



Welcome

EIC EDGE – Week 1

Software Development Methodology

 **Develop**Intelligence

A PLURALSIGHT COMPANY

Hello

HELLO
my name is

Allen Sanders
with DevelopIntelligence,
a Pluralsight Company.

About me...



- 25+ years in the industry
- 20+ years in teaching
- Certified Cloud architect
- Passionate about learning
- Also, passionate about Reese's Cups!



Prerequisites

This course assumes you:

- Have some education and experience in Computer Science and software development
- Have completed basic onboarding

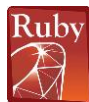


Why study these subjects?

Quality software engineering is built upon a solid software development methodology

- Knowing how iterative software development works helps you confidently contribute from your role
- Understanding Agile principles helps you work more effectively in teams
- Understanding DevOps (and DevSecOps) orients you to Capital Group's tech culture and enables secure continuous improvement
- Being familiar with JIRA and how it's used helps you create, track and contribute to technology workflows
- Knowing how to use GitHub positions you for divide-and-conquer on project completion

We teach over 400 technology topics



You experience our impact on a daily basis!





My pledge to you

I will...

- Make this interactive
- Ask you questions
- Ensure everyone can speak
- Use an on-screen timer



Objectives

At the end of this course you will be able to:

- Describe key aspects of iterative software development and understand your role in the process
- Participate in Agile work efforts in a confident and meaningful way
- Understand and help to improve the DevSecOps culture at Capital Group
- Work within JIRA to manage work and break down complex tasks
- Use GitHub to effectually manage and contribute on important source code assets



Agenda

- **Day One**
 - Software development methodologies and iterative software development
 - Software Development Lifecycle (SDLC)
- **Day Two**
 - The Agile Manifesto and best practices in Agile/Scrum
 - Working with JIRA



Agenda

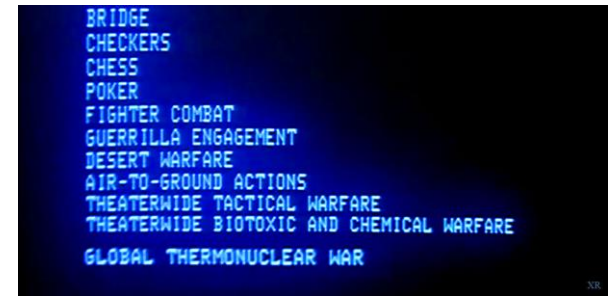
- **Day Three**
 - Introduction to DevSecOps
 - Introduction to Test-Driven Development (TDD)
- **Day Four**
 - GitHub Part 1
- **Day Five**
 - GitHub Part 2



How we're going to work together

- Lab work will take place in individual workspaces – isolated, browser-based virtual machines
- Classroom labs may include individual work as well as group discussion and work
- I welcome being interrupted – if you need more info, or clarification, or anything else, just break in and ask. I am here to help you.

Would you like to play a game?



- To make it interesting, let's add a competition factor to our time together 😊
- For questions posed to the larger group, 1 point will be awarded for each answer provided by a participant
- 2 points will be awarded for acting as a spokesperson for a breakout room during one of our group discussions
- There will be a rubric provided for the capstone (targeted for week 4) which will define a points breakdown for the implemented solution



Let's explore the development environments we'll be using...

- Information about individual VM you'll be logging into and using throughout the course will be provided
- We'll do a review of available software offerings that you'll have access to in the virtualized environments
- We'll make sure you have a GitHub account (separate from your corporate Capital Group account) available for use during the course
- We'll make sure you have access to the software needed during our time together
- We'll install a local installation of Jira for labs

Software Development Methodologies and Iterative Software Development

Common Approaches to Software Development

- Waterfall
- Iterative



Waterfall Software Development

A sequential development process that flows like a waterfall through all phases of a project (requirements, design, implementation, testing, and deployment for example), with each phase completely wrapping up before the next phase begins.

Key aspects:

- Majority of research done up front
- More accurate time estimates
- More predictable release date

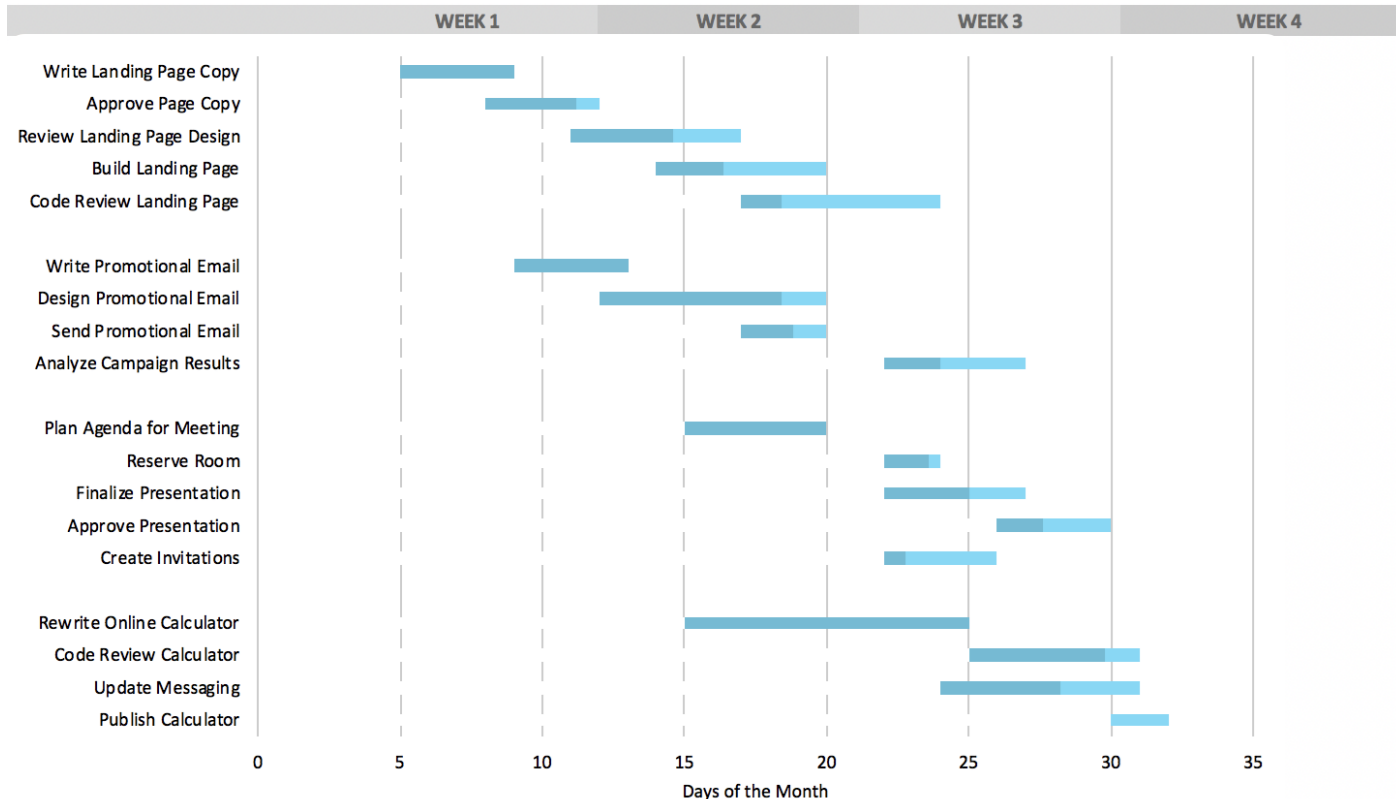
Cons:

- Process is brittle – can't pivot easily in terms of changing requirements
- Long lead times – difficult to respond to rapid business evolution

Tools:

- Commonly use Gantt charts to track projects, subtasks and dependencies

Typical Gantt Chart





Agile Software Development

A group of software development methodologies based on iterative development.

Requirements and solutions evolve through collaboration between self-organizing cross-functional teams.

Key aspects:

- Incremental delivery
- Always ready to ship
- Continuous inspection of work product and process provides feedback for continuous improvement

We will explore Agile in more detail later in this class

ESSAY:

Your experience with
various software
development systems
(3-5 sentences)

Describe the software development or project management approaches you have worked with, and your opinions on the pros and cons of each

Notes:

- This shouldn't be a long essay, and doesn't need to conform to university standards 😊
- It's more important that you communicate how the systems worked, in plain language – and what it was like to work in those systems. Talk about WHY you think something worked well, or didn't
- Please send your 3 – 5 sentence answer to the instructor via direct chat



Iterative Software Development

Key elements:

- Incremental improvement to code and process
- Always ready to ship
- Iterative feature development
- Working with the Product Owner
- Pivoting mid-development



Incremental improvement to code and process

Principle: Using feedback obtained from regular examination of product and process lets you incrementally improve both

Key aspects:

- Work in short cycles (1-6 weeks) composed of overlapping phases: requirements, design, programming, testing.
- Build on what was built before and produce a working product at each stage. Gather customer feedback at the end of each iteration.
- Each iteration, evaluate the team's performance. Change internal process as needed to improve quality of estimation and delivery.



Always ready to ship

Principle: after the first iteration, the product is always "ready to ship". That is, the product does something useful and is in a releasable state.

Key aspects:

- This is a risk mitigator – if the project is cancelled or greatly reduces in scope, there is still something of value to show for the work efforts
- Projects are less likely to be cancelled, as there is always a working product that shows the project's value and the team's competence

Key practices that support this principle:

- Do daily software builds
- Fix bugs right away
- Relentlessly use automated testing to mitigate regression



Iterative Feature Development

Principle: Instead of building all features and then releasing (waterfall), develop one feature, then the next, always keeping the application in a shippable state. Continually re-assess priority of feature development, including refinement of existing features, based on feedback.

Key aspects:

- Work closely with all concerned parties
- All involved need to understand and appreciate that development on project will be iterative
- Application reviewed at demo is still “work in progress”
- Focus is on gathering “real-time” feedback so course corrections can be made if required

Key practices that support this principle:

- Set expectations for all stakeholders
- Educate stakeholders on iterative development so they are effective contributors
- Be willing to change



Working with the Product Owner

Principle: working closely with the Product Owner enables the incremental production of shippable features that are valuable to the end user

Key aspects:

- Consult with the product owner before each iteration, agreeing on the scope and details of work for that iteration.
- During the iteration, the external customer or project manager cannot change the scope for that iteration, but the development team may change the scope by dropping features if the end date will not be met.



Pivoting mid-development

Principle: An agile, iterative process can allow a project to pivot in small or large degree, mid-process, without excessive cost of time or resources

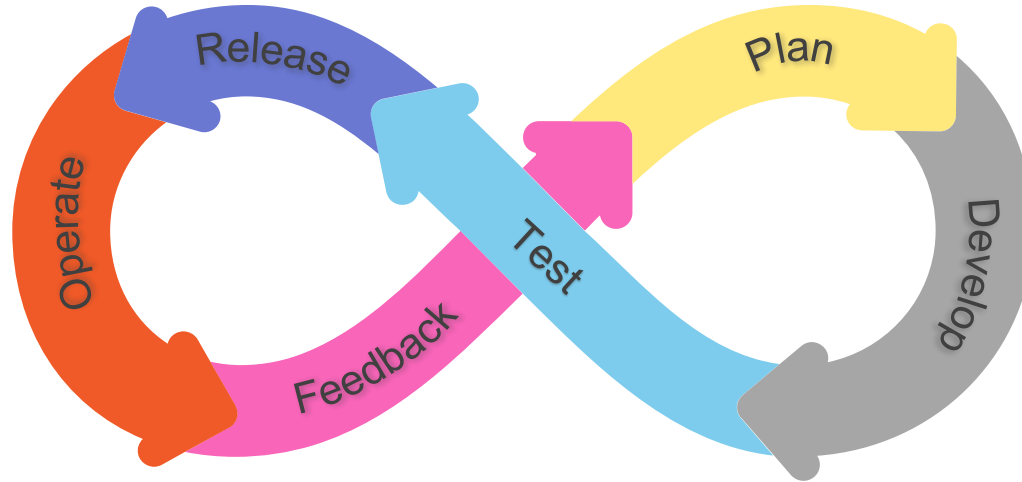
Key aspects:

- Manage to meet business goals, due dates and budgets. Be willing to change requirements to fit these, not the other way around.
- Learn as you go; be adaptable to new or changing business needs that become clear only after development begins.
- Analyze existing implementations frequently to determine that they are meeting business goals.



Software Development Lifecycle (SDLC)

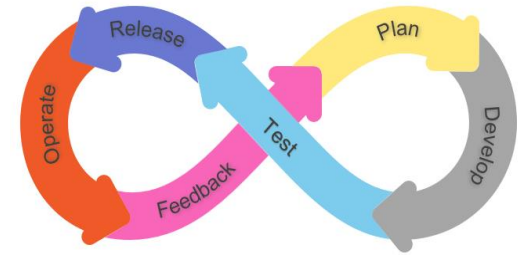
Software Development Lifecycle (SDLC)



Plan Phase

Focus:

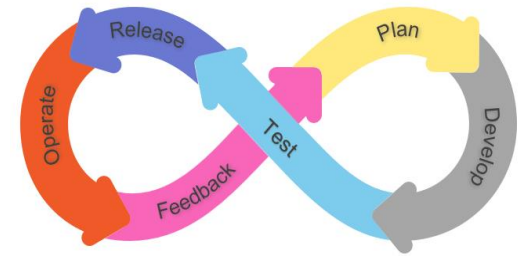
- Deciding on the set of user stories for team focus over the upcoming iteration
- Refining those stories to ensure all required detail to complete is in place
- Identifying key requirements (both functional and Quality of Service) that need to be met



Develop Phase

Focus:

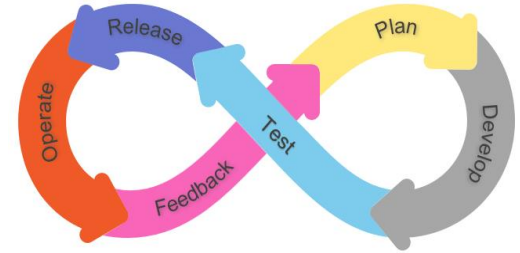
- Executing coding tasks that will complete the set of user stories in scope for the iteration
- Implementing automated unit tests that can continue to confirm requirements met
- Addressing any potential security vulnerabilities in the code being built
- Ensuring the full body of source code created to date (including new code) remains “clean”
- Ensuring new code gets correctly merged with existing code and code added by other members of team



Test Phase

Focus:

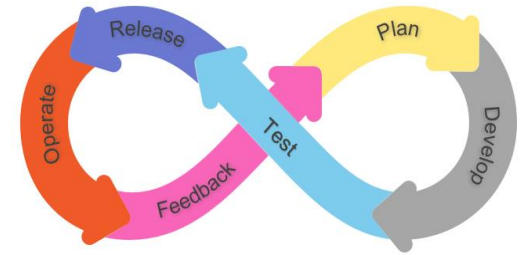
- Code updates from all team members built as a single, comprehensive unit
- Seeks to validate that all the code operates correctly together
- Through automated integration tests, verifies requirements successfully satisfied
- Likely includes deployment to pre-production environments for testing
- Can include automated security scans as well to help find security vulnerabilities early



Release Phase

Focus:

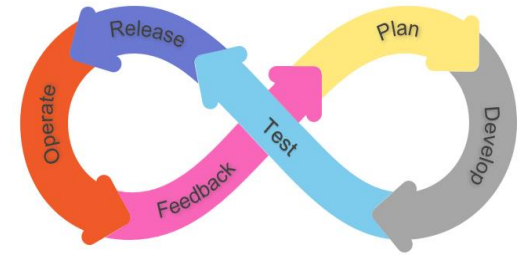
- Installing required runtime, compiled binaries, and configuration to the production environment
- May include small set of “smoke tests” (test transactions that can be run through the system to validate at a macro level)
- Verification that all infrastructure and configuration is in place for user access
- Ensuring that required documentation is up-to-date for production support



Operate Phase

Focus:

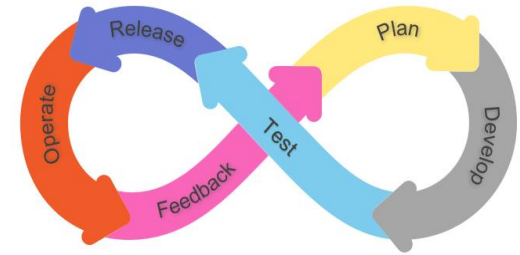
- Ensuring that the application remains healthy in production
- Addressing any production incidents encountered during application operation
- Identifying and tracking any issues/bugs that need to be elevated from “incident” to “problem” status



