PACIFIC NW SOFTWARE QUALITY CONFERENCE

Why We Need New Software Testing Technologies

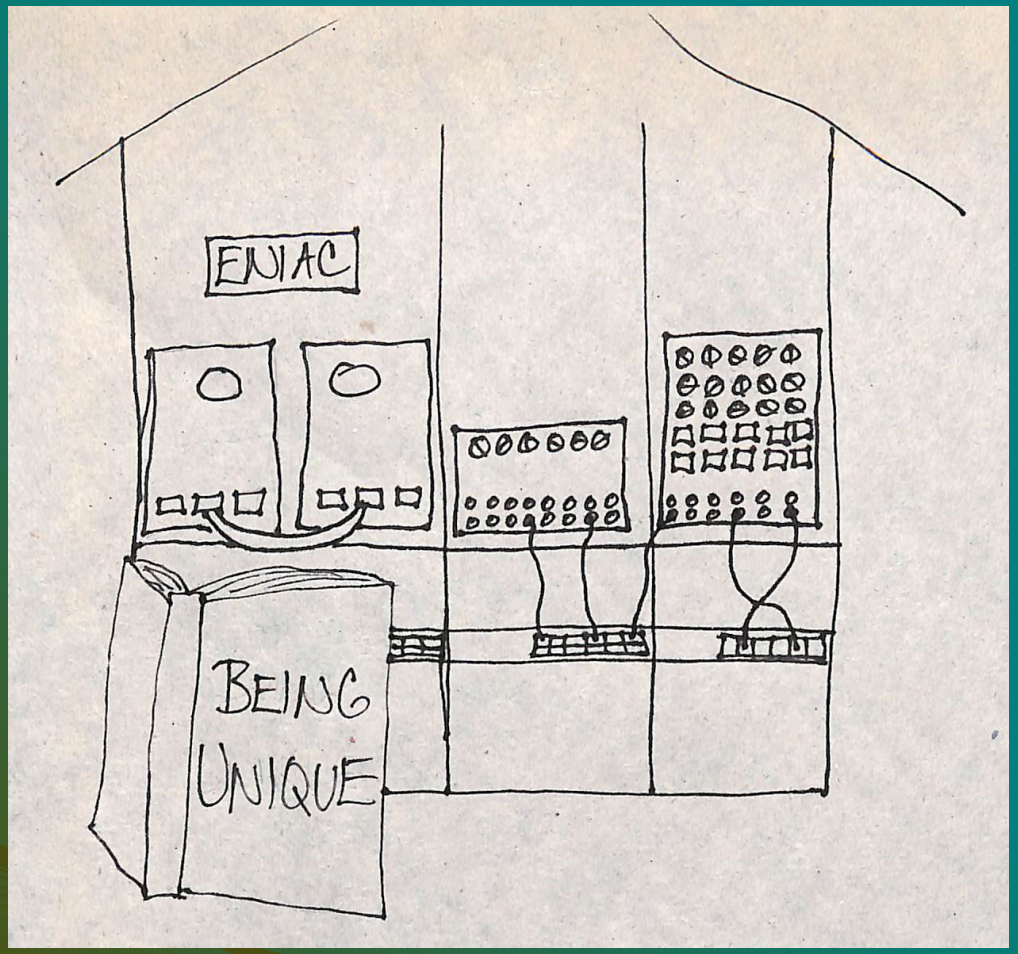
CAROL OLIVER, PH.D.
OCTOBER 2019

PACIFIC NW SOFTWARE QUALITY CONFERENCE

How has the Context of Software Changed?

MAJOR PHASES OF COMPUTING AND SOFTWARE TESTING

The Earliest Computers

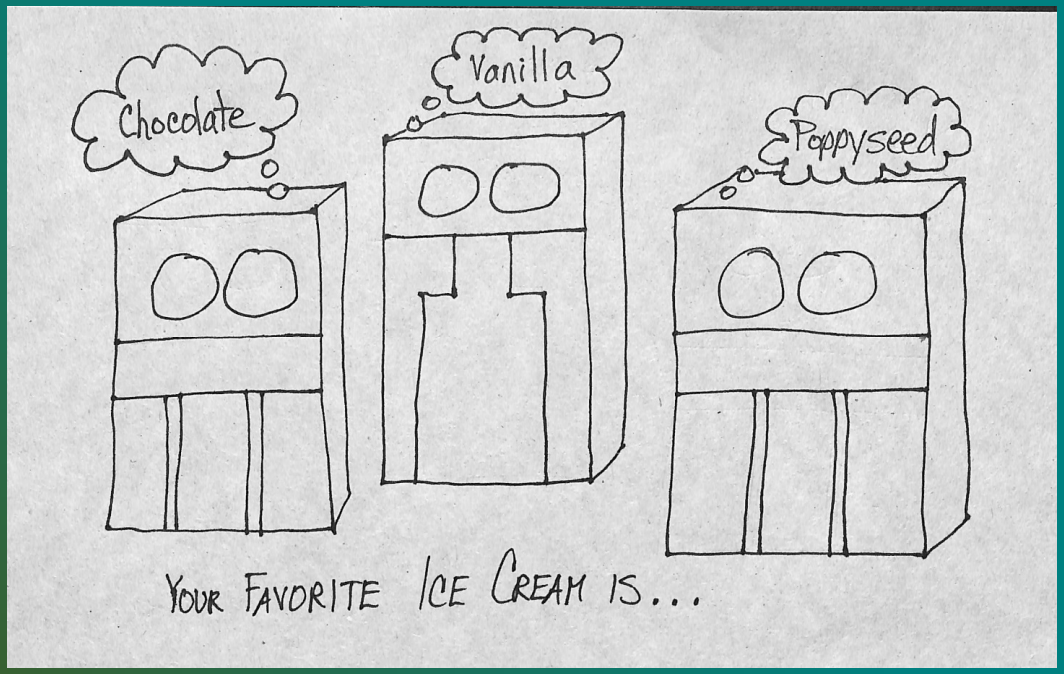


- Extremely Few Computers
- Each an Individual
  - Built for Special Purposes
  - Its Own Language

Software Execution Environment:

- *100% Known*

Commercial Mainframes

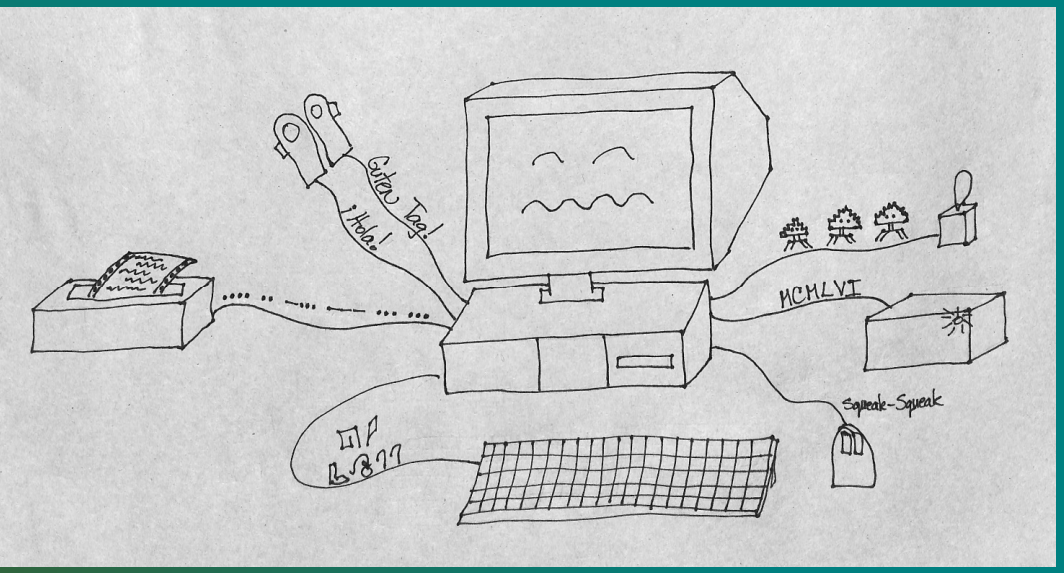


- Still a Small Number of Computers
  - Tens and Hundreds
  - A Few Types / Manufacturers
- Built for General-Purpose Use
- Languages reused across many computers

Software Execution Environment:

