Testing Fundamentals

Diagram

Description automatically generated

Concept of operations

* Specifications of objectives and goals of the system
* Includes strategies, policies and constraints and contextualises the system in the wider business
* Defines responsibilities and authorities
* Specifies processes for initiating, developing, maintaining and retiring the system

System Requirements

* Requirement analysis
  + Requirements must be clear, complete, consistent and unambiguous
  + Formal recording of requirements (documentation) can be in the form of use cases, stories, formal lists and specification

High-level design

* Describe platforms, systems, products, services that the developed system will depend on
* Cover any significant changes that need to be made to any external systems
* Usually include an architecture diagram
  + Contain interface, components and networks that need to be further specified or developed

Detailed design

* Specifies how the system will be constructed
* Functions, object and methods that should be included
* Once detailed design is generated, it should be possible to write tests before the actual system is implemented

Testing

Unit testing

* Testing individual blocks of code
* In procedural programming usually at the function level
* In OO programming usually at the object/class/method level
* The type of testing that I am used to

Integration testing

* Testing the individual components after being grouped together or integrated
* Closer to being the final system
* Often gives rise to unexpected behaviour

System testing

* Test the entire system
* Checks that it doesn’t cause issues or decrease performance on the wider environment
* Affecting broader resource availability on the OS
* Affecting other applications working on the same system

User acceptance testing

* UAT is performed in a user environment, that resembles the production environment and uses realistic data
* Ensures the final system meets the user’s requirements and is ready for use in real time
* This is more about validation rather than verification

+ of V model

* Well structured
* Phases completed one at a time
* Works well if requirements are clear and do not change throughout development
* Simple to understand and implement
* Easy to manage
  + Each phase is well defined and easy to review and evaluate

- of V model

* Very difficult to go back and change functionality once a project is in testing
* No working software created until late in the cycle

Testing categories

* Static Testing
  + Assessing documents
  + Review or walk throughs (usually paper based)
* Dynamic Testing
  + Execute program code
  + White box testing
    - Where the internals of the program are known and can test explicitly paths through the system (code coverage)
  + Black box testing
    - Treats code as a black box and is therefore only concerned with functionality
  + Grey box testing
    - Know both the required functionality and some of the implementation (and/or access to data that are not user accessible in the live system)
    - The code base is however not code base exposed

White box vs Black box

* Code can be seen so many paths can be tested in the program
* Can easily fix the issues
* However must be very thorough and takes a lot of time
* Knowledge of internals can influence test design and lead to bias
* Black box will lead to 0 bias, but not all “paths” will be tested

Code coverage

* Measures how much of your code has been tested
* Looks at code directly
* Different measures of coverage are often referred to
* Often inaccurate or hard to come up with a definite estimate
* Types
* Function/method/(sub)routine coverage
  + Has each method had a test written and run on it
  + Includes constructors and invocations of methods not externally accessible
* Statement coverage
  + Has each line of your code been processed by at least one of the tests
* Decision coverage
  + Has every decision edge been traversed in your code (if/else)
* Predicate coverage
  + Has every condition been tested (not always the same as decision coverage)

EMMA

* Open source code coverage tool developed for Java

Corner cases

* Related to edge cases (variable takes a max or min possible value)
* Where many variables are at their extremes it is a corner case
* Most tests are focused on typical system uses, missing situations where multiple extremes occur
* Usually most difficult bugs, due to rarity and therefore difficulty of repeating causal situation when they occur in real use (unless the code is instrumented in some way)
* Modern code from is usually instrumented in order to help track errors in the field, tackle corner cases and other unexpected performance

Limits of testing

* Obviously not possible to test every signle aspect of a system
  + # of inputs too large
  + Too many paths through the system
* Testing cannot assert software functions correctly under all conditions
* Can only assert that it functions incorrectly under specific conditions