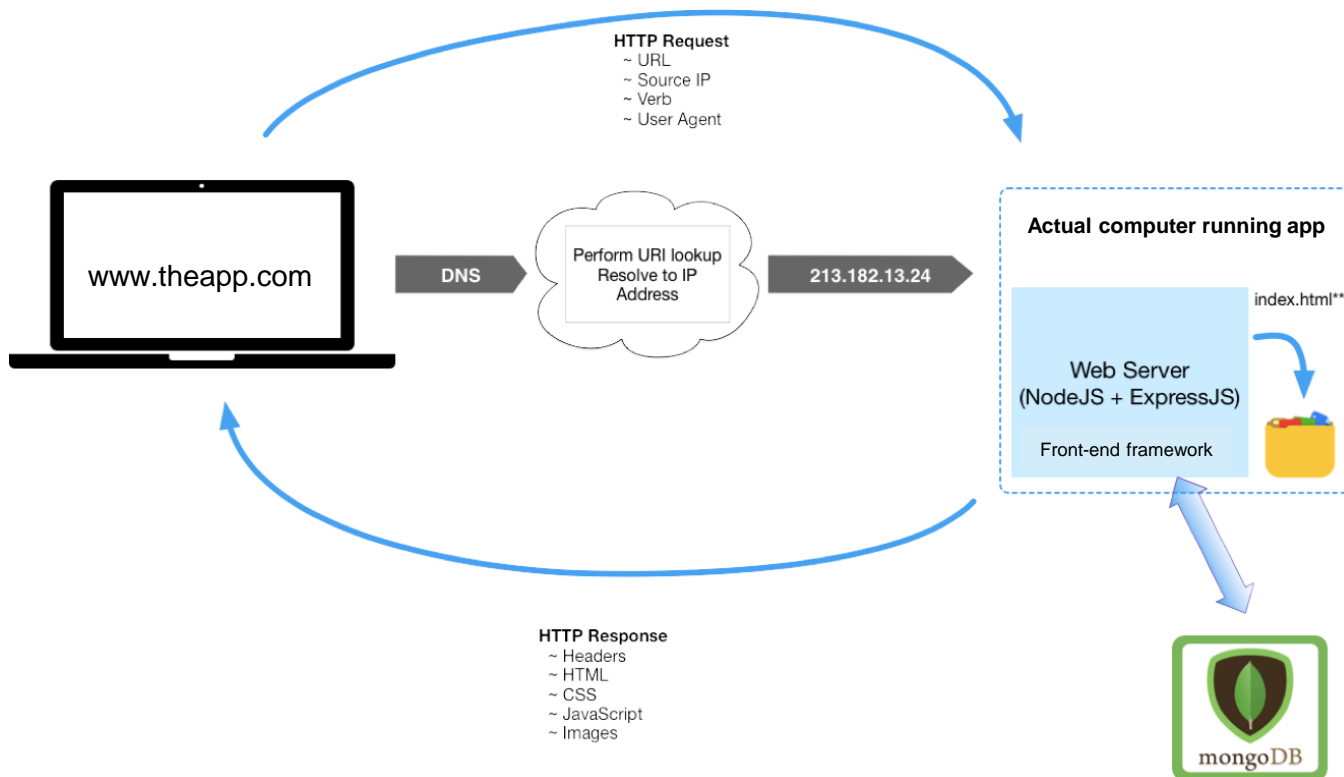
Feedback Phase

Focus:

- Gathering business stakeholder, technology stakeholder, and user feedback on how application is operating post deployment of most recent iteration (and overall)
- Can include explicit demo to project team as part of closing out current sprint
- Goal is uncovering opportunities to continuously improve the product and getting those opportunities documented for inclusion in future sprints



Elements of a Full-Stack App and HTTP Request/Response Cycle



WORKSHOP:

Looking at the
“Reference
Implementation” from
an iterative approach

Group discussion considering the Reference Implementation from the following points of view:

- Itemize the set of high-level features that will be needed to implement this application?
- Which features should be implemented first? Which features should ship first?
- How might your approach need to pivot, based on market or other factors?

WORKSHOP:

Looking at the
“Reference
Implementation” from
an iterative approach

Group discussion considering the Reference Implementation from the following points of view:

- Based on your prioritization of the features that should be implemented/shipped first (from the previous discussion), what should your first two iterations look like?
- Include tasks for each phase (Plan, Develop, Test, Release, Operate)
- What are some practical ways to gather feedback on the features deployed as part of the first two iterations?

The Agile Manifesto and Best Practices



The Agile Manifesto and its Origins

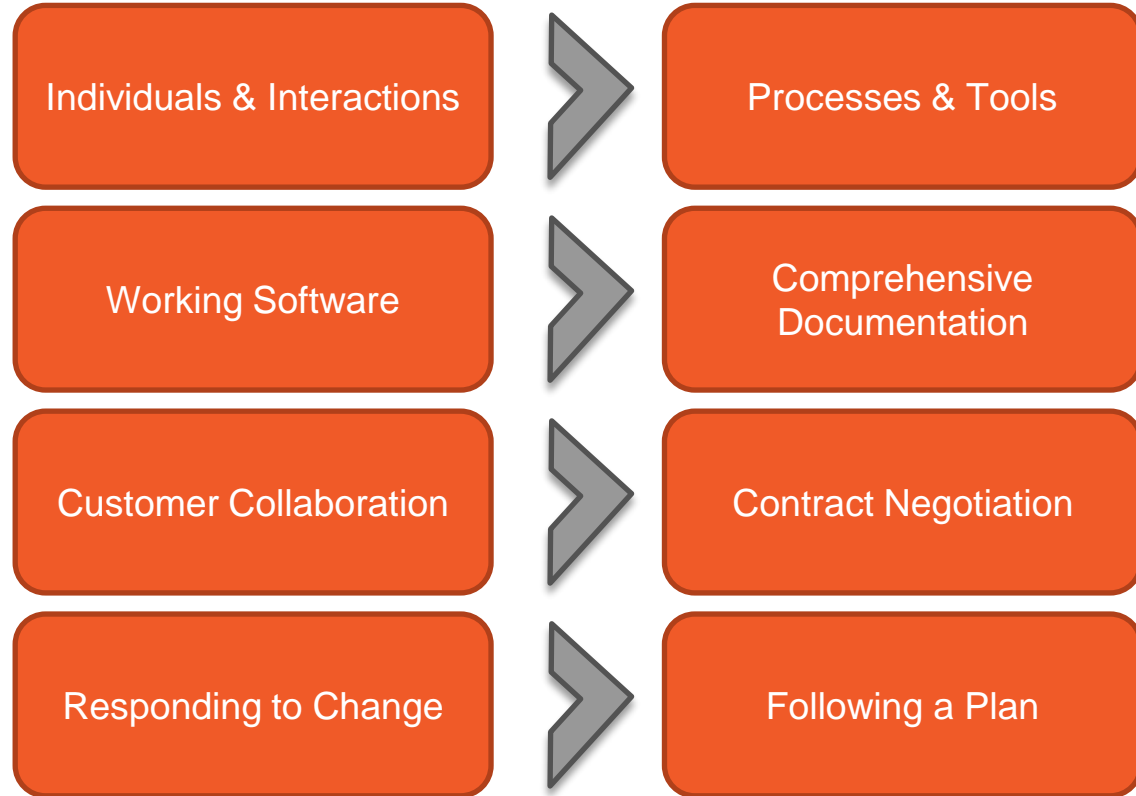
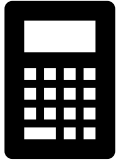
What is it? A statement of values regarding approaches to how teams create and ship software

When was it made? 2001, by 17 top software developers and consultants

Key aspects:

- 12 values were agreed upon
- These values are NOT a process, implementation or standard; they are opinions
- Many popular project management methodologies have arisen that are based on these values
- Many Agile implementations are poorly executed; it takes great cooperation between business leadership and a competent, self-guided dev team to make it work

Agile Values





Key Elements of the Agile Approach

The Agile method is an iterative and incremental tactic to software design that utilizes constant planning, understanding, upgrading, team partnership, development, and delivery.

It is driven by the principles of providing value and collaborating with stakeholders.

It starts with customers defining the end uses of the final product and the kind of problems the final product attempts to address.

Designated teams start to plan and work on a complete process through planning, implementing, and appraising.

Since the development process is iterative, errors are resolved in the intermediate stage of the project.

Modern Approaches to Agile Software Development

Several project management approaches implement Agile values in some way. Often, they bring in elements of other approaches.

Popular methodologies:

Kanban	Scrum	Scaled Agile Framework (SAFe)
<ul style="list-style-type: none">• Uses visual boards to view & organize tasks• Uses elements of “just in time” lean manufacturing strategies• Emphasizes throughput	<ul style="list-style-type: none">• Aligns closely with Agile values• Breaks down product development into iterative sprints• Makes use of exclusive roles (“Scrum Lead”, “Product Owner”)• Emphasizes constant communication	<ul style="list-style-type: none">• Workflow and organizational patterns to help deploy Agile at scale• Can support large organizations

All three of these approaches are in use in some fashion at Capital Group.



Common Agile Practices at Capital Group

Common practices are:

- Close collaboration between an engaged and knowledgeable Product Owner and a self-organizing dev team with strong ownership for code and team quality
- Relentlessly working to break complex tasks down into smaller, more discrete and defined elements
- Making use of Scrum boards to track and prioritize work
- Sprints run using Scrum principles and practices
- Continuously evaluate and improve how we choose scope of work for each iteration

Typical Kanban Board

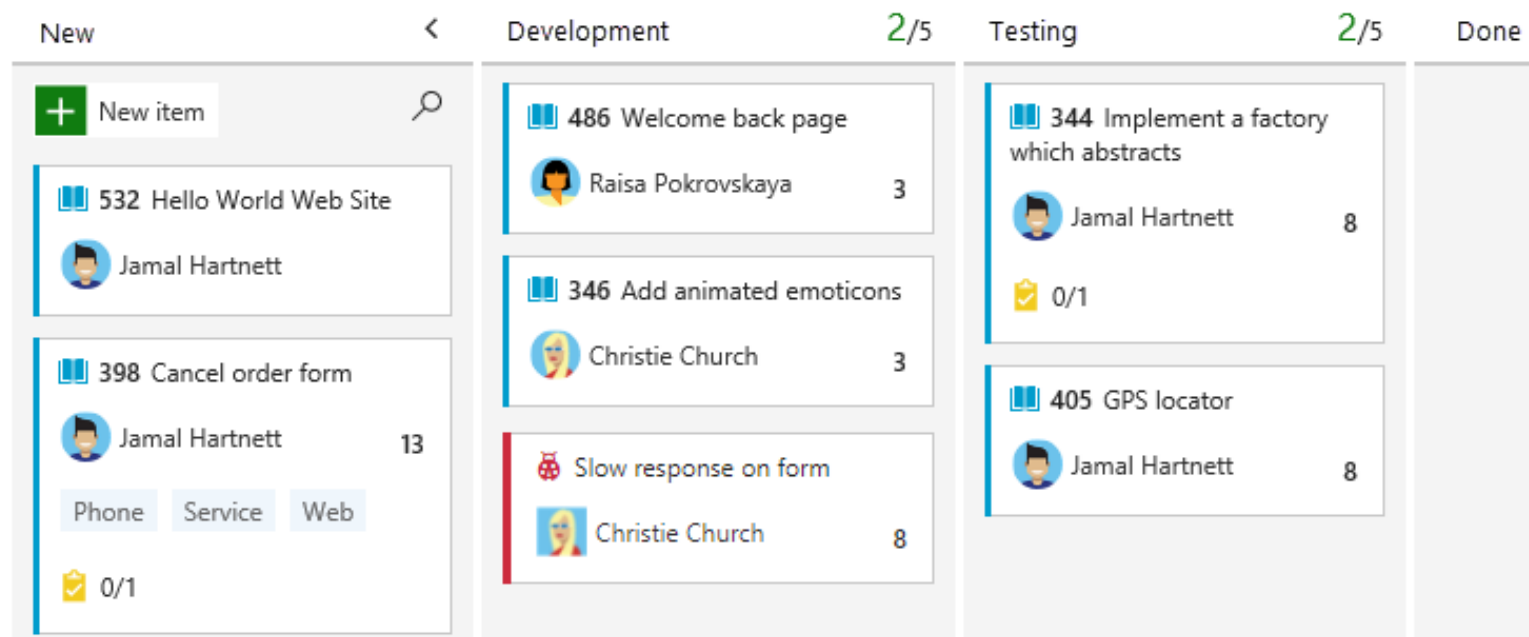
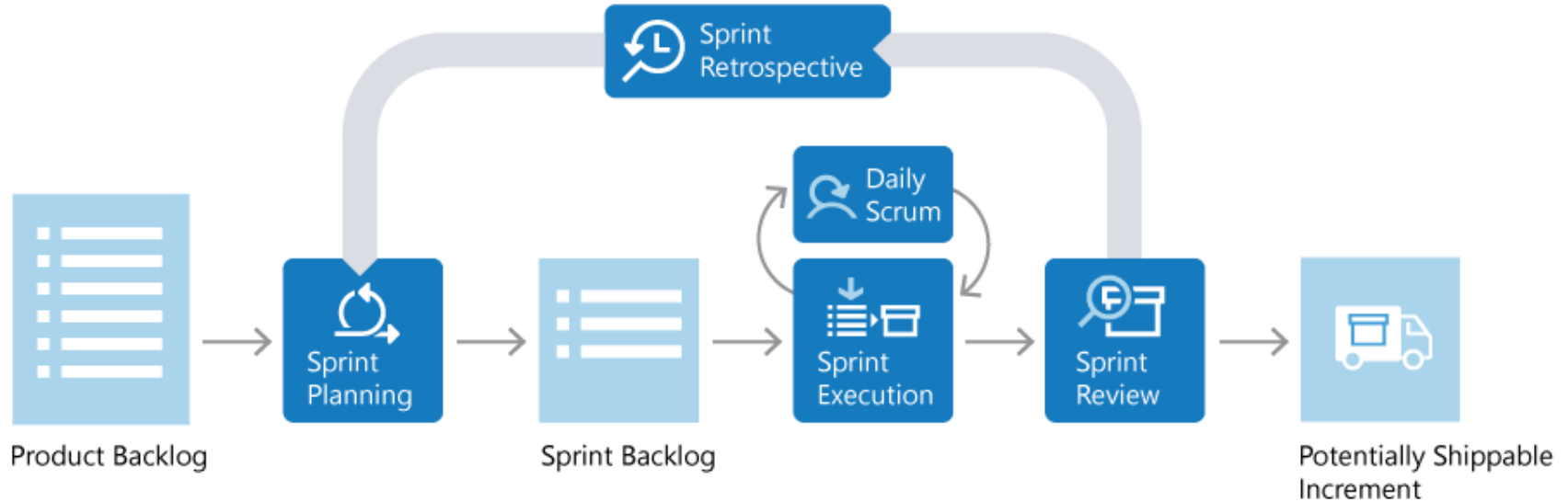


Diagram of Typical Scrum Sprint



Source: <https://docs.microsoft.com/en-us/devops/plan/what-is-scrum>

Scrum Roles

Product Owner

Scrum Lead

Scrum Team

Scrum Roles

Product Owner

- Responsible for what team builds and why
- Keeps backlog up to date and in correct priority order

Scrum Lead

Scrum Team

Scrum Roles

- Ensures that Scrum process is followed by team
- Responsible for the fidelity of the process
- Part coach, part team member, part cheerleader

Product Owner

Scrum Lead

Scrum Team

Scrum Roles

Product Owner

Scrum Lead

Scrum Team

- People building the product
- Responsible for product build (and associated quality)

Product Backlog



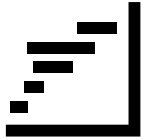
- Prioritized list of work the team will deliver
- Product owner manages – adds, changes, reprioritizes as needed
- Items at the “top of the list” get worked first

