Headers (15)
▼ Form Data   view source   view URL encoded
   diagnostics: true
   q: show tables
   format: json
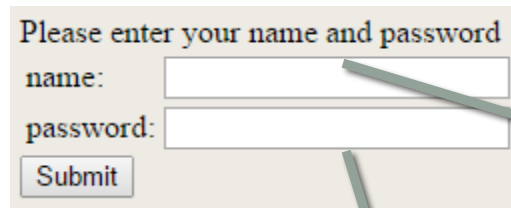   crumb: fQ9eX9fKcxU
   _rand: 145
   _p: hs
▶ Response Headers (12)

GET/POST data length limit

- GET
  - Usually limited by browser – commonly 2K
- POST
  - Set limit in language - php_value post_max_size 20M
  - Common server size 10M
  - Larger sizes could be a symptom of a DoS attack.
- Test with stock data file
  - 2048 characters ending with # (to detect file truncation)
  - 4096 characters ending with #

SQL injection

- Knowing how SQL is written and inserting or adding extra code to statement being executed

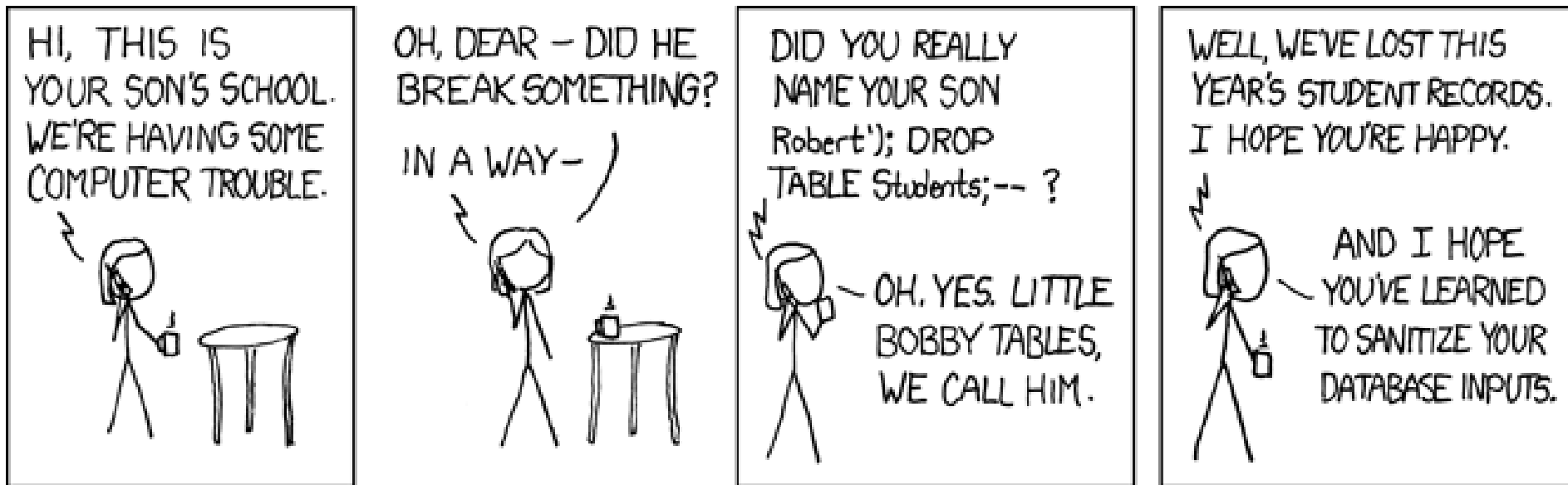Please enter your name and password
name: [   ]
password: [   ]
[Submit]

- **execute** "SELECT * from USERS WHERE name = ' " + **name** + " '
AND password = ' " + **password + " ';"
- ' OR ''=' for both fields creates the statement (single quotes only)
- **execute** "SELECT * from USERS
WHERE name = '' OR '' = '' AND
password = '' OR '' = '' ;"
- '' is an empty string but acts to continue the statement
- http://sqlzoo.net/hack/

SQL injection

- Exploits of a mom – XKCD

Form testing

- Multiple submits of POSTs are asynchronous

→ Please verify the information below. If everything is correct, press the "Submit Payment" button.

If you are paying an amount other than the full balance, please contact a Patient Account Representative to make payment arrangements.

Please do not click the [Submit Payment] button more than once as it may result in a duplicate payment.

Selenium

- http://www.seleniumhq.org
- [Selenium IDE](); a Firefox add-on that will do simple record-and-playback of interactions with the browser.
  - Sep 2017 – deprecated, use programming
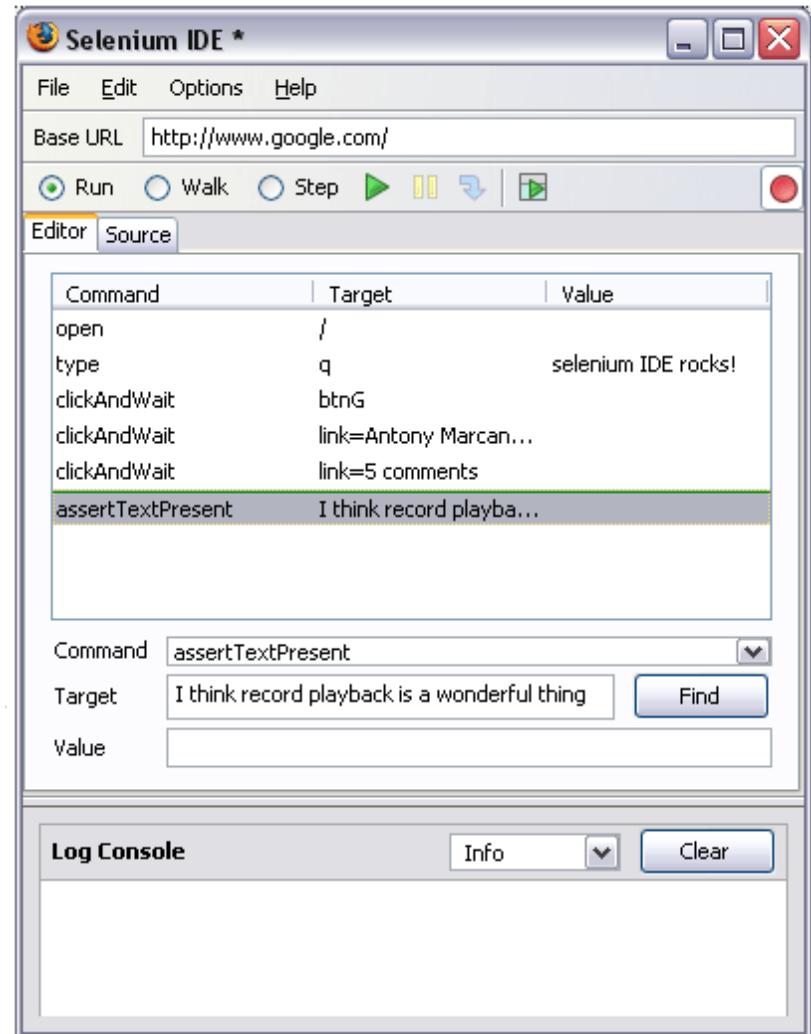
Selenium

