Development Practices - Assessment Questions

Whiteboard Session

Whiteboard the development pipeline

- story definition
- assignment
- branching
- dev pull code (duration)
- dev change on local and compilation (duration)
- dev push (what branch?, CI enabled?)
- unit tests
- static code analysis
- review
- merge
- QE validations
- release candidate build
- how bugs are fixed in different phases (production fix, release hardening fix, dev fix)

Questionnaire

Question	**Why we ask?**	**How to score/rate?** **(1 lowest - 5 highest)**	**Weight?**	**What data can we pull to quantify?**
What branching model is used for development?	The branching model will help determine what integration points and gates are utilized. Branching is the key foundation that defines the development process and capabilities.	1 - branching model that involves long living feature/team branches 3 - no branching (master only) 5 - GitFlow branching strategy		
Is the branching model consistent across all services in the organization?	This will let us know if the model is an organization wide strategy or if it is different between each development team.	1 - no standardized branching model 3 - two branching models and adoption depends on type of application/service 5 - all services in the organization use the same branching model		Can pull the list of branches for each repo using the SCM rest api
Does each services have its own repository?	This will help us determine if code is stored in one large repository where development cycles/branching is striped across all services, or if the services are each in their own repository and capable of being versioned, branched, and deployed independently.	1 - on large repository for all services 3 - multiple repositories, but a repository can have more than one service 5 - one repository per service		
How are apps/services versioned?	This will help us understand if versioning is in place and if it is systematic across the organization.	1 - No versioning 2 - versioning for production uses SNAPSHOT versions 3 - non-numeric versions 4 - numeric versions, but not semantic versioning 5 - semantic versioning		
When is a release cut?	Need to determine when a release is cut so we can understand the timeline and the tasks that are involved leading up to release candidate creation and validation. Is it after the end of every sprint, after every other sprint?	1 - no release is cut 3 - release is cut at the beginning of release development 5 - release is cut once there is enough content to release		
What is the process for cutting a release?	This will give us an idea of the steps and gates that are part of cutting the release. We will also determine what the versioning strategy is wrt bumping for the next release, creating a release candidate, etc.	N/A		

How are development dependencies managed? For example, in order to develop my feature, a backend service change needs to be available.	This will give us an idea on how story dependencies are planned, developed, and integrated. It will also determine if there are strategies around mocking dependent changes that are not fully ready and also get a better idea on how api contracts are defined and communicated across the team.	N/A		
Are service dependencies (compile and runtime) clearly defined/documented?	This gives us an idea on whether development teams monitor and understand dependencies and know when they have changed.	1 - dependencies are not clearly defined 3 - either compile or run-time dependencies are defined but not both 5 - both run-time and compile dependencies clearly defined		
What is the unit test code coverage percentage for each application/service?	This will help determine if unit tests are an integral part of the development cycle	1 - < 10% unit test code coverage or not measured 2 - 10-30% unit test code coverage 3 - 30% - 60% unit test code coverage 4 - 60% - 80% unit test code coverage 5 - 80% > unit test code coverage		This can be gathered from Sonar or the build orchestration logs.
Are unit test code coverage enforced?	This will help us understand if there is standardization across the organization that enforces unit test code coverage.	1 - No code coverage enforcement 2 - Some applications require code coverage percentage to be met. 3 - All applications require code coverage to be met. 4 - All applications require code coverage to be met with the lowest threshold greater than 50% 5 - All applications require code coverage to be met with the lowest threshold greater that 80%		
Does the build pipeline fail if unit tests do not pass?	This will help us understand if there is standardization across the organization that enforces successful unit test runs.	1 - Unit tests are not executed for all applications 3 - Unit tests executed for all applications but do not enforce that the tests pass 4 - Unit tests executed for all applications but only enforce that they pass for some applications (>50% of apps) 5 - Unit tests executed for all applications and pipeline fails if any tests fail.		• Can check the pom.xml to verify that application is not configured to skip tests • Can check orchestration tool to ensure that tests are executed and configured to fail pipeline on test failure
Are feature branches created for unit of work?	This will give us an idea if featuer branches are created and if they are used for a small unit of work and merged often	1 - no feature branches created 3 - feature branches created but the branch includes multiple stories and/or developers 5 - feature branch created for small unit of change (story) and is worked by one developer		
What languages are the services developed on?	This will give us an idea of the type of compilation tool is used for build in the application.	N/A		