Backlog Grooming

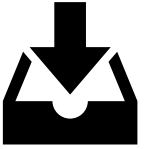


- Regular team activity which allows group to verify that quality of defined sprint items remains high
- Team discusses each item and ensures that all required information to successfully complete is in place
- For example, has the item been given an adequate description and does the acceptance criteria provide team members with the information needed to assess “done”?
- Backlog item is assigned “points” representing relative level of effort to complete
- Points are not directly correlated to hours – instead, about amount of effort required for the task in comparison to previous tasks completed or other tasks planned
- There will likely be some number of points for items that is considered “too high” to complete as a single item
- Provides threshold for determining when to divide a large task into smaller parts

Sprint Planning



- Team chooses items from the backlog to target for upcoming sprint (based on prioritization by Product Owner)
- Intended to represent amount of work team thinks they can complete within the sprint
- Items in the sprint may be broken down into a set of more granular tasks
- Historical data from previous sprints can help a team determine its velocity (i.e., how many points the team can complete in an iteration)
- Results in a “line” for the sprint within the backlog – items above the line are “in”, everything else is pushed to future iteration
- If something below the line needs to move above, it will need to be swapped with an existing task to ensure capacity to deliver



Sprint Execution

- Team executes on the items identified for inclusion in the iteration
- Short daily meetings are held among team members (called daily Scrums) – usually no more than 15 minutes
- Each member briefly describes what they worked on the previous day, what they're planning to work on today, and any “blockers” (issues preventing progress)
- As part of daily review, a sprint burndown chart can be used to assess whether or not the team is on track to complete what was committed to

Sprint Review



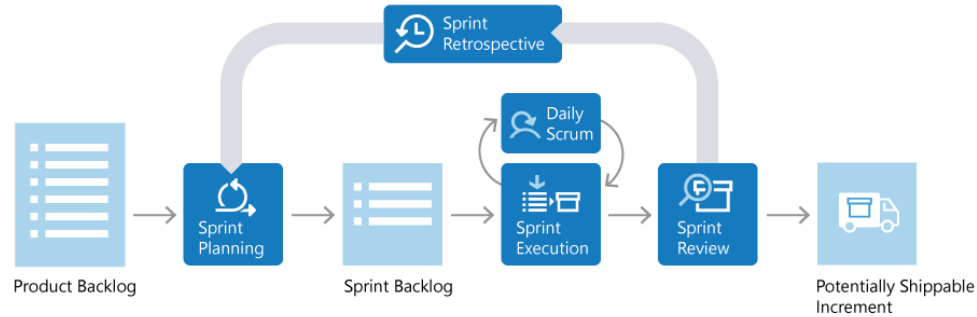
- At the end of the iteration, team demonstrates for stakeholders what's been accomplished during the sprint
- Goal is to show incremental value



Sprint Retrospective

- Team reflects on what went well during the last sprint
- Just as important (if not more so) is identifying specific areas for improvement to take into future sprints
- Outcome should include a set of action items to help steward improvement

Team Reflection



In your assigned breakout room, discuss the following questions. Nominate someone to share results of the discussion.

- Which of the sprint activities do you feel would be the easiest to complete, and why?
- Which of the sprint activities do you feel would be most difficult, and why?
- Which of the sprint activities would you consider to be the most important, and why?

WORKSHOP:

Building the Team Definition

Using information gathered during the previous discussion, work with your breakout room to build your Team Definition. Team Definition represents the team's working agreement. Go through each activity executed during a sprint and set expectations for how your team will perform and ensure ongoing fidelity of each ceremony.



Knowledge Check

This role is ultimately responsible for ensuring that all scrum ceremonies are correctly followed:

- A. Scrum team
- B. Product owner
- C. Team leader
- D. Scrum lead

Enter the letter corresponding to your answer in chat



Knowledge Check

True or False: Points assigned to sprint items should directly correspond with the number of hours the task is expected to take to complete.

Enter your answer in chat



Knowledge Check

This role is ultimately responsible for keeping the backlog up to date and in correct priority order:

- A. Scrum team
- B. Product owner
- C. Team leader
- D. Scrum lead

Enter the letter corresponding to your answer in chat

Working with JIRA



Overview of Project and Issue Tracking Software

There are many applications in use to manage tech work – some oriented around service tickets; some around dev tasks, some around both

Popular project and issue tracking software:

- JIRA
- Asana
- Trello
- Monday
- Zoho

There are MANY others; this is a crowded space. In dev work, integration with tools used in the full dev cycle is important – testing, deployment, monitoring, etc.



Overview of JIRA

JIRA is part of a family of products from Atlassian. It can be configured to align with popular Agile methodologies.

Key functionality:

- Provides a central hub for design, coding, testing, collaboration, release, and maintenance
- Integrates with CI/CD systems
- Provides a feature backlog for planning purposes
- Enables Sprint planning



Basic Building Blocks of JIRA

JIRA is built around three basic concepts:

- Projects: A collection of Issues, all oriented to one product or work initiative
- Issues: The basic element of work in JIRA. Includes the work to be done, as well as metadata about the work. This can be net new work, bug fixes, service tickets, etc. – any type of work
- Workflow: A record of the life of an Issue. Issues can only have one state at a given time. Transitions from one state to another can be tracked and controlled as needed

DEMO:

Lifecycle of a JIRA Issue



Using a Scrum Board

Your Scrum board visually represents your workflow – the sequential state changes of your team's work. A common configuration at Capital Group might be:

New Issues (in the form of features/stories/tasks/bugs, etc.) begin in a Product Backlog, as they are identified

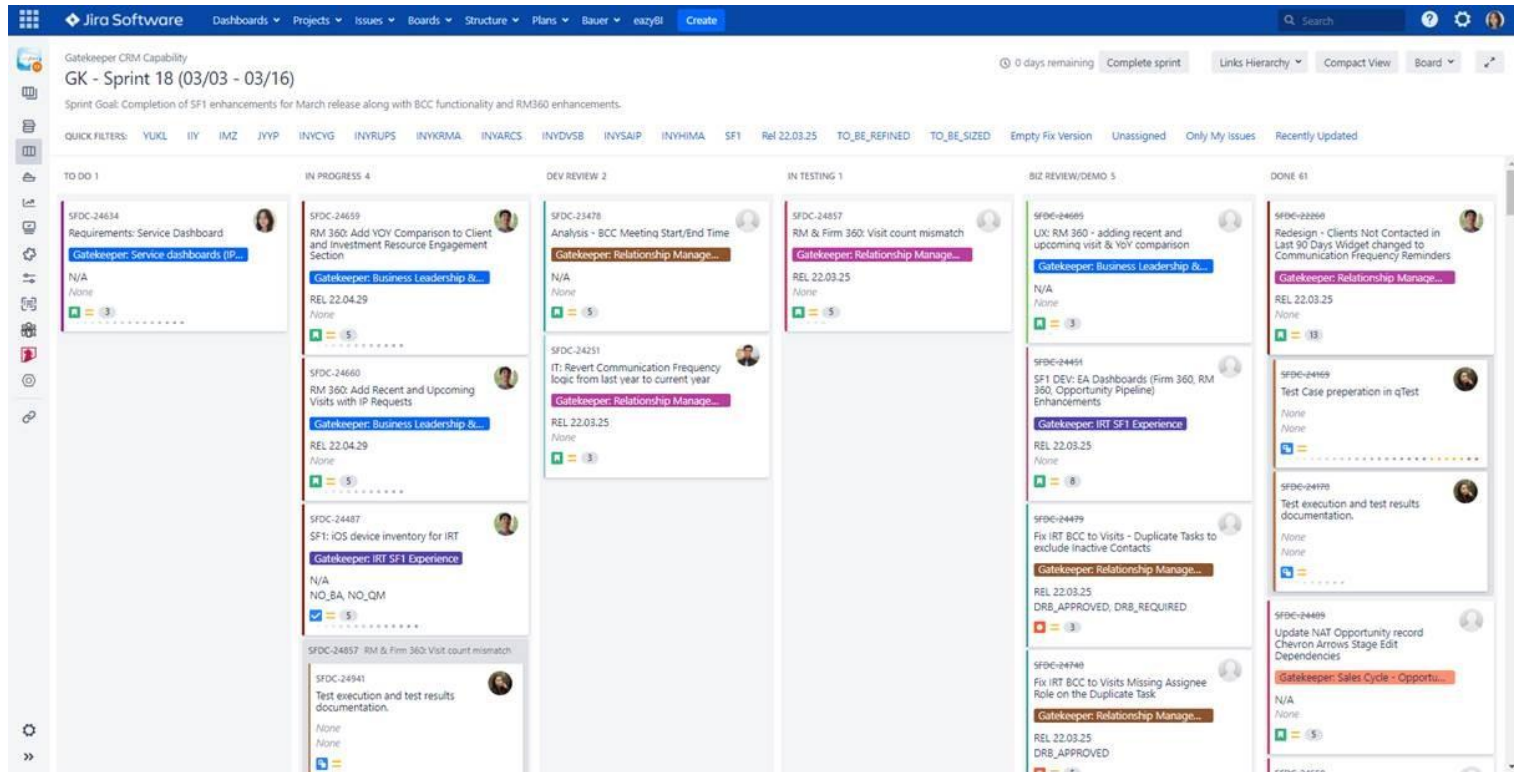
In Sprint Planning, Issues are refined and moved into the Sprint Backlog

Whether assigned a work item, or pulling it yourself, the board lets you choose an Issue and move it through the states in the defined Workflow for the Project


Often, your actions in other tools will result in a state change for an Issue; the board will update to reflect that change

Because so much of the work activities is tracked electronically, you can also use JIRA's Control Center to view important metrics about the work of the team and of its members. It tells you about things like: How long an issue sits in a state; average time to complete an Issue; etc.

Example Scrum Board





Team Activity – Story Review

 Vitals / VITAL-1465
UPDATE TO Portfolio button

[Edit](#) [Add comment](#) [Assign](#) [More](#) [Dev Complete](#) [Stop Progress](#) [Resolve](#)

Details

Type:	 Story	Status:	IN PROGRESS (View Workflow)
Priority:	 Medium	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	Vitals 2.5.6
Labels:	None		
Story Points:	2		
Risk Exposure:			
Acceptance Criteria:	Portfolio toggle between custom and autogenerated has been moved to the portfolio page.		
Epic Link:	Vitals 2.5 Release		
Sprint:	Vitals-S Sprint 73		
Last Transition:	Start Progress		

Description

who-as-a developer


what-remove portfolio button from lefthand side, create a toggle button similar to dark/light mode that will allow directors to toggle between their custom created portfolio and a portfolio of products in which they are listed as owners on Vitals.

why-this will allow directors to quickly view their auto-generated portfolios toggle to their custom portfolios

Attachments



Discuss as a breakout room and nominate someone to share your group's findings.

Review the two stories displayed here – what are some key differences you see between the two? Which of the two do you think is better structured, and why? For the one less structured, how might you go about correcting it?

 Cloud Platform-Architecture Team / CPAT-95
CDD Phoenix Fact data POV

[Edit](#) [Add comment](#) [Assign](#) [More](#) [Start Progress](#) [Deferred](#) [Resolve](#)

Details

Type:	 Story	Status:	READY (View Workflow)
Priority:	 Medium	Resolution:	Unresolved
Labels:	None		
Story Points:	1		
Assigned Team:	Data Arch		
Risk Exposure:			
Acceptance Criteria:	Complete analysis and propose solutions.		
Epic Link:	Critical Business Consulting		
Last Transition:	Ready		

Description

[Click to add description](#)

Attachments

LAB:

Breaking down a large
story into smaller
tasks

Story description: As a customer of the online ordering system, I want to be able to add new items to my shopping cart, update items already in the cart (e.g., adjust quantities), and remove items from the cart that I no longer wish to purchase so that I can effectively manage and submit an order.

Acceptance criteria: I can add a new item to the cart, I can update quantities on an item that is already in my cart, and I can remove items that I no longer wish to purchase.

Story Points: 13

As a team (in your breakout room), create new stories that break this larger task down into more manageable “bites”. Include description, acceptance criteria, and estimate points for relative level of effort. Also, identify any additional concerns or edge cases that should be considered; capture that information in a card as well.

LAB:

Grooming the backlog

Share your team's stories with another team in a different breakout room. The other team will examine the stories you've created and provide feedback for the following on them.

Is all required detail included? Is anything missing?

With the provided stories, do you feel like you have enough information to successfully complete that feature?

Would you suggest any updates to the story to improve?

Review the number of story points assigned – do you agree with the number? Why or why not, and how would you adjust?

Introduction to DevSecOps



“Shift Left” Testing Approach

DevOps (and its CI/CD automation tools) are all about creating trust – in the tools and processes, and in the code itself. Key to this is testing.

A popular movement in DevOps is called “shift left” testing. It is an approach to software testing and system testing in which testing is performed earlier in the lifecycle (i.e., moved left on the project timeline)

This means developers are getting involved more in testing.

A common tradeoff to examine here: How long does a full set of tests take? How often are devs checking code into the branch that is tested? Do the tests take so long that code changes are getting backlogged? How to we prioritize tests in this situation?



Common Security Attacks

- Denial of Service (DoS or DDos)
- SQL injection
- Large files
- Cross Site Scripting (XSS)
- Credential stuffing



Denial of Service (DoS)

- Server “flooded” with so much bogus traffic that systems are unable to serve valid requests
- Alternatively, instruction(s) received that trigger a server or system “crash”



Denial of Service (DoS)

Common flooding attacks:

- Buffer overflow (most common type)
- ICMP flood (AKA “smurf attack” or “ping of death”)
- SYN flood



Denial of Service (DoS)

Crash attacks:

- Often involves send of data targeting common classes of bug
- Request used to crash or severely destabilize the system



Distributed Denial of Service (DDoS)

- Similar profile as DoS but uses multiple systems to orchestrate the attack
- Provides attacker with advantages



Distributed Denial of Service (DDoS)

Potential advantages for attacker:

- More agents means more power behind the attack
- Location of attack is difficult to detect (often globally-routed)
- Easier to shut down a single attack machine than multiple
- Identity of attacker is more easily disguised



Defending Against Denial of Service

- Ensure service has good AuthN/AuthZ in place
- Utilize a proxy or gateway and configure throttling
- In Production, disable ICMP pings or use rate-limit for ICMP requests (e.g., using iptables)



Knowledge Check

When we shift left in Devops, what exactly is being shifted left?

- A. Planning
- B. Testing
- C. Deploying

Enter the letter corresponding to your answer in chat

DEMO:

Denial of Service



SQL Injection

- Attacker injects SQL (Structured Query Language) queries into app flow (e.g., UI)
- In some cases, injected SQL used to retrieve additional sensitive detail
- In other cases, injected SQL used to alter or damage a company's critical data



Types of SQL Injection Attack

- In-band
- Blind



In-band SQL Injection

- Uses existing channel of communication for an attack
- Error-based – attacker performs actions that cause errors in order to gather “intel” about database structure
- Union-based – attacker takes advantage of UNION SQL operator to fuse multiple SELECT statements into single response



Blind SQL Injection

- Attacker sends separate, independent data queries to a server
- Called blind because results of query are not sent back to attacker
- Instead, attacker observes results to infer vulnerabilities



Blind SQL Injection

- Relies on response and behavior patterns of server so slower to execute
- Boolean – attacker sends SQL query to database and determines if attack valid based on response received (true or false)
- Time-based – attacker sends SQL query to database and determines if attack valid based on amount of time taken to process



Defending Against SQL Injection

- Use well-defined contracts to explicitly map expected results from application
- Sanitize inputs and use parameterized queries in code
- Use a Web Application Firewall that includes protections at the application layer (including protection against SQL Injection)

LAB:

SQL Injection

SQL Injection

Walk through and execute the tutorial available at <https://realpython.com/prevent-python-sql-injection/>. This lab will demonstrate some options for protecting against SQL injection in Python using libraries for PostgreSQL.

If you do not have access to an installation of PostgreSQL where you can create a database/table, consider running PostgreSQL in a Docker container with:

```
docker run --name some-postgres -e POSTGRES_PASSWORD=<your password here> -p 5432:5432 -d postgres
```

If you have issues installing psycopg2 using `python -m pip install psycopg2`, try using `python -m pip install psycopg2-binary` instead.