- *Largely Similar*

Early Desktop Computing



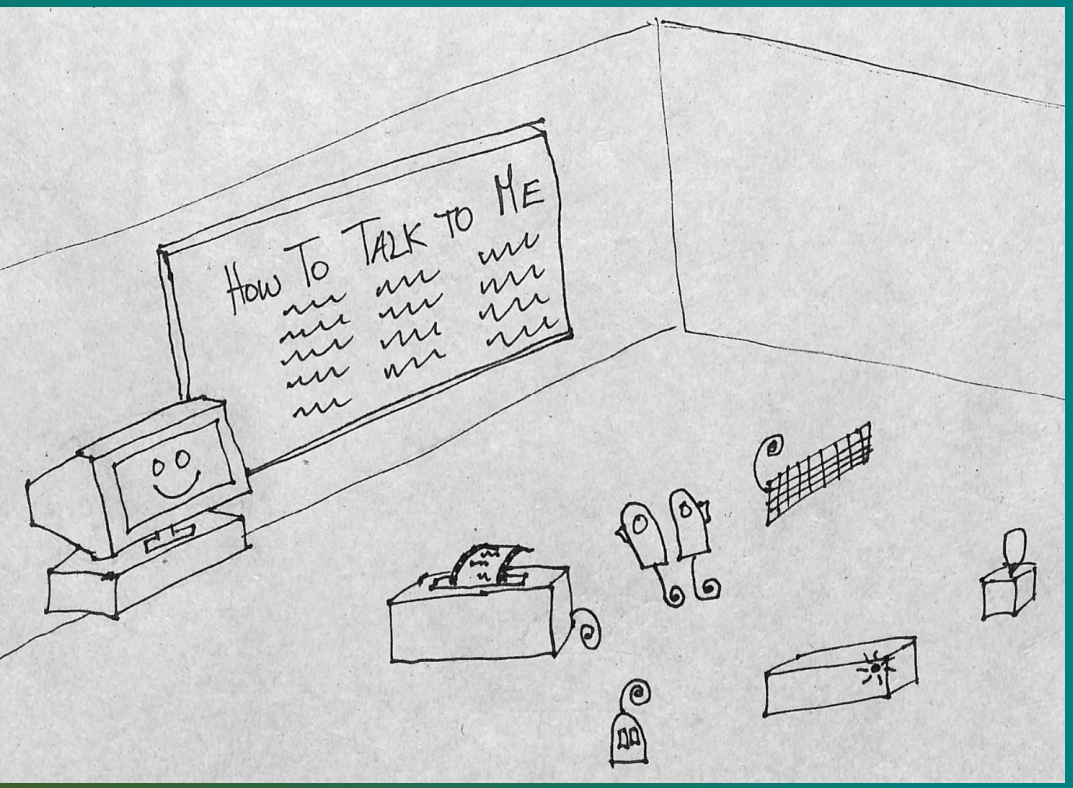
- Many Computers
  - Thousands for General-Purpose Use
  - Dozens of Types / Manufacturers
- Customizable: Many Different Peripherals
  - Each speaking its own language
  - Each with unique capabilities
  - Each with specific restrictions

Software Execution Environment:

- *Many Variations*
- *Wildly Unpredictable*

  (Dozens by Hundreds of Variations)

Late Desktop Computing

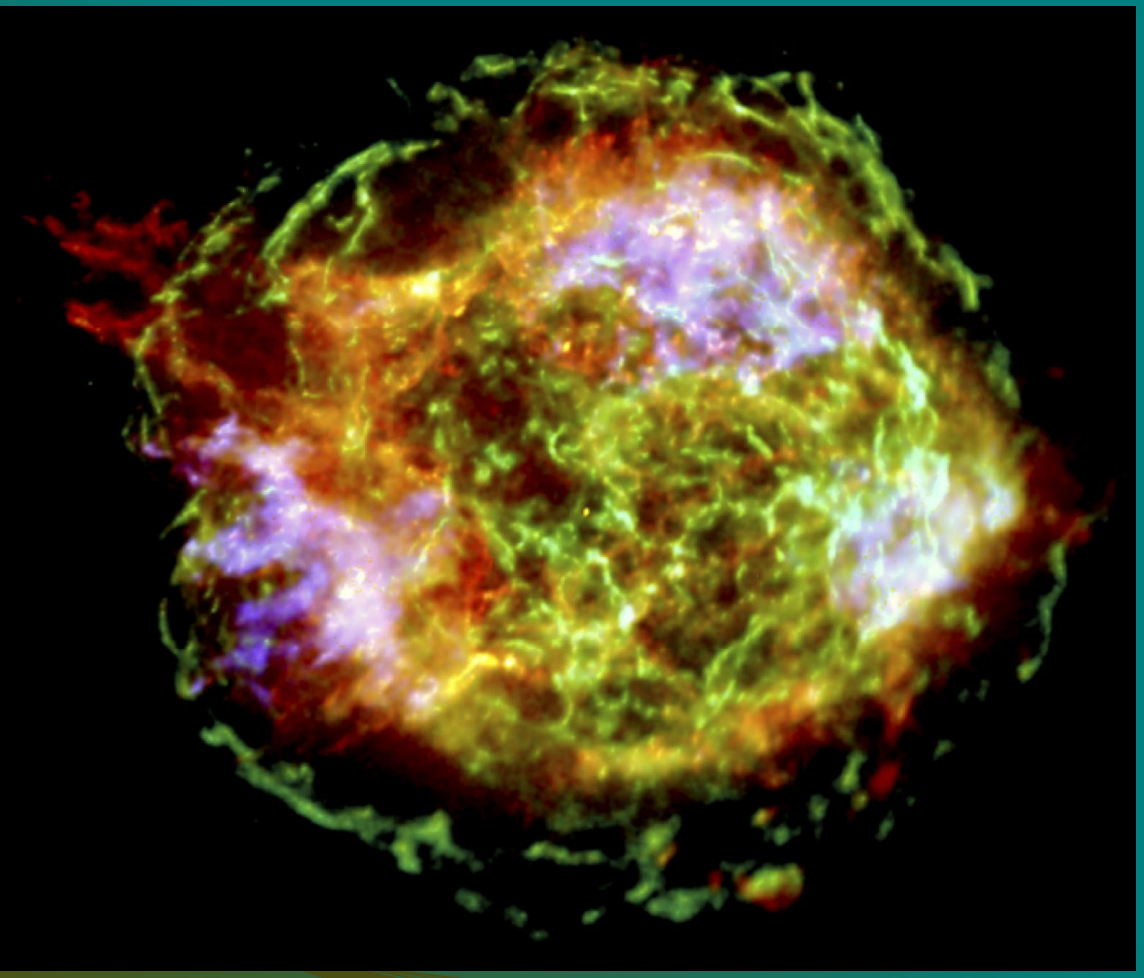


- Many, Many Computers
  - Tens of Thousands of Customizable Computers
  - Dozens of Types / Manufacturers
- Operating Systems Defined Interfaces for Peripherals to Use to Communicate
  - Capabilities of Peripherals Standardized

Software Execution Environment:

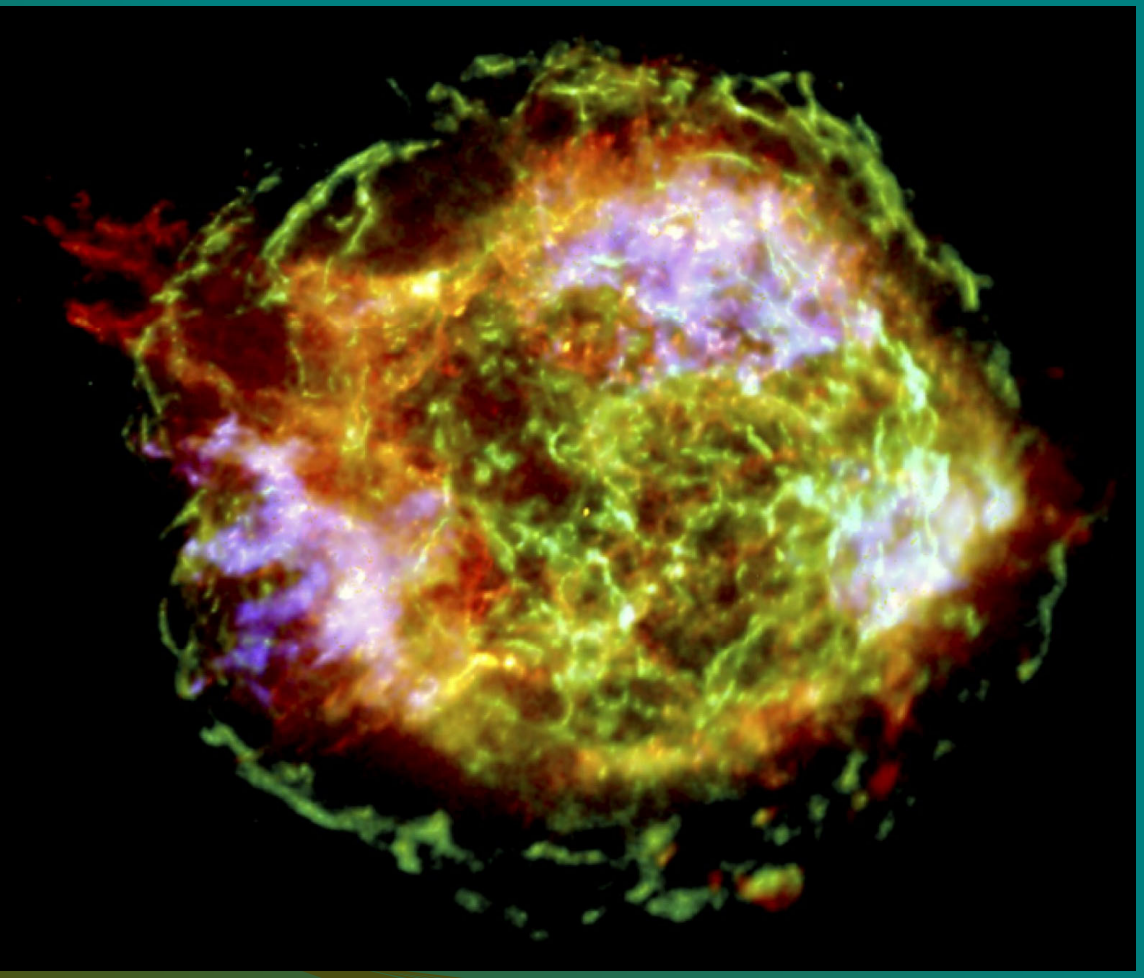
- *Many Variations*
- *Relatively Predictable*
  (Dozens by Dozens of Variations)