- Selenium WebDriver; a collection of language specific bindings to drive a browser -- the way it is meant to be driven.
  - http://www.seleniumhq.org/projects/webdriver/
- WinAppDriver - Test any app with Appium's Selenium-like tests on Windows
  - http://www.hanselman.com/blog/WinAppDriverTestAnyAppWithAppiumsSeleniumlikeTestsOnWindows.aspx

Visual Studio Test Manager

- Integrated in Visual Studio
- Requires Team Foundation Server

Windows Problem Steps Recorder

- Built in to Windows

- Process
  - Turn on
  - Record
  - Turn off
  - Save file.

- Since Windows 7
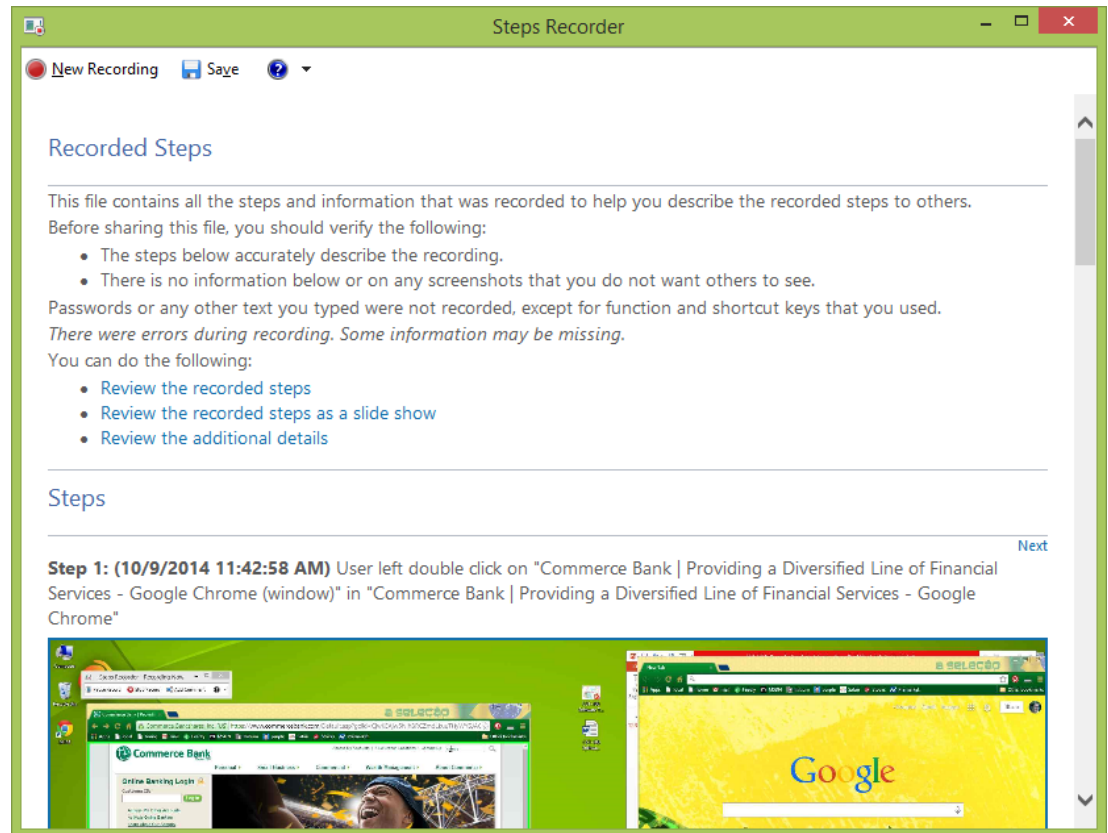
Chrome DevTools

- F12
- Responsive web sites
  - Layout
  - Screen size
- Latency
  - Network tab
  - Choose throttling
  - Click and hold refresh button and choose Empty Cache and Hard Reload.
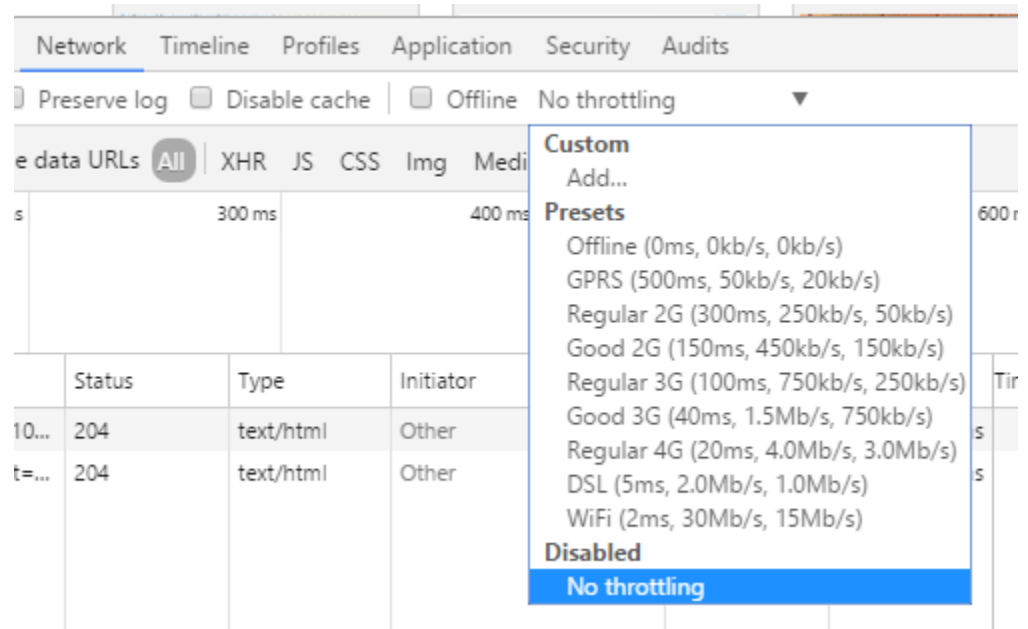
JSFiddle

- Save HTML, CSS, SCSS, JavaScript, CoffeeScript, libraries to URL
  - Bug reports
  - Design suggestions
  - Mobile testing also!
- Plunkr - https://plnkr.co/
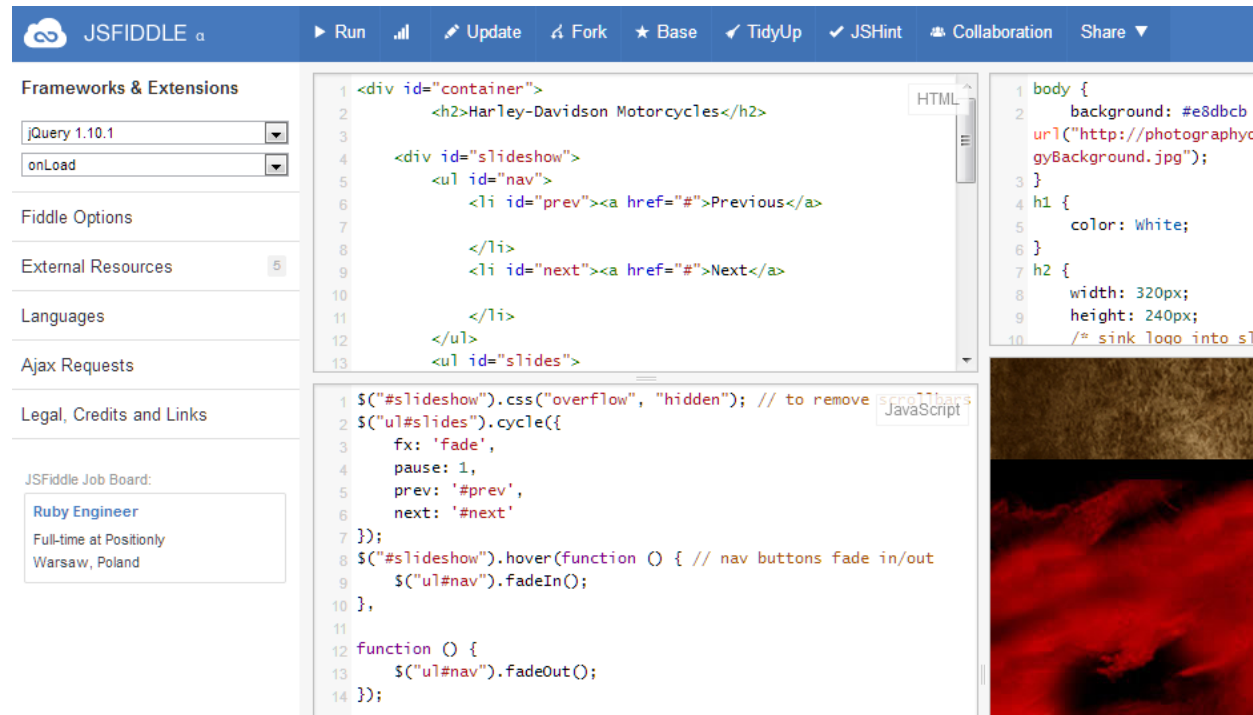
Jasmine / PhantomJS

- Jasmine
  - Unit testing of JavaScript for programmers
- PhantomJS
  - Headless website testing
  - Screen capture
  - Page automation
  - Network monitoring
  - http://phantomjs.org/

Telerik

- Microsoft components
- Fiddler - http://www.telerik.com/fiddler
- Test Studio - http://www.telerik.com/teststudio

Browser testing

- Eggplant
  - http://www.testplant.com/
- Browsersync
  - https://www.browsersync.io/

Server testing

- Hamms
  - a badly behaved HTTP server
  - https://github.com/kevinburke/hamms

UX/GUI TESTING

Goals

- Consistent
  - Validate design guidelines on all pages/screens
- Simple
- Functional
  - Validate link destinations
  - Validate button actions
- Accessibility

Design guidelines

- Get users involved.
- Know the users' culture, experience, and environment
- Verify often by creating quick prototypes
- Use the users' workflow to drive the design
- Use GUI features that users understand and don't over-design
- Create the GUI, help, and training concurrently
- Don't expect users to remember hidden commands or functions. Put it on the menu.
- Think ahead about possible mistakes and don't penalize the user for them.
- Show status as often as possible
- Keep it simple.