Large Files

- Sometimes resembles another form of Denial of Service
- Attacker attempts to send one (or several) very large files as upload
- Could also occur with extremely large payloads (JSON or XML bodies)
- As a result, network connectivity to servers or services can become “clogged”



Defending Against Large Files

- Use configuration to limit file/payload sizes and number of concurrent connections from a client
- Utilize timeouts judiciously to prevent large file operations from completing
- Can also leverage MIME types as a way to limit acceptable types of data
- Finally, proxies or gateways (WAF) can be configured for mitigation at the network layer



Cross Site Scripting (XSS)

- A type of injection – but script instead of SQL
- Attacker uses inputs to attempt injection of a `<script>...</script>` element
- An example could be posting a comment with a link that routes to a malicious site



Cross Site Scripting (XSS)

- Without inspection for malicious content, `<script>` can be returned (and executed) in user's browser
- Malicious content can include JavaScript, HTML, Flash, etc.
- Really, any code that browser can execute



Cross Site Scripting (XSS)

Common forms of attack:

- Stealing cookie or session information
- Redirects to web content controlled by an attacker
- Executing malicious operations on user's machine
- Leveraging impersonation for elevated privilege



Cross Site Scripting (XSS)

Stored XSS attacks:

- Injected script permanently stored on target servers
- Could include storage in database, forum, comment field, etc.
- Malicious script returned to browser as part of retrieval from storage



Cross Site Scripting (XSS)

Reflected XSS attacks:

- Malicious script is indirectly transferred back to browser
- When user clicks on malicious link, code gets injected into vulnerable site
- Malicious code then reflected back to user under the cover of “valid” site interaction for immediate execution



Cross Site Scripting (XSS)

DOM-based XSS attacks:

- Takes advantage of sites that copy input to DOM without validation
- Similar to reflected in that victim is tricked into sending malicious code to vulnerable site
- However, input lands in DOM in the browser for execution instead of being reflected back



Defending Against Cross Site Scripting

- Encode and validate everywhere – do not trust user inputs, escape outputs, and manage response headers
- Use libraries with utility handlers where possible (e.g., Jinja or Django)
- Quote every attribute of every tag in HTML



Defending Against Cross Site Scripting

- Use HttpOnly directive on custom cookie response headers (i.e., “Set-Cookie” header)
- Use the “X-Content-Type-Options: nosniff” to prevent MIME type sniffing (i.e., dynamic changes to Content-Type header)
- Leverage network components like WAF with built-in protection to intersect at the network layer

DEMO:

Cross Site Scripting



Credential Stuffing

- Attackers use lists of compromised credentials to try and find a breach
- Based on assumption that many users reuse same credentials across sites
- Uses bots, automation, and scale



Credential Stuffing

- Like a brute force attack
- However, instead of random strings, uses existing lists of known credentials
- Powered by broad availability of compromised info and increasing sophistication of bots & automation



Credential Stuffing

- Attacker sets up bot able to attempt login for multiple accounts in parallel
- Uses an automated process to test effectiveness
- Monitors for breaches and pulls/retains sensitive detail when found
- With parallel attempts, often fakes IP addresses to make difficult to trace



Defending Against Credential Stuffing

- Leverage MFA (Multi-Factor Authentication)
- Use a CAPTCHA (though I hate them!)
- Gather details about user devices to create a “fingerprint” for incoming sessions – if same “fingerprint” is logged several times in sequence, block



Defending Against Credential Stuffing

- Leverage IP blacklisting
- Rate-limit non-residential traffic sources (like public Cloud)
- Block headless browsers based on JavaScript calls used
- Disallow e-mail addresses as user IDs



Defending Against Credential Stuffing

- Use Capital Group's primary Identity Provider (Okta) – includes built in credential stuffing protection and alerting
- Includes additional risk-based authentication that also considers device, location, request rate, etc.



Knowledge Check

Which attack best describes when an attacker uses a bot to login to multiple accounts in parallel?

- A. Credential stuffing
- B. Cross Site Scripting
- C. SQL Injection

Enter the letter corresponding to your answer in chat



Knowledge Check

In which attack does the attacker attempt to “clog” the network

- A. Credential stuffing
- B. Cross Site Scripting
- C. Large files

Enter the letter corresponding to your answer in chat

DevSecOps

What is it?

Discipline that takes into consideration how People, Processes, and Tools Automation combine to ensure we have:

- Disciplined build of security into all phases of SDLC
- Just-in-time security assessment and testing
- Security as a “shared responsibility”

DevSecOps

Why is it important?

To maintain a competitive advantage:

- We need to deliver software quickly
- We need that software to have high quality
- We need that software to be secure

We want to merge speed & agility with security & quality!

DevSecOps and how can businesses benefit from it?

“DevSecOps integrates software development with IT/SEC operations, leveraging **automation** to **shorten the product life-cycle** and **enable frequent deliveries** aligned with the business needs, while **ensuring quality**”

DevSecOps Building Blocks



Continuous integration

Automated testing and single source code mgmt. system



Continuous delivery

Ability to deliver software when ready



Continuous deployment

One-click release of software to live



Continuous monitoring and autonomic operations

Highly automated IT Operations, including security, vulnerability scanning



Standardized and automated infrastructure mgmt. with built in security

Automate environment with infrastructure-as-a-code (IaC),

Impact Levers



Speed

Release in days, not months



Quality

Enable speed and enhance stability with rigor and confidence



Efficiency

Operate in automated environment and focus on high value tasks

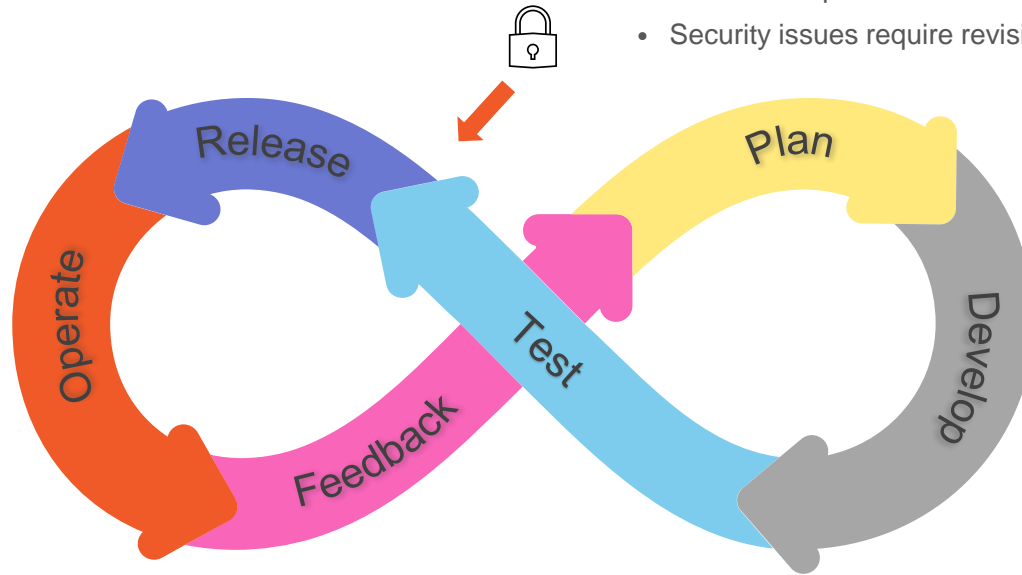


Security

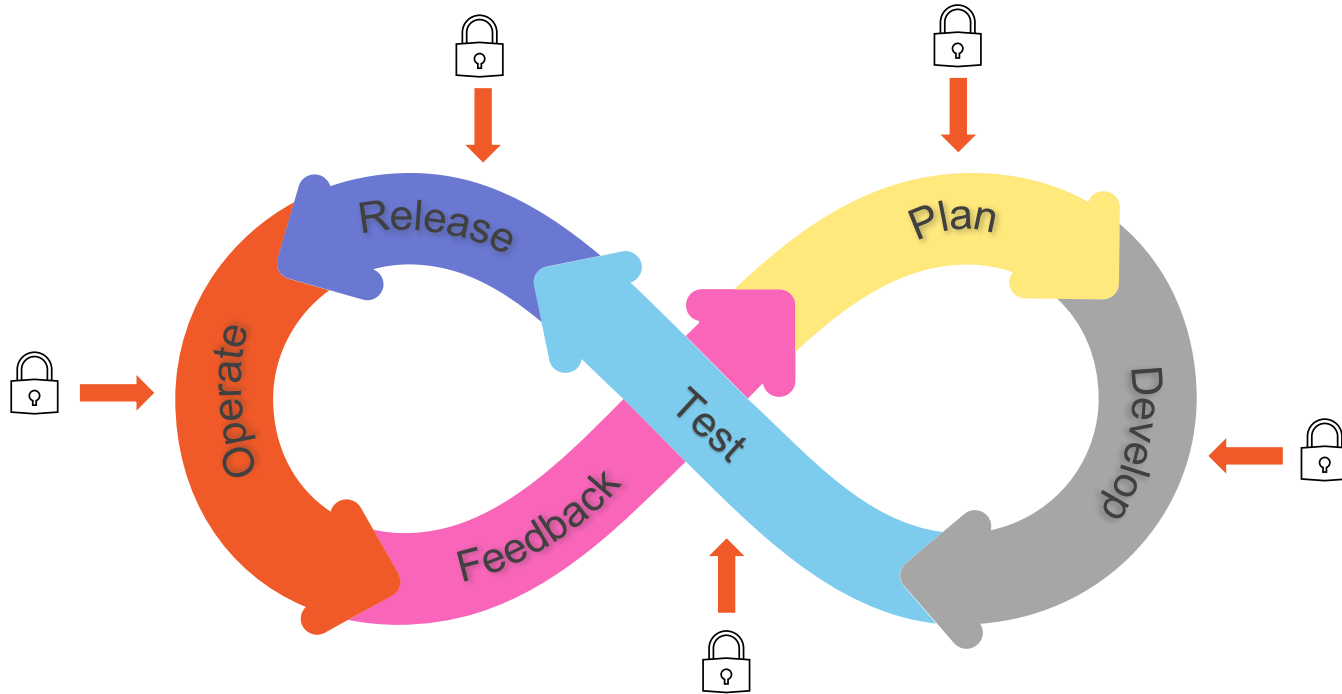
Baked-in and fully integrated into SDLC

Security in Software Engineering – The Traditional Way

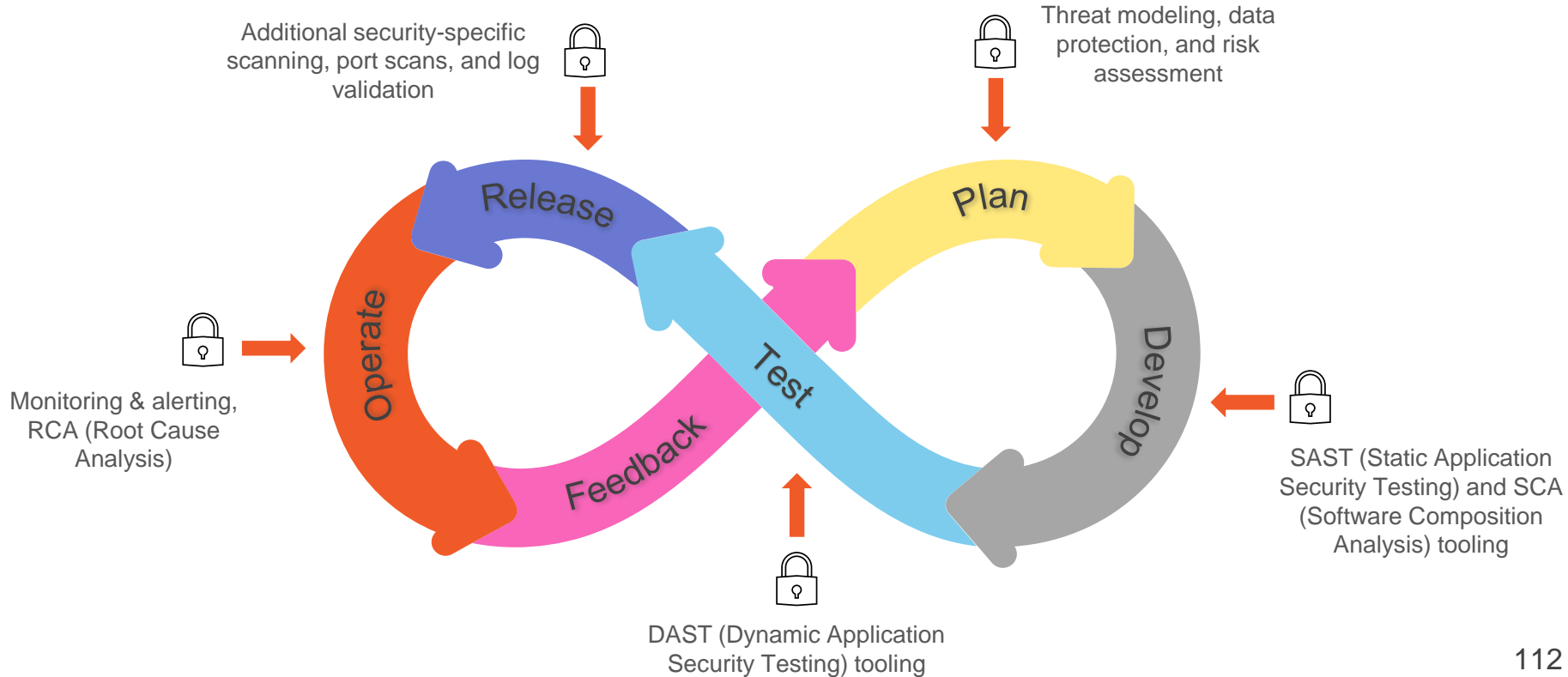
- Not enough time in schedule to absorb changes required to remediate
- Activities required to remediate may be complex
- Security issues require revisit of one or more previous phases



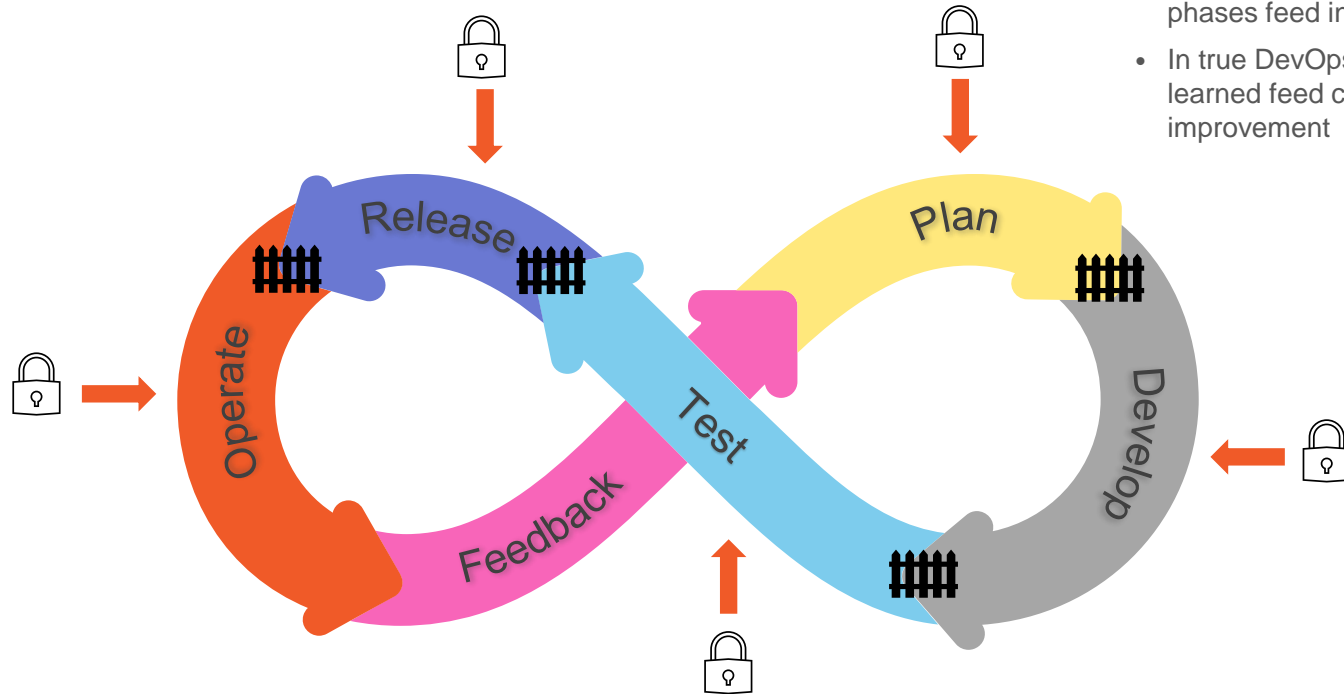
Security in Software Engineering – A Better Way



Security in Software Engineering – DevSecOps



Security in Software Engineering – DevSecOps



- Information gathered from early phases feed into later phases
- In true DevOps fashion, lessons learned feed continuous improvement



Quality gates guard against moving security defects forward



Assessing Security Risks (Plan Phase)

- To secure a solution, attack surfaces and potential threats must be identified
- Common practice utilizes something called threat modeling
- Includes modeling and analyzing possible attack vectors based on application

Assessing Security Risks

- Risk assessment should account for different “zones” of execution
- Security requirements must be understood in context of specific use cases
- Ideally, threat modeling would be executed during design & dev phases



Hardware

Software

Network

Database

Threat Modeling





Threat Modeling

Can be viewed in two different, but related, contexts:

- Implementation of controls mapped to security requirements & policy (prevention)
- Implementation of countermeasures against possible known attacks (remediation)

LAB:

Threat Modeling

Take a look at the telecommunications reference architecture described at <https://github.com/chanakaudaya/solution-architecture-patterns/blob/master/industry-specific/Telecommunication-reference-architecture-pattern.md#adding-value-to-your-subscriber-base>.