Mobile Computing (1 of 2)



- Computers, Computers Everywhere!
  - Millions of Computers
  - Thousands of Types / Manufacturers
- Many with Built-in “Peripherals”
  - Sensors and Interfaces
  - Each with Possibly Unique Capabilities
  - Each with Possibly Unique Restrictions
  - Each with Possibly Unique Communications Interface
- OS Components Possibly Customized (common on Android)

Mobile Computing (2 of 2)



Software Execution Environment:

- *Many Variations*
- *Wildly Unpredictable*

  (Thousands by Hundreds of Variations)

NASA image of supernova remnant Cassiopeia A.
Picture NASA ID 0401563. Not copyrighted, see
https://www.nasa.gov/multimedia/guidelines/index.html

Therefore…

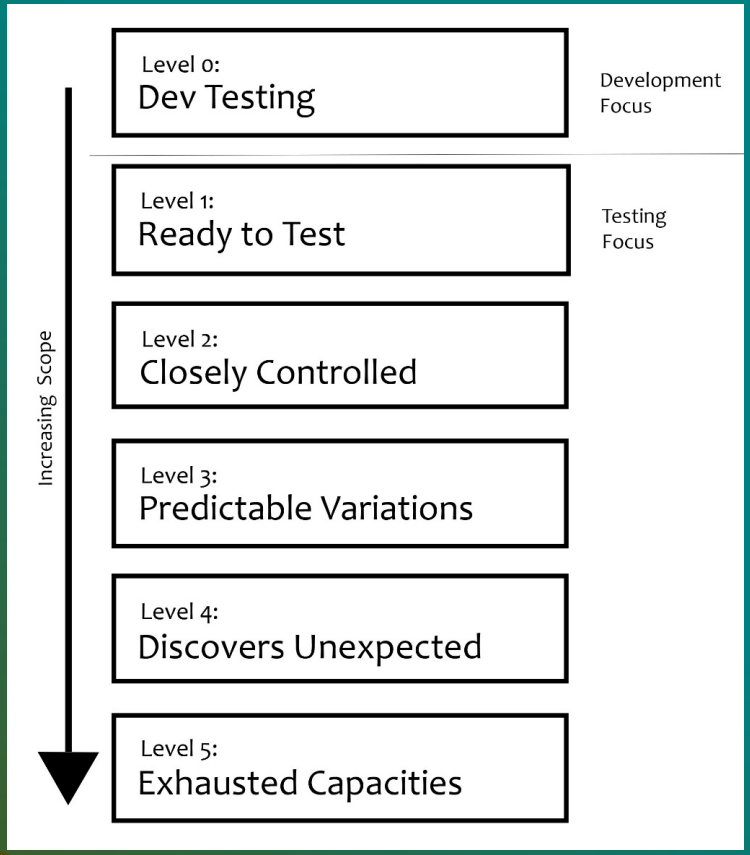
- Vaster scale and scope of environmental complexity in Mobile and IoT era
- Requires testing that efficiently handles vast environmental complexity

PACIFIC NW SOFTWARE QUALITY CONFERENCE

What Types of Testing Efficiently Handle Vast Environmental Diversity?

RELEASE-READINESS LEVELS FRAMEWORK

Release-Readiness Levels Framework (1 of 2)



Development Focus

- About finding a way to make something work

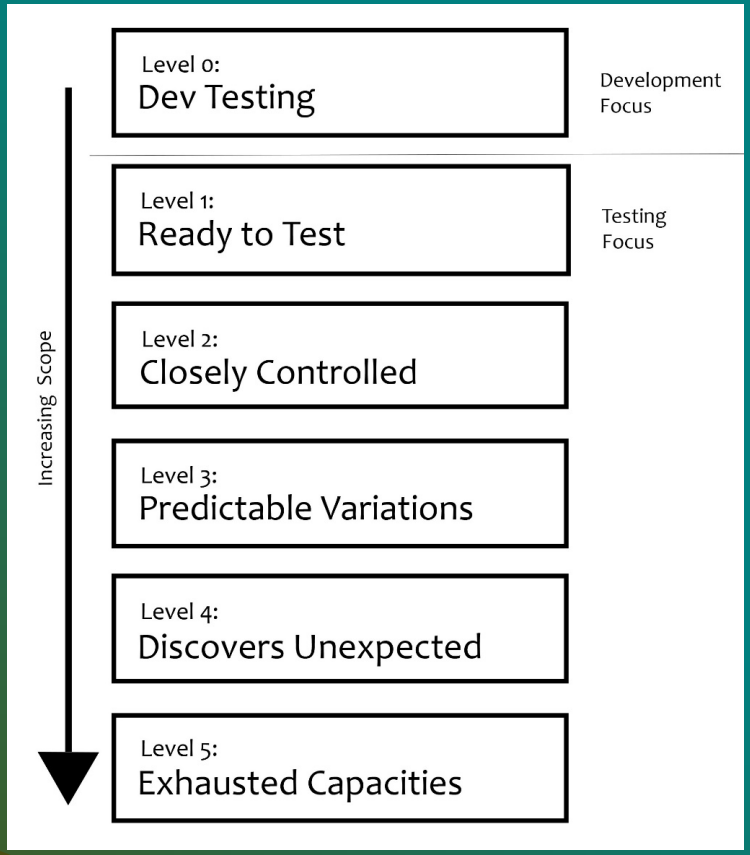
Testing Focus

- About challenging what's been created to see if it will hold up no matter what adverse circumstances occur

Two Creative Efforts

- Shared Purpose: Produce Successful Software
- Different – and Contradictory – Goals

Release-Readiness Levels Framework (2 of 2)



This framework is NOT about:

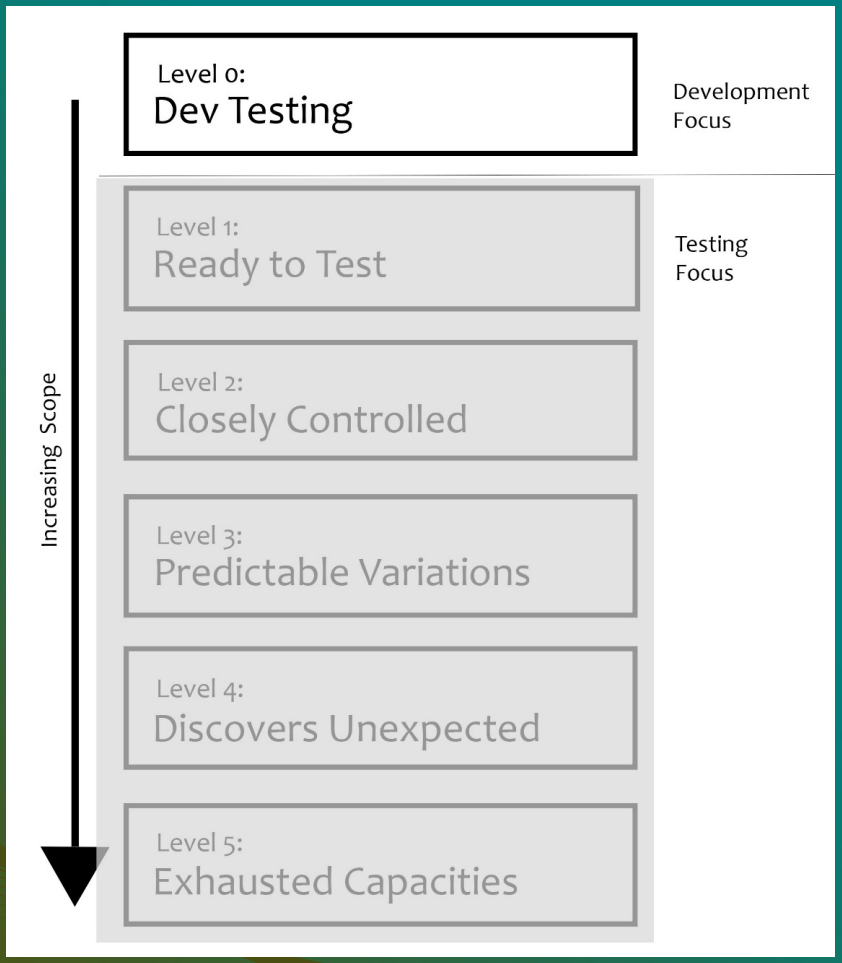
- Any specific SDLC
- Who does what and when

This framework IS about:

- What work it is possible to do
- Why doing that work may be worthwhile

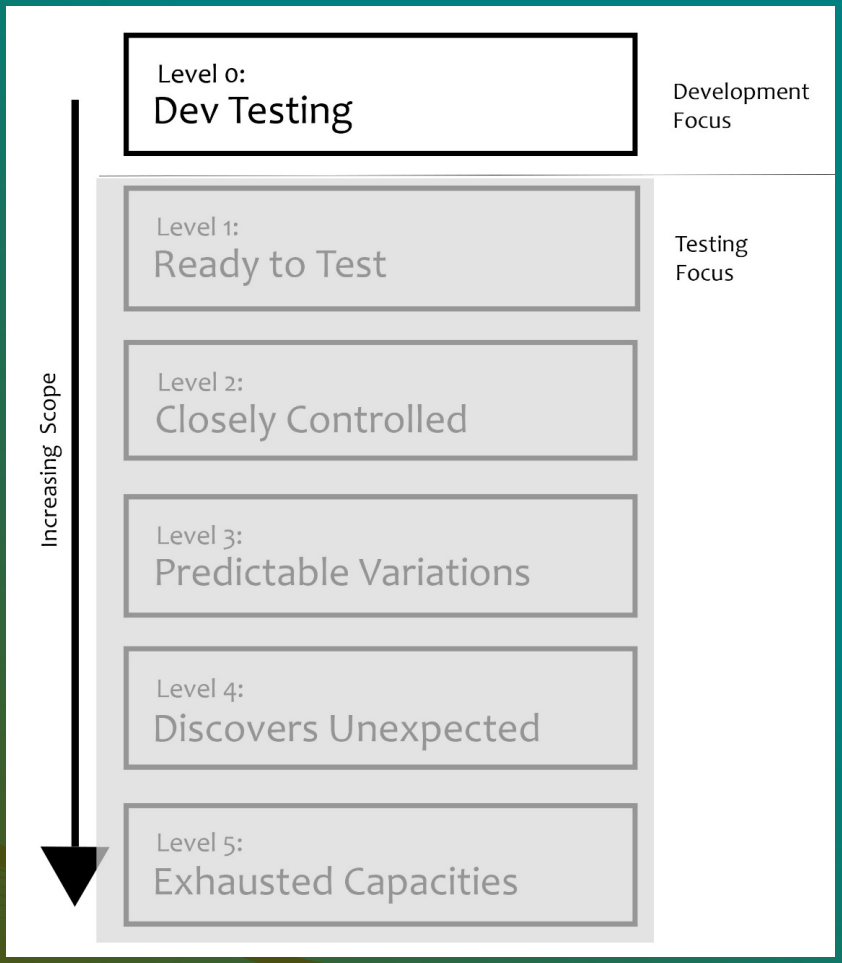
This talk limits its attention to Functionality Testing.

Level 0: Dev Testing (1 of 2)



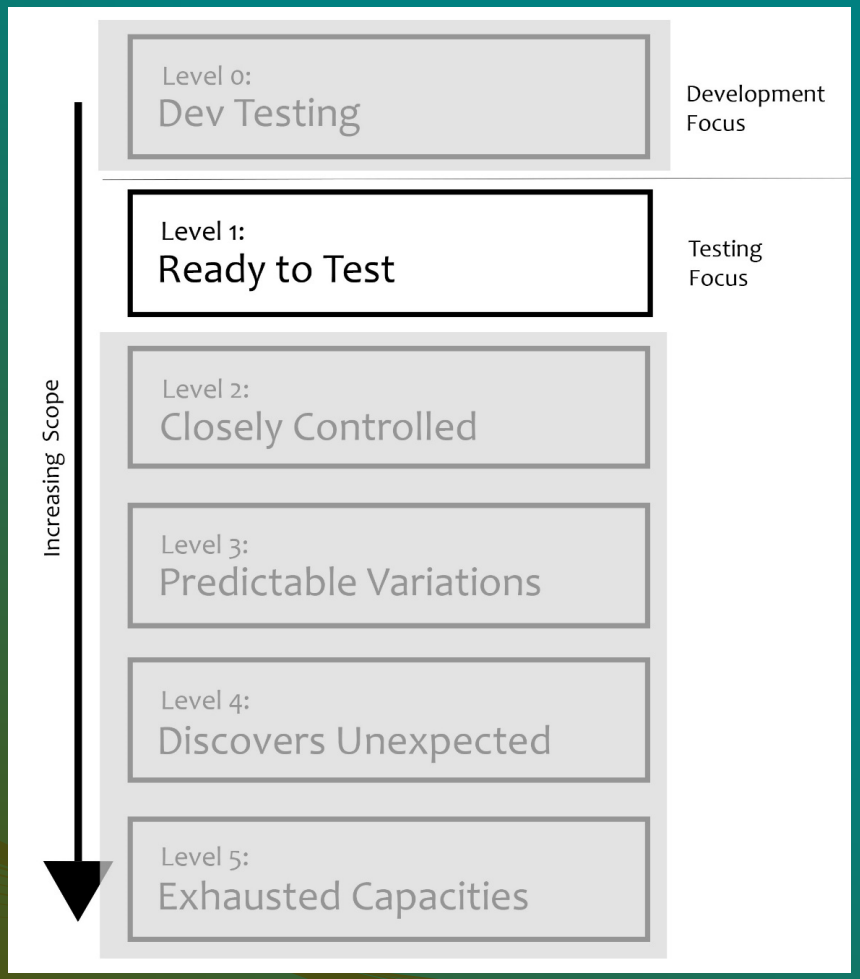
- Goal: Has intended functionality been implemented?
- Typical Tests: Very small aspects of behavior analyzed separately from all other behaviors of the program.
- Examples:
  - Unit Tests
  - Style Checkers
  - Other Static Code Analyzers

Level 0: Dev Testing (2 of 2)



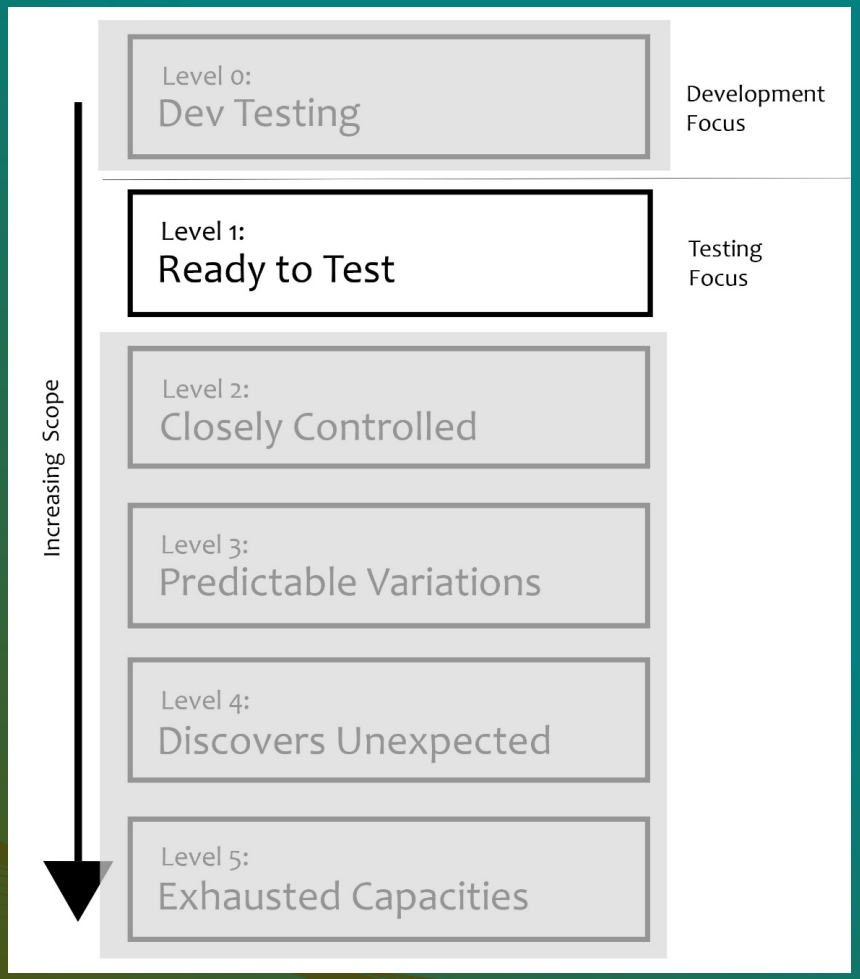
- Software Released at Level 0:
  - May contain missing, partly-functional, and incomplete features
  - May not install cleanly into any environment other than Development Environment
  - May not work as a cohesive whole even if it installs cleanly
- Professional software development shops usually test further than this before product release.

