Shift Left Testing with DevSecOps

Hasan Yasar

Technical Director, Adjunct Faculty Member
Continuous Deployment of Capability Directorate

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM22-0618

Testing my knowledge!

- 25+ years of software development experiences
- Certified Scrum Practitioner
- Certified Ethical Hacker
- Various roles throughout SDLC ; Manager, Architect, Tester, Developer, QA, IT Manager, Project Manager, VP…
- Started with waterfall in 1990
- Started with agile in 2003
- Started with DevOps in 2010
- Instructor on delivering DevOps course at CMU, SEI since 2015
- DevOps, DevSecOps community organizer, frequent Speaker
- PC members in various research conferences,
- Editorial board member, IJSS, AJSE
- Co-Author IEEE/ISO 2675 DevOps Standard

Agenda

- **Testing Fundamentals**
  - Incremental Approach
  - Testing types
  - HWIL testing
  - Testing with Agile & DevSecOps
- **A Strategy for shift left Testing**
  - Dev/Test Candance
  - Tools for Test Automation
  - End2end Testing Reference Architecture
- **Takeaway**

Shift-Left testing

Shift left testing requires:

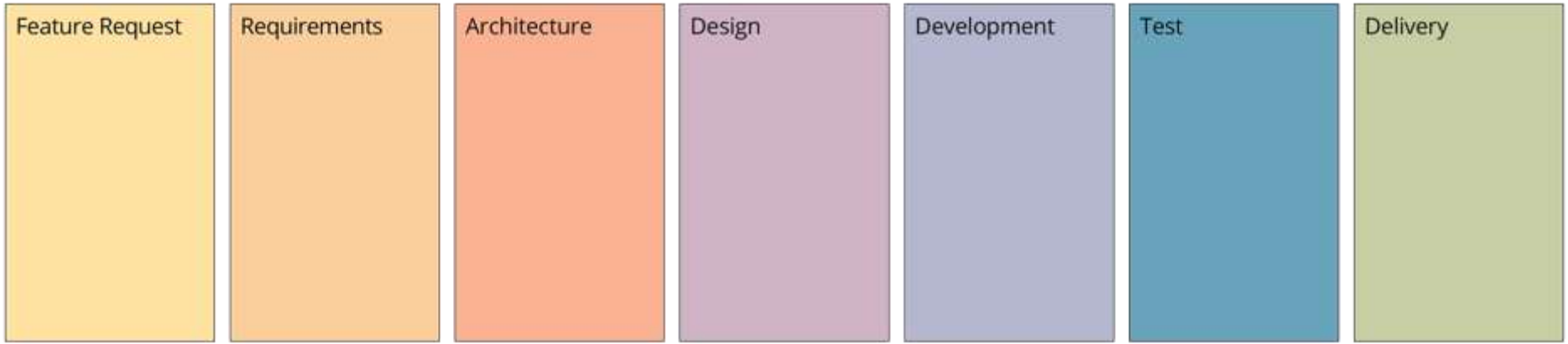
- Practices and test tools
- Timing of tests
- Time to develop automate testing
- Planning for testing(what & wow)
- Level of testing
- **Culture**

None of these changes will be easy;

they require *energy, commitment, resources and mindset*

Testing Fundamentals

Reminder: SW Development Phases



“Responding to change *over following a plan*”

Focus on predictability ⟷ Focus on responsiveness

	Waterfall	Spiral	Iterative	Scrum
Defined processes	Required	Required	Required	Planning & Closure only
Final product	Determined during planning	Determined during planning	Set during project	Set during project
Project cost	Determined during planning	Partially variable	Set during project	Set during project
Completion date	Determined during planning	Partially variable	Set during project	Set during project

Completion contracts, the supplier commits to a finished product; typically the price is fixed

Best effort contracts, the supplier commits to making a good faith attempt to fulfill all the requirements with quality work, but makes no commitment to a finished product; work continues as long as the contractor is paid for progress

Adapted from:

- Jeff Sutherland, Ph.D. and Ken Schwaber (Co-Creators of Scrum), The Scrum Papers: Nuts, Bolts, and Origins of an Agile

Different Incremental Approaches

Incremental development:

- Single increment of work, delivered once in a single package

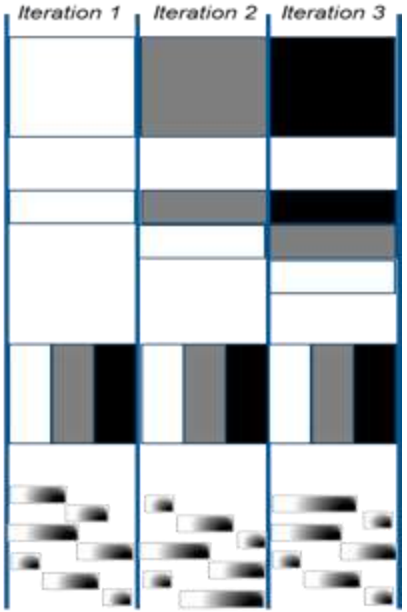
Incremental Development, Single Delivery

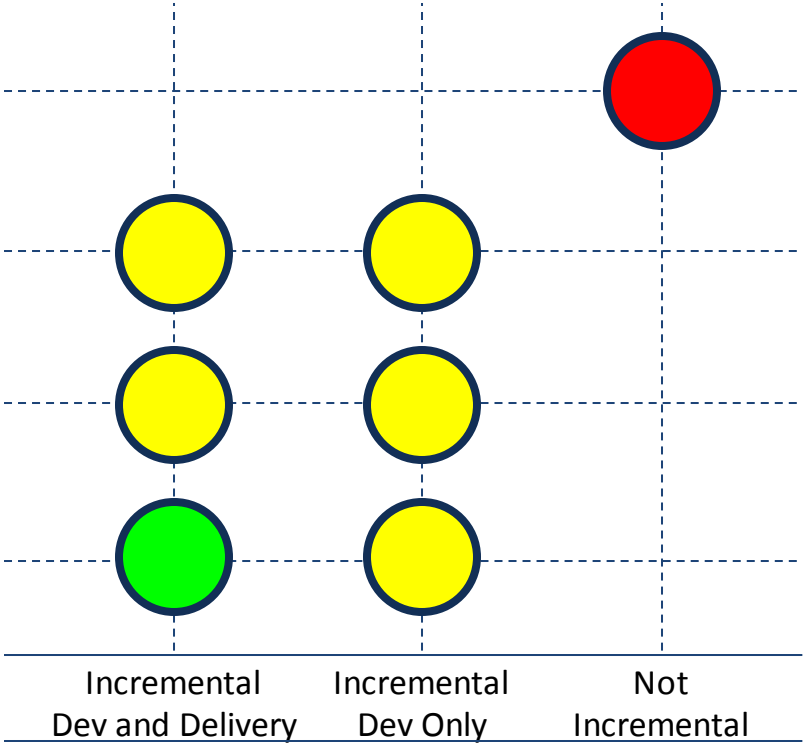
- Work divided into logical subsets for development in pieces, delivered once in a single package

Incremental Development & Delivery

- Work divided into meaningful slices of the total end result, delivered in gradually more complete versions
- Alternatively, delivering new pieces rather than total new versions