In your assigned breakout room, spend time discussing and documenting potential threats and attack vectors that you see in the defined architecture. Start by brainstorming as many potential attacks as you can think of. Then, as a team, organize by risk vs. impact.

Do not worry about being too technical in your descriptions - you are simply looking to think about and identify potential threats by type. Don't forget to include the following considerations:

- Hardware
- Software
- Data
- Network

Nominate someone from your breakout room to gather/document the results of the discussion. That person will present to the larger group when we come back together from our sub-teams.



Threat Modeling

Multiple approaches:

- Attacker-centric (think like an attacker!)
- Asset-centric (what do we have to lose?)
- Application-centric (what are we building & testing?)



Threat Modeling – Key Considerations

Valuable principles:

- Defense in Depth
- Principle of Least Privilege
- Secure by Default

Shifting Security Left – Develop



- Ideally, security flaws in code would be caught on a developer's machine
- Can use plugins for IDEs to provide automated static analysis
- In some cases, tools can be configured with custom rules and priorities

Shifting Security Left – Develop



- Rulesets and priorities should be driven by results of threat modeling
- Scanning tools if used should be a standard part of all developer's code tooling, to ensure consistently applied (standard dev build)
- Goal is to catch issues VERY early on in the SDLC

Moment of Reflection



What is the most “secure” programming language
in use in the industry today?

Type your answer in chat

Shifting Security Left – Build



- Changes from a developer are ready to be integrated with others
- Typically driven through submission of Pull Request in GitHub
- Notifies a peer that changes are pending and need review

Shifting Security Left – Build



- In addition to logic/syntax errors, peer reviewer can focus on areas of high risk/high impact identified in threat modeling
- In some cases, a specialist peer reviewer (e.g., one from InfoSec) can be engaged to validate security of code

Shifting Security Left – Build



- Coupled with manual review, SAST (Static Application Security Testing) tools can be integrated into CI/CD to automate security checks
- Tools like SonarCloud/SonarQube can be integrated into pipeline via plugins (<https://sonarcloud.io/>)
- Some even include ability to capture scan results in resulting build report and use results of scan as quality gate to prevent move forward on failure

Shifting Security Left – Build



- SCA (Software Composition Analysis) tools also provide benefit
- Tools like Nexus Sonatype (<https://www.sonatype.com/products/open-source-security-dependency-management>) can be integrated into CI/CD for automated security scans of Open-Source components
- Ideally, governance would be in place to monitor and manage “accepted” set and versions of Open-Source tools approved for usage

Shifting Security Left – Build



- SCA tools can provide a couple of benefits
- Primarily, security scans can be executed, and Open-Source vulnerabilities identified
- Secondly, can help an organization catalog the Open-Source components (and versions) “in play” already

Shifting Security Left – Build



- Resulting report can be used to identify components that have not been properly vetted through governance
- Results can be merged with overall build output
- Quality gates (manual or automated) can be built around results to prevent move forward if insecure

Moment of Reflection



Answer the following question: Using an open-source library is more secure than building something custom? Yes or no, and why.

Type your answer in chat

Shifting Security Left – Test



- Includes changes ready for move to QA for additional testing
- Dynamic Application Security Testing (DAST) tools provide more sophisticated vulnerability checks
- Dynamically exercise application's runtime interfaces using injected data

Shifting Security Left – Test



- DAST tools can use complex combinations of invalid input via fuzzing
- Provides multiple layers of protection
- A tool like OWASP Zed Proxy or OWASP ZAP is an example (<https://www.zaproxy.org/>)

Shifting Security Left – Test



- Two types – passive scan and active scan
- Passive scan less aggressive and minimizes use of fuzzing
- As a result, not as robust in its ability to find exposure but also takes less time to run

Shifting Security Left – Test



- Because of that, passive scan can be good fit for CI/CD pipelines, especially those that look to push updates at a greater frequency
- For the C (continuous) part, need flows to complete quickly

Shifting Security Left – Test



- Active scans are more aggressive in their attack approach
- Make extensive use of fuzzing to elevate level of attack sophistication
- Send multiple requests, continually altering request data to simulate attack payloads

Shifting Security Left – Test

- Can also make use of a technique called “spidering”
- Allows the scan to crawl a site or service to simulate sequenced or multi-level attacks (stateful attacks)
- Can be very effective in securing a site or service but also takes much longer to execute (due to added complexity and sophistication)



Shifting Security Left – Test



- Because of its more aggressive nature, should only be executed against owned sites and in a sandboxed environment (risky)
- Often active scans are scheduled to occur out-of-band on a regular basis (weekly, nightly, etc.)

Shifting Security Left – Test

- In addition to the automated scans, traditional functional test scripts and manual testing may be used
- Should include security-focused testing as well (or in separate PEN testing)
- Tests should be informed by prioritized set of vulnerabilities identified during threat modeling



Moment of Reflection



Which type of dynamic scan (passive or active) is more effective, and why?

Type your answer in chat

Shifting Security Left – Release

- Software confirmed ready for release
- Often deployed to a pre-prod environment (like staging)
- Can include automated “smoke” testing



Shifting Security Left – Release



- This phase may also include security-specific scans as a “last mile” protection
- SAST, SCA, and DAST scans may be repeated in this environment as a double-check
- Port scans can also be used (depending on how closely this env matches prod) to help validate exposure at network communication layer

Shifting Security Left – Release



- Additionally, this stage can utilize application logging as a type of validation
- Provides a unique opportunity to confirm correct sanitization of log detail
- Also, can include verification of specific AuthN/AuthZ controls prior to a prod release

Moment of Reflection



What other types of security checks (besides rerunning SAST, SCA, and DAST scans) are effective as part of the Release phase?

Type your answer in chat

Shifting Security Left – Operate



- In this stage, monitoring and alerting become paramount for capturing and notifying support of any security exposure “in the wild”
- System logs, telemetry, and event detail become useful
- Likely not everything will be found prior – ongoing vigilance is a must

Shifting Security Left – Operate



- When (not if) something happens, Root Cause Analysis (RCA) is important
- Helps with understanding the why
- Can also help with identifying short-term workarounds (to stop the bleeding) and long-term fixes (for enhancement prioritization)

Shifting Security Left – Operate



- Preceding detail can feed dashboards or reports to help stakeholders visualize the security posture of the system and org
- System logs can also be used to satisfy auditing requirements (depending on the industry)
- Key Performance Indicators (KPIs) can be used to measure

Shifting Security Left – Operate



Examples

- Number of security-related tickets
- Build or release delays caused by security alerts
- Mean time to compliance
- These can all help support the Continuous Improvement function in the area of security focus

Shifting Security Left – Operate



- One additional layer of protection that a company can employ involves security bug “bounty hunters”
- Ethical hackers that can be paid to independently attempt to attack a site or system
- Can result in a set of reports on uncovered vulnerabilities and recommended remediations
- A SIEM (Security Information and Event Management) solution can also help with detection, analysis, and response

Moment of Reflection



What is one Key Performance Indicator (KPI) or metric you would recommend using (or have used) to evaluate the security posture of a system running in production?

Type your answer in chat

Other Key Considerations - SAST

- Be aware that SAST tools have the propensity for false positives (depending on tooling employed)
- Tools may err on the side of caution with security scans
- Might require that an organization customize the tool to more accurately report on environment and application
- Otherwise, teams run the risk of wasting time on non-value add activities, impeding “frictionless security”



Other Key Considerations - Containers



- If application profile includes containers or container orchestration (k8s), additional consideration required relative to security of images
- Tools like Aqua (<https://www.aquasec.com/>) and Prisma Cloud (<https://docs.paloaltonetworks.com/prisma/prisma-cloud/>) can help with scanning of container images

Other Key Considerations – Sensitive Config



- Sensitive configuration data and app secrets (e.g., connection strings, passwords, etc.) should never be hardcoded in source
- Run the risk of checking into and exposing via source control
- Tools like truffleHog (<https://github.com/dxa4481/truffleHog/>) can help with GitHub repo scanning

Other Key Considerations – Sensitive Config

- Other platforms include built-in support
- Services like AWS KMS or Azure Key Vault can provide dynamic linking of secure config into the CI/CD pipeline



DevSecOps

Potential Benefits



- Security “baked” into the process end-to-end
- Promotes secure by design and defense in depth
- Increased efficiency in verifying security through automation
- Faster recovery times if security breach occurs
- Enables measurement and objective assessment of security posture

DevSecOps

Potential Benefits



- Leveraging established Capital Group best practices, existing reference architectures, and existing patterns helps minimize risk
- Also, helps ensure you meet critical security requirements in a “low friction” manner
- Means as a developer you don’t have to try to mitigate all threats on your own – there are resources available to assist

DevSecOps

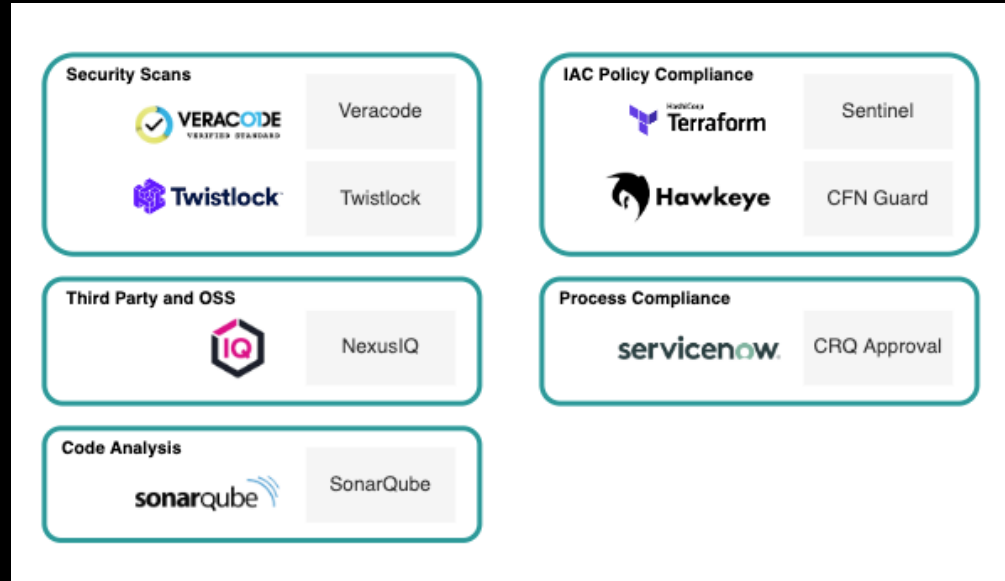
Potential Issues



- Knowledge gaps about security, a complex subject
- Additional complexity and friction in pipeline
- Potential for developer overload
- False positives breed “noise” and can lead to “boy who cried wolf”

LAB:

Review the Capital
Group DevSecOps
Toolset



Google each of the tools above used at Capital Group. Create a 2 – 3 sentence description for each based on the results of your research.

Where Can I Go For Help?

- At Capital Group, there are security architecture resources available to assist engineers
- Dedicated team is available to oversee and assist with threat modeling
- Given the complex nature of getting security “right”, organizational resources like there can prove invaluable

Introduction to Test-Driven Development (TDD)

Benefits to be experienced with automated testing



Greater shared success
(as tests can be tied to
requirements)



Cleaner code



Guaranteed system
documentation



Improved quality



Reduced fear



Improved regression
protection

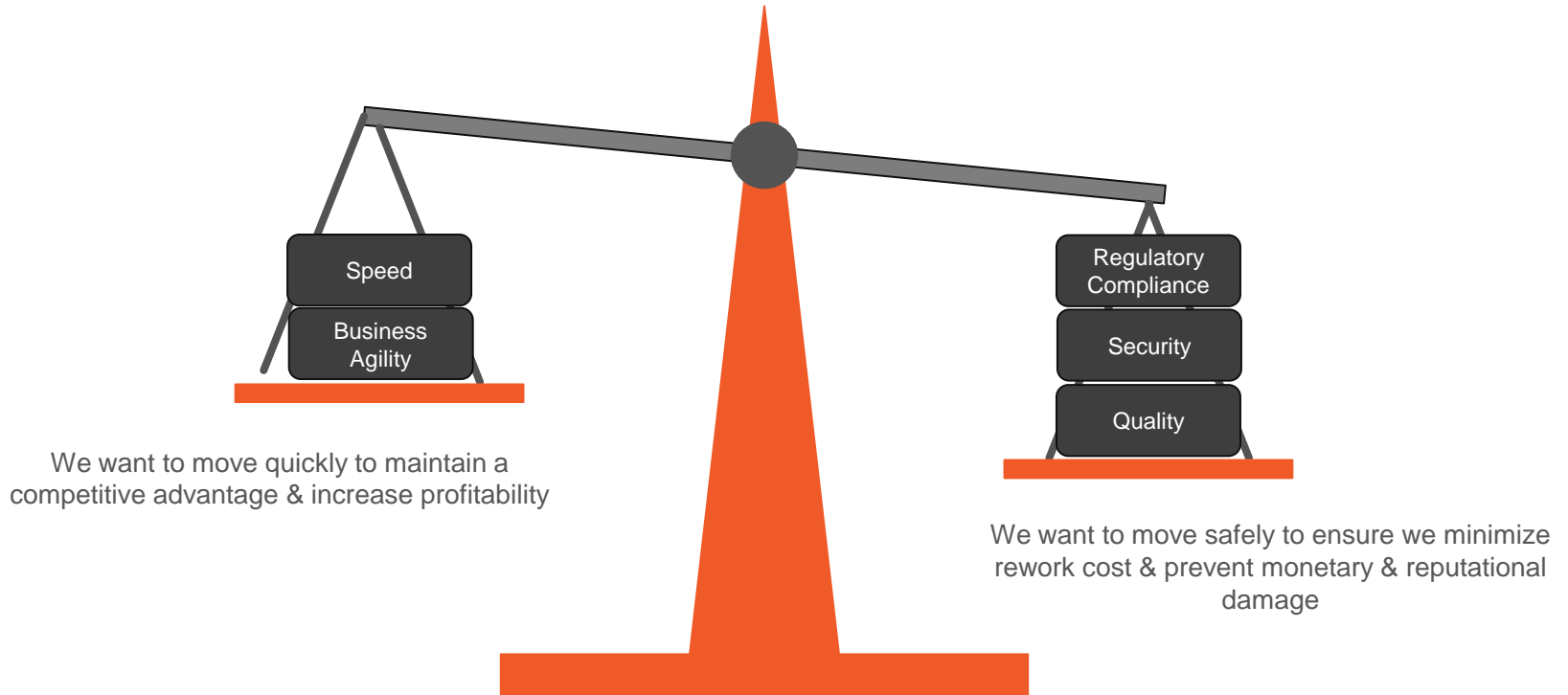


Greater streamlining to the
approach
(small bites, avoidance of
premature optimization,
structured flow)



Discussion | Which of these benefits resonate most with you?

Balancing Speed & Quality





What if we wrote our tests *first*?