GUI tests

- Design standards specify
  - wording for prompts, errors and help
  - color, highlights, and cursors
  - shortcut key usage
  - screen elements that must be in a certain place like logos and disclaimers or copyright notices
  - choice of fonts
  - choice of colors
  - Containers - windows, frames, dialog boxes
    - web form layouts
    - content layouts
    - sequence of screen parts
  - Navigation - menus, context menus
  - Forms and controls
  - Icons

Design documents

- Windows UX Guide
  - http://msdn.microsoft.com/en-us/library/windows/desktop/aa511258.aspx
- Windows Store apps UX guidelines
  - http://msdn.microsoft.com/en-us/library/windows/apps/hh465424.aspx
- iOS Human Interface Guidelines
  - http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html
- Android User Interface Guidelines
  - http://developer.android.com/guide/practices/ui_guidelines/index.html
- All others
  - http://www.theuxbookmark.com/2010/08/interaction-design/a-monster-list-of-ui-guidelines-style-guides/

MOBILE TESTING

Mobile differences

- Screen size
  - Responsive web design comparison tool
  - http://quirktools.com/screenfly/#u=http%3A//centriq.com&w=1024&h=600

- Connectivity / Speed
  - Ability to be disconnected

- O/S
  - Proprietary widgets – form controls
  - Unique controls and behavior

Emulators

- Ability to test?
  - Screen size – yes, but better to use Chrome device toolbar
  - Connectivity – no, but can use Chrome throttling
  - Touch – somewhat in Chrome
  - O/S – mostly no
- Prefer a real device
  - Go to a phone store

Prioritize

- Know your users' devices
  - Check your server logs
  - Use Google Analytics or another admin tool
- Know what stories are important to mobile users
  - Mobile users are anxious
  - Mobile users are bored
  - Mobile users have bad vision
  - Mobile users can't click well

Remote debugging

- Weinre - https://people.apache.org/~pmuellr/weinre/docs/latest/Home.html
- Android: Remote Debugging on Android with Chrome
- iOS: Enabling Web Inspector for iOS
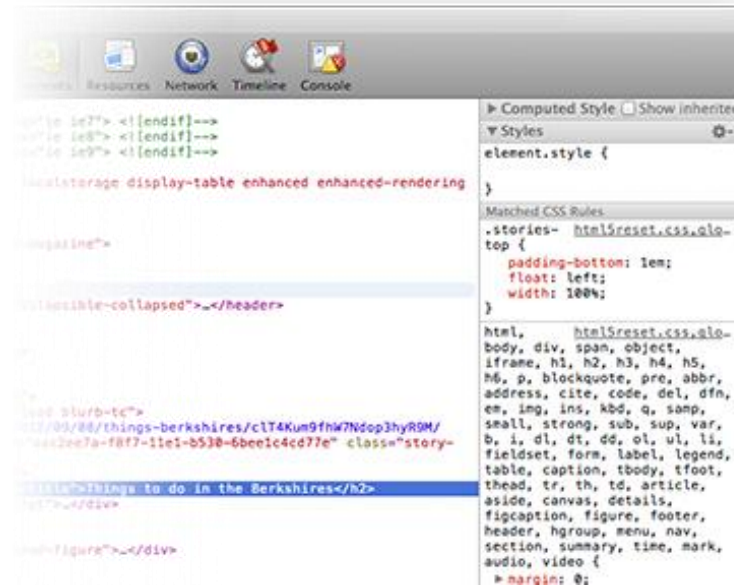
Chrome DevTools

- Features
  - Touch emulation
  - Preset sizes
  - Screen shots
  - CSS media query viewing
  - Throttling
- Test for
  - Layout at different sizes
  - Load times at different throttling
  - JavaScript errors in Console

Test services

- BrowserStack
  - https://www.browserstack.com/

What is an equivalence class?

If two tests with different data produce the same expected results, they are equivalent. The tests form an equivalence class:

1. if they all test the same type of thing (the facet or category)
2. if one test catches a bug, the others will likely catch it also
3. if one test misses a bug, the others will likely miss it also

Equivalence class tests

Tests can be put into the same equivalence class if:

1. they involve the same type of input variables
2. they result in similar operation in the program
3. they affect the same type of output variables
4. they either, all or none, force error handling to occur

Common facets/categories

- Numeric value – whole, floating point
  - Also include zero as a special case
- Numeric value – non-digit
  - One, Roman numerals…
- Numeric format
  - Numeric length (including zero), commas
- Text encoding – ASCII, Unicode, ISO 8859-1
- Text length
- Text language
- Web form drop-down order
- Location distance
- Legal values

Characters

- Unicode
  - http://www.unicode.org/charts
- Look up Unicode
  - http://www.amp-what.com/
- Type a character
  - Alt (left side of space bar) + numeric keypad decimal in four digits
  - Alt + 0241 = ñ
- Use a special characters file to reuse for text inputs

Race conditions and time dependencies

- Vary execution by
    - running on different speeds of machines
    - interrupting the program like hackers do to get the program to give you access to something that is a state that shouldn't be exposed
- Press keys and try to infiltrate the system when it is ready to time-out

Facet partitioning and boundary test data

- **Equivalence class** partitioning uses categories/facets of test types
- **Boundary test data** selects the best representative of each facet nearest to the boundary if possible

Facet	Valid Min / only	Valid Max	Invalid max / only	Invalid min
Input length	1	8	9	0
Numeric integer value	-2,147,483,648	+2,147,483,647	+2,147,483,648	-2,147,483,649
Time length	0	20 minutes	20 min, 1 sec	
Temperature	32°F	212°F	213°F	31°F
State	Missouri or Kansas		Nevada	

Simple example

- A program that should accept whole numbers from 1 to 99 has at least **five** equivalence classes.
  - **Facet type**: description of members of class set
- Valid
  1. **Value**: Any whole number from 1 to 99
  2. **Format**: Any floating point number equivalent to a whole number.
- Invalid
  1. **Value**: Any whole number less than 1.
  2. **Value**: Any whole number greater than 99.
  3. **Format**: Any floating point number not representing a whole number (1.01)
  4. **Format – non-digit-** Anything not understood (parsed) as a number.