Level 1: Ready to Test (1 of 2)



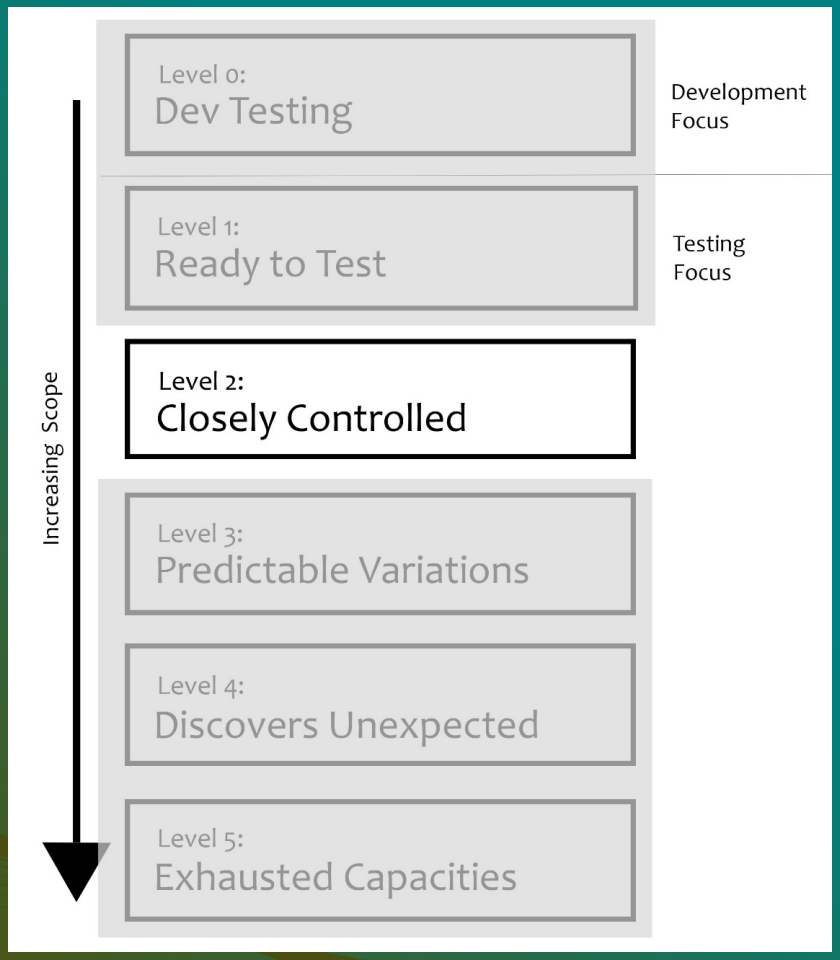
- Goal: Is software ready to be challenged by Testing Focus work?
- Typical Tests: Surficial functionality testing, just enough to show whether program crashes with trivial ease or permits access to all features of current interest.
- Examples:
  - Smoke tests
  - Simple happy-path scenarios
  - Most common / predictable error scenarios

Level 1: Ready to Test (2 of 2)



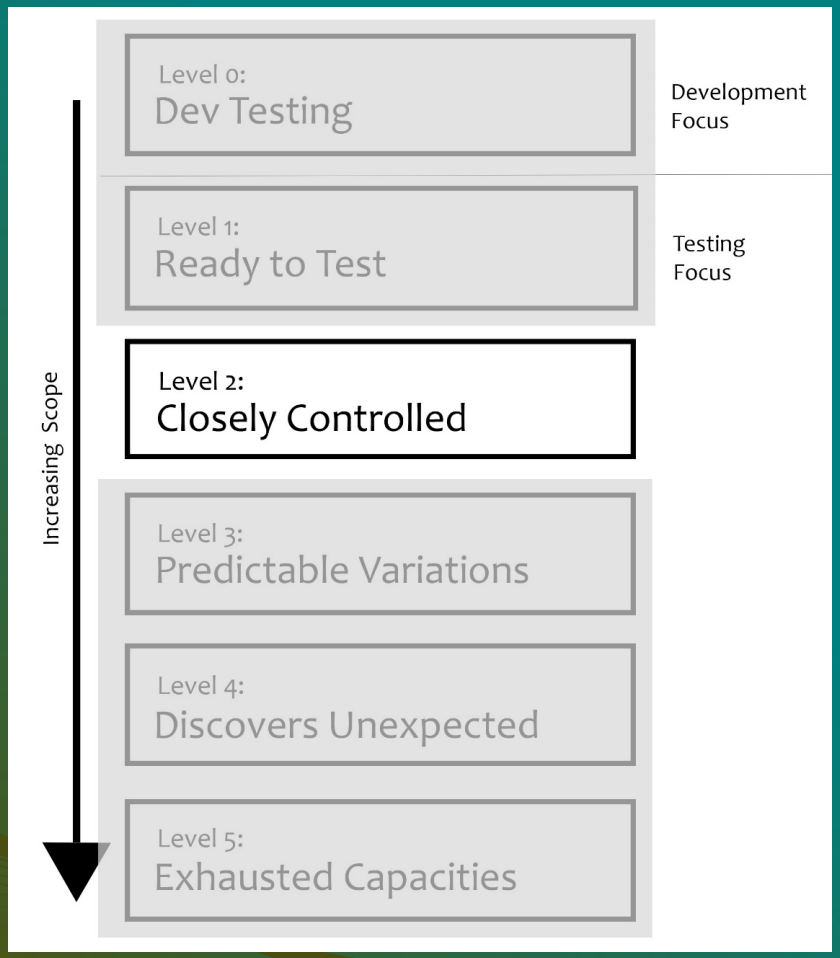
- Software Released at Level 1:
  - Should install properly into expected operating environments
  - May break if used even slightly differently than anticipated
    - Via user action
    - Via data state
    - Via non-tested operating environment

Level 2: Closely Controlled (1 of 2)



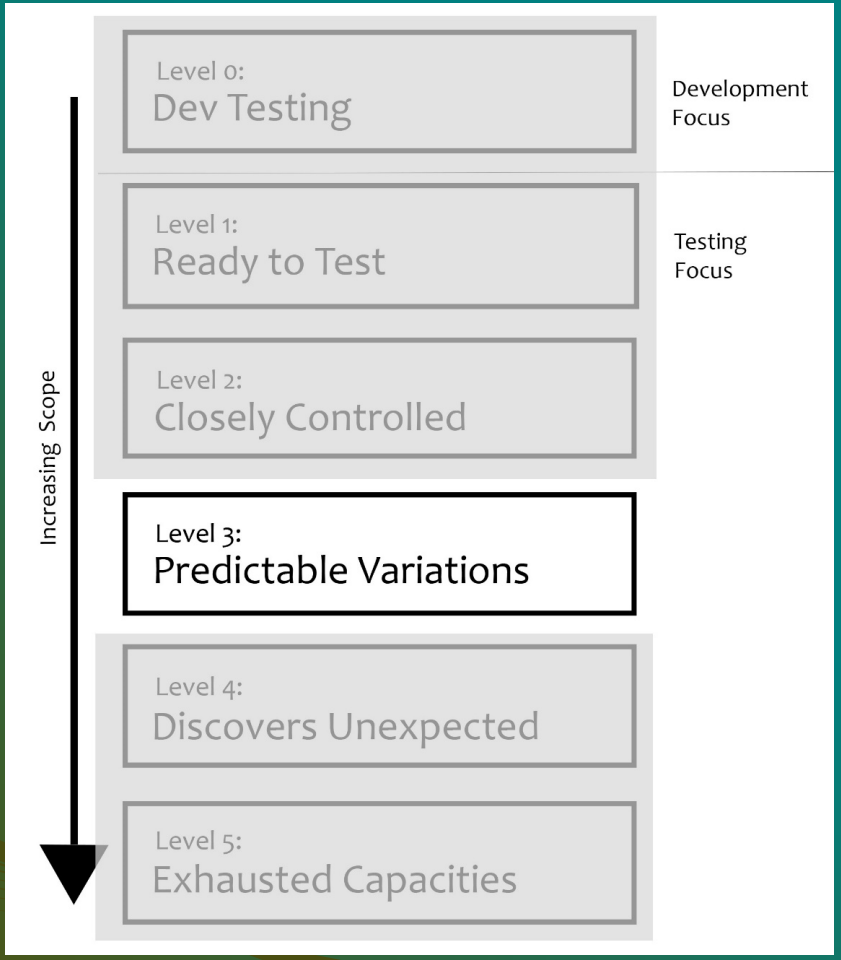
- Goal: Do features work when challenged by tests with carefully controlled parameters?
- Typical Tests: Very brief runs of the app to test a specific behavior, followed by resetting app to a clean state before running next test.
- Examples:
  - Tests with hardcoded test data
  - Tests using data drawn from small lists for different runs

Level 2: Closely Controlled (2 of 2)