Test Driven Development reverses the usual

Devs must devise a test that will verify correct operation of a function *before they even write the function*

This forces you to write functions that “do one thing, and do it well”



Test Driven Development – basic sequence

Write a unit test for the function you're about to create

Run the test – it should fail

Write the function

Test it

If it fails, re-examine your test and change it if your logic was wrong

Otherwise, refactor your function until it passes

Write another test - repeat

Test-Driven Development (TDD) and the 3 Laws

1

The First Law

“You are not allowed to write any production code until you have written a unit test that fails due to its absence”

2

The Second Law

“You are not allowed to write more of a unit test than is sufficient to fail, and failing to compile is failing”

3

The Third Law

“You are not allowed to write more production code than is sufficient to cause the currently failing test to pass”

Understanding TDD

Scenario: Determine discount based on purchase volume



- SaaS module is external service defining and enforcing business rules
- Several levels of discount based on purchase volume ranges
- Need to verify that the invoicing module operates correctly with the different levels of discount
- What if the call to the SaaS module fails?



- Tie test cases directly to specific business requirements (e.g., each discount level)
- Write **just enough** code to pass the tests
- By extension, have written **just enough** code to meet the business requirements
- **Mocking** enables tight control over what gets returned from the downstream dependency, including any error conditions
- Result is verified quality when built and **automated regression testing** against future changes

Best practices with TDD

TDD supports the following best practices:



Don't commit to a methodology, commit to impact and a positive result



Ensure that the engineering team has a clear understanding (and practice) for unit vs. integration testing



Automated test code is code – just like application code – estimate and plan accordingly



Find the right % of test coverage – quality & quantity!

To be effective, automated unit tests need to follow FIRST:

F

Fast

or they won't be
executed

I

Independent

i.e., minimal
dependencies
between tests (keeps
them simpler)

R

Repeatable

i.e., every run against
unchanged code
should operate the
same

S

Self-validating

i.e., Boolean output
(pass or fail)

T

Timely

if not first, early and
often



Challenges

What are some of the challenges you face with software testing in your jobs (i.e., why don't we test as much as we want to or should)?



Challenges

- Time pressures
- Complexity
- Requires more code
- More debugging (up front)
- More cost (up front)
- “Can’t we just ensure quality with other types of testing?”



Knowledge Check

Which of the following best describes TDD?

- A. Write tests before writing code
- B. Write higher quality tests
- C. Write more tests

Enter the letter corresponding to your answer in chat



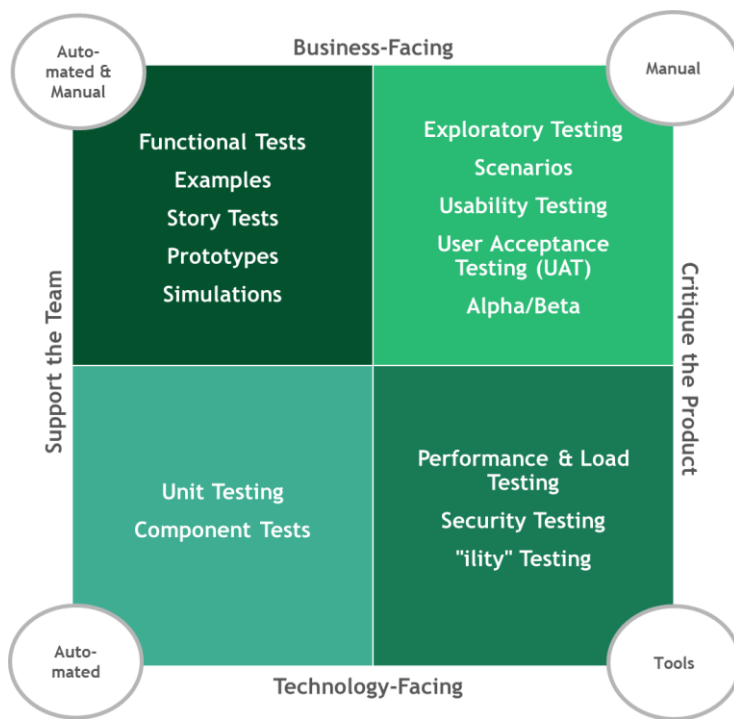
Knowledge Check

Why do we first write a failing test in TDD? Why not a passing test instead?

Type your answers in chat

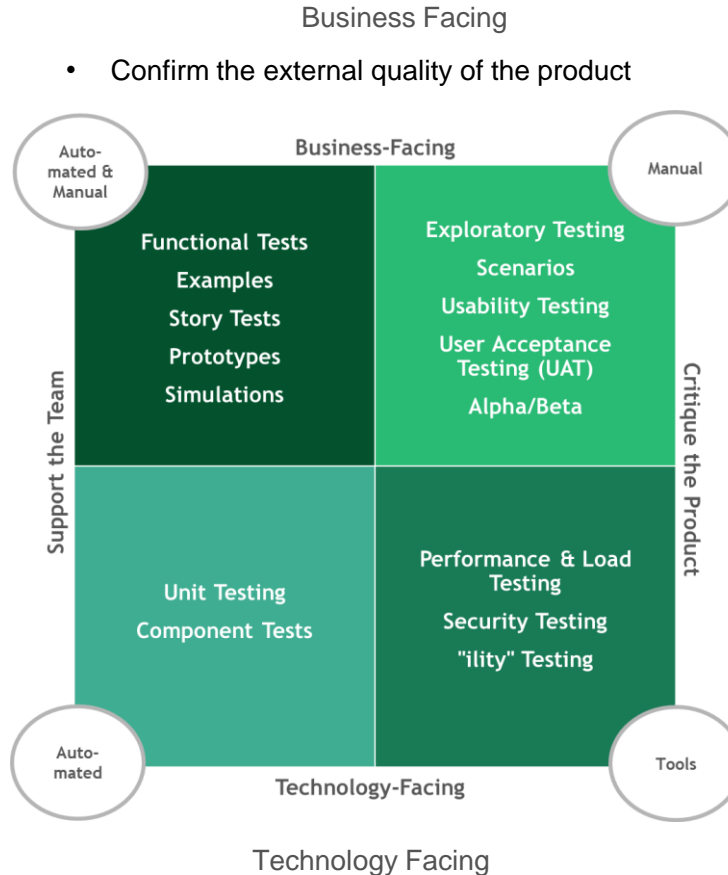
Testing Strategies

Using the Agile Testing Quadrant



Testing Quadrant

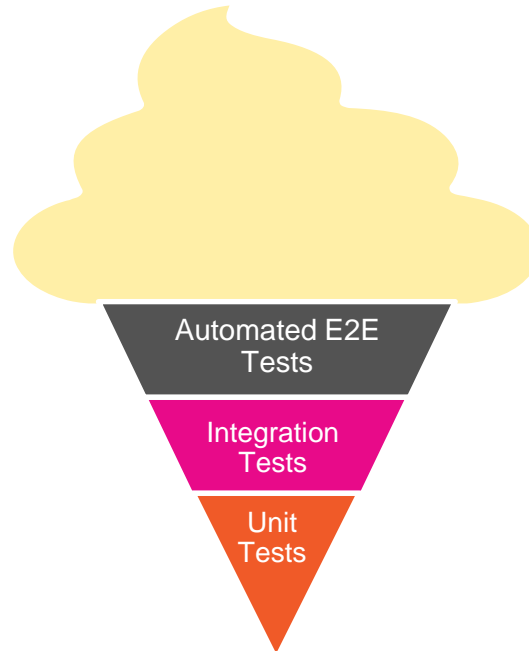
- Support the Team
- Focused on supporting the teams that underpin the product & its delivery
 - Goal is to help guide the quality of the product & prevent functional defects from making their way into production



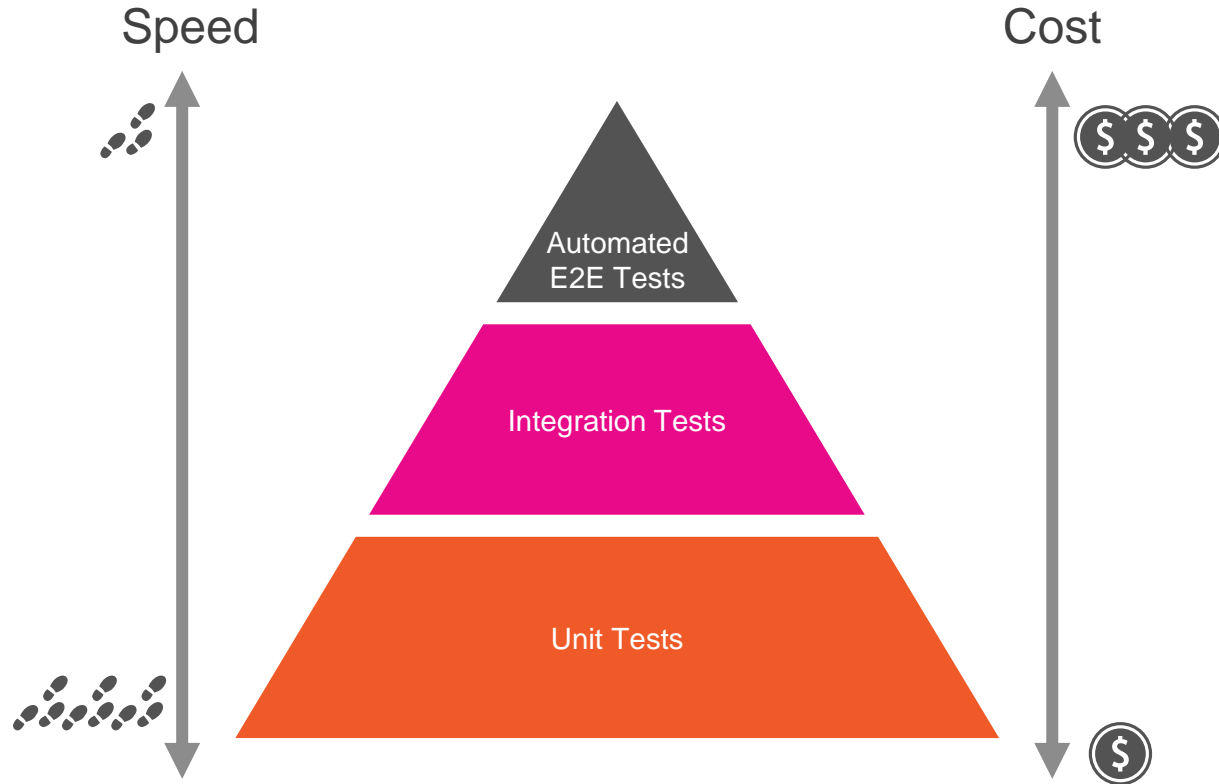
- Critique the Product
- Focused on evaluating overall product quality
 - How will software that meets the functional requirements perform (on multiple levels) when deployed to production?
 - Ensuring that Quality of Service (QoS) are met

- Maintain the internal quality of the application

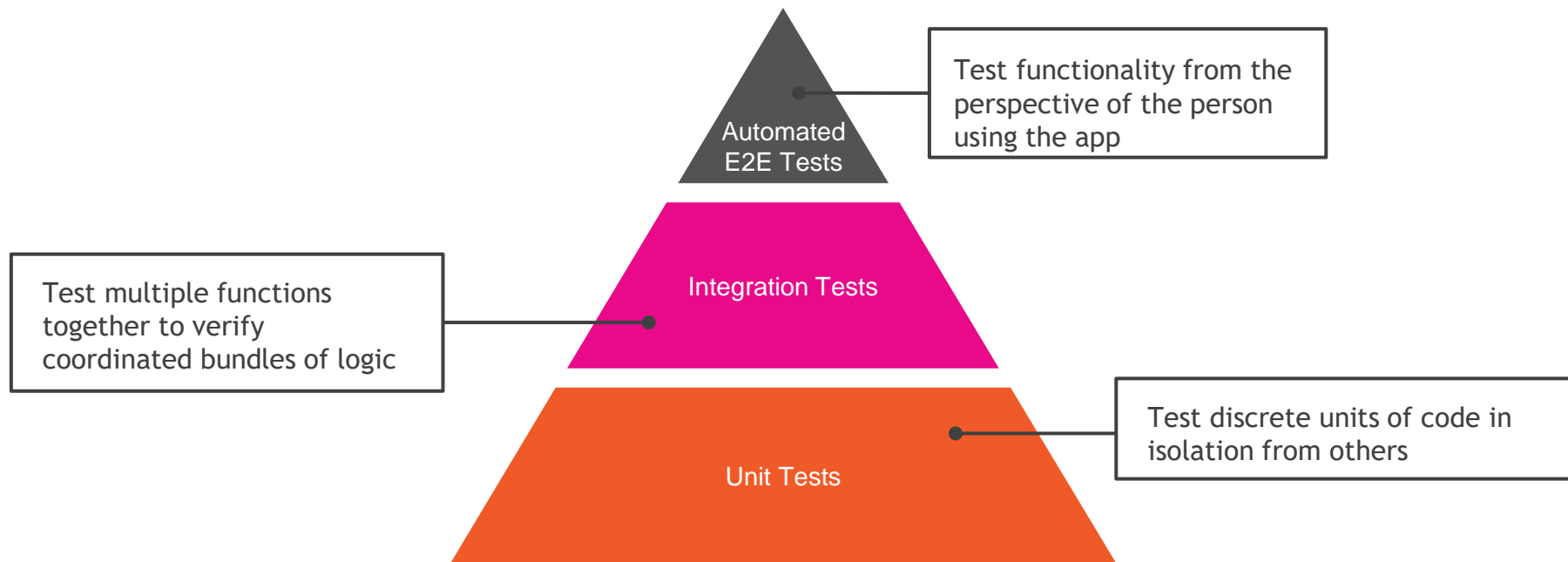
At times, without vigilance, we may find that our software testing resembles an ice cream cone rather than a pyramid



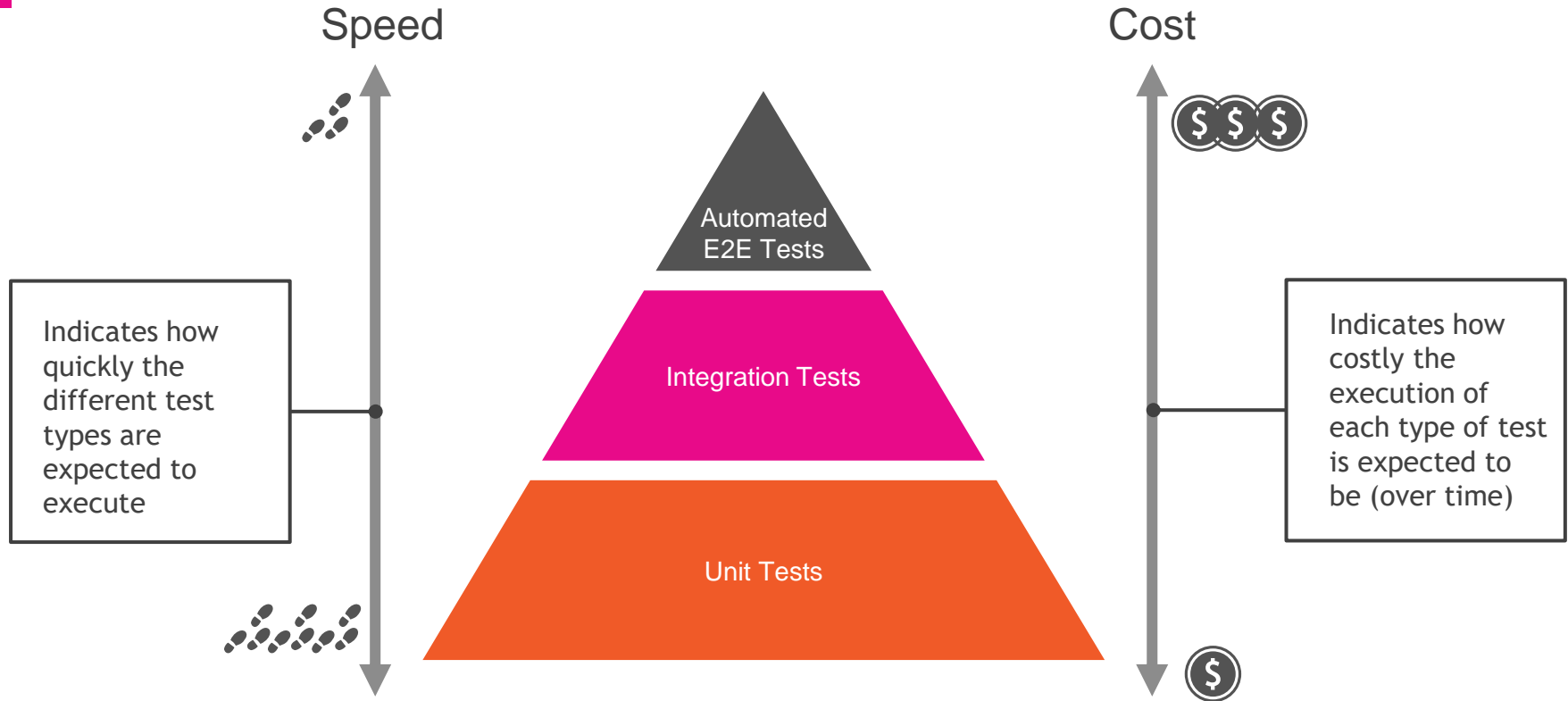
Test Pyramid



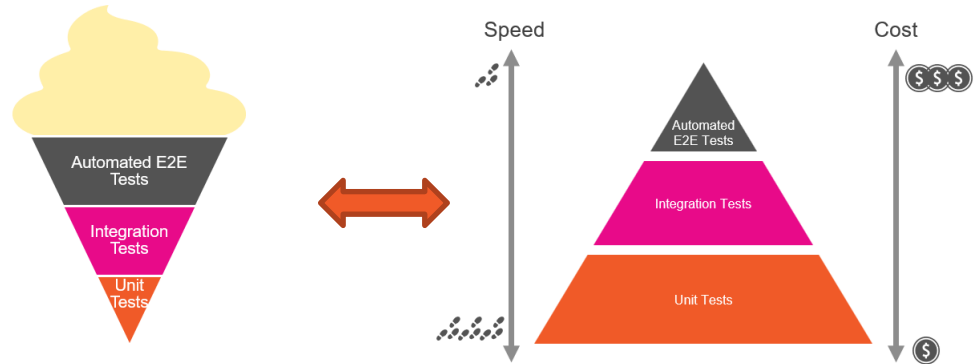
Test Pyramid



Test Pyramid



Reflection



1. What are some of the root causes that you think could cause a shift from the pyramid to the ice cream cone?
1. During project execution, what are some specific warning signs we can look for that will alert us to a trend towards that shift?
1. What are some practical steps we can take to move from the ice cream cone to a proper testing pyramid?

