Managing Technical Debt

Lessons from the field about pitfalls to avoid
and how to set you and your startup up
for future success

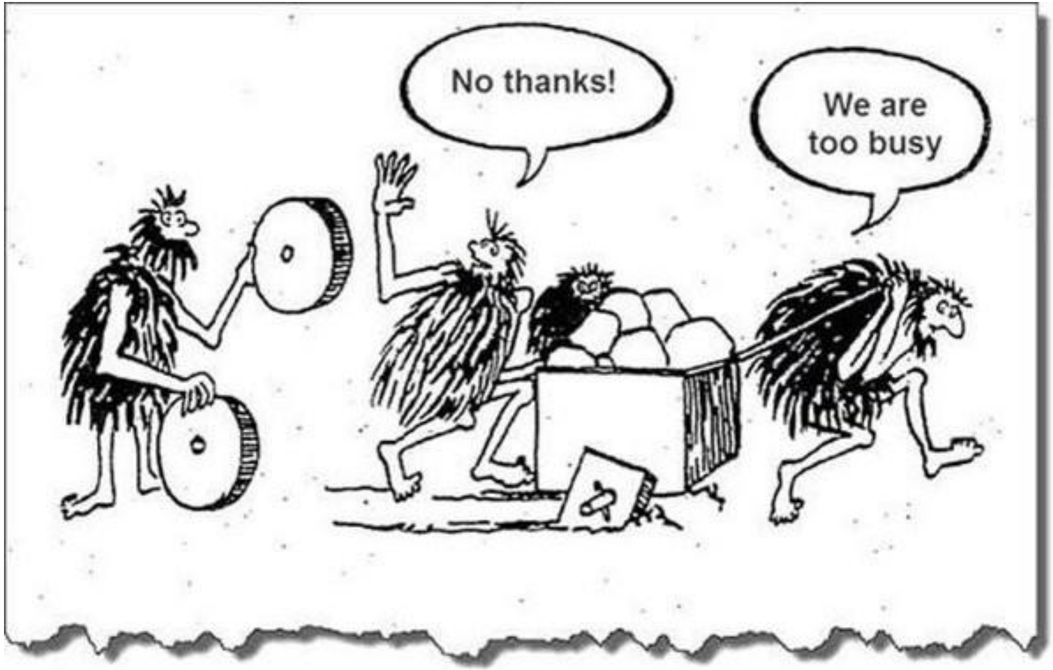
Your friendly neighborhood technical co-founders

What is Technical Debt?

Expedience Now, Pain Later

Substandard design caused by:

- ➔ Incomplete/bad design specification
- ➔ Lack of ownership
- ➔ Unrealistic timeframes
- ➔ No standardized development process
- ➔ No automated testing strategy



The cause of technical debt in many organizations

Get You A Spec

Gathering Requirements

How can you build something if you don't know what it is?

- → Whiteboard
- → Wireframe
- → Workflow
- → High-fidelity mockups
- → Specification

balsamiq® Sketch

Adobe

Case study #1

Struggling to maintain a coherent vision

- ➔ Overview
- ➔ What went wrong:
    - ◆ High-fidelity mockups are important, but they should not be your starting place
    - ◆ Determine goal of product and make workflows first

Building Your Product

Picking a language or platform or framework that actually makes sense

- Avoid “New and Exciting”
  - Usually Unproven
  - Difficult to hire for
- Short timeframe? Great time to leverage current expertise
- Consider tooling, breadth, and depth of platform
  - Language syntax is irrelevant
  - Test, test, test!

Case study #2

Implementing a DHT in Rust

- ➔ Overview
- ➔ What went wrong:
  - ◆ Testing infrastructure
  - ◆ Debugger (!)
  - ◆ Network stack
  - ◆ Quality of standard libraries

Test your biz

Testing

Thou *shalt* test thy code

- Prove features work as specified!
- Lets you safely refactor
- Quickly exposes design flaws
  - Tight coupling
  - Performance issues
- Defects can be tied to test cases
  - Reproduce defect in failing test
  - Fixing defect should fix the test

Unit Testing

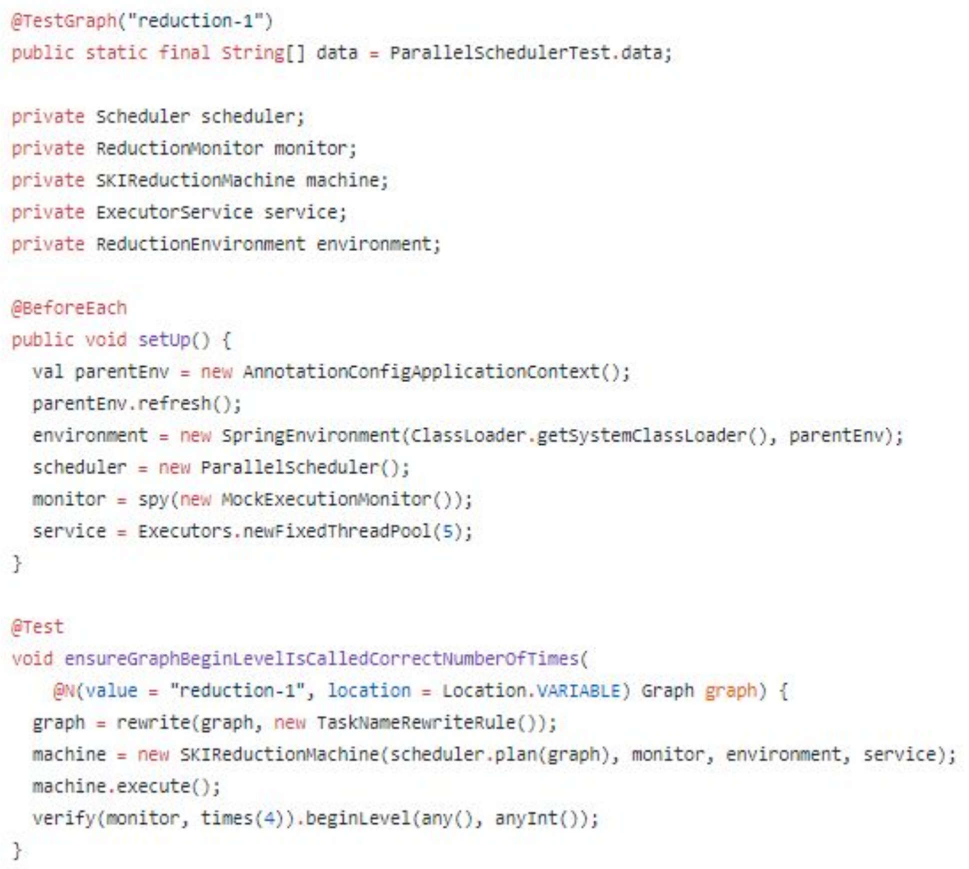
- Narrow scope
  - Test only “unit” (function/method, class)
- Quick to run
- Easiest practice to improve velocity and quality
- Tools:
  - JUnit, NUnit, RSpec, Spock
- Contractors must provide tests!