Incremental and Iterative Combinations





Start from Planning (capturing the right requirements)

Traditional

- The requirements form a mutually exclusive and collectively exhaustive expression of the user needs and wants
- Complete. Each requirement must fully describe the capability to be delivered
- Unambiguous. All readers of a requirement should arrive at a single, consistent interpretation of it
- Verifiable. It should be possible to objectively determine whether the system properly implements each requirement
- Consistent. A requirement must not conflict with other requirement

Good User stories (INVEST)

Independent. The requirement can be developed and tested on its own

Negotiable (Definable). The requirement is a promise to have a conversation in due time to define the details of whatever is being built. Is more about learning than negotiation

Valuable. The requirement must provide a benefit the customer could appreciate

Estimable. It should be possible for the team to forecast the effort it will require to implement it

Small. The requirement should be small enough to be able to be completed in an iteration

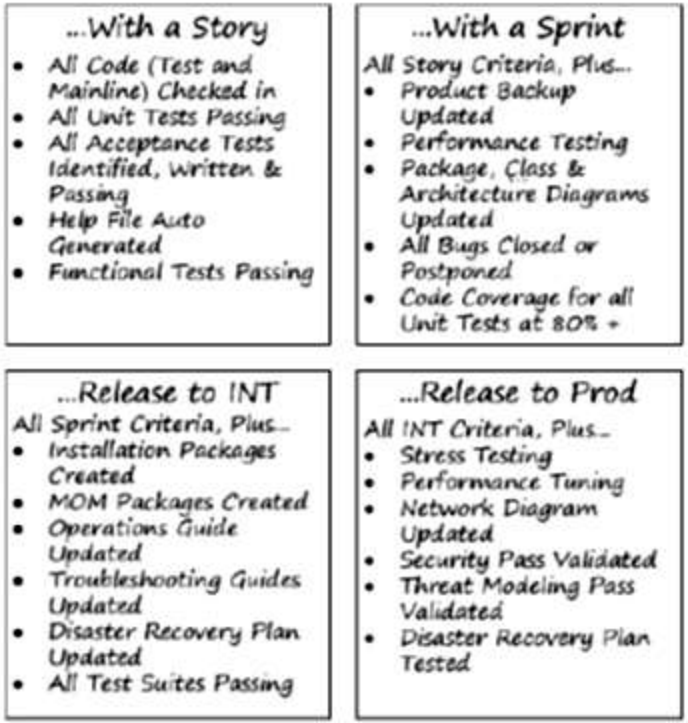
Testable. The requirement must provide enough information to make it clear how it will be verified

Definitions of Ready & Definitions of Done

DoR

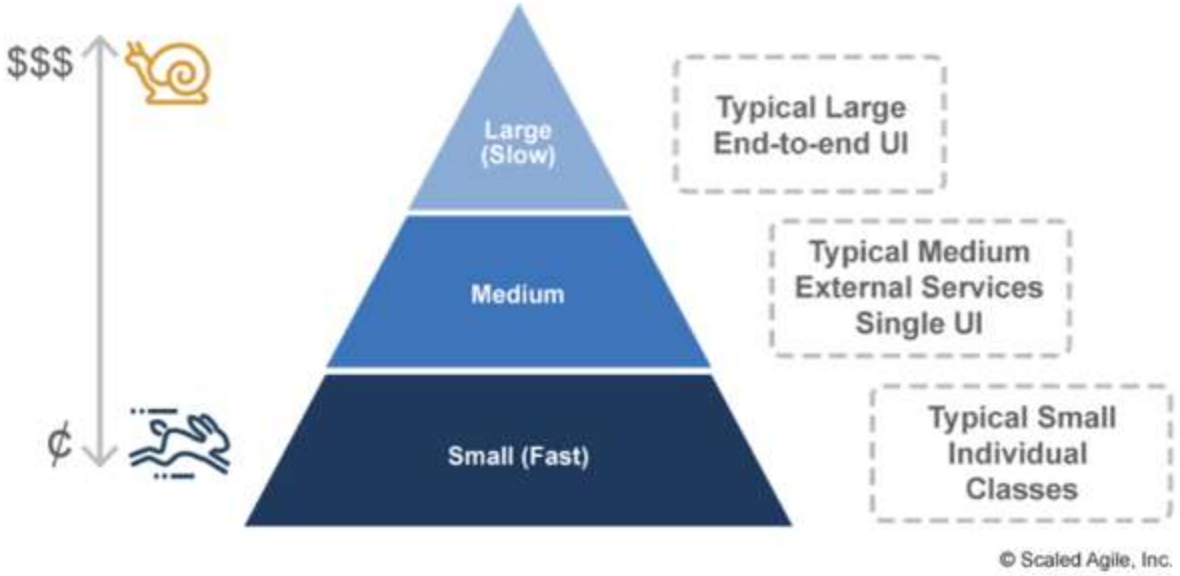
- Business value is clearly articulated
- PBI details are understood by the development team
- Dependencies are identified and resolved (e.g., not known external dependency should block the work once started)
- The PBI is estimated and small enough to comfortably fit in one sprint
- Acceptance criteria are clear and testable.
- Performance criteria, if any, are defined and testable

DoD



M. Lacey, How Do We Know When We Are Done?, 2008

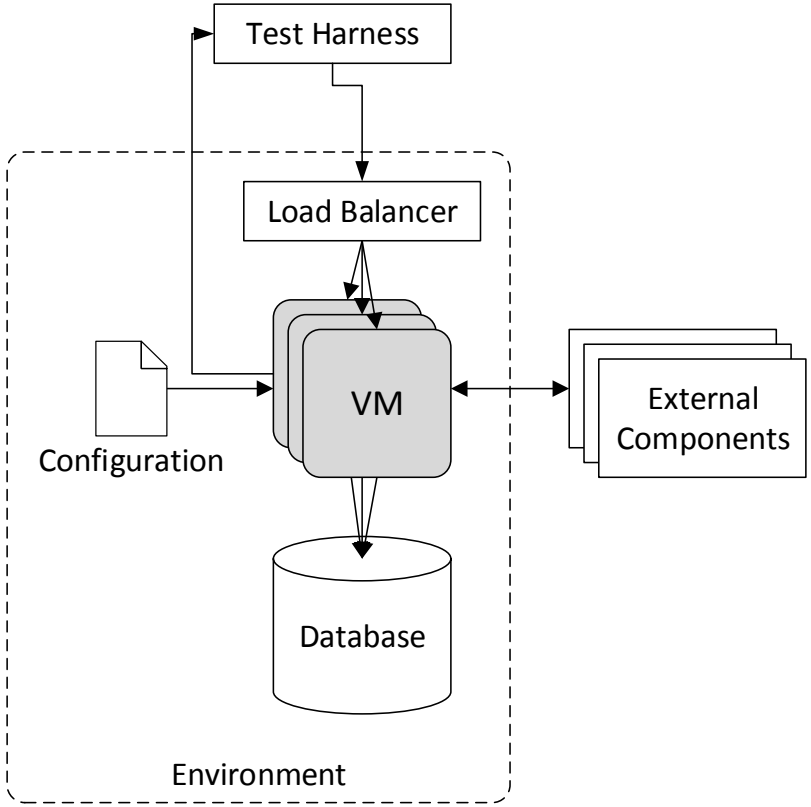
Testing in General



Have as many cheap, fast running tests as possible and minimize the number of expensive and slow tests

Test Harness

A test harness generates inputs and compares outputs to verify the correctness of the code.