How aggressive are compile errors addressed on integrated and release branches?	This will give us an idea of how quickly compile errors are addressed	1 - compile errors last more than 24 hours on integrated/release branches 2 - compile errors resolved within 24 hours 3 - compile errors are resolved within 4 hours 5 - compile errors are resolve within 1 hour		
Are integration branches gated?	This will give us an idea on whether there are gates to ensure that the integrated branches are releasable at any point in time.	1 - integrated branches are not gated 3 - integrated branches can only be merged to by a select few 4 - integrated branches can only be merged by a select few, must contain code review, and must have passing build 5 - integrated branches can only be merged by a select few, must contain code review, must have passing build, and must have passing quality gate		
How often is code commit and pushed to the centralized repository?	This will give us an idea of the size of commits, how often it i tested, and if commit and push are happening late in the cycle thus reducing frequency of continuous integration.	1 - just before merging to integrated branch 3 - just before code review 5 - at least daily even when coding is not complete		
What is the current technical debt for the application/service?	This will help determine if technical debt is measured and addressed on a consistent basis.	1 - technical debt is not measured 2 - technical debt is greater than 30 days 3 - technical debt is less than 30 days 4 - technical debt is less than 7 days 5 - technical debt is less than a day		This can be determined using Sonar
Is there a process to ensure that services are using current versions of compile dependencies (internal and third party)?	This will help determine if dependencies and where they are referenced are kept up to date.	1 - no process in place 3 - manual process in place as part of the development practice 4 - automated process to identify outdated dependencies 5 - automated process to identify and address outdated dependencies.		
Only release artifacts are deployed to staging and production (no snapshot versions)	This will ensure that release builds can be replicated if required and ensures that only tested artifacts are released.	1 - snapshot artifacts can go to production 5 - only release artifacts can go to staging and production		
How are the deployment binaries packaged?	This gives us an understanding of how the package the binaries for deployment. Examples can be a Docker image, RPMs, MSI, etc	1 - no standardized packaging in pace 3 - packaged as a standard compressed file (tar, zip, etc) 5 - packaged using a deployable artifact (rpm, msi, docker file)		

How are configurations packages and managed?	This will give us an idea on how configurations are managed and deployed. Are the independent from the app binaries? How do you handle configurations that different from each deployment environment?	1 - they are manually applied to the deployment server 3 - they are packaged with the application binaries, but no tokenization in place for differences between environments 5 - They are packaged independently and tokenized. they are mapped to application release versions		
When is code reviewed?	Need to figure out when code is reviewed and how often. This will help us determine if code is reviewed at the last minute with a large number of lines/files changed. Reviews can be lengthy and most cases are not reviewed in detail.	1 - no code review enforced 3 - code review occurs as a big bang close to release. 5 - code review occurs as part of small feature branch changes before they are merged to the integrated branch		
What is the production deployment strategy?	This will give us an idea if there are large downtime windows require ed for a application/service upgrade	1 - in-place deployment requiring an outage greater than 1 hour 2 - in-place deployment requiring an outage greater than 30 mins hour 3 - in-place deployment requiring an outage greater than 2 mins 4 - blue/green deployment requiring less than 2 min outage 5 - canary deployment with zero outage		
Any architectural limitations that prevent certain deployment strategies (ex: canary)?	This will help determine what is required to achieve more efficient deployment strategies	1 - yes 5 - no		
What type of rollback strategy is in place	This will give us an idea on whether there are rollback strategies and/or scripts.	1 - no defined rollback strategy 2 - rollback strategy defined but takes more than 30 mins to rollback 3 - rollback strategy in place and takes more than 5 mins to rollback 4 - rollback strategy in place and takes less than 5 mins to rollback 5 - rollback strategy in place and is automatically initiated when failure detected or requirement not met.		
What is the size of an average release (lines of code, stories, story points, number of services deployed)	This will give us an idea of the magnitude of a release, the likelihood of introducing a new big, the amount of testing time required, etc.			
Is the organization following Agile?	This will help determine if the organization is following agile.	1 - no 3 - yes, but more scrum-waterfall 5 - yes		

