CECS 434 SWE – Testing Strategies Wk #14 vLect #3 4/22

o22. SW Testing Strategies p 466

ITG = Independent Testing Group,
- o- better at finding bugs
- o- costs more

Kinds:
- o- **Unit** T (CF chapter 23) // TDD == Test-Driven Dev
- o- **Integration** T (for joining (modules) diff pgmrs code)
- o- **I&T** == **Integration & Testing** (for joining major parts)
  - o-- Major APIs work as expected
- o- **Smoke** T: See if joint "build" boots (and maybe try a couple features)
- o- **System** T (for joining HW & SW)
- o- **V&V** == Verification and Validation (V2 can be Acceptance T on-site)
  - o-- Verif == works per spec/reqts
  - o-- Valid == users like it (fair users)
  - o-- I V&V == Independent Verification and Validation (by outside group)
- o- **Acceptance** T: Validation on-site – (sometimes)
- o- **Alpha** T: give sys to **in-house** pseudo-users (not on dev team; fresh eyes)
- o- **Beta** T: give sys to small sample of **external** users (often with NDA)
- o- No such thing as a "Gamma T"
- o- **Recovery** T: (can sys recover?) **Inject faults** and see if sys recovers
- o- "**Pen"** (Security) T: (AKA Penetration) can sys be crashed/captured by BGs
- o- **Stress** T: check max sys loading (eg, transactions per second)
- o- **Performance** T: does sys work at req'd perf level? (per non-fcnl reqts)
- o- **Config** or **Deploy** T: does sys work on all target OS's, in all configs

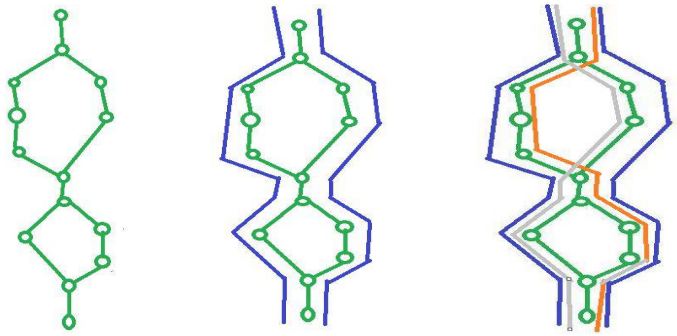
nox Installation T: does this come up on user's H/W config w features they bought

(*) Regression T**: Verify old bugs remain fixed/dead -- don't regress
- o-- Every bug fix has a test; it is added to the "regression test suite"

(*) You are Never done testing -- always bugs left

o23. Testing Conventional Apps p 496
** You can't see bugs until you test.

Black box T: can't see the code inside the box – painted black
o- Test the API & the "concepts" the box represents
White/Glass box T: can see code working
o- **All stmts** run **All paths** run -- expensive



To check, put outputs at the start of each branch (including after a loop)
o- **All loop iterations,** run to end
o- **All conditions** (w all their var values?) – else how do you know they all work?
EX: `if ((3 <= xx) && (xx < len) && (KRED == color))`
==> **3 conds** ==> 2^3 = **8 test combos**
o- All var/slot values run -- never done
o- **Call Tree**: log **entry** to every fcn at fcn body start.
o- **Entry-Exit Tree**: log both **entry and exit** to each fcn.
o- **Entry-Exit State Tree**: log **state+args on entry** and **state+retval on exit**.

Black box T: can't see the code inside the box – painted black
o- All I/O ports run: argl + retval
o- All Input vals run – usually too expensive
o- All EIO pairs

Range of Values T: EX: an int value, enum months, floats
o1. **All "Boundary" values** (both inside & outside)
EX: **int xx** = 0; // xx runs **from 0 to len-1**.
"Corner Cases":
Bds: **inside**: {0, len-1} should work.
Bds: outside: {-1, len} for catching errors?
o2. **Random sample** of interior values
EX: xx in {5, 31, 66, 82}

o31.1.4 The Project [Plan +Control]

Plan, Track and Adjust the project

The Problem with Projects

** **Capers Jones-04 Study**

250 large SW projects from 1998-2004

o- **10%** = 25 **within the plan (AKA the original Estimate)**

o- 20% = within 35% over the plan (AKA **overran** by up to 1/3rd)

o- **70% = failed, or nearly failed (AKA > 35% overrun, but got more money)**

So 30% success or "modest" overrun.