Test data

Test data is real (ish) but limited.

- Need to worry about exposing private data to developers/Contractors
- Limited so that tests will run fast and keep build queue from growing
- If tests change database, it must be refreshed before next set of tests.
- No results should be sent to any production service – results in corrupting production version.

Testing in Development Environment

- Unit Testing/component testing
  - Verify the functionally of specific section of code
  - Includes static code analyzers, data flow, metrics analysis or peer code reviews
- Acceptance Testing
  - Done by developer prior to integration or regression
- Smoke/Sanity Testing
  - Reasonable to proceed with further testing or not
  - Minimal attempts to operate software

Functional testing

- Created image is subjected to a collection of automated tests
- Tests are performed using a test harness
  - Tests are the result of
  - User stories
- Regression tests
- Rainy day scenarios
- Trade off between comprehensiveness of tests and time it takes to run the tests
- CI server reports results through e-mail or a web page.

Testing in Continuous Integration Environment

- Much like testing in the development environment except with multiple modules rather than a single one
  - Unit Testing/component testing
- Built system testing
  - Detect defects in the interface and interaction between modules
  - Module/subsystem integration

Testing in Staging Environment

- Regression testing
- Smoke testing
- Compatibility testing
- Integration testing
- Functional Testing
- Usability Testing
- Install/uninstall testing
- Performance testing
- Security testing

User Tests

Real (or simulated) users exercise the system.

- Having real users is expensive and difficult to ensure coverage.
  - Record/playback works in some contexts
  - Teeing actual input works in some contexts
- Having real users allows for exception testing in a better fashion than automated tests

Performance tests

- Test for performance with generated loads
- Environment should be as real as possible
  - Load balanced
  - Auto scaled
  - Multiple instances

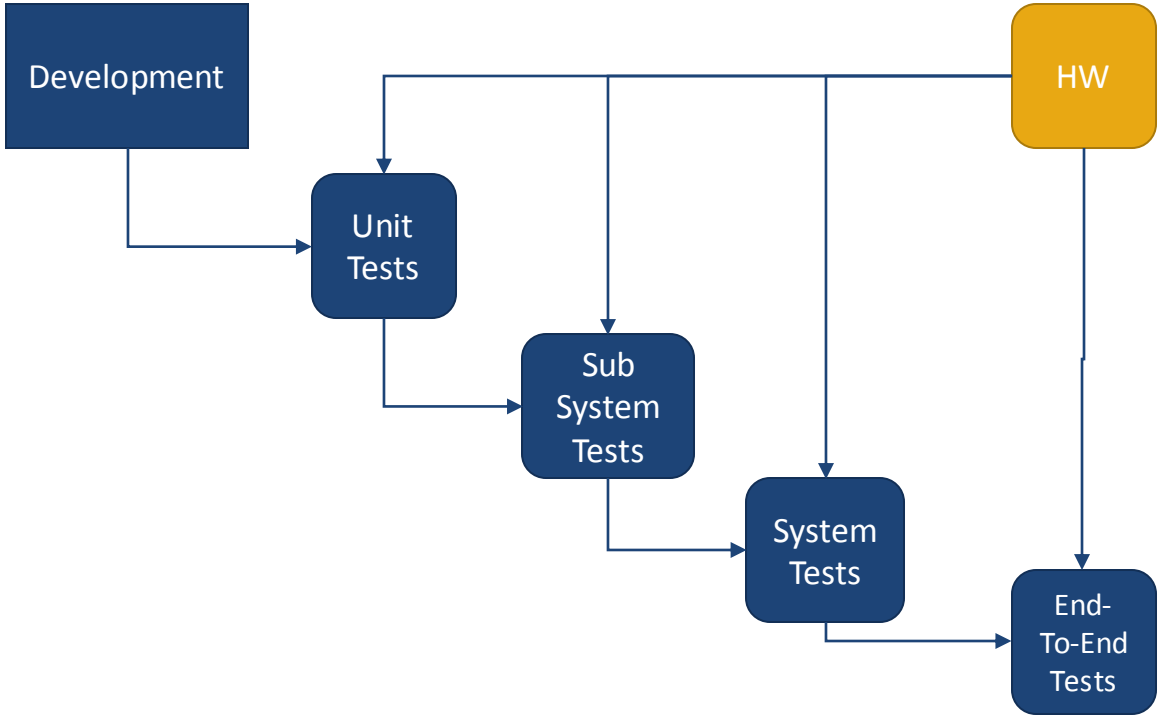
Test Database – It is a big challenge for segregated environments

- Use as close to real data as possible.
  - One organization uses database that is one hour old (created as a back up by cloud vendor).
- Constraints
  - Restore database after each test
  - Keep personal information private

Security testing

- Common Vulnerabilities and Exposures data base (CVE)
  - Buffer overflow
  - SQL injection
- Penetration testing tools
  - Static analyzers
  - Fuzz Testing
- Model checking

Testing for Embedded systems & HW



Development boards and prototypes

- Typically used during initial prototyping but can also extend into testing
- Convenient due to multitude of available IO options already built in
- Very limited as they usually represent the controller, no custom hardware, sensors, etc.
- Not the actual hardware, requires system level tests on actual hardware