Organize with an outline or table

- A table can make a lot of classes visually appealing but gets bulky. An outline is easier to organize on a computer in Word but isn't as easy to read without indentation.
- Enter a number in a text field
  - **Valid sets**
    - **Whole values:** between 1 and 99
    - **Format - length**: one, two
  - **Invalid sets**
    - **Whole values:** < 1
    - **Whole values:** >99
    - **Calculated values**: any
    - **Format - digits**: more than one decimal point
    - **Format - Non-digit characters** :
      - Unicode less than '0' except the decimal point
      - Unicode greater than '9'
    - **Format - length** : Zero
    - **Format - length** : Longer than 2

Ranges

- When there is an ordered set, select by min/max in range
  - Whole numbers
  - Floating point precision
  - Digit lengths
  - Unicode
- Ranges/continuums include a valid equivalence class as well as the two invalid classes above and below it, and the non-numeric class.
- Sometimes there are multiple valid ranges.

Set members (enums)

- A valid equivalence class may use a country's name like "USA" but also include "United States" and "United States of America" as well as "U.S.A."

  - Should "America" be allowed as a member of this group?
  - How about "Les Etats-Unis"?
  - How about "USA " or " USA"? (The space can cause programs to get very upset.)

Analyze responses to lists and menus

- Look at the list of possible inputs and check to see if others are valid.

- How might a person respond if they couldn't find an appropriate response?

- A common set of valid responses could be ones that ignore case
  - "yes", "YES", "Yes" , "yEs"

Look for variables that must be equal

- Sometimes, a selection only gives you one choice because the others are no longer available.

Create time-determined classes

- Some events on a computer can crash it depending on when you create that event.

- The equivalence classes become
  - stuff you do much before a task,
  - stuff done at about the same time before the task, and
  - stuff done after the task but before the application can do anything about it.

Dependencies and constraints

- Triangle problem: (see appendix) The triangle must have three inputs that are related by having to add up to 180 degrees.

- If the program is supposed to output into a standard range of values, then the inputs will be a class.

Facets in operating environments

- If the program runs only with more than 256K that becomes an equivalence class.
- Other types of classes might group together types of monitors, storage units, etc.

Boundaries of equivalence classes

- The best ones are at the boundaries of ordered classes such as the
  - biggest, smallest,
  - soonest, latest
  - shortest, longest,
  - loudest, quietest
  - fastest, slowest, or
  - nicest, meanest.
- The characters @ and [are just beyond the ASCII/Unicode codes for A and Z. Try them.
  - ` and { for a-z

Exercise – equivalence classes

- Using the calculator's data entry screen, analyze the values you can enter into the field. The tests that will use a set of values described by a range is an equivalence class.
- Fill in the chart with the classes and any candidate boundary tests.

Functional/ non-function item	Equiv class used	Valid / Invalid	Description of boundary sets or best candidates
Enter a number	Numeric value – whole (ordered)	Valid	Min: -99,999,999 Max: +99,999,999
	Numeric value – whole (ordered)	Invalid	-100,000,000 +100,000,000
	Numeric precision – floating point	Valid	Min: 2 places Max: 8 places
	Numeric precision – floating point	Invalid	1 place 9 places
	Format – decimal point	Valid	Min: 0 decimal points Max: 1 decimal point
	Format – decimal point	Invalid	2 decimal points
	Time between digit entry (ordered)	Valid	Min: ~1/100th second Max: amount of time up to auto shut off
	Time between digit entry (ordered)	Invalid	Less than ~1/100th second More than time for auto shut off
	Temperature	Valid	Min: 0°F ? Max: 200°F ?
	Temperature	Invalid	Min: -1°F ? Max: 201°F ?
Show the result of a calculation	Numeric precision – floating point	Valid	Min: 2 places Max: 8 places with rounding (see rounding rule)
	Numeric precision – floating point	Invalid	1 place 9 places with error message
	Invalid math results (error state)	Valid	Imaginary numbers (square root of any negative number) Divisions by zero Infinity (questionable, how do you get this?)

Exercise - File access test design

- Think up as many ways as you can (not including combinations of ways) to test a file/hard drive that might pose a problem in a client-server environment (cloud based storage: Dropbox, OneDrive, AWS, Google Drive etc.)

Exercise - Numerical equivalence classes

- Think up as many ways as you can (not including combinations of ways) to type in numbers on a web site that fall within valid or invalid bounds.

- Also give an example of the tests using the valid input range of 2,000 to 8,000.

- Suggest a good test representative for the set.

Exercise - Web text equivalence classes

- Think up as many ways as you can (not including combinations of ways) to type in a text to a form input field on a web site that might pose a problem to the text entry.

- Also give an example of the test.

- Also see if you can come up with any special tests for file names or proper names.

Exercise -
IP and character encoding equivalence classes

- See how many ways you can come up with (not including combinations of ways) to type in a web site address in a browser's address bar that will connect you to a web site without using character encoding.

Other equivalence class ideas - time/date

- Timeouts
- Time Difference between Machines
- Crossing Time Zones
- Leap Days Always Invalid Days (Feb 30, Sept 31)
- Feb 29 in Non-Leap Years
- Different Formats (June 5, 2001; 06/05/2001; 06/05/01; 06-05-01; 6/5/2001 12:34)
- Daylight Savings Changeover
- Reset Clock Backward or Forward
- Double digit months – sometimes yield results
- 2038 – a good special number to try, see Wikipedia "year 2038 problem"

Credit cards

- Rule (calculation) - Luhn algorithm
- Credit card security standards
  - OWASP - https://www.owasp.org/index.php/Handling_E-Commerce_Payments
  - https://www.owasp.org/index.php/Category:OWASP_Testing_Project
  - PCI - https://www.pcisecuritystandards.org/

Other equivalence class ideas

- Violates Domain-Specific Rules
  - an IP address of 999.999.999.999
  - Email - http://rumkin.com/software/email/rules.php
  - an age of -1

- Violates Uniqueness Constraint

STATE TESTING

State chart

- a model of what states are triggered by what
- test all paths people will likely follow
- if you think a choice might affect another screen, test all of the choices and their effects
- try a few random paths

Exercise - state chart

- Complete the states of the calculator in this UML state chart diagram below.
- Fill in on the lines which transition from one state to another with the keys that will take the calculator to that state

PRIORITY BASED TESTING

AKA risk based testing

What is priority/risk based testing?

