Debugging and Testing

Di Fan
2022.09

About Me

https://www.linkedin.com/in/difan/

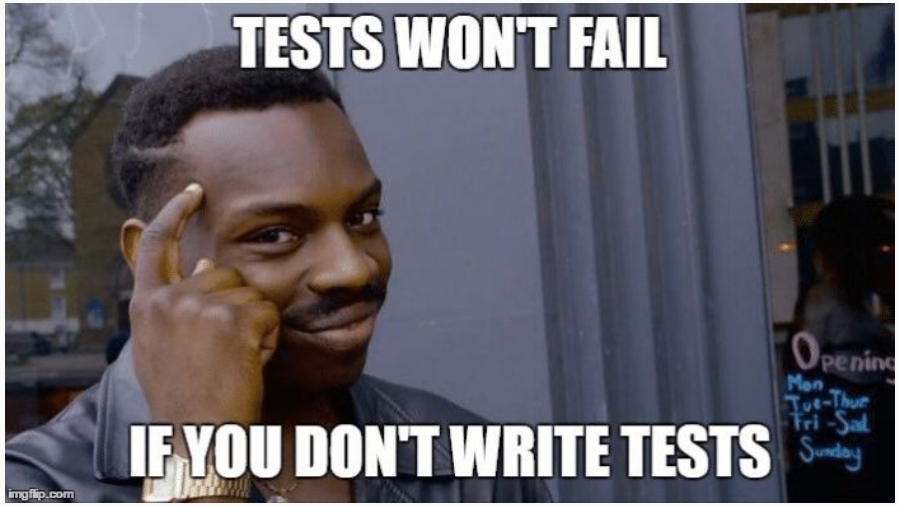
- Senior Software Engineer at Waymo, Simulation
- Previously:
  - Google
  - YCharts
  - Tax accounting
- Passionate about deleting code

Agenda

- The Basics
- Testing in Production
- The Philosophical Challenges
- Q&As

Basics of Debugging and Testing

Why Testing?


TESTS WON'T FAIL
IF YOU DON'T WRITE TESTS
Mon
Tue-Thur
Fri -Sat
Sunday
imgflip.com

Why Testing?

- Developer Velocity = time to:
  - Plan how to make a change +
  - Make the change +
  - Rollback (if there is an issue) +
  - Rollforward

Why Testing?

- Developer Velocity = time to:
  - Plan how to make a change +
  - Make the change +
  - Rollback (if there is an issue) +
  - Rollforward
- Detect issue early

Always Test Your Code

Two major reasons a change are sent back without actually being reviewed

- It was not tested
- It was too big

Unit tests are generally expected for standalone modules

The Easiest Thing to Test Ever

```
const is_even = x => x % 2 === 0
const square = x => x * x
const sum = (x, a) => x + a

[1, 2, 3, 4, 5]
    .filter(is_even)
    .map(square)
    .reduce(sum, 0)
```

Easy to Test

- No state
- No side effect - Hermatic
- Data transformation

Is This Easy to Test?

