Ecse 429 Final

- People commit mistakes, which lead to defects, which execute and become failures, leading to incidents
- Defects - faulty requirements, communication failure, deviation from requirements, logical design errors, coding errors, documentation errors
- Software quality factors - correctness, reliability/availability, efficiency, integrity, usability, maintainability, flexibility, testability, portability, reusability, interoperability
- SQA - software quality assurance - reasonable confidence that software conforms to requirements, budgetary requirements, and helps manage efficiency of development, maintenance, etc
- 3 SQA principles - know: what you are doing (structure), what you should be doing (requirements, specs), how to measure the difference (pillars: formal methods, software testing, metrics, inspections & reviews, SQA of external participants)
- Verification - are we building product right, validation - are we building right product
- Roles - moderator, recorder, reviewer, reader, producer
- V & V - requirements > system design > architecture design > component design > implementation < unit testing < integration testing < system testing < acceptance testing
- SQA - defect prevention, detection, removal
- Lifecycle - sequential (v-model) - deliver complete, iterative (scrum) - deliver incrementally
- Continuous: integration - build per commit, deployment - release per commit, delivery - easy release mechanics for each step
- White-box cannot reveal missing functionality, black-box cannot reveal unexpected functionality
- TDD - test driven design - listen, create test, code, run tests, repeat
- MISRA C - no unreachable code, identifiers distinct from macro names, right hand operator of && or || cannot have persistent side effect, if else terminates with else
- Dead code - reachable, operation does not alter behaviour
- source code + (lexer + parser) → abstract syntax tree (AST); if AST fails, generate error report
- Graph pattern matching (eg forbid concatenation of empty string); if no matches, no error found through SA
- Soundness - if prover says P is true, then P is true (trivial: nothing)
- Completeness - if P is true, SA says P is true (trivial: everything)
- CFG - control flow graph - join nodes so long as they have at most one exiting and one entering edge; nodes cannot have mutation
- Branch/edge coverage - every decision executed; condition coverage - each condition true and false at least once; modified condition/decision coverage - each condition T or F at least once, one condition changed at a time from previous tests and output must change as well, requires n + 1 cases for 1 decision, n conditions
- Multiple condition > modified condition/decision > condition/decision > condition; condition/decision > branch/edge > statement/node, path > branch edge, path $\not>$ condition, branch/edge $\not>$ condition
- Path sensitization - variable for each declaration/mutation, find ranges for original variable for desired path
- DFG - data flow graph - cu (computational), pu (predicate), d (definition), k (kill), notation *(v, n), write conditions alongside pu
- All-p-use some-c-use - at least one dp path for all d to reachable p, otherwise at least one dc path, all-p > branch/edge
- Fault based testing - modify one thing at a time, if all changes detected, mutant is dead and test set is adequate
- Stillborn - killed by compiler, trivial - killed by most test cases, equivalent - always same output as original program
- Unit test code smells - obscure, eager, mystery guest, general fixture, hard coded test data, test code duplication, test logic in production
- Unit test behaviour smells - assertion roulette, erratic, fragile tests
- Myer's test selection - equivalence classes - until all valid ECs covered, cover as many remaining ECs as possible per test; until all invalid ECs covered, cover one new EC per test
- BVA - boundary view analysis - min, min+, nom, max-, max; 4n + 1 test with n vars, 6n + 1 for robust testing with min- and max+, 5^n for worst case, 7^n for robust worst case
- Decision table - list of conditions, unique combination of conditions \\ list of actions, list of selected actions
- Test generation - all explicit, all-variants (all implicit), all-true (with outcome), all-false (without outcome)
- ROBDD - reduced ordered binary decision diagram - convert BDD from L to R, canonical w.r.t. variable ordering
- Cause effect modeling - label unique conditions and effects, match each effect with condition constraints (eg e2 = (c1 and c3) or (c2 and c4))
- Graph: $\widetilde{\vee}$ nor, flip for nand, O exactly once, E at most one, I at least one, M mask (A implies not B), R require (A implies B); for latter two, arrow towards B
- DNF - disjunctive normal form - terms with and, joined by or
- Absorption - $A \vee (A \wedge B) = A \wedge (A \vee B) = A$
- Each-condition/All-condition - for both, add variant where only one variable is true; each → or logic, all false; all → and logic, all true, note that variant includes its associated negation (eg $\not{C}$ is false when C is true)
- Unique true points - generate tests where only one term true at a time
- Near false points - invert one literal, make selected term true, make all others false given constraints from first term, result: with negation all false, with flip selected term true, so full function true
- Full predicate coverage - make each predicate T or F at least once, change one predicate at a time from previous tests
- System integration - bottom up (no stubs, lots of drivers, count nodes), top down (only driver for main, lots of stubs, count arrows for max, can use module directly after tested), risk driven - start with high risk node
- Stub replaces module - input module → passes test data, output module → returns test data
- Drivers - used to call test modules, eg parameter passing
- Integration testing strategies - big bang - everything at once, sandwich - logic top down, middle, operational bottom up, function/thread-based, top-down, bottom-up, risk-driven
- Integration testing comparison criteria - fault localization, effort (stubs, drivers), degree of testing, parallel dev
- OOP compared to procedural introduces classes, hidden variables, and potential problems with inheritance or attribute redefinitions
- Kung - I subclass, C composition (global ref), A association
- CFW - class firewall, contains all classes referencing self
- Abstract class - tested with level of subclass (eg E(A))
- Kung cycles, remove associations
- Forced error test - trigger error, check if proper error message/recovery occurred, Usability - accessibility, responsiveness, efficiency, comprehensibility, Performance - stress testing, load, durability, endurance, Configuration, Compatibility - eg backwards, forwards
- Combinatorial method - find pairwise combinations one at a time, create table with full combinations while minimizing extra cases
- Security - buffer overflow
- Reliability - MTTF mean time to failure, MTTR ... repair, MBTF ... b/t failure
- Statistical testing - based on usage models, test frequent behaviour, better at reliability estimation but worse at finding defects
- Other system testing - multitasking, recovery, installability, serviceability
- Formal verification - can prove absence of errors and analyze all execution traces
- State machine - event(arguments) [condition]/operation(arguments)
- Statecharts for design - auto synthesis of executable code, check if code generator is working correctly and corresponds to model
- Statecharts for test gen - auto synthesis of test data, check if implementation corresponds to spec
- Round-trip path tree - flatten state model, when adding new transitions, mark as terminal if already encountered
- Conformance test cases: ID, start state, event, condition, reaction, new state
- Kripke structure - flattened synchronous statechart
- Temporal logic - X next, F future, G global, $P \cup q$ P until q, given F q
- Temporal connectives - E exists, A for all
- LTL - linear, CTL - computational