Managing Software Quality



Using Test Coverage to evaluate the completeness & effectiveness of our tests

The What

- Measurement mechanisms are built into many languages & development frameworks
- These tools analyze source code under test against automated test runs, identifying source code that is covered (exercised by one or more tests) vs. not covered (excluded from any test)

The How

- Ranges from 0 - 100%, and the higher the number, the better the coverage
- A low test coverage percentage indicates a deficiency in automated testing
- Test coverage measures are often integrated into CI/CD pipelines, failing the build if a minimum level of coverage has not been achieved



Wrangling “Technical Debt”

- Involves borrowing from the future to address the immediate
- Manifests as choosing to go with a sub-optimal solution to move a project forward, instead of spending the time to build out the optimal approach
- Results in higher compounding costs in the long-run (e.g., future development cost, long-term maintainability, speed, quality, etc.)



20-30%

of sprint capacity, ideally, should be allocated for bug fixes, automation improvement, POCs, etc.

Tactics to mitigate technical debt:

1. Find opportunities to pay down debt as part of a future enhancement
1. Remove debt when modernizing a legacy system – don’t simply do a “balance transfer” when building the replacement
1. Embrace pay down of technical debt as a regularly-planned activity within your sprints (if possible)



End-to-End (E2E) Automated Tests

- Everyone owns quality – quality from start to finish
- Integration testing – system level, point-to-point interface testing
- Testing across multiple products critical as well
- At Capital Group, there are hundreds of systems – any large program change can potentially impact 20 – 50 investment systems
- Critical that this be automated to verify quality of external/customer user experience across our systems

A thick yellow L-shaped line that starts at the top right, extends left, then turns down and extends left again, framing the title.

Unit Testing Frameworks



Purpose of a unit testing framework

- Help you write and run unit test

- Provide test reporting

- Manage collections of unit tests

- Implement testing strategies that vary according to testing needs



Popular unit testing frameworks for Python

PyTest

Robot

Unittest (we will use this for our work)

DocTest

Nose2



Where does a unit test come in the development cycle?

VERY IMPORTANT FACT: Unit tests are NOT executed on a running instance of your application. They are run using ONLY your source code

This means they can be run before you use valuable resources to build and deploy your app, catching problems early

A thick yellow L-shaped line that starts at the top right, goes left, then turns down and goes left again, framing the title.

Writing Unit Tests in UnitTest



unittest Module

- Included in Python standard library
- Provides a framework for building and running unit tests against Python code (functions, classes, etc.)
- A Python file houses code that will be used to automate the testing



unittest Module

- Test code imports unittest and module under test
- Test code includes a Python class that inherits from `unittest.TestCase`
- We write Python methods to exercise our business logic and verify expected outputs



unittest Module

- unittest includes multiple “assert” methods for checking different kinds of results (expected vs. actual)
- Can exercise tests through unittest framework
- Results will indicate success (or failure)

unittest Module

Options for executing tests

- *python -m unittest module_name*
- *python -m unittest module_name.TestClass*
- *python -m unittest module_name.TestClass.test_method*
- *python -m unittest path/to/test_file.py*

Can also use “discover” mode by running *python -m unittest*



unittest Module

- If test run fails, could be for 1 of a couple of reasons:
 - Code under test is incorrect
 - Test code is incorrect
- Verify the expected test results against your requirements
- If the expected result aligns, then fix the code under test



unittest Module

Assert methods include:

- assertEquals/assertNotEqual
- assertTrue/assertFalse
- assertIn/assertNotIn

<https://docs.python.org/3/library/unittest.html#assert-methods>



unittest Module – Setup and Teardown

Rather than repeat common logic across tests, unittest provides a couple of options for capturing:

- setUp
- tearDown
- Includes a couple of forms



unittest Module – Setup and Teardown

At the individual test level:

- setUp() – logic executed before each test is run; provides chance to setup common conditions
- tearDown() – logic executed after each test is run; provides chance to clean up (keep tests isolated)



unittest Module – Setup and Teardown

At the TestClass level:

- setUpClass() – logic executed before any tests in a class are run; requires decoration as a class method
- tearDownClass() – logic executed after all tests in a class are run; requires decoration as a class method



unittest Module – Skipping Tests

While permanently skipping tests can be a “smell”, unittest offers options for temporarily skipping:

- `@unittest.skip()` decorator
- `@unittest.skipIf()` decorator
- `@unittest.skipUnless()` decorator
- `self.skipTest` method – call within test to initiate a skip



Mocking – Testing Components in Isolation

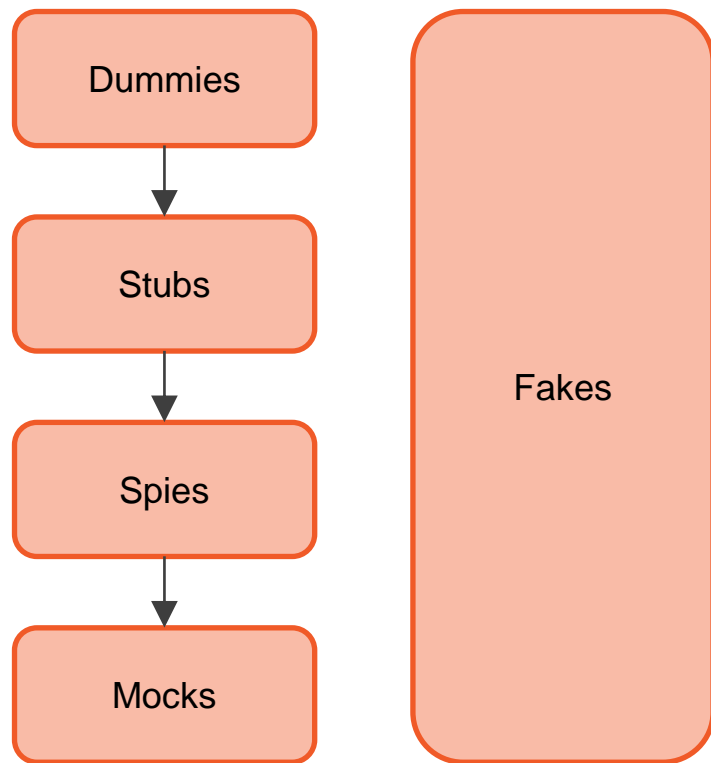


Mocking / Test Doubles

We use mocking/test doubles:

- To help isolate our testing to focused areas of code
- To manage dependencies and help control testing against them
- To force special cases that exercise specific code flows
- To bring stability and consistency to our tests
- Helps make the “unit” in unit testing possible

Mocking / Test Doubles





Dummies

- Simplest form of test double
- No real functionality – mostly a placeholder
- When dependency exists but is not needed for the test
- Create and use an empty or straightforward copy
- Returns meaningless value on every call



Stubs

- Same response regardless of parameters passed
- Used to test different paths through code
- Allows you to control the value of the dependency
- Can also use to force an exception
- Create a copy of the dependency and return same value



Spies

- Can return a value like a stub
- Also provides info about how called
 - Verify member call with specific parameter values
 - Verify member call a specific number of times
 - Verify whether member called or not
- Helpful with testing at third party boundaries
- Helpful with managing side effects of dependencies



Mocks

- Programmable spies
- Can return a set value like a spy
- Difference is ability to setup test data in the double
- Provides additional flexibility - can code the setup
- Still only returns set value but it's the value coded
- Trade-off – requires setup work



Fakes

- Most powerful type of test double
- Acts like the class/component it's doubling for
- Double that's more sophisticated (but still controlled)
- Useful for managing downstream dependencies
 - Gating database connections
 - Gating web service calls
 - File operations
- Can appear to integrate but control results



Coverage



Coverage

- Coverage in unit testing measures the % of your code under test that is exercised by a unit test
- Coverage value can range between 0 and 100 %
- Question: What is the right target % of coverage to aim for?



Coverage

- `coverage.py` can be used to measure test coverage in your Python app
- Install using pip (*`pip install coverage`*)
- Use *`coverage run`* to measure
- For unittest, use *`coverage run -m unittest <target>`*



Coverage – Measuring Results

- Use *coverage report* to report coverage results
- Alternatively, *coverage.html* can be used to view coverage results in a more user-friendly manner

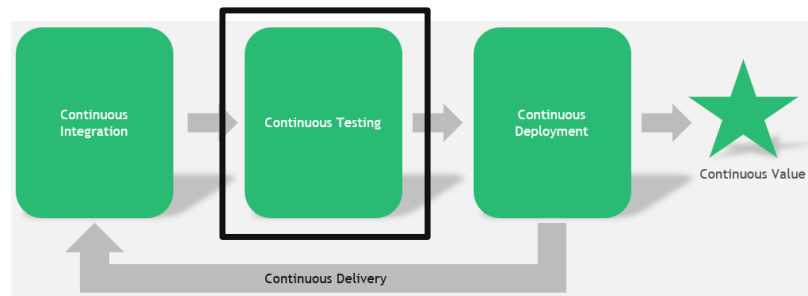


unittest, Mocking, and coverage - Demo



unittest, Mocking, and coverage - Lab

Continuous Testing



Common Types	Description
Unit testing	Testing discrete units of code in isolation from other code (e.g., testing a single method). Leverages mocking to stabilize other system dependencies so that tests are repeatable. Code coverage often used to assess quality & sufficiency of unit testing. Should be automated.
Regression testing	Testing that verifies that newly introduced changes for development of a feature do not inadvertently break other, existing features. Automated unit tests can also fill the role of a regression test suite nicely. Quickly confirming no breakages.
Integration testing	Testing that verifies that groups of components & component features operate correctly when combined with other features (e.g., multiple functions or modules that provide a larger “chunk” of functionality). Usually represents a layer above unit testing but below functional testing. Should be automated.
Functional testing	Testing that verifies functional requirements for critical workflows in the software end-to-end. Can be (and probably should be) implemented as automated black box tests, requiring minimal knowledge & minimal assumptions about the inner workings of the software.
Acceptance testing	Testing that verifies the functionality of the software against specific acceptance criteria defining the difference between “good” & “bad”. May include manual testing/utilization by a subset of testers to verify that the software will operate correctly when leveraged by end users in production. AKA does the software do what it is supposed to from the end user’s perspective?
Security testing	Testing that verifies the functionality of the software against critical security requirements as identified & prioritized via threat modeling. Through a combination of static & dynamic testing, assesses the software for security vulnerabilities or deficiencies from a compliance & regulatory perspective (A.K.A. DevSecOps). Examples include SQL Injection and Cross-Site Scripting.
Performance testing	Testing that verifies that the software will meet defined SLAs (Service Level Agreements) when exercised under load or stress. Can also be used to validate the software’s ability to elastically scale (out or in) based on volumes. How does the software perform under load?

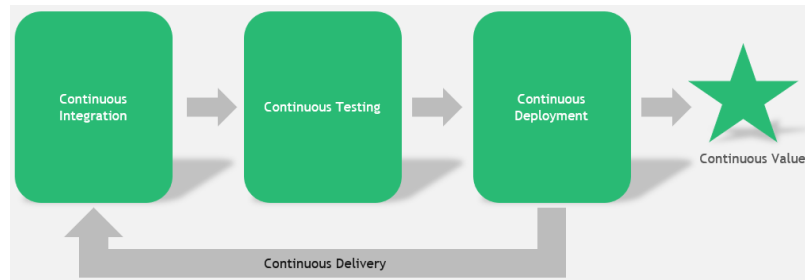
“Shift Left” Testing Approach - Redux

DevOps (and its CI/CD automation tools) are all about creating trust – in the tools and processes, and in the code itself. Key to this is testing.

A popular movement in DevOps is called “shift left” testing. It is an approach to software testing and system testing in which testing is performed earlier in the lifecycle (i.e., moved left on the project timeline)

This means developers are getting involved more in testing.

A common tradeoff to examine here: How long does a full set of tests take? How often are devs checking code into the branch that is tested? Do the tests take so long that code changes are getting backlogged? How to we prioritize tests in this situation?



Distributed Version Control and Git



What is Version Control?

Programs AND work practices that we use to track and control text documents

What documents? Source code - the text files we use to create our executable program

Helps facilitate multiple contributors on a project

Allows us to revert to previous iterations of code base as needed



The Past: Client/Server Version Control

Prior to 10-15 years ago, most version control used client/server model:

One computer (version control server) had all source code files; it was usually on-site

Client computers (devs) could “check out” a file with the server – that would lock the file to changes

When dev done with work, submits change to server

Once change is made, the file is unlocked for others to modify

Disadvantage: geographically distributed teams weren't possible; multiple devs can't work on the same file



The Present: Distributed Version Control

The source code files are distributed – a full copy of all files, and their entire history, exists on every developers' computer

Multiple devs can work on the same file simultaneously – when they merge, a computer program will handle routine merge actions; merge conflicts are raised for human interaction

There are a few popular programs:

- Subversion
- Mercurial
- Git

Git is by far the most popular

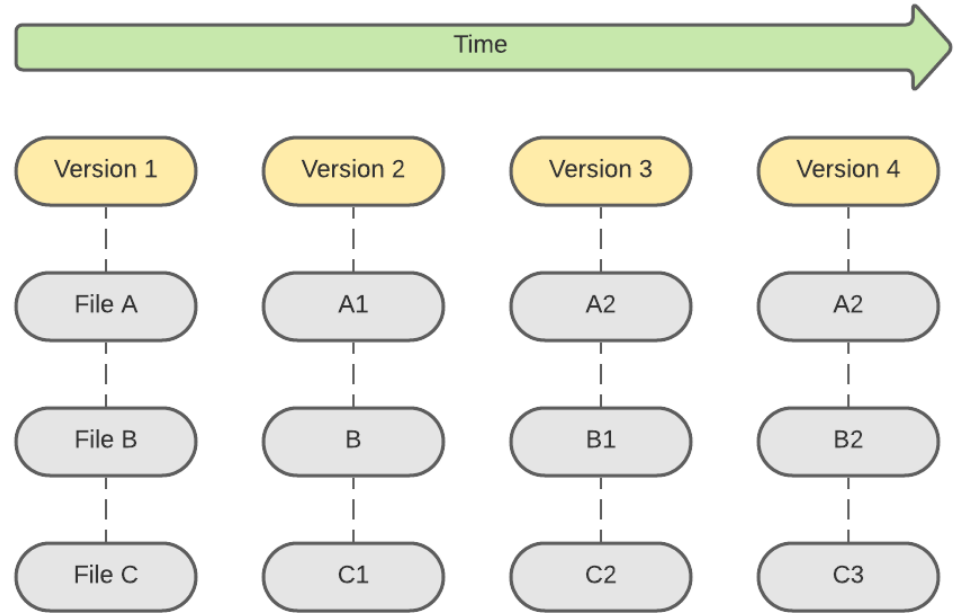
Git

What is it?