- **Priority** – scope ranked by impact and urgency
- **Risk** = probability of failure based on history * confidence, complexity
- **Risk factors / Failure modes**: design tests to expose a type of failure.
  - Product element (or, component) failures
    - Operational failures
  - Quality attribute failures
  - Common programming errors
- **Operational profiles**: Focus testing on the most common patterns and minimize time on things that no one will do.

Exercise – risk based tests

- List out some failure modes that you think are appropriate for your calculator

Recommendations

- Test every limit of the system that the **documentation** says it can handle

- Use **boundaries of equivalence sets**

- Try out your intuitive ideas
  - Exploratory testing

Recommendations

- Try sensitivity analysis
  - where small differences in inputs cause large differences in output
  - where values in the test and the reference function may not agree exactly
- Test randomly selected inputs
- Try all-pairs or orthogonal array testing
- Do a full test by following through

SYSTEM TESTING

System test types

- Concerned with performance and overall fitness of use
- Performed by internal groups
- Oriented towards technical issues

System test types

- Performance test
  - to verify and validate response times, transaction rates and other time sensitive requirements
- Security test
  - to ensure the integrity and confidentiality of the data
- Volume test
  - to determine if the system can handle large volumes of data
- Stress test
  - tests capabilities when conditions overload the resources of the system especially noting performance degradation in processing time
- Compatibility test
  - how the system works with other systems

System test types

- Conversion test
  - verifies the conversion of existing data and loads a new database
- Usability test
  - sees how the user uses it, one screen, flow through screens
- Documentation test
  - verifies accuracy of the docs and that the procedures and screen shots are correct
- Backup test
  - verifies that the system can make a backup if required

System test types

- Recovery test
  - verifies that the system can recover from a software or hardware failure

- Installation test
  - does it install successfully

- Localization testing
  - adapting software for a new place

Exercise – System test definition

- For each test, describe some sort of system test to perform for a web application such as Amazon.com.

Design system fragment tests

- After integrations before full system build

  - sample subsets of full system tests used for each iteration

- security testing

- UX testing

- performance testing

Identify potential acceptance tests

- Optional tests that the user runs at the end of testing
- Redundant testing if the user has been a part of the process up to now.

System test evaluation

- Evaluation
  - Analyze the metrics
    - Test case execution status
    - Defect gap analysis
    - Defect severity status
    - Test burnout tracking
  - Refine the test schedule
  - Identify requirement changes

Exercise – Stress tests

- The calculator does have some stress related problems. Try these tests and see if you can get the defects:
  - Press any digit or operator in sequence fast enough by using two fingers so that you enter so quickly the system can't respond to it.
  - Turn off the system by pressing several keys (but not the OFF key!) simultaneously multiple times. Try to isolate which two / three / four keys make it work and how many times to strike that makes this a repeatable error.
- Find any other action that would cause it to malfunction during normal use.

Report results

- Perform data reduction
  - Ensure all tests were executed and resolved
  - Consolidate test defects by test number
  - Post remaining defects to matrix
- Prepare final test report
  - Prepare the project overview
  - Summarize the test activities
  - Analyze and create metric graphics
  - Develop findings and recommendations
- Review and approve the final report
  - Schedule and conduct the review
  - Obtain approvals
  - Publish the final test report

Testing cheat sheet

- **Variable Analysis**
- **Touch Points**
- **Boundaries**
- **Goldilocks**
- **CRUD**
- **Follow the Data**
- **Configurations**
- **Interruptions**
- **Starvation**
- **Position**
- **Selection**

- **Count**
- **Multi-User**
- **Flood**
- **Dependencies**
- **Constraints**
- **Input Method**
- **Sequences**
- **Sorting**
- **State Analysis**
- **Map Making**
- **Users & Scenarios**

SECURITY TESTING

Special rules

OWASP

- Open Web Application Security Project
  - https://www.owasp.org/
- OWASP Top 10
  - awareness document for web application security
  - perhaps the most effective first step towards changing the software development culture within your organization into one that produces secure code

Security testing tools

- ZED Attack Proxy (ZAP) – from OWASP
  - allows you to probe an existing site for security vulnerabilities in an automated fashion
  - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
- ToolsWatch hackers arsenal
  - https://www.toolswatch.org/ - top 10 tools every year

Google skipfish

- https://code.google.com/p/skipfish/
- active web application security reconnaissance tool
- prepares an interactive sitemap for the targeted site by carrying out a recursive crawl and dictionary-based probes. The resulting map is then annotated with the output from security checks. The final report serves as a foundation for professional web application security assessments.

LOAD TESTING

Determine performance levels and the breaking points of an application

Tips

- Tune early, and tune often
  - optimize your functional use cases / user stories for one actor/role/use
  - prevents you from doing more load testing
- Agree on performance goals
  - setting metrics is part of design
  - discuss response times, web site load, acceptable error levels, expected number of concurrent users with requirements
  - testing with virtual users is not real life
- Stabilize and prepare the environment
  - control the environment
  - alert the admins

Tips

- Gather ownership from all of the development units
  - each area should have a representative available for debugging and making changes
  - requires cooperation from all sides to determine points of failure
- Prepare for the reiterative test process
  - you never can complete load testing
  - take steps to ensure that the tests can be repeated and that benchmarks are recorded so that you can compare the results with the results of previous tests
  - define mitigation plans for failure
  - define contingencies for stopping the test and resuming it once bottlenecks have been resolved

Tools - optimization

- Google PageSpeed Tools
  - https://developers.google.com/speed/pagespeed/

Tools - load test

- Tsung
  - http://tsung.erlang-projects.org/
- Apache Benchmark
  - http://httpd.apache.org/docs/current/programs/ab.html
- httperf
  - http://code.google.com/p/httperf/
- Apache JMeter
  - http://jmeter.apache.org/
- Unix curl command
  - curl -s http://google.com?[1-1000]
- Low Orbit Ion Cannon
  - http://en.wikipedia.org/wiki/Low_Orbit_Ion_Cannon

Tools - load test

- Selenium WebDriver
  - use functional scripts in a Headless browser environment where you can run multiple instances

- PhantomJS or Selenium HTMLUnitDriver
  - Wrap the load tests in a JUnit class. Fork threads for the tests using either Maven Surefire or Gradle (with a filtering test task using maxParallelForks). Gradle or Maven will create a JUnit report (using CSS) that shows each test and the time each one took.