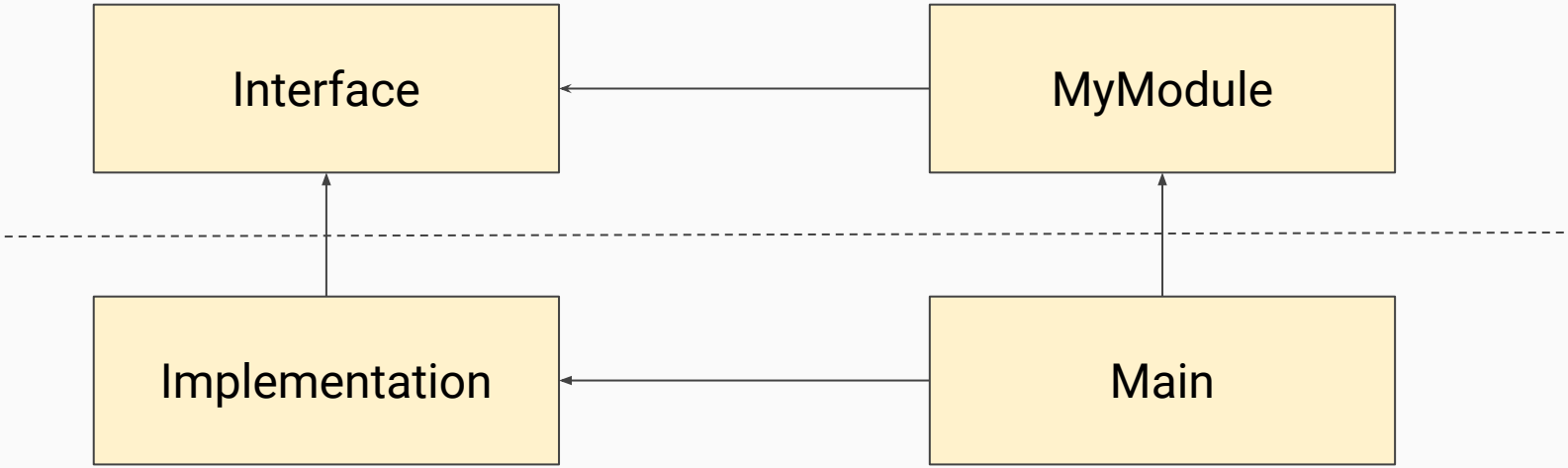
```
int32_t MaxPlayerScore(std::span<const PlayerID> player_ids) {
  std::vector<int32_t> scores;
  auto score_db = MakeScoreDB();

  for (const auto& player_id : player_ids) {
    scores.push_back(score_db.get_score(player_id));
  }
  return std::max(scores);
}
```

Dependency Injection

```
int32_t MaxPlayerScore(
          std::span<const PlayerID> player_ids,
          const std::function<int32_t(const PlayerID&)> fetch_score_fn) {
  std::vector<int32_t> scores;
  for (const auto& player_id : player_ids) {
    scores.push_back(fetch_score_fn(player_id));
  }
  return std::max(scores);
}
```

Testable Components - Dependency Inversion



Program Correctness Bugs

```
Context InitializeContext() {
  std::unique_ptr<Database> db = MakeDb();
  return Context{.db = db};
}
```

How to avoid?

- ASan test: https://github.com/google/sanitizers
- Code Reviews
- Use Rust / languages with GC

Running Unit Tests in Alphabet

- Build with Bazel: https://bazel.build/
- Automatically re-run test on code changes: https://github.com/bazelbuild/bazel-watcher

Debugging

```
DiffResults DoDiff(int64_t test_id, int64_t diff_id) {
  auto data = FetchData(MakeQuery(test_id));
  auto diff_data = FetchData(MakeQuery(diff_id));
  auto joiner = MakeJoiner(data, diff_data);
  joiner.DoJoin();
  return joiner.DiffResults();
}
```

Debugging

```
DiffResults DoDiff(int64_t test_id, int64_t diff_id) {
  auto data = FetchData(MakeQuery(test_id));
  auto diff_data = FetchData(MakeQuery(diff_id));
  auto joiner = MakeJoiner(data, diff_data);
  joiner.DoJoin();
  return joiner.DiffResults();
}
```

Recall Code is AST. Tree can be binary-searched!

Debugging - With Logging / Printing

```
0  DiffResults DoDiff(int64_t test_id, int64_t diff_id) {
1    auto data = FetchData(MakeQuery(test_id));
2    LOG(INFO) << data.size();
3    auto diff_data = FetchData(MakeQuery(diff_id));
4    LOG(INFO) << diff_data.size();
5    auto joiner = MakeJoiner(data, diff_data);
6    joiner.DoJoin();
7    return joiner.DiffResults();
8  }
```

Debugging - Retained Logs

- Machines retain logs from applications they run
- Debugging in production usually starts with reading these logs and mentally binary search them
  - Across Binaries
  - Across Code Modules

Integration Tests

- Testing integration with your backends
  - Databases
  - RPC / HTTP services
  - Remotely hosted data files
  - Message brokers
- Using the real backend vs. Using a sandbox / blackbox
  - Is the backend internal in your organization?
  - Org-wide testing framework, e.g. recording and replaying RPC responses

Integration Tests

- More costly, potentially more flaky
  - Need to run less frequently
- Presubmit
  - Before sending out for review
  - Before submission
- Continuous Integration Test (Postsubmit)
  - As Cron
  - In release pipelines

E2E Test

- How do you setup such a thing?
- Once you set it up, how different is it from production?
- Why not just test in production then? :)