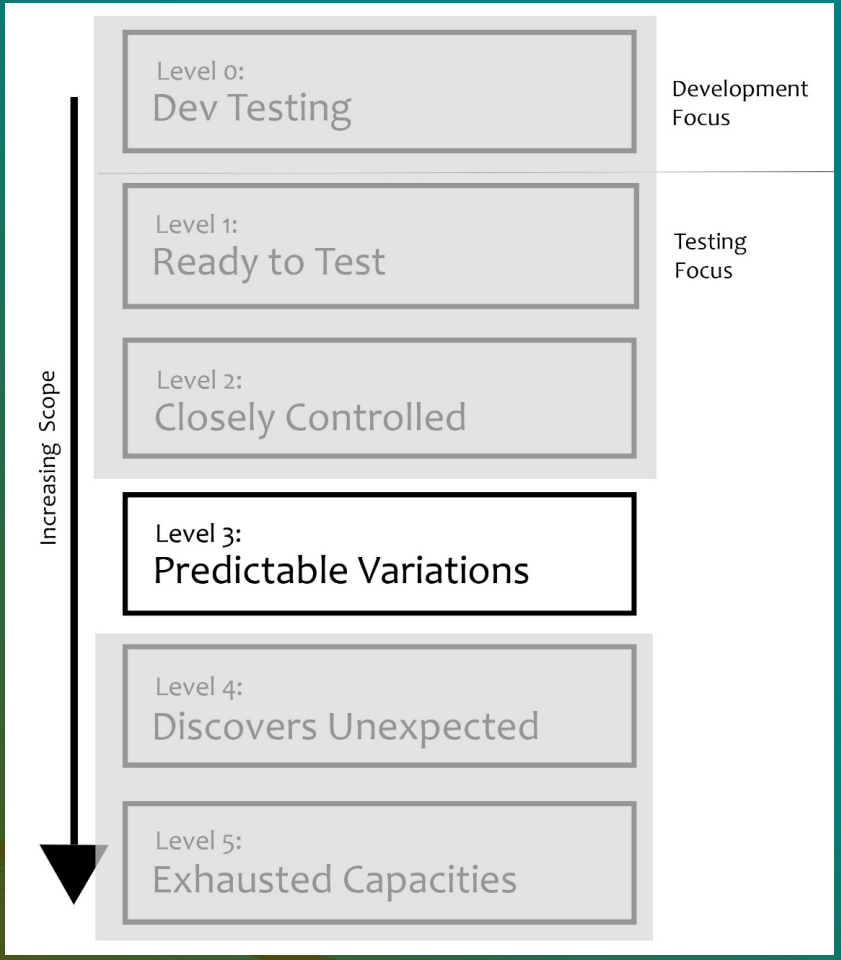
- Software Released at Level 2:
  - Likely to suffer many field failures on systems that differ from Development and Testing Environments
  - May break if used in unanticipated ways
  - May break if used on a continuous basis for some block of time

Level 3: Predictable Variations (1 of 2)



- Goal: Do features work when test conditions are significantly loosened?
- Typical Tests: Intensely vary one or a few variables while holding others constant, to assess impact of those specific variations.
- Examples:
  - Very large sets of very similar hardcoded tests
  - Tests that iterate through large, hardcoded lists of possible values
  - Short runs of tests that generate data at runtime based on a seeded random number generator

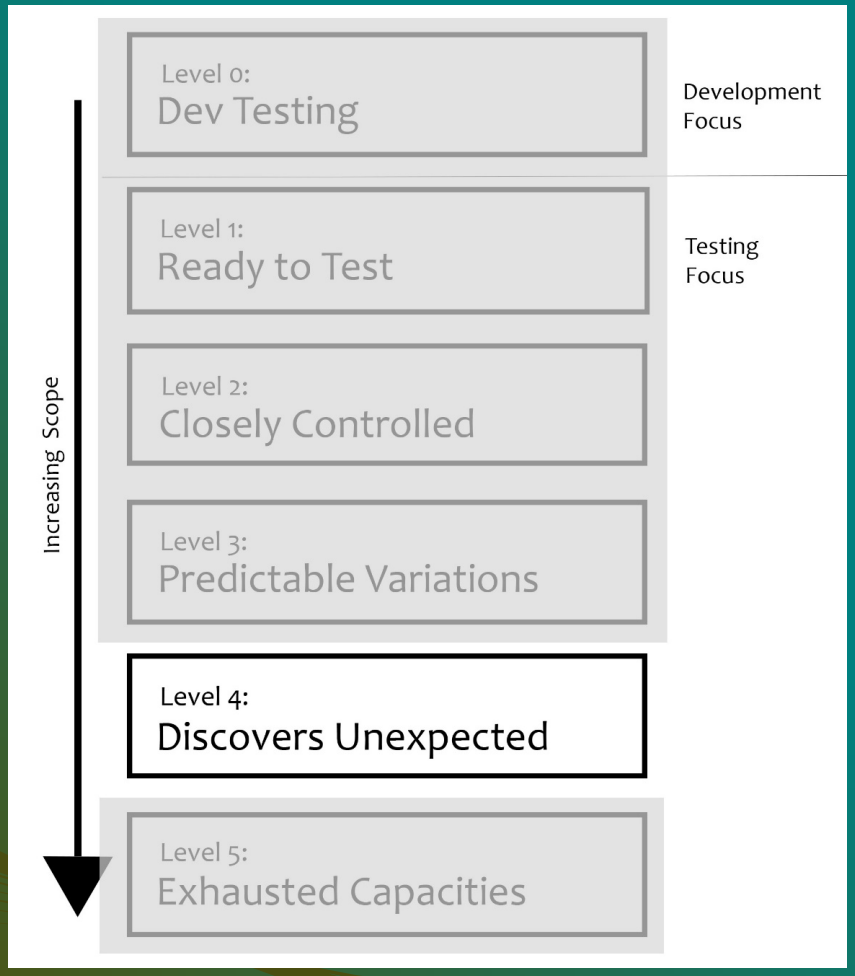
Level 3: Predictable Variations (2 of 2)



- Software Released at Level 3:
  - Markedly more stable in a diverse variety of conditions than software released at Level 2 (Closely Controlled)
  - May still experience a moderate number of field failures
    - Unexpected patterns of use
    - Untested environmental conditions
    - Untested data conditions

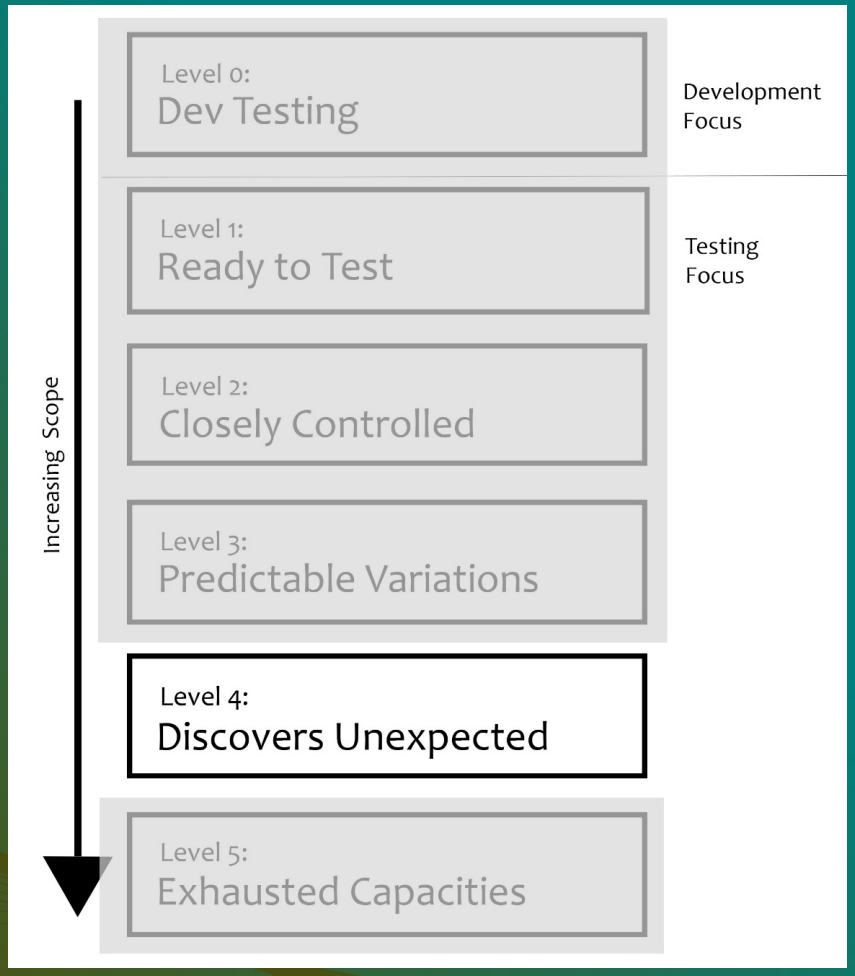
The more aspects of environmental unpredictability not tested, the higher the likelihood of finding many problems after release.