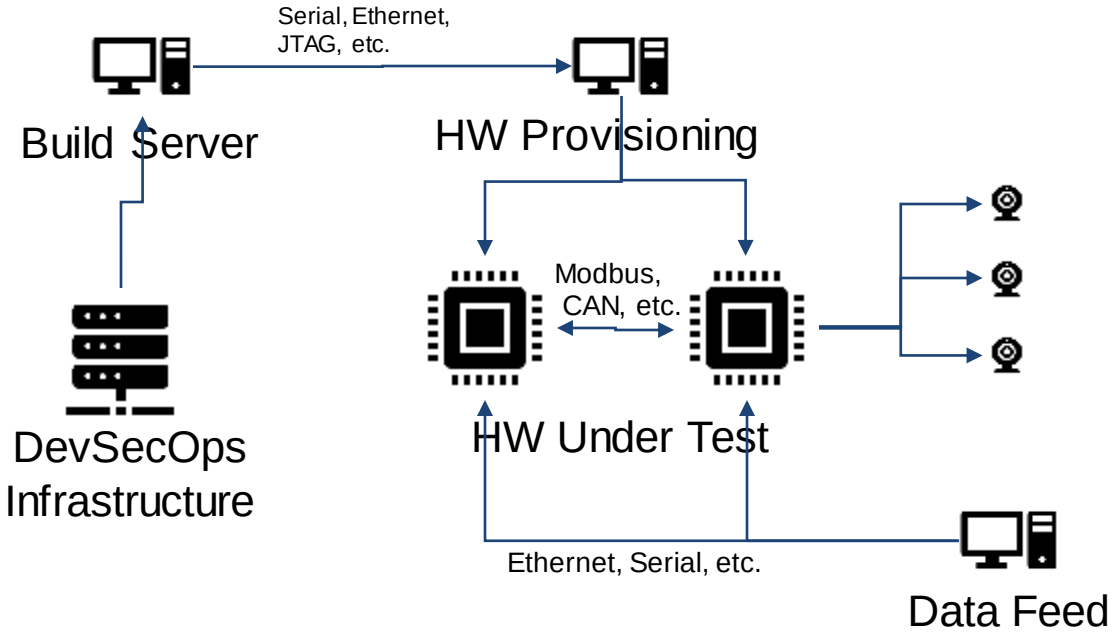
Hardware simulation

- Fully or partially implemented hardware functionality in software
- Emulation is often down to the instruction set
- Many forms exist, each has its limitations
- Emulation introduces latency
- Significant effort to create and maintain a functional simulator
  - Many companies have a dedicated simulator “SIM” team
- Efficient development of complex systems “requires” a simulator, often custom
- Off the shelf simulators exist, provide generic simulation and test integration capabilities
  - May not be sufficient for a complex system
  - Proprietary technology maybe difficult to extent

Hybrid Approach

- Start up the simulation team early in the dev process
- Perform early SW development with API-based substitution and HW dev boards
- Basic simulator ready in time for sub-system and unit tests in CI
- Hardware testbed and simulator management with configuration and memory snapshots to improve test stability
- Daily/Weekly “arming” tests with real HW
- End-to-End tests on real hardware once ready

Hardware-Based (HWIL) Testing with DevSecOps

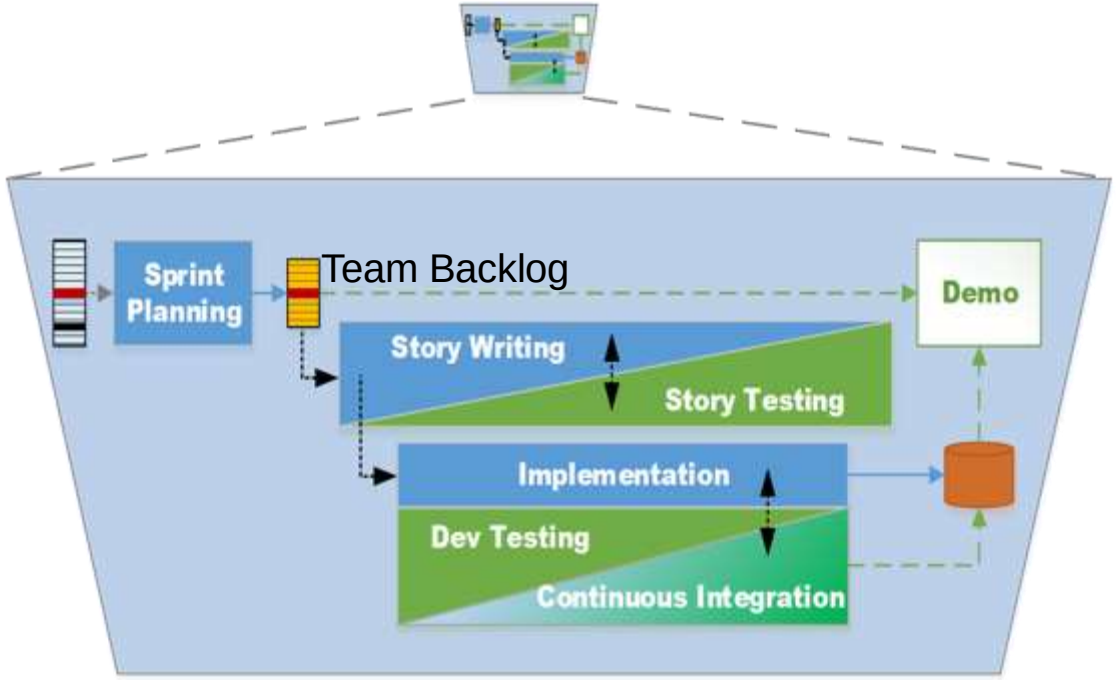


Overall, View of Testing

- Tests are the final requirements for the system!
  - We use tests to determine if the software is what we want and have become synonyms for the requirements
  - They're the only objective measure of the system
- Consequences:
  - Every requirement **must** have one or more tests associated with it
  - If we're practicing TDD, then code should only be written to make a failing test pass
  - Tests will have to be developed incrementally (or we're back to waterfall)

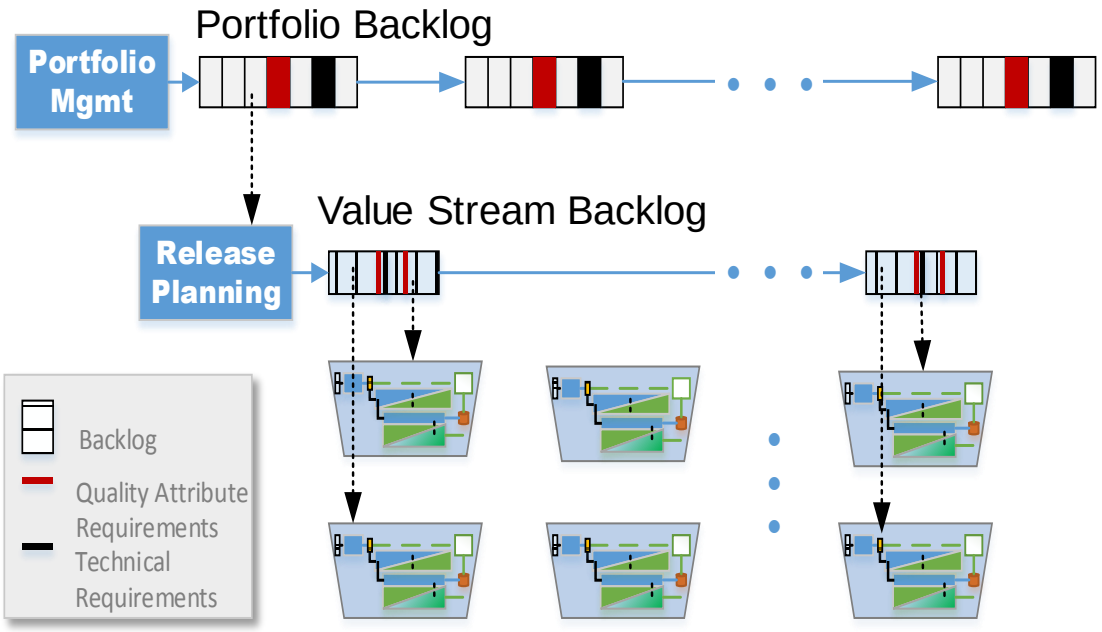
Agile “Inter-twingles” Development and Test

- Development and test are not separate or standalone
- Tests are added to a repeatable test suite
- Test suite is repeated for each change
- Demos ***not*** a replacement for testing



Agile iteration relies on more testing earlier

- Testing used for rapid refinement of loosely understood requirements, architecture, and design
- Requirements and implementation are finely sliced
- Each slice is tested immediately and repetitively