Testing in Production

Testing in


I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO, I DO IT IN PRODUCTION
quickmeme.com

Why Testing in Production

- Behaviors that can be difficult to replicate in test environment
  - Machine configurations
  - AAA
  - Frameworks / implementations, e.g. server logging, metrics & monitorings
- Statistical metrics that need to collect over a significant amount of traffic
  - Latency
  - Resource usage
  - Behavior metrics - Search quality as measured by XYZ
- Not always a direct translation to a pass-or-fail

Why Testing in Production

- ~~Testing in Production~~ Continuous Deployment
- Instead of having a pass-of-fail results, i.e. that's blocking, testing in production act as fitness functions that guide the system's evolution

Tools of Testing in Production

Taking a Single Server as an example:

- Monitoring
- Experiments
- Automated & Staged Releases

Generating Metrics - Monitoring

- Counters!
  - Request Count
  - Error Count
  - Latency
- Counter dimensions
  - Data center, Job name, Software version, Endpoint Method, Error Code
- Convert Counter to Dashboards and Alerts
  - Bucket the data stream

Generating Metrics - Logging

- e.g., Average length of user sessions and CTA rate
- Log individual events (e.g. API request, UI event) and join using data pipelines
  - Server framework for per-request logging
- Convert to dashboard / alerts

Live Experiments

- The most common form: A/B test
- Examples
  - Rolling out new UI to users
  - Testing different page size of a List API
  - Experimenting a different search algorithm / ML model

Experiments - How Does It Work

- Put your code-change behind an experiment flag
- Experiment flag’s value is controlled by the experiment configurations
- Experiment config is released through data push instead of binary releases
- Collect metrics over time to determine if the experiment leg is favorable
- Ramp up experiments gradually, all the way to 100%, if needed

Automated Releases

Performing the following steps using an automated job:

- Cut a release
- Run all tests at the commit
- Build the binary (as a package) and label it
- Running any validation tests with the new binary
- Update the running servers to use the new binaries
    - Which running servers?

Automated Releases - Rollback

Code rollback

- Rollback a change and submit it
- Cherry-pick into current release

Why not binary rollback?

- When is binary rollback possible?

Staged Releases

Again, changes don't *really* go straight to production

- Autopush -> Staging -> Production

Before Testing in Production…

- Run a server locally
- Run a server as a one-off dev instance
  - Use the same deployment tools as the production workflow

- Make the server as a blackbox as other servers' backends

Complex Systems

- Orchestration
  - e.g. Testing change to a service which is primarily used in a data pipeline
  - Hot-swap not always possible
- Component Interface & Contract
  - Did an upstream component pass the correct input to its downstreams?
  - Compatibility concerns

Global Systems

- Examples
  - Cluster Management
  - Quota Management
  - Service Framework and Registry
- Rolling out by clients
- Highest Risk
  - Config change as the #1 source of outages
  - Modeling impact

The Philosophical Challenges

What Does Testing Do, Anyway?

- That code *passes* the *test cases*
- Questions
  - Do test cases cover all real world possibilities?
  - Is it even possible to have test cases cover all real world possibility?
  - Is it even necessary to have test cases cover all real world possibility?

The Knowledge Problem

- *“The code will not fail for these test cases”* is different from:
- **This code will work**
- It’s impossible to prove the code will work, just like how it’s impossible to prove there is no black swan

The Knowledge Problem

- *“The code will not fail for these test cases”* is different from:
- **This code will work**
- It’s impossible to prove the code will work, just like how it’s impossible to prove there is no black swan

Example - Autonomous Driving Systems

Say we want to make sure an autonomous vehicle does not crash into another object (car, motorcycle, pedestrian etc) - an interesting event

- Of course that happened before, which becomes TP test cases
- Note the asymmetry:
  - When collision happens, someone could die

Example - Autonomous Driving Systems

- Smoke testing
  - Run against past collisions / difficult scenarios to make sure the software behave correctly
- Statistical Discovery
  - Run against as many inputs as possible
    - All past driving logs
    - Artificial scenarios
  - A lot of FP that requires human discretion to distinguish from TP
- …and everything in between

Example - Autonomous Driving Systems

Open questions

- What should be signals? Can we have enough signals?
- Cost and the ever growing dimensions of parameters
- Real world inputs from past vs. artificial inputs to improve coverage
- The unknown unknown

Q&A