Level 4: Discovers Unexpected (1 of 2)



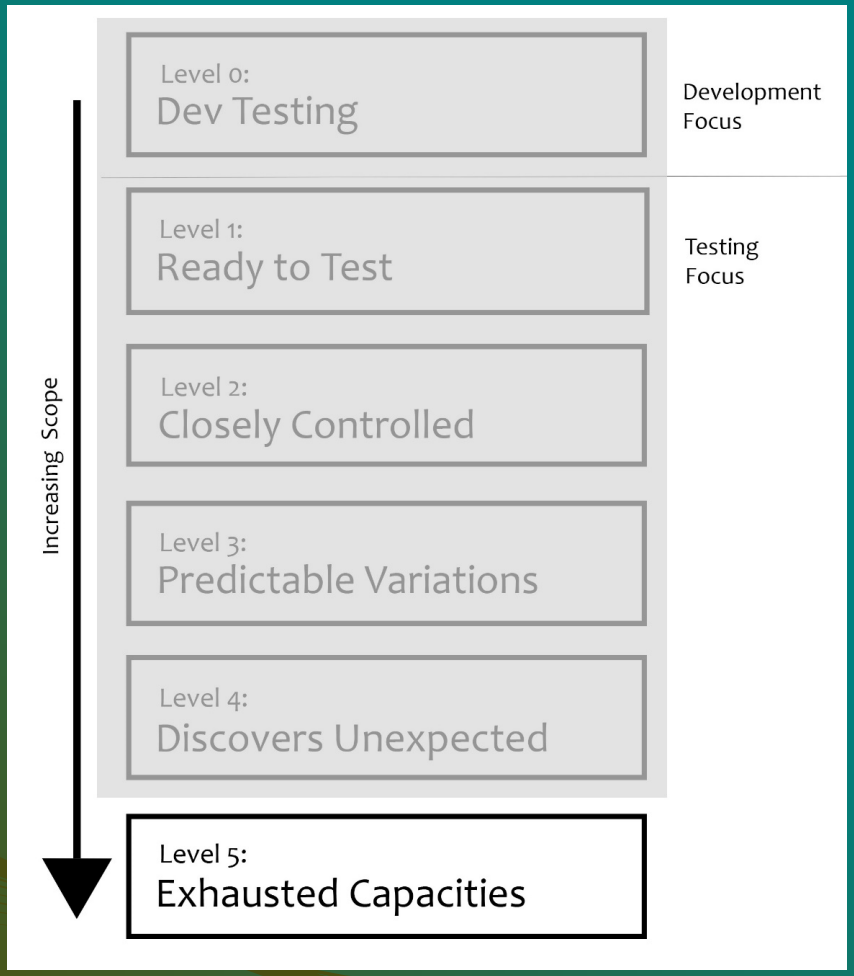
- Goal: Uncover problems that no one on the project could predict, sometimes that no one on the project could imagine.
  - Modern programs are complex and contain many tacit dependencies with themselves, their operating environment, and their data.
- Typical Tests: Stress the program and its operating environment at great scale.
  - Many use HiVAT (High Volume Automated Testing) techniques to efficiently run hundreds of thousands, millions, or scrillions of specific tests

Level 4: Discovers Unexpected (2 of 2)



- Examples:
  - Long-duration runs of tests that do not tidy up between tests
  - Random variations of structured input (fuzzing)
  - Long-sequence tests interleaving a few actions that should not have side effects, repeated many, many times
  - etc.
- Software Released at Level 4:
  - Experiences some field failures, when combinations not reached by intensive testing discovered in the field
  - Failures are rarer than at earlier levels

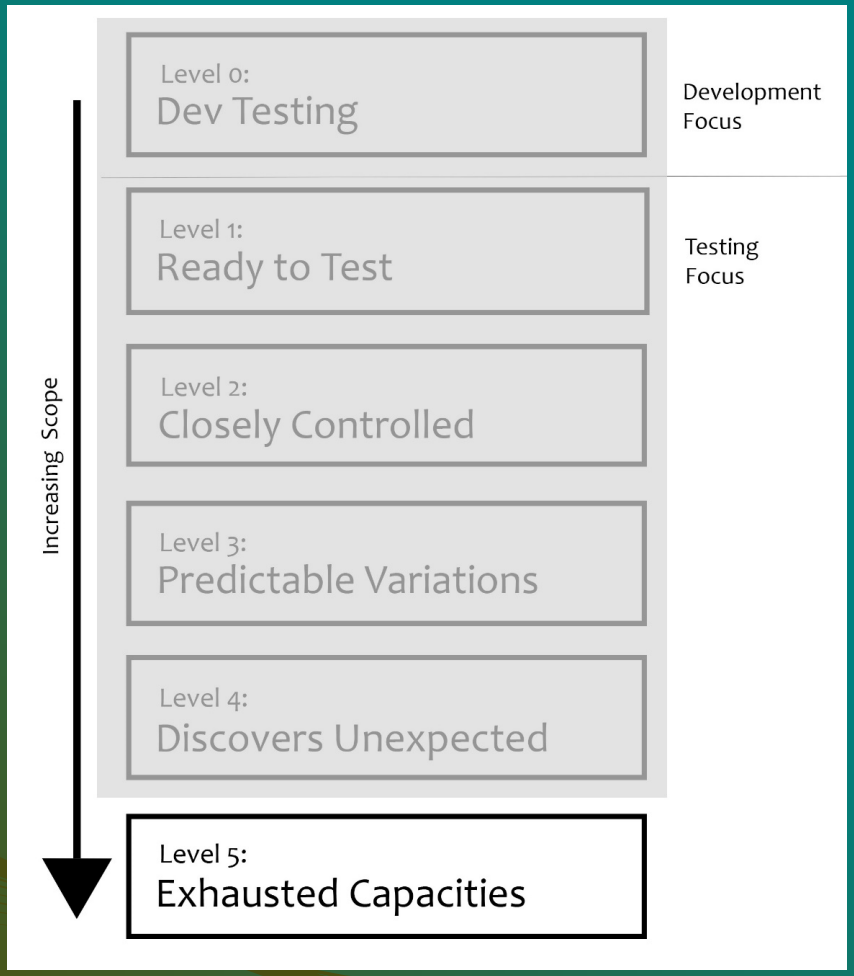
Level 5: Exhausted Capacities (1 of 2)



- Goal: For testing efforts to exhaust their capacities, given the capabilities of available tools and techniques and the time available.

- Typical Tests:
  - Dramatically increase stresses upon the software and its operating environment
  - Sampling scope approaches closer to exhaustive
  - System testing increases the variations in exercising interleaved, cooperating, and coexisting features

Level 5: Exhausted Capacities (2 of 2)



- This level of testing intensity typically done when the risk of failure is extremely high
  - E.g. Data-processing logic that could corrupt the database if done wrong
  - E.g. Life-support systems
  - E.g. High-impact infrastructure systems, etc.
- Historical Example: Y2K Testing
- Software Released at Level 5:
  - Experiences the fewest possible field failures
  - But some failures remain possible

Levels vs. Environmental Diversity

Release-Readiness Level	Environmental Diversity
0 (Dev Testing)	None
1 (Ready to Test)	Barely Any
2 (Closely Controlled)	Very Little
3 (Predictable Variations)	**Little to Moderate**
4 (Discovers Unexpected)	**Vast Capacity**
5 (Exhausted Capacities)	**Vast Capacity**

- Handling Environmental Unpredictability for Mobile and IoT Software:
  - Requires Level 3 (Predictable Variations) tests
  - Requires Level 4 (Discovers Unexpected) tests
- Some critical behaviors within many apps deserve Level 5 (Exhausted Capacities) tests
- Some whole systems may merit Level 5 (Exhausted Capacities) tests
  - Example: Autonomous Vehicles

PACIFIC NW SOFTWARE QUALITY CONFERENCE

Which Testing Levels are Enabled by Current Technologies and Tools?

CORE SOFTWARE TESTING TECHNOLOGIES

Core Software Testing Technologies

Present in today's readily available tools and the academic research literature:

1. Physical Devices
2. Virtual Devices (Emulators and Simulators)
3. Simulation Environments
4. Mechanisms for interacting with the Visual GUI Tree
5. Image-comparison
6. Code Manipulation (e.g. xUnit, code instrumentation, etc.)
7. System Monitoring

Each technology:

- Enables certain kinds of tests
- Is better-suited to some types of investigations than others

Existing tools:

- Fulfill a technology's potential to differing degrees

1. Physical Devices (1 of 3)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

What:

- Actual physical devices, local or remote

Technology Strength:

- Trustworthy realism about how apps will behave on that device.

Greatest Weakness:

- Difficulty scaling to many thousands of platforms.

1. Physical Devices (2 of 3)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

Current Best Solution: Cloud-based access to Physical Devices

- Hundreds to a Few Thousands of device/OS version combinations
- Devices located in server rooms
  - Little variety in physical or network environments
  - Little scope for exercising embedded sensors and other equipment on devices
- Testing limited to features of interfaces (frameworks) provided by the cloud service

1. Physical Devices (3 of 3)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

Release-Readiness Levels Assessment (at scale):

- Level 2 (Closely Controlled)
  - Very little can be inferred about stability of software on other devices based upon its behavior on one device.
  - Testing tens or even a few hundreds of devices is still a small fraction of the possible many thousands of devices.
- At scale, only robustly testing combinations of basic hardware and OS version.
  - No varied environmental conditions such as seen by mobile devices out and about in the real world.

2. Virtual Devices (1 of 3)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

What:

- Software illusions of real devices using different computing hardware and resources. (Simulators and “Emulators”)

Technology Strength:

- Easy, cheap access to a variety of device variations.

Greatest Weakness:

- Trustworthiness.

2. Virtual Devices (2 of 3)

Virtual Mobile Devices model:

- CPU, OS Version, RAM
- Screen size, resolution, pixel density (if device has a screen)

Virtual Mobile Devices commonly do NOT model:

- Camera, GPS, accelerometer, microphone, speakers, gyroscope, compass
- Sensors for ambient light, pressure, temperature, humidity, proximity
- Network connections like Bluetooth, Wi-Fi, NFC (Near Field Communication)
- Cellular connections like 3G, 4G, etc.