DoD Acquisition requirements are unique

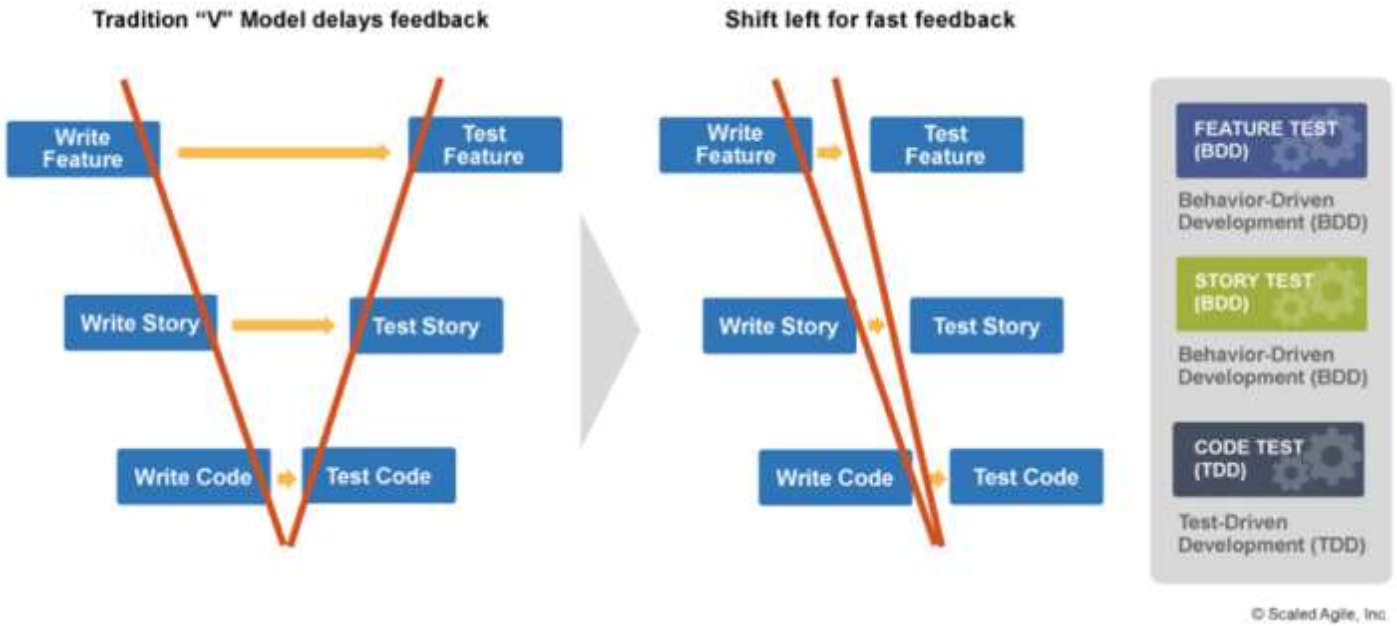
DoD requires large systems to plan for and undergo independent Operational Test & Evaluation

- Planned and executed by a different organization than the Program Office acquiring the software system
- Requires a very early Test and Evaluation Strategy more compatible with a “big bang” delivery than the incremental delivery typical in Agile

USAF guidance(AF99-103) now aligns independent testing with a more incremental approach

- Integrated testing and integrated test teams are a specific strategy called out
- Incremental testing is specifically discussed and encouraged prior to full operational testing of a deployed capability