- Visual Studio Ultimate 2012
  - http://msdn.microsoft.com/en-us/library/vstudio/dd293540.aspx

Exercise - Track processor load

- Windows menu / Accessories / Command Prompt
- typeperf -qx | more
- typeperf "\Processor(_Total)\% Processor Time"
  - writes the values for the local computer's performance counter **\Processor(_Total)\% Processor Time** to the command window at a default sample interval of 1 second until CTRL+C is pressed.
- typeperf "\Processor(_Total)\% Processor Time" -si 5 -sc 50 -f CSV -o processorLoad.csv
  - writes the values for "\Processor(_Total)\% Processor Time" to the comma-delimited file **processorLoad.csv** at a sample interval of 5 seconds until 50 samples have been collected.
  - create an Excel line chart with this

Exercise - Run a load test on your site

- http://loadimpact.com/
- About a seven minute test

Exercise – Run a load test on any web server

- Download Apache web server and run the ab.exe file
  - http://httpd.apache.org/download.cgi
- ApacheBench can simulate hundreds of HTTP/1.0 clients simultaneously accessing same resource on the server. Run it with command:
 - ab -n 10000 -c 10 http://centriq.com/
  - ./ab … for PowerShell

METRICS AND TRACKING

Defining

- You can't control what you can't see.
- Admit you have a problem/risk, then data collection becomes valuable.
- Characteristics
  - **measurable** – define unit and quantity
  - **independent** – human interaction should not affect the measurement
  - **accountable** – any interpretation must be backed up by the original data and provide an audit trail
  - **precise** – metrics must be explicitly documented in a data collection process and if there is variance, it's measured as a range or tolerance.

Types

- the business **result** aggregate
  (critical success factor - CSF)
- **specifics** of the result
  (key performance indicator - KPI)
- Defect analysis
- Test effectiveness
- Development effectiveness
- Test automation
- Test cost
- Test status
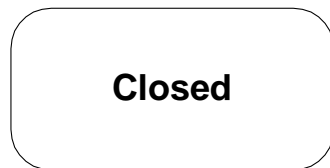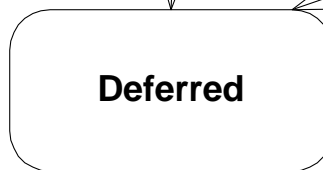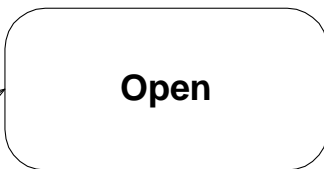- Test extent
- User involvement

Tester enters
new bug report

Visible as bug

Review
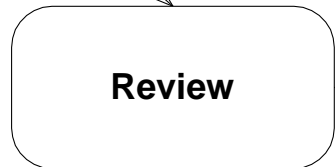
Open

Assigned

[bug needs
lots of rework]