2. Virtual Devices (3 of 3)

Android Virtual Devices:

- Apply a stock version of Android OS
- Do NOT include customizations done by the hardware providers (usually proprietary)
- Do NOT include customizations done by the cell service providers (usually proprietary)

Release-Readiness Levels Assessment:

- Level 1 (Ready to Test)
  - Information obtained is only a ghost of reality

3. Simulation Environments (1 of 2)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

What:

- Virtual representations of various aspects of the world surrounding a mobile or IoT device.
- Theoretically allow an in-house lab to generate many thousands of realistic deployment scenarios, as well as scenarios that are questionable in terms of reality but powerfully informative for testing purposes.

Technology Strength:

- Rich exercise of widely varied environmental conditions.

Greatest Weakness:

- Accessibility.

3. Simulation Environments (2 of 2)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

Release-Readiness Levels Assessment:

- Level 3 (Predictable Variations) potential
- Level 4 (Discovers Unexpected) potential
- Level 5 (Exhausted Capacities) potential

- Used to a limited extent in academic research
- No apparent access for practitioners (at this time)

4. Visual GUI Tree (1 of 3)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

What:

- Interact with the display-rendering hierarchy of objects drawn for the GUI.
- Contain visible elements like buttons but also invisible elements like grids that organize the visual layout.

Technology Strength:

- Excellent at replicating how humans interact with a GUI.

Greatest Weakness:

- Depends on uniquely-identifiable objects.
- GUI Programming is changing, so that objects with unique identifiers are less likely to exist.

4. Visual GUI Tree (2 of 3)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

Visual GUI Tree Tools are the dominant form of test tools today.

- *Mobile Native Apps: Appium, Calabash, Espresso, Robotium; Android UI Automator, XCTest, XCUITest*
- *Cloud Services Examples: AWS Device Farm, App Center Test, Bitbar, Experitest, Kobiton, Mobile Labs, pCloudy, Perfecto Mobile, Sauce Labs, Xamarin Test Cloud*
- *Web applications: Selenium*

4. Visual GUI Tree (3 of 3)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

Release-Readiness Levels Assessment:

- Level 2 (Closely Controlled)
  - Tests tend to hardcode data which are repeated verbatim each run
  - Tools rarely offer features to parameterize the data
  - Tools even more rarely offer features to determine test data programmatically at run time

Cannot trigger situations initiated by sensors, only those initiated by user actions.

- Does not touch most of environmental complexity problem.

5. Image-comparison (1 of 2)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

What:

- Full-Image Comparison: Every pixel must match.
- Part-Image Comparison: Finds a specific small image within a full-screen picture.
  - Examples: Eggplant, Sikuli

Technology Strength:

- Excellent for handling Visual Meaning scenarios.
  - Visual Meaning scenarios = Those in which a picture most accurately conveys the meaning and all else merely points at the meaning.

Greatest Weakness:

- Only handles fully-visible elements of GUI.

5. Image-comparison (2 of 2)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

Release-Readiness Levels Assessment:

- Level 2 (Closely Controlled)
  - Tests look for specific, hardcoded reference image data
  - Existing tools poorly handle variations in size, style
  - Cannot address functionality not visible onscreen

Cannot trigger situations initiated by sensors, only those initiated by user actions.

- Does not touch most of environmental complexity problem.

(Excellent complement to Visual GUI Tree tools but NOT a complete replacement of them.)

6. Code Manipulation (1 of 2)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

What:

- Many types of code analysis and modification.
  - All utilize insight into the code's details.
  - All techniques require programming.

Technology Strength:

- Immense potential, limited only by human capacity to address the problem.

Greatest Weakness:

- Rapid Complexity
  - Some of the required knowledge is very deeply technical.
    - People with that knowledge are rarely in testing positions.

6. Code Manipulation (2 of 2)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

Release-Readiness Levels Assessment:

- Level 3 (Predictable Variations) potential
- Level 4 (Discovers Unexpected) potential
- Level 5 (Exhausted Capacities) potential

Tools that propagate to industry typically encapsulate a small set of behaviors.

- xUnit frameworks
- Code coverage tools
- Style checkers
- etc.

7. System Monitoring (1 of 2)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

What:

- All mechanisms for observing and noting system behavior while an app is running.
- Including built-in features to a program providing extensive diagnostic interface.

Technology Strength:

- Monitors are robust for core computing elements and network communications.

Greatest Weakness:

- Little available to monitor sensors and other embedded equipment.

7. System Monitoring (2 of 2)

1. Physical Devices
2. Virtual Devices
3. Simulation Environments
4. Visual GUI Tree
5. Image-comparison
6. Code Manipulation
7. System Monitoring

Release-Readiness Levels Assessment:

- Most commonly: Level 2 (Closely Controlled)
- Level 4 (Discovers Unexpected) potential
  - Requires extensive built-in diagnostics features

Technology vs. Testing Levels

Testing Technology	Testing Levels Enabled by the Technology – At Scale	Testing Levels Enabled by Accessible Tools – At Scale
1. Physical Devices	1 (Ready to Test) 2 (Closely Controlled)	1 (Ready to Test) 2 (Closely Controlled)
2. Virtual Devices (Emulators and Simulators)	0 (Dev Testing) 1 (Ready to Test)	0 (Dev Testing) 1 (Ready to Test)
3. Simulation Environments	**3 (Predictable Variations)** **4 (Discovers Unexpected)** **5 (Exhausted Capacities)**	**Limited availability in academia** **No apparent practitioner access**
4. Visual GUI Tree Interactions	2 (Closely Controlled)	2 (Closely Controlled)
5. Image-Comparison	2 (Closely Controlled)	2 (Closely Controlled)
6. Code Manipulation	**3 (Predictable Variations)** **4 (Discovers Unexpected)** **5 (Exhausted Capacities)**	**Tools typically custom one-offs** **Programming expertise required**
7. System Monitoring	2 (Closely Controlled) **4 (Discovers Unexpected)**	2 (Closely Controlled)

Therefore…

- Vaster scale and scope of environmental complexity in Mobile and IoT era
- Requires testing that efficiently handles vast environmental complexity
- Existing tools leverage technology ill-suited to this need
- Therefore we need new tools leveraging suitable technology

PACIFIC NW SOFTWARE QUALITY CONFERENCE

What Kinds of New Testing Tools?

VISION OF A NEW GENERATION OF SOFTWARE TESTING TOOLS

Requirements for Next Generation Testing Tools

1. Directly target functionality dealing with embedded sensors and equipment

2. Scale easily to vast variations in data readings, data fidelity, data delivery methods, etc.

3. Constrain technological complexity to be within reach of non-specialists

By "non-specialists" I mean:

- Experienced software testers
- With competent, generalist programming skills
- Without special technical expertise in any subsystems comprising mobile/IoT computing environments
- Without special technical expertise in mathematical modelling

Further down the road, it may be possible to create more generally-accessible tools, as has already happened with code coverage tools and style checkers.

A Tool I've Imagined

- Combine Integration Testing and Compiler Knowledge
  - Tester defines both a Beginning Point and 1+ Ending Points for data flow of interest
    - May also define specific Intermediate Monitoring Points to snapshot data state
  - Tester defines static starting-state date
  - Tester defines pattern for operational data submitted to program's environments via sensors
  - Tool executes that path within the program for the specified duration
  - Tester and Tool interpret the data results harvested from the Ending Points

- Enables rich Simulation Environment testing on custom slices of functionality
  - Without requiring fully-featured reality-based environments
  - Without requiring testers to understand all the underlying code
  - Just the intent of the feature, how to manipulate data at the endpoints, and how to monitor intermediate data states

Example: Bicycle Ride Tracking App

- Scenario: A long, mountainous training ride
- Among its features, one tracks Total Height Climbed during the ride
  - Likely involves Accelerometer, Gyroscope, GPS
  - If available, also Magnetometer and Barometer
- Happy-Paths: Perfectly functioning sensors with instantaneous responses

- Realistic Environment Challenges:
  - GPS outages, low signal strength, slow responses, erroneous data
  - Sensor outages/erroneous data when very hot (valleys) or very cold (mountain tops) or quickly transitioned between extremes
    - A mountain descent can easily change 50F in 20 minutes
  - Other sources of interference, affecting sensor operations: Humidity, metal bridges, high power lines, etc.

Example: Things to Vary

- Starting position (including altitude)
- Ending position (including altitude)
- Length of training ride

- Pattern of noise in each sensor's data during the ride
  - % Good data
  - % Late responses
  - % Error codes
  - % Missing responses
  - % Nonsensical data values

Recap

- Mobile and IoT computing operate within a fundamentally different scale and scope of environmental complexity than in prior computing eras.

- Accurately assessing field performance of software for such devices requires testing at
  - Level 3 (Predictable Variations)
  - Level 4 (Discovers Unexpected)
  - Level 5 (Exhausted Capacities)
- Existing testing tools enable testing at
  - Level 1 (Ready to Test) – common
  - Level 2 (Closely Controlled) – very common
  - Testers making tools reach partially into Level 3 (Predictable Variations)

- Mobile and IoT need testing tools that
  - Directly handle vast environmental unpredictability
  - Operate efficiently at great scale

In Closing

Large and Complex Problem

What types of tools can you imagine that would be useful to the field?

carol@carolcodes.com