How long are sprints?	This will help determine the development cycle and how often things are released.	1 - greater than 3 weeks 2 - greater than 2 weeks 3 - two weeks or less but application not released on every sprint 5 - two weeks or less and application released on every sprint		
How many story points for a day's worth of work?	Need to understand the story point size relative to the velocity of the team and the duration of the sprint.	N/A		
What is the done criteria of a story?	Need to understand when a story is considered done. It should include testing the story.	1 - no defined done requirement 3 - dev complete 5 - dev and test complete		
Frequency of standup meetings?	This will help determine how often standup meetings occur	N/A		
Duration of standup meetings?	This will help us understand how much time a standup takes and if there are opportunities to make them more efficient. Need to also determine what is discussed in the standup meetings.	1 - grater than 30 mins 3 - less than 30 mins 5 - less than 15 mins		
Distribution of code push to the integrated branch within the sprint?	An indicator of a potential issue is when there is limited code check-in during the sprint until the end of the sprint. Late check-in means: • QA load increases • QA timeline to test shortens • Higher percentage for late merge conflicts that can impact the release timelines • Dev productivity decreases at the end of sprint due to the larger number of reviews condensed at the end of the sprint • Late integration	1 - 80% of code pushed at the last 2 days of sprint. 2 - 60% of code pushed at the last 2 days of sprint 3 - 50% of code pushed at the last 2 days of sprint 4 - 40% of code pushed at the last 2 days of sprint 5 - 30% of code pushed at the last 2 days of sprint		
Where are feature branch builds deployed and tested?	This will help determine if feature branch builds are deployed and tested prior to merging to integration branch	1 - not deployed and tested 2 - deployed and tested locally 3 - deployed and tested in a shared environment that may not be isolated with only the feature branch changes and may not be reliable 5 - deployed and tested in a isolated feature environment that only has the feature changes		
Are sprint durations fixed?	Need to be cautious with organizations that extend sprint cycles on a adhoc basis?	1 - sprint durations can be extended 5 - sprint durations are static across the organization		
Does the team have a sprint retrospective?	This will help us understand if the teams discuss what went well during a sprint and what could be improved	1 - no 5 - yes		
Is the team velocity defined and adjusted as needed?	This will help determine what the team's velocity is and how it compares to sprint allocation	1 - no 5 - yes		
How often do stories miss the sprint deadline?	This will help determine if there is a potential sizing/allocation issue.	1 - on average more than 5 stories miss a sprint 3 - on average more than 3 stories miss a sprint 4 - on average at least 1 miss a sprint 5 - only once in a while a story may miss a sprint		

How often are stories resized within a sprint?	This will help determine scope creep	1 - almost every sprint 3 - at least once in every three sprints 5 - on isolated rare occasions		
How long does it take to compile application code locally on developer workstation?	This will give us an idea on how long it takes for developer to iterate over changes. It may also uncover opportunities to better utilize compile dependencies instead of compiling everything every time.	1 - greater than 1 hour 2 - greater than 40 mins 3 - greater than 30 mins 4 - greater than 15 mins 5 - less than 15 mins		
Can a developer start up the application locally for testing?	This will give us an idea if developers are able to locally verify their changes prior to the changes being compiled and deployed by CI/CD. Not being able to compile and run locally increases the time it takes to iterate between changes. The conversation will also lead us to impediments that currently blocks them from testing locally (i.e. database, large number of runtime dependent services that are not mocked or simulated, etc)	1 - no 5 - yes		