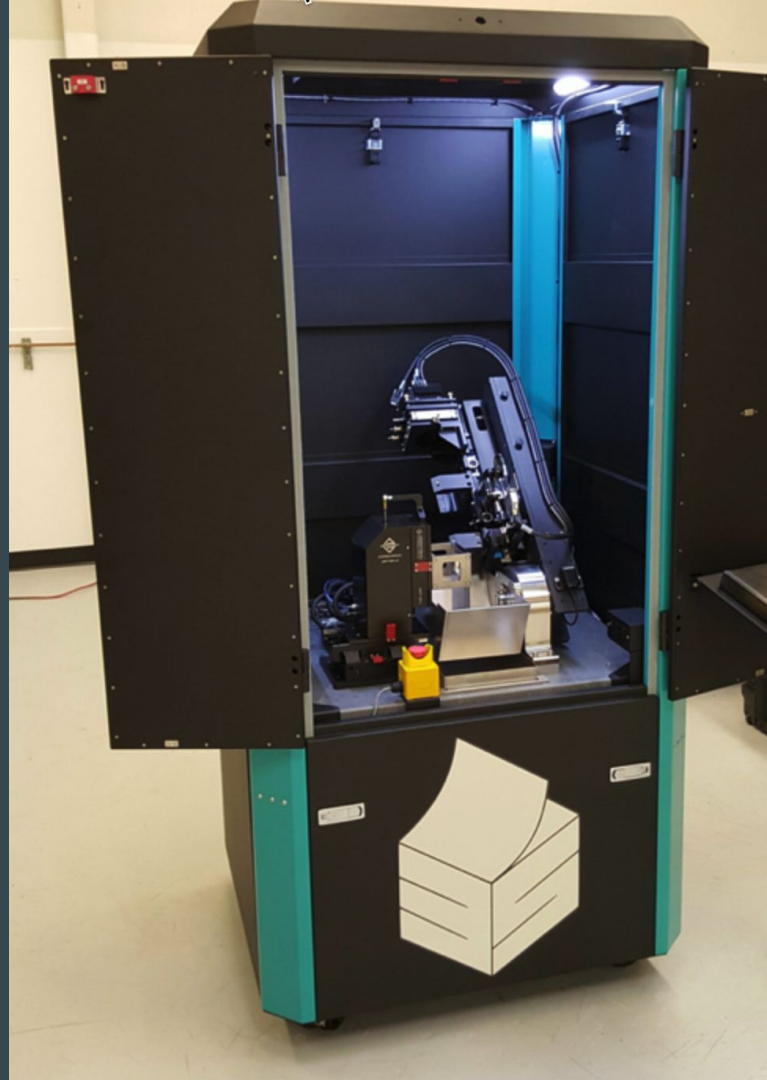
Testing Practices in the Wild

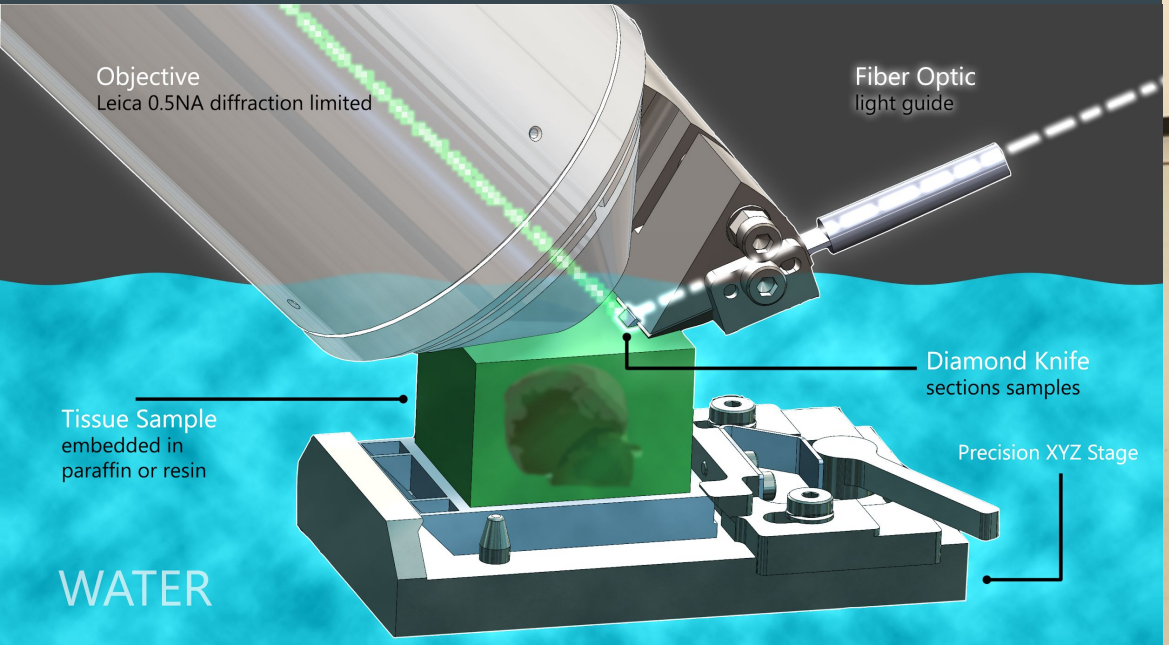
Ilica Mahajan