Deferred

Reopened

Rejected

Closed
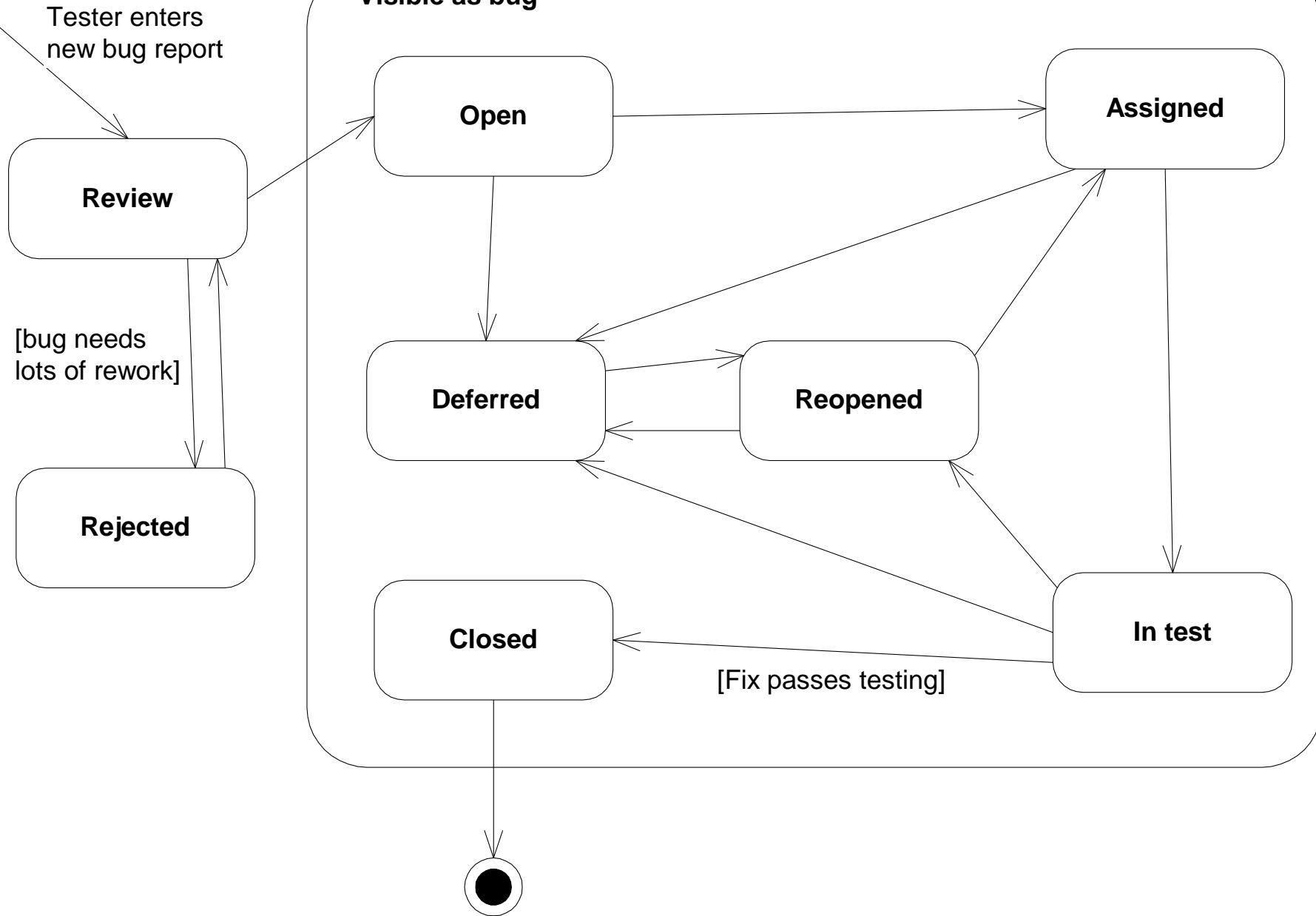
In test

[Fix passes testing]

Test escapes

- something that your team could **reasonably** have found but didn't and was found by the customer
- the bug
- defect detection percentage (DDP)

Exercise - Bug Reporting

1. Write these two sections of a bug report for the bugs in the workbook
   - Problem Summary
   - Problem Description

2. What other tests should you run? Why? Write down your list.

3. Exchange your bug report with another student and read each other's reports.
   - How good is the summary?
   - How clear is the description?
   - How complete is the description?
   - How accurate is the description?
   - How promising is your list of ideas?

Exercise - Product Quality Problems Analysis graph

- Based on the log of the bugs for an application, discuss the chart. Discuss whether this is good or bad and why.

Test-Driven Development

What is TDD?

- a unit testing approach made popular by Kent Beck
  - created Extreme Programming
  - original Agile Manifesto member
  - Smalltalk
  - CRC cards
  - JUnit testing framework that spawned NUnit and others
  - http://www.threeriversinstitute.org/blog/
- underlying assumption is that analysis has not been done and is left up to the programmer
- useful with Agile user stories that skim the analysis

The to do list

- Invoke test method
- Invoke setUp first
- Invoke tearDown afterward
- Invoke tearDown even if the test method fails
- Run multiple tests
- Report collected results

Writing tests

- Arrange — Create some objects
  - usually called initialization

- Act — Stimulate them
  - execute some code with them

- Assert — Check the results

Implementing tests

- Pick a test that you know will work and can teach you something.

- Make your small-scale test work

- Reintroduce the larger scale test

- Make the larger test work quickly using the mechanism demonstrated by the smaller test

- Notice potential problems and note them on a to-do list instead of addressing them immediately

Writing/implementing tests

- Test first, assert first
  - not really possible, it's more like prepare your assertions for the code that define the boundaries of the variables that should have been defined

- Write a list of independent tests

Red / green bar

- Fail / succeed
- Make it simpler and easier, increase the complexity gradually
- Mock up resources when necessary

Mastering TDD

- Use smaller steps to increase coverage or to do refactoring

- Test your code until you have confidence in it (boredom)

- Make tests easy, simple to run, and to use shared setups.

- "Code for tomorrow, design for today"
  - make units closed to further modification but easy to use

Feedback

- How many tests do you need?
    - based on experience
    - complexity is not readily apparent without analysis or trial & error
- When do you delete tests?
    - When your confidence remains the same.
    - When they handle the same scenario for the same role
    - When one is less useful than the other

The process

- Red
  - the test fails
- Green
  - the test works at the minimal level
- Refactor
  - gradually make the code work correctly
- Repeat

Extreme programming (XP)

- Pair programming

- Work fresh

- Continuous integration

- Simple design

- Refactoring

  - remove duplication

- Continuous delivery

Terms

- service
- web service
- mocking
- stubbing
- Test doubles - http://martinfowler.com/bliki/TestDouble.html

Web API testing

- http://katrinatester.blogspot.com/2015/09/api-web-services-microservices-testing.html
  - Katrina Clokie, Bank of New Zealand - @katrina_tester
  - http://www.testingtrapezemagazine.com/

Tools

- Mountebank
  - http://www.mbtest.org/
  - open source stubbing tool to provide cross-platform, multi-protocol test doubles over the wire (Thoughtworks Adopt)
- Postman
  - https://www.getpostman.com
  - Chrome extension that acts as a REST client in your browser, allowing you to create requests and inspect responses (Thoughtworks Adopt)

RESOURCES

Videos

- Cem Kaner - Black Box Testing
  - http://www.testingeducation.org/BBST/
- https://www.udacity.com/course/cs258 - Software testing course
- http://guru99.199tech.com/software-testing.html - video tutorials
- Windows 98 crashes during live demo - http://www.dailymotion.com/video/x23mv0_microsoft-bill-gates-windows-98-cra_news#.Ua_9YEC858E
- Steve Jobs pivots from too slow internet to static photos during live demo of Retina display - http://www.youtube.com/watch?v=-I1KWAX3y50 (4:30 - 5:30)

Groups

- http://www.kcqaa.org/

Sites

- Testing Trapeze
  - http://www.testingtrapezemagazine.com
- Modern Analyst
  - http://modernanalyst.com/

Software

- Bug bounty as a Service
  - Bugcrowd
  - HackerOne