Integration Testing

- Wider scope
  - Test interactions between multiple units
- Includes interaction with database
- May take longer to run
- Can be more difficult to write
- Helps Refactorability/Design!
- Usually “built in” to framework
  - Spring Test, Ruby On Rails, etc

System Testing

- Widest Scope
  - Entire system under test
- Simulates real usage
- Can be hard to develop and expensive to maintain
- Greatest value usually after product functionality “frozen”



```
@TestGraph("reduction-1")
public static final String[] data = ParallelSchedulerTest.data;

private Scheduler scheduler;
private ReductionMonitor monitor;
private SKIReductionMachine machine;
private ExecutorService service;
private ReductionEnvironment environment;

@BeforeEach
public void setUp() {
  val parentEnv = new AnnotationConfigApplicationContext();
  parentEnv.refresh();
  environment = new SpringEnvironment(ClassLoader.getSystemClassLoader(), parentEnv);
  scheduler = new ParallelScheduler();
  monitor = spy(new MockExecutionMonitor());
  service = Executors.newFixedThreadPool(5);
}

@Test
void ensureGraphBeginLevelIsCalledCorrectNumberOfTimes(
    @N(value = "reduction-1", location = Location.VARIABLE) Graph graph) {
  graph = rewrite(graph, new TaskNameRewriteRule());
  machine = new SKIReductionMachine(scheduler.plan(graph), monitor, environment, service);
  machine.execute();
  verify(monitor, times(4)).beginLevel(any(), anyInt());
}
```

System test example from Sunshower.io

Scale your stuff

Scalability Considerations

You're not Google. Google wasn't Google.

- Scaling is *hard* (past a certain point)
    - No one-size-fits-all solution
- Previous design decisions impact scalability
- Cost considerations
- Vertical vs horizontal scaling
- Storage *will* be your bottleneck
    - Also probably where you'll spend the most $$

Platform

Platform doesn’t really matter

- Most applications will not be held back by the platform they’re in
  - Time spent in code: ~0.1%
  - Persistence and network dominate CPU time
- Most modern platforms within a factor of 2 of each other (PHP, Ruby On Rails, J2EE)

Platform

Up to a point ...

- ➔ Languages with many features can be hard to standardize
- ➔ “Functional programming” usually means “write-only”
- ➔ Bottleneck is still the database

Speak of the DB: SQL

Getting this wrong will ruin *everything*.

- ➔ SQL offerings have decades of maturity and optimization and zillions of features
- ➔ Scale to TB of data and 10k+ users without really doing much
- ➔ Relatively easy to get expertise
- ➔ Details *really matter*
- ➔ Replication Strategies

NoSQL

This is a big topic

- ➔ NoSQL does not automatically mean "Big Data" or "Highly Scalable"
- ➔ NoSQL = everything that's not SQL
- ➔ Wide-column store vs. Document
- ➔ How much data do you *really have*
- ➔ Rule of thumb: use NoSQL only if you know exactly what problem you need to solve

Case Study #3

NoSQL and no plan for Datacake

- ➔ Overview
- ➔ What went wrong:
  - ◆ Deployment = difficult
  - ◆ Testing = difficult
  - ◆ Expensive to scale
  - ◆ Analytics implementation was less expressive than SQL
  - ◆ Ad-hoc data format = really bad

Put it on the cloud

Serverless or Function-as-a-Service

The new kid on the block

- ➔ "Pay for what you use"
- ➔ The most "microservicey"
- ➔ Network usage and function startup time = expensive
- ➔ Hellooooo vendor lock-in!
- ➔ Data integrity can be *really hard to enforce*
- ➔ Pricing the hardest to reason about

Containers

Your application in a box

- Typically requires “orchestration”
- Can simplify *complex* deployments
- Microsoft “native support” still pretty rough
- Vendor-specific orchestration flavors
    - AKS, EKS, GKS still require a “cluster” of cloud Instances
    - Markup is pretty steep (+100-200/month)

Cloud Instances

The workhorses of the cloud

- ➔ Inexpensive if done correctly
- ➔ No need to worry about network connectivity or failing parts
- ➔ No vendor lock-in (a server’s a server’s a server)
- ➔ Intentionally obfuscated pricing
- ➔ “Hidden” pricing
    - ◆ Network traffic
    - ◆ Storage costs

Q&A

email lisa@sunshower.io or josiah@sunshower.io

for any other questions or for help scaling your cloud