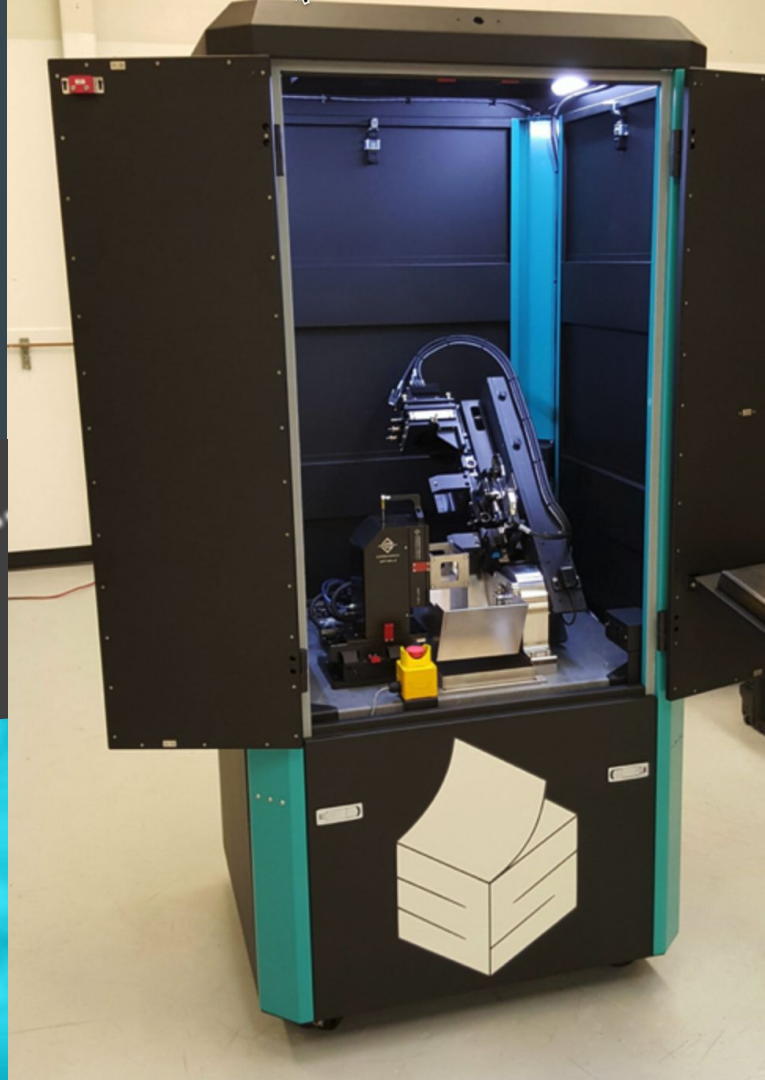
Former Distributed Systems Engineer at 3Scan



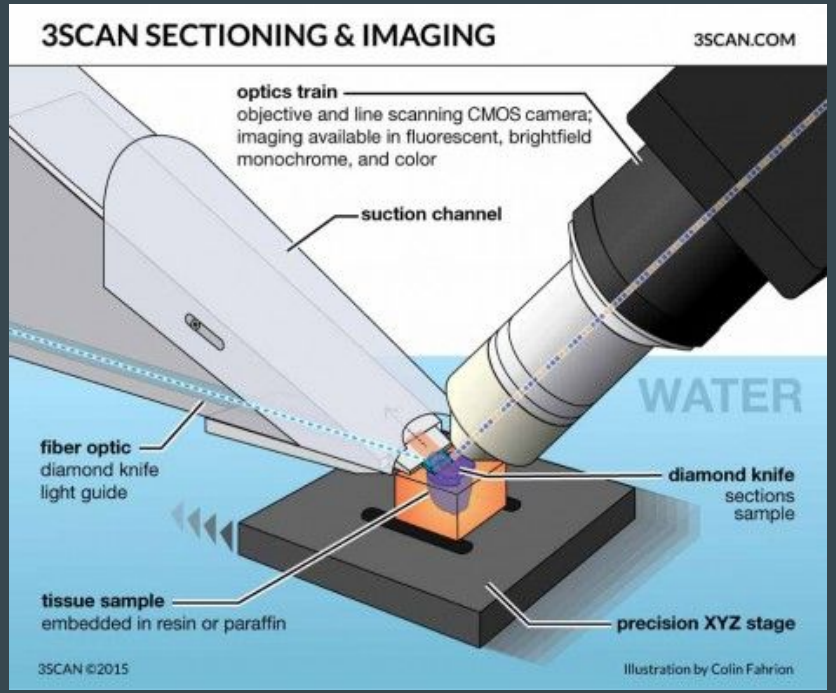
What does 3Scan do?