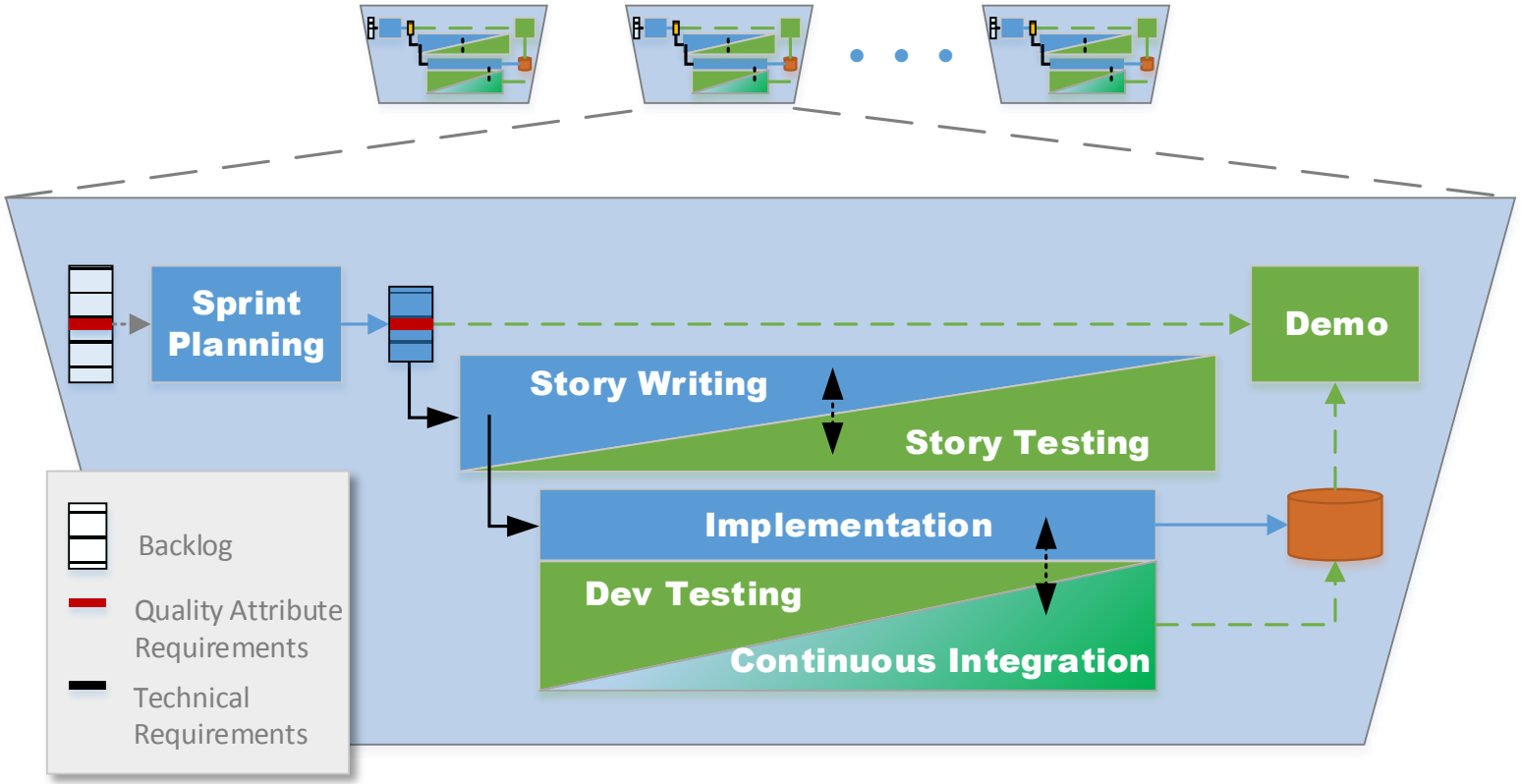
Shift-left testing



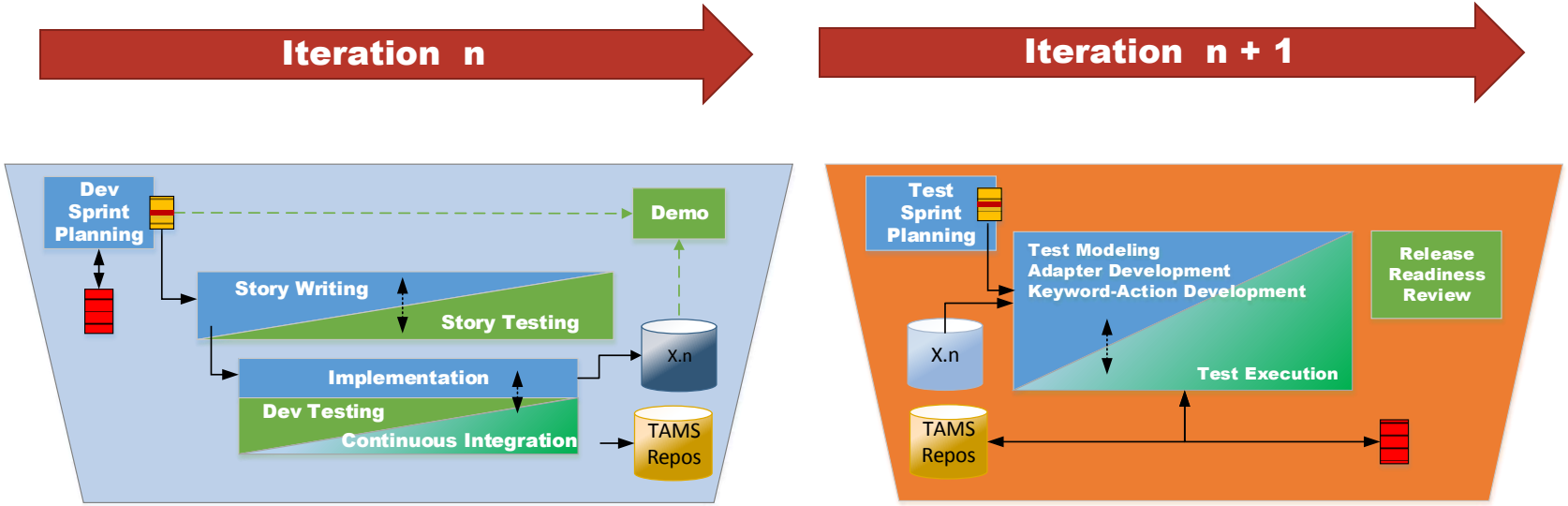
We may not be able to run the tests, but we're thinking about them early

A Strategy for Shift-Left Testing

Classic Agile inter-twingles development and test



Agile High Assurance Dev/Test cadence



Development Sprint – Development Team
Version *x.n* developed and stabilized

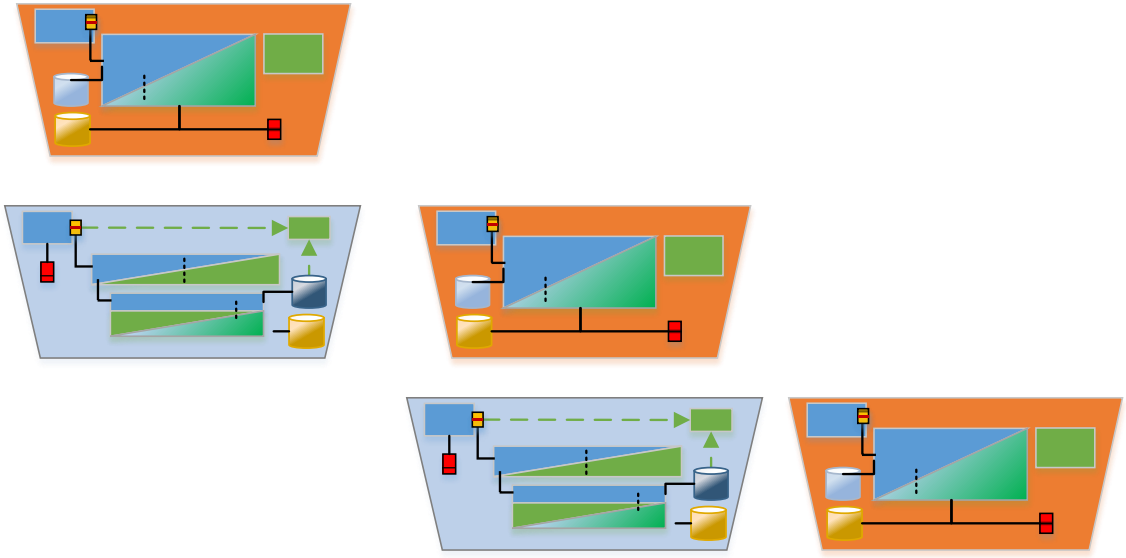
Test Sprint – Test Team
Version *x.n* modeled and stressed

Agile High Assurance Dev/Test cadence

Version 0.1

Version 0.2

Version 0.3



Agile High Assurance Dev/Test Cadence

2-4 week Iteration/Sprint

	0	1	2	3	4	5	6	Release
0	Release Planning	Test Env Prep						
N		Dev 0.1	Test 0.1					
N+1			Dev 0.2	Test 0.2				
N+2				Dev 0.3	Test 0.3			
N+3					Dev 0.4	Test 0.4		
N+4						Dev 0.5	Test 0.5	
Release Candidate							Dev RC	Test RC, Release
								Support, Retro

Version

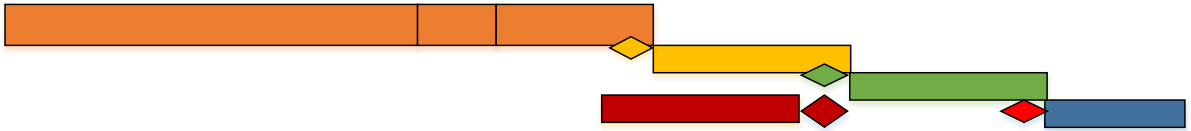
Team Sprint

Dev Sprint

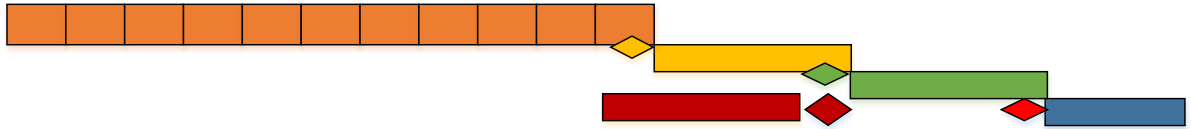
Test Sprint

Shift left Testing

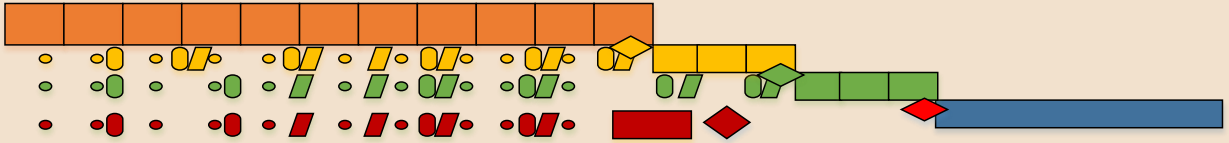
Traditional Vee-process



Agile development with traditional DT and OT (Hybrid)