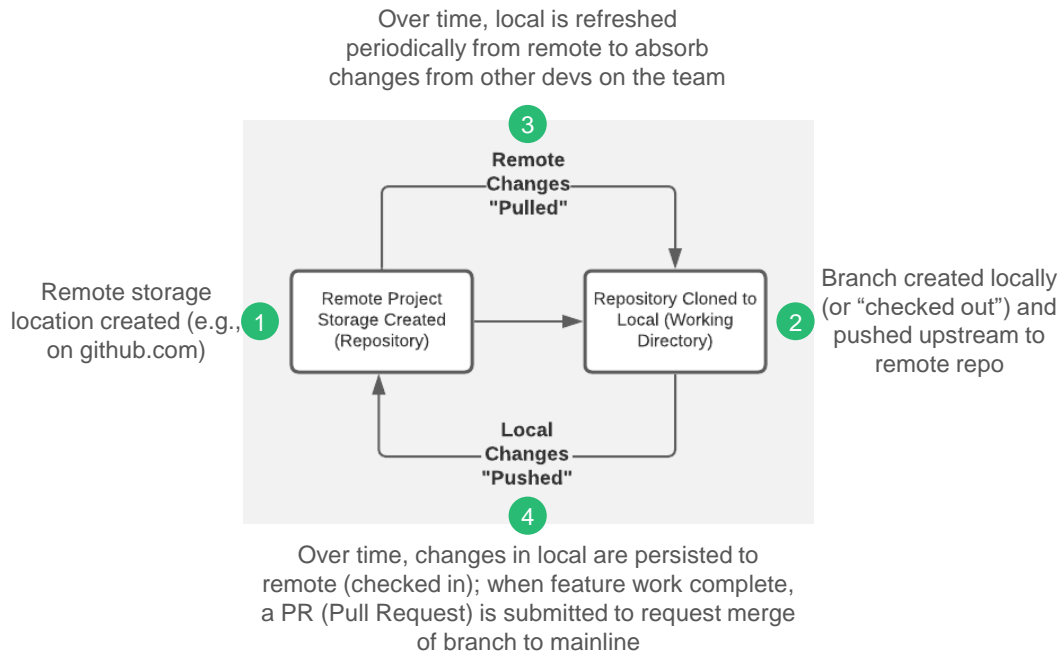
- Open-source distributed version control system
- Takes “snapshots” of an application folder as changes occur
- Includes both a local and remote component
- Changes are local and are synced with remote as needed
- Keeps a log of history of changes and allows return to previous version if required
- Available at <https://git-scm.com/> for download & installation

Check-ins Over Time in Git

- Other version control systems track a running list of differences in each file over time
- Git keeps snapshots of the entire folder at each check-in over time
- Uses checksums to verify that contents of snapshot not tampered with

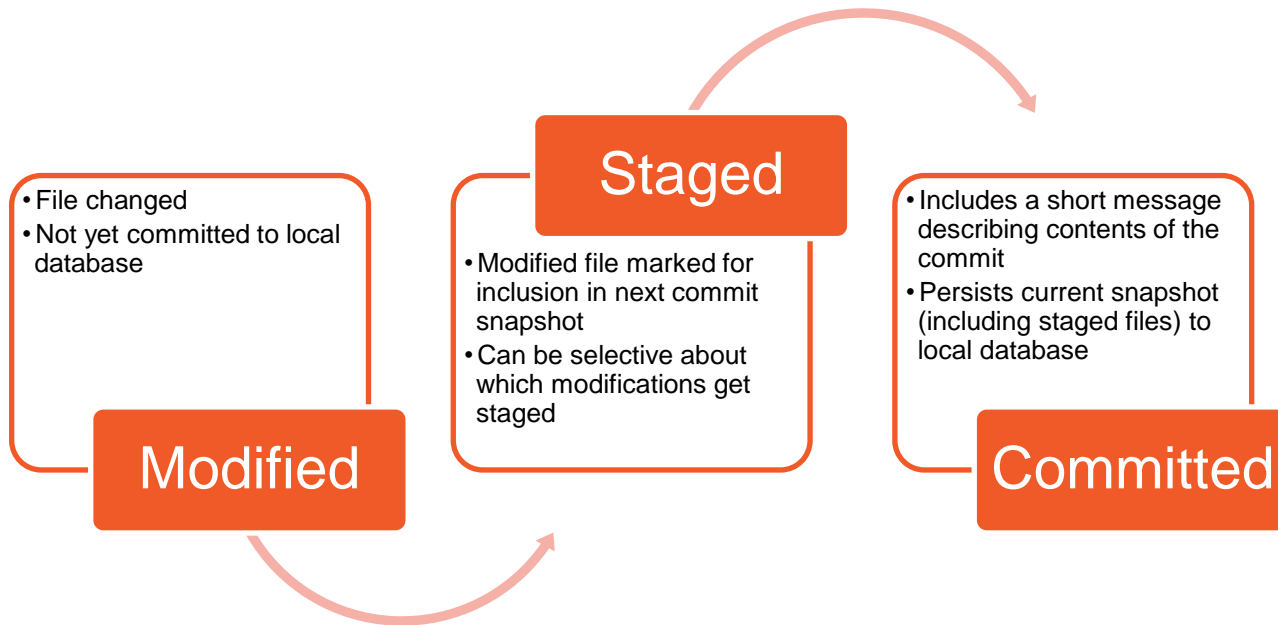


Typical Workflow



NOTE: Also, able to initiate repo creation from local command-line

The Three Git States





Demo

Configuring Git

- *git --version* to confirm Git is installed and to check version
- *git config --global user.name "<username>"* to set username
- *git config --global user.email "<email>"* to set user email address
- File changes will be identified in Git history using this information

Configuring Git

- Use *git config --list* to see all configuration settings
- Use *git config <setting name>* to view a specific setting
- For example, *git config user.email*

Create a Repository

- Public or private repository can be created with hosting service (e.g., GitHub)
- To link with local snapshot of remote repo, have a couple of options:
 - Initialize local from command line, add files locally, add/push to remote
 - Use URL from remote to pull to local via clone

Push to Remote

- Create and/or navigate to target folder at local
- With an existing repo at hosting provider (e.g., GitHub), execute the following:
 - `echo "# <initial README.md text>" >> README.md`
 - `git init`
 - `git add README.md`
 - `git commit -m "<commit message>"`
 - `git remote add origin <HTTPS URL to repository>`
 - `git push -u origin main`

README.md

- `echo "# <project title>" >> README.md` creates new README file
- Usually, a Git repo will include a “markdown” file with info about the repo
- Markdown syntax is used to apply formatting characters to raw text

README.md

- When displayed in “markdown” reader, formatting is applied
- For example, “# “ in the echo statement formats text as a header
- One great option for previewing the markdown is using the “Open Preview” feature in Visual Studio Code
- Cheat sheet for markdown syntax at <https://www.markdownguide.org/cheat-sheet>

git init

- *git init* initializes local repo using configuration template
- After executing, folder recognized as local Git repository

git add

- *git add README.md* “stages” the README.md file for commit in Git
- Reminder – files in Git can be in 1 of 3 stages:
 - Modified – file changed but not yet staged or committed
 - Staged – file ready for commit
 - Committed – file changes persisted to local repo; ready for potential push to remote

git commit

- *git commit -m "<commit message>"* persists staged files to local repo
- -m flag used to specify message to attach to current commit snapshot
- At this stage, file changes are persisted locally but not yet remotely

git remote add

- *git remote add origin <HTTPS URL to repository>* used to add remote
- Configures link between local and remote at hosting provider
- If target remote is a private repo, will require additional security
- Can use a personal access token (with GitHub)

git push

- *git push -u origin main* used to persist branch and changes to remote
- -u used to specify “upstream” (or remote) branch target
- origin is reference name for remote repo
- *main* is name of branch to push
- Again, if private repo, will require additional security to complete

git clone

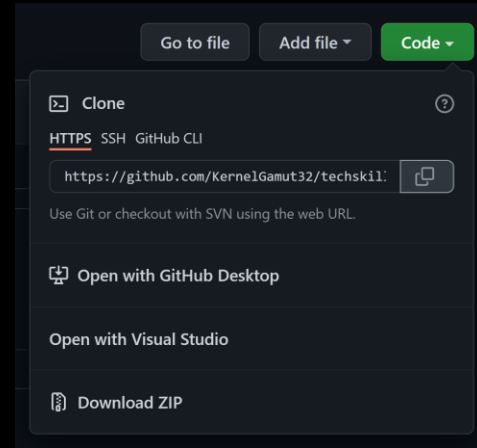
- For initial pull of repository from remote (e.g., GitHub) to local, use *git clone*
- Create and/or navigate to folder where new repo folder should exist
- Format is *git clone <HTTPS URL to repository>*

git clone

- If repo is private, user will need to be granted access
- Public repos can be cloned without additional access being granted
- Upon completion, new folder will contain local snapshot of remote repo

git clone

- On repo “home page” in GitHub, clone repository URL can be found/copied using the “Code” dropdown



git pull

- Local repository can be refreshed from remote (as required)
- Canonical use case is updating local version of shared branch with others' changes
- With branch checked out locally, use *git pull origin <branch name>* or *git pull* (which will default to currently checked out branch name)



Lab

Branching in Git

- Branches provide segregated snapshots of the codebase for parallel workstreams
- Often, branches created from mainline for development of a specific feature
- Structured process used to merge branches back into mainline

Branching in Git

- Use `git checkout -b <branch name>` in local working directory to create new branch
- Use `git push -u origin <branch name>` to push branch to remote for tracking
- `git branch --list` used to list existing branches

Branching in Git

- *git checkout <branch name>* used to switch to different local branch
- *git fetch* used to refresh local list with list of newly added remote branches

Making Code Changes

- Checkout target branch for development
- Make necessary code changes – including adds, updates or deletes
- Use *git status* to see the state of files in the current branch

Making Code Changes

- Use *git add* to stage changes for push to remote
- Either *git add <filename>* or *git add -A* (for all updates in directory)
- Use *git status* to check status of modified vs. staged artifacts

Making Code Changes

- To unstage 1 or more files, use *git restore --staged <file or wildcard pattern>*
- *git restore --staged .* used to unstage current directory
- Use *git add* to restage

Pushing to Remote

- Committed files are pushed to remote using *git push*
- Format is *git push origin <branch name>* or *git push* (to push to currently checked out branch in remote)
- User will need permissions on repo to push
- May be prompted for credentials with hosting provider to complete

Pushing to Remote

- To prep staged files for push to remote, use *git commit*
- Format is *git commit -m <commit message>*
- *git status* will reflect status change

Pull Requests

- When feature complete and ready for merge, engineer submits pull request (PR)
- Allows peer review of proposed changes
- Usually governed by policies around number of required reviewers, successful build in branch before merge, etc.



Lab

Merging in Git

- Best practice is to favor multiple smaller PRs over a few larger ones
- Helps reduce review risk and chances for conflicts
- Also, helps to minimize “drift” between in-progress work and mainline

Merging in Git

- The larger the PR, the longer it can stay open
- Other features are being merged into mainline while development continues
- If Git is unable to automatically merge and resolve multiple changes to the same file, results in a conflict

Merging in Git

- Good practice to regularly refresh branch with pull from mainline
- Helps to keep branch in sync with mainline
- Allows developer to find and fix merge conflicts in the branch before submitting PR

Managing Merge

- With target branch checked out, use *git merge* to pull in updates
- For example:

git checkout branch-name

git merge origin main

Managing Merge

- Any merge conflicts will be called out on the command-line
- Use *git status* and *git diff* to list merge conflict details
- Manually update files to resolve and push changes to remote

Managing Merge

- Once all conflicts have been resolved, run *git merge* once more
- Do a final push to remote for the branch to refresh
- To “cancel” merge, use *git merge --abort*

Managing Merge

- Alternatively, many IDEs will help you automatically manage via UI



Demo



Lab

Branching in Git

Branching Strategy



- A consistent strategy for branching is important
- Define a standard set of strategy components that enable:
 - Simplicity
 - Stability
 - Efficient and repeatable releases

Feature Branches

- Feature branches are branches created from the main branch
- Use for new features as well as bugs
- Helps with tracking evolution of software over time
- Keeps developers' parallel work separate
- Use a consistent naming strategy for the branch (e.g., user-name/feature-name)

Use Pull Requests

- Feature branch changes should be reviewed and merged via PR
- Repo can be configured with a minimum number of reviewers to enable merge
- Can also integrate build steps into the PR process so successful build is confirmed before merge
- Keep PRs small

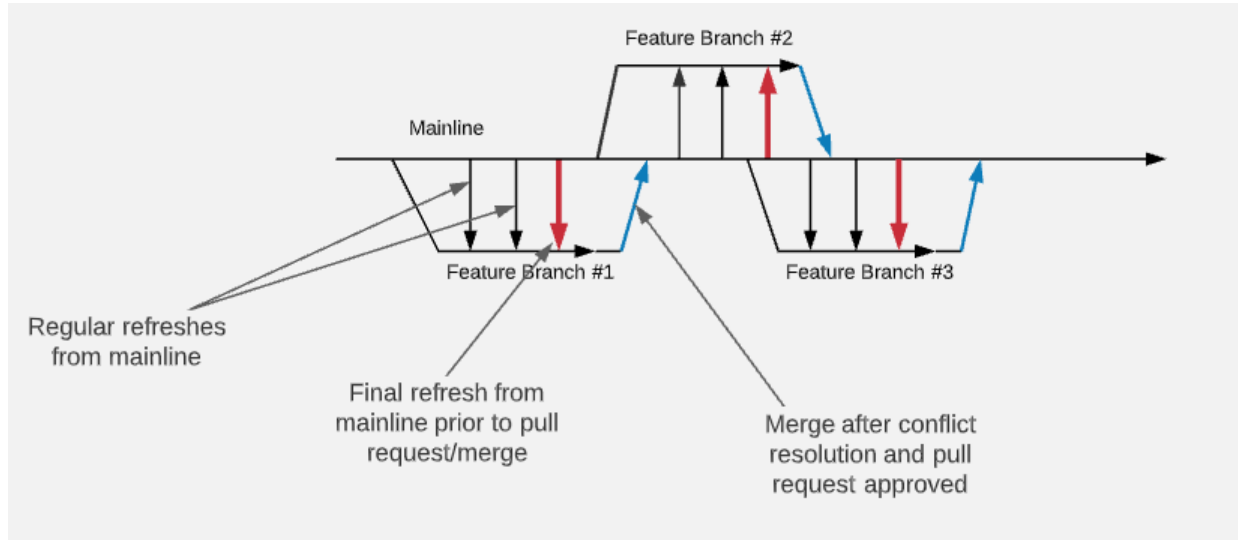
Protect Mainline

- Mainline should be clean and kept up-to-date
- Provides a stable, common starting point for developers creating feature branches from it
- Use branch policies to protect mainline
- Avoid direct merge there – use PRs instead

Developing in a Branch

- The longer a feature branch exists (and the larger the amount of code changes it contains), the higher the likelihood of a merge conflict
- To help minimize conflicts, a developer should keep their feature branch up-to-date with latest
- Can be done by regularly merging mainline into the branch during development

Developing in a Branch



Managing Releases

- Optionally, release branches can be used for managing releases of the software
- Provides a point-in-time version snapshot
- If bugs are found in a release, fixes need to be made in both release branch and mainline
- Couple of options: fix in mainline and port to release branch or fix in release branch and backport to mainline



Knowledge Check

Which of the following best describes why we use feature branches?

- A. It's the branch where all major release will be built from
- B. It helps developers keep feature updates/bug fixes separate from the main branch

Enter the letter corresponding to your answer in chat



Knowledge Check

Why do we use pull requests?

- A. To ensure code is reviewed before merging to the main release branch
- B. To ensure that unit testing is done before merging to the main

Enter the letter corresponding to your answer in chat



Knowledge Check

How do we mitigate merge conflicts with our long living feature branches?

- A. By running continuous tests on our feature branch code
- B. By regularly merging the main branch into the feature branch during development

Enter the letter corresponding to your answer in chat



Thank you!

If you have additional questions,
please reach out to me at:
erik-gross@pluralsight.com