- Build robotic microscopes that serially dissect tissue samples
- Render that data in 3D for analysis






3D Reconstruction
3SCAN
VASCULAR FINGERPRINT



Knife Edge Scanning Microscope

Why was testing important?

- We had production systems that were expected to perform and deliver on deadlines to customers

- We had other teams relying on our systems to do their job

- We wanted our lives to be easier and stress free

More than just about finding current bugs...

Why testing is doing yourself a favor:

- Sure, prevents bugs in production
- Robust to refactoring
- Defining the interface
- Ensures one method, one purpose
- Defines and aligns expectations with reality
    - Documentation can lie to you!
- Helps others understand your code

Code Coverage Tools

- Automatic checks on all new pull requests to check that ~90-95% of lines of code were tested

- Integrated with GitHub to block the merging of PR's with non-adequate coverage

- Be warned: 100% code coverage does not mean your code is well tested
  - All lines were run, not that the expected behavior was tested

Encourage good behavior from engineers (or users) by making the wrong thing hard to do

Blocking/Warning on the Merge Button

- Tests didn’t pass
- Code coverage was not high enough
- Lint checks failed
- Code not reviewed
  - Code not reviewed by at least one “owner” of that portion of the code base

Continuous Integration

- GitHub integrates with tools that run all tests before a merge
  - CircleCI
  - Jenkins

- Specified which tests run in parallel and which serially

- Docker containers managed and run in the cloud

Unit Testing

- Each file's unit tests housed in a parallel tree structure

- Every method, every failure case, every branch of behavior, etc.

Tests should be written around the same time as the code
(or before as in TDD)

Tests can help clarify the task at hand.

What should this method do?
What information does it need to do that?
What's the easiest structure for the method to take?
What failure cases does it need to be able to handle and how should it fail?

Integration Tests

- Systems that interact need to be tested for their compatibility

- Also test how one system behaves and recovers when another fails

Acceptance Tests

- Testing all or most of your systems together in a dedicated staging environment with real database (not your production database)

Staging Environments and Mocks

Bug Tracking

- Customer support should have a way to report bugs to you in a way where engineers can see what is going wrong (Jira)

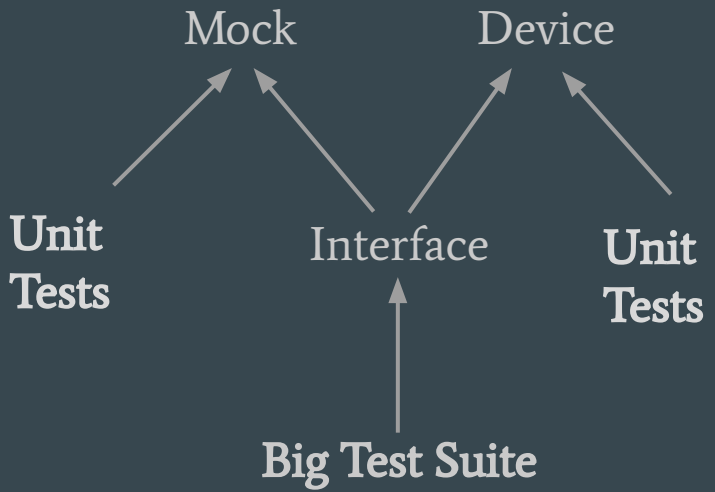
- GitHub Jira integration allowed for tickets to be closed automatically when pull requests dealing with them were merged

Static Code Analysis

- Intellij (IDE) would highlight tons of potential issues like duplicate method names, unused returned objects, duplicate code, unused methods

Your IDE is your friend.

Our Hardware/Software Testing Methodology



- Software mock of a hardware device. Test it!
- Software unit tests that use an interface.
- Allows unit tests to run against real hardware when needed!

Learn keyboard shortcuts.
[& Learn vim or emacs...]