Agile development with traditional DT and OT, early integration synch points



Agile High Assurance Dev/Test Cadence

OEM Dev | Developmental Testing (DT) | Cyber | Operational Testing (OT) | Deployment

Moving from phased and siloed testing to Agile testing is the "Big Deal"

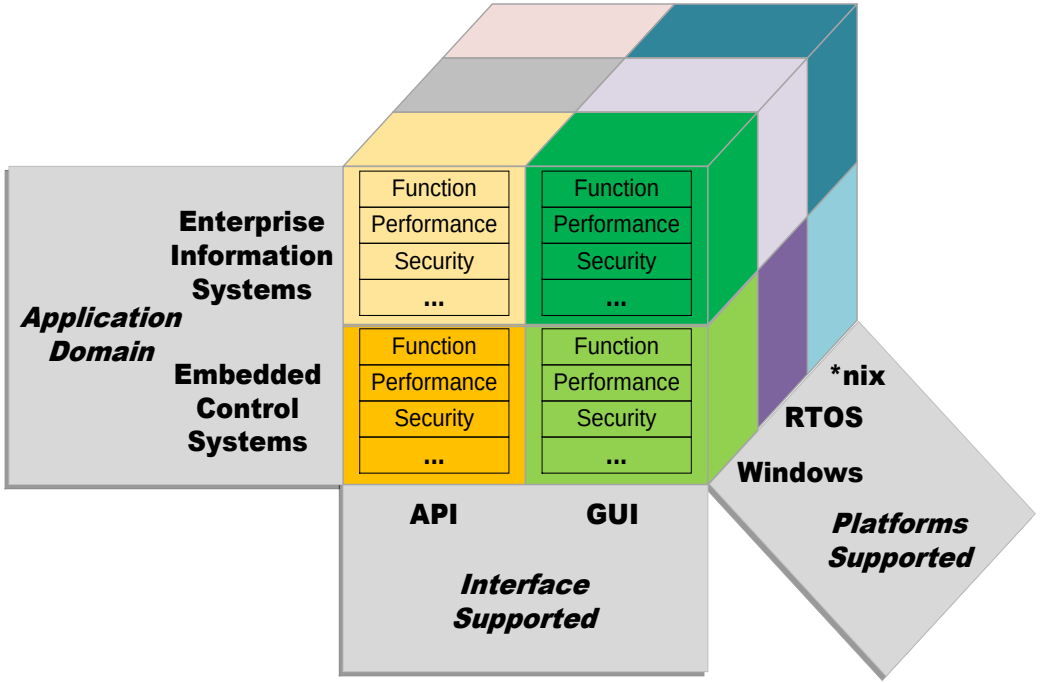
Integrating Agile cadence with DT/OT is a key challenge

Tools for Test Automation

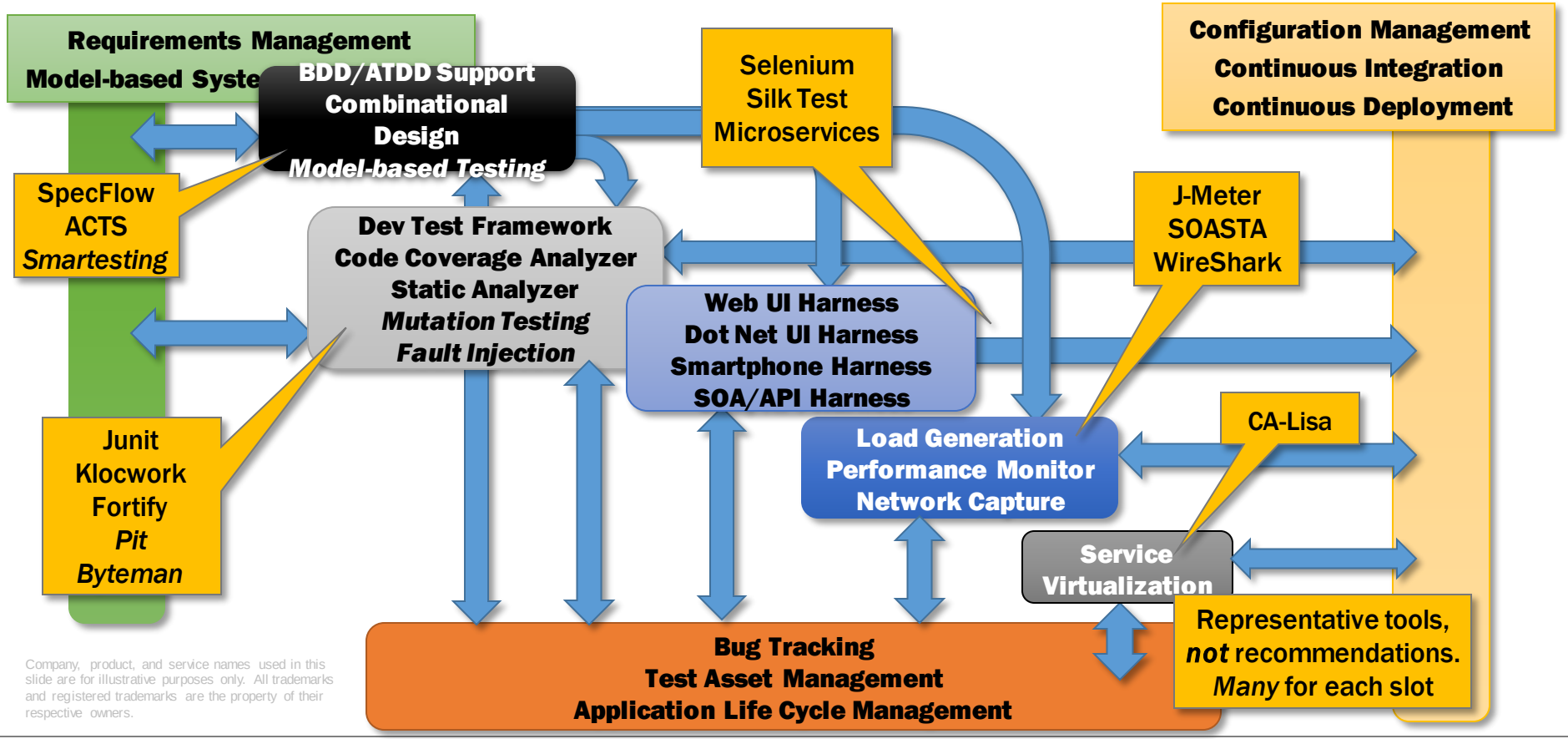
- There are *hundreds* of COTS, FOSS, and GOTS software testing tools

