19: Beyond testing; debugging
Rest of the V

- Product requirements
- Software requirements
- High level/architecture design
- Integration testing
- (System-level) Software testing
- Acceptance testing
Scenario: check available funds at ATM

Integration test sends these messages:
- User sends insert card message to ATM.
- ATM requests PIN from User.
- User enters PIN and sends request.
- ATM validates PIN and sends PIN valid to Bank server.
- Bank server sends get available funds to ATM.
- ATM sends send available funds to Bank server.
- Bank server sends available funds to ATM.
- ATM displays available funds.

But also keeps track of sequence/timing of these messages.
Message formats

Messages are data structures with multiple fields

Example: message to validate pin may have fields

- Message header (message #, timestamp, origin ID, etc)
- Message type (“validate” encoded into bits)
- (Encrypted) user ID
- (Encrypted) PIN
- Checksum

Part of integration testing is checking that message formats are consistent between sender/receiver
What are some coverage criteria for integration tests?
System testing

Tests all system requirements

Tests entire system: do inputs at external interfaces cause the correct overall behavior?

Software testing: tests from software POV

Acceptance testing: tests from customer POV

Expensive to instrument, very expensive to repair
Other styles/kinds of testing

Smoke testing - *turn the system on and see if the system works at all (way to check if rest of the system is worth testing)*

Exploratory testing - *skilled tester exercises the system by hand*

Beta test - *product tested by a representative group of users*

Regression test - *did bug fix introduce new bugs?*

Robustness test - *does system hold up to invalid inputs?*

Security test - *can an attacker compromise the system?*

Performance test - *bandwidth, speed, data usage...*
Testing plans

What should be tested?

What level(s) of testing? (unit, integration, system)
What kind(s) of testing for each level?

How should it be tested?

Define testing frameworks, mock functions
Make an argument for sufficient isolation from interference

How thoroughly should you test?

Define coverage goals
Accountability

Testing plan should be written *before* testing

Failing tests should be reported, with a plan to triage/address them

Diagnosing and fixing failing tests is an art in itself (*but good methodology/defects caught at unit level/clean and intentional testing helps!*)

**How do you know testing matches design at each level?**
Traceability applies to tests too!

Example: trace unit test to FSM

Number each test - put each number in row of traceability matrix

Put each transition (and non-transition!) in columns of traceability matrix

Make sure each column has at least one x
Failed tests as process red flags

- Testing is not done in a vacuum – it is one part of a thorough design process
- Finding a bug during system testing probably means:
  - Unit/integration tests missed it
  - Documentation was not thorough
  - Code style guides were not being followed
  - Traceability was not complete/correct
  - Requirements were incomplete/incorrect

Koopman reading from HW9 (23.3.2) has a great and detailed explanation of this
What about this part of the chart?
Peer reviews

Structured meetings for people to review artifacts
Catch defects/discrepancies early
Can be done at every stage (requirements, design, implementation, test plan)

Healthy projects find more than half of all defects in peer review!
What are some anti-patterns (things you shouldn’t do) for peer reviews?
Fagan-style inspections

Participants have familiarity with project
Producer explains artifact but is not present for review
Roles assigned (reader, moderator, recorder)
Identify defect and move on
Give list of defects to producer and rework
Peer review best practices

Review the artifact, not the producer
Limit meeting length (<2 hours)
Have clear roles
Have set goals (checklist); agreed upon beforehand
Do not fix problems

Inspect early and often
Bonus: what about debugging?

Which of the following are your favorite/most important software/hardware tools?

- Oscilloscope
- Debugger
- Compiler/assembler
- IDE
- Logic analyzer
- JTAG/BDM
- Software libraries
- ICE
- Linux tools
- Static analysis tools
- Starter, evaluation kits/boards
- Hardware emulators
- Software testing tools
- Software drivers
- Automatic code generation
- Configuration management tools
- Graphical Design tools
- Network debuggers
- Simulation modeling tools
- FPGA-based prototypes
- Trace

(Top 21 shown)

2019 (N = 434)

2017 (N = 580)

Image source
At a high level, how does a debugger (GDB, IDE debuggers, etc) work?
Hardware debuggers

Run a GDB (or similar) server

Connect directly to pins of chip for access to stack/memory/registers
Oscilloscopes

View and analyze electrical signals
Logic analyzers

View and analyze digital signals
Often have advanced features (decoding common communication protocols)

Image source
Summary

Testing should be done for every level of the V model; should trace back to left side

Coverage helps set goals for how much to test

The earlier the testing, the cheaper (and peer reviews are the cheapest of all!)