Tools for Test Automation

- There are *hundreds* of COTS, FOSS, and GOTS software testing tools
- Each tool is specialized for a certain kind of testing
- Each tool is specialized for a tool stack, target stack, and target interface

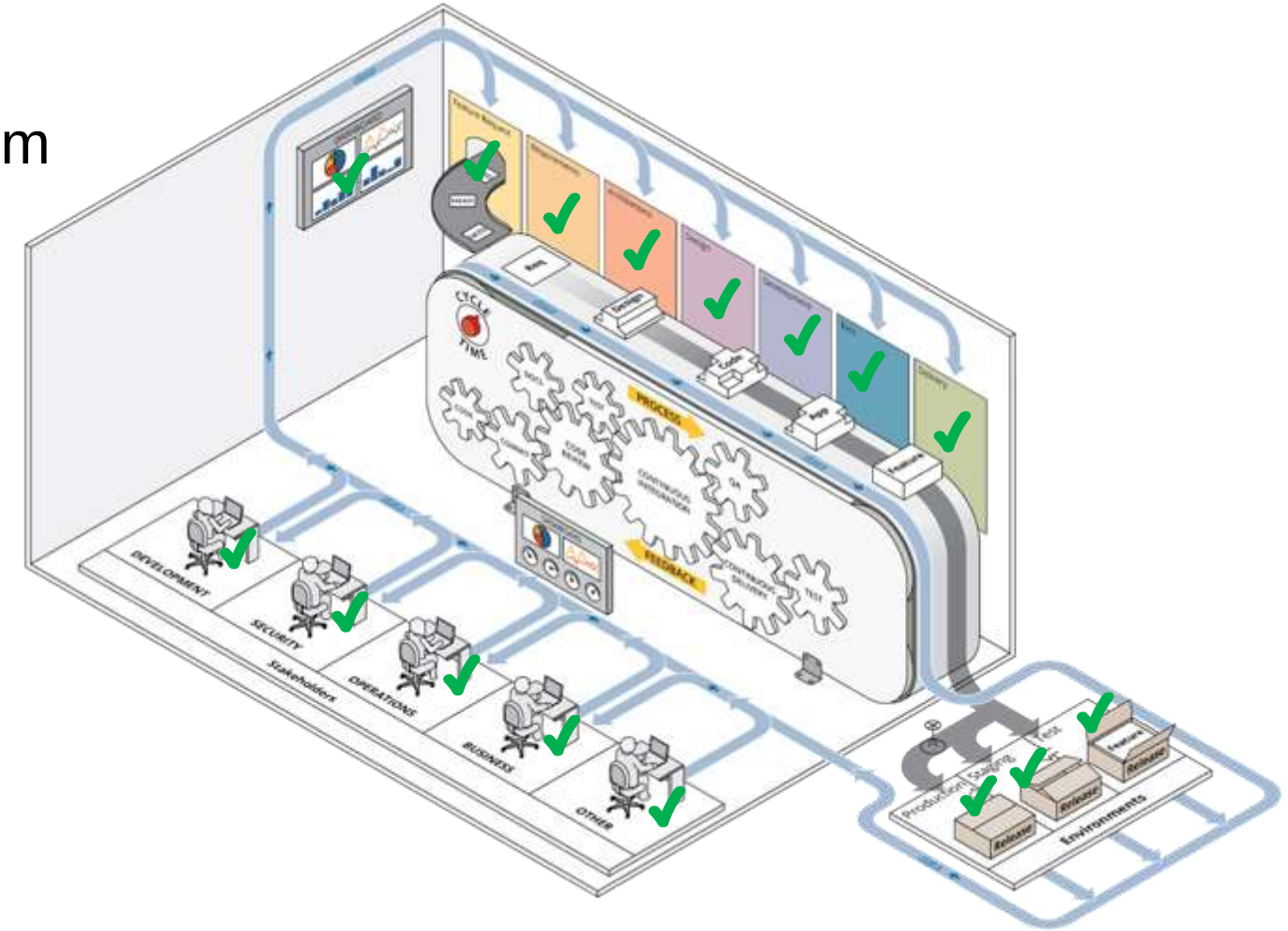


Exemplary Test Automation Reference Architecture

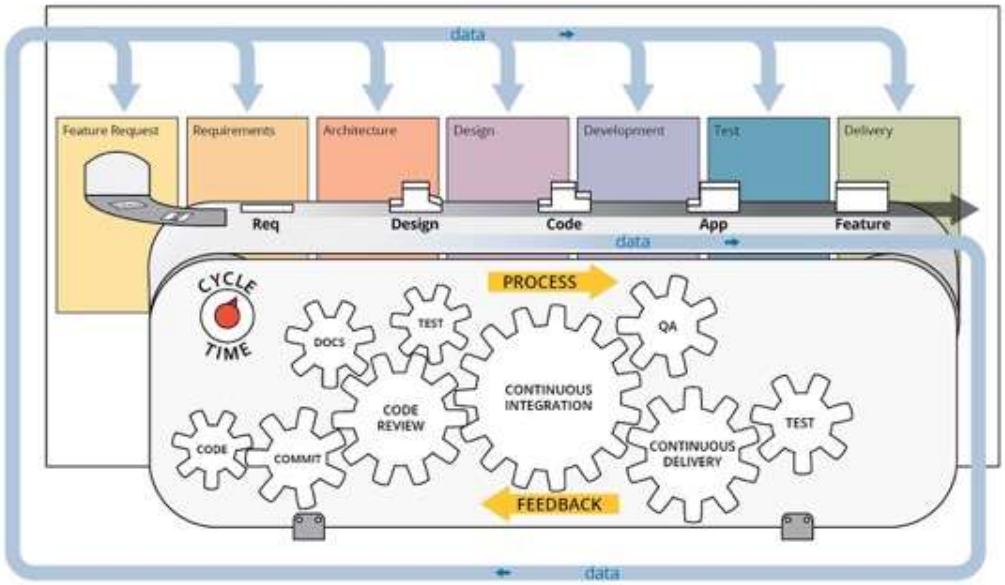


Takeaway

Apply Testing from *Inception* to *Deploy* and improve every delivery



Next Steps



- Bring tester community into development early
  - Ideal time is in backlog grooming; testers can develop acceptance criteria
  - Particularly true for SAFe Features and Capabilities – use BDD to define acceptance
- No surprises
  - No hidden tests – provide tests to developers as soon as you have them
  - Independence (e.g., of OT&E) is not isolation
- Integrate. **Integrate. INTEGRATE**
- Automate as much as possible
- Tests should be delivered with the code

For more information…

DevOps: https://www.sei.cmu.edu/go/devops
DevOps Blog: https://insights.sei.cmu.edu/devops
Webinar : https://www.sei.cmu.edu/publications/webinars/index.cfm
Podcast : https://www.sei.cmu.edu/publications/podcasts/index.cfm

Thank you

Hasan Yasar
Technical Director, Adjunct Faculty Member
Continuous Deployment of Capability,
Software Engineering Institute | Carnegie Mellon University
hyasar@cmu.edu

It is a question-and-answer time!