Microsoft

AZ-400.7 Module 1: Planning for DevOps

Lesson 01: Transformation Planning

Lesson 1: Overview

- Separating Transformation Teams
- Defining Shared Goals
- Setting Timelines for Goals

Separating Transformation Teams

- There are several challenges when creating teams:
  - Availability of staff
  - Disruption of current procedures and procedures
- To overcome the challenges, create a team that is:
  - Focused on the transformation
  - Well respected in their subject areas
  - Internal and external to the business

✔ A transformation project can conflict with on-going business needs

Defining Shared Goals

Projects must have a clearly-defined set of measurable outcomes, like:

- Reduce the time spent on fixing bugs by 60%.
- Reduce the time spent on unplanned work by 70%.
- Reduce the out-of-hours work required by staff to no more than 10% of total working time.
- Remove all direct patching of production systems.

✔ One of the key aims of DevOps is to provide greater customer value, so outcomes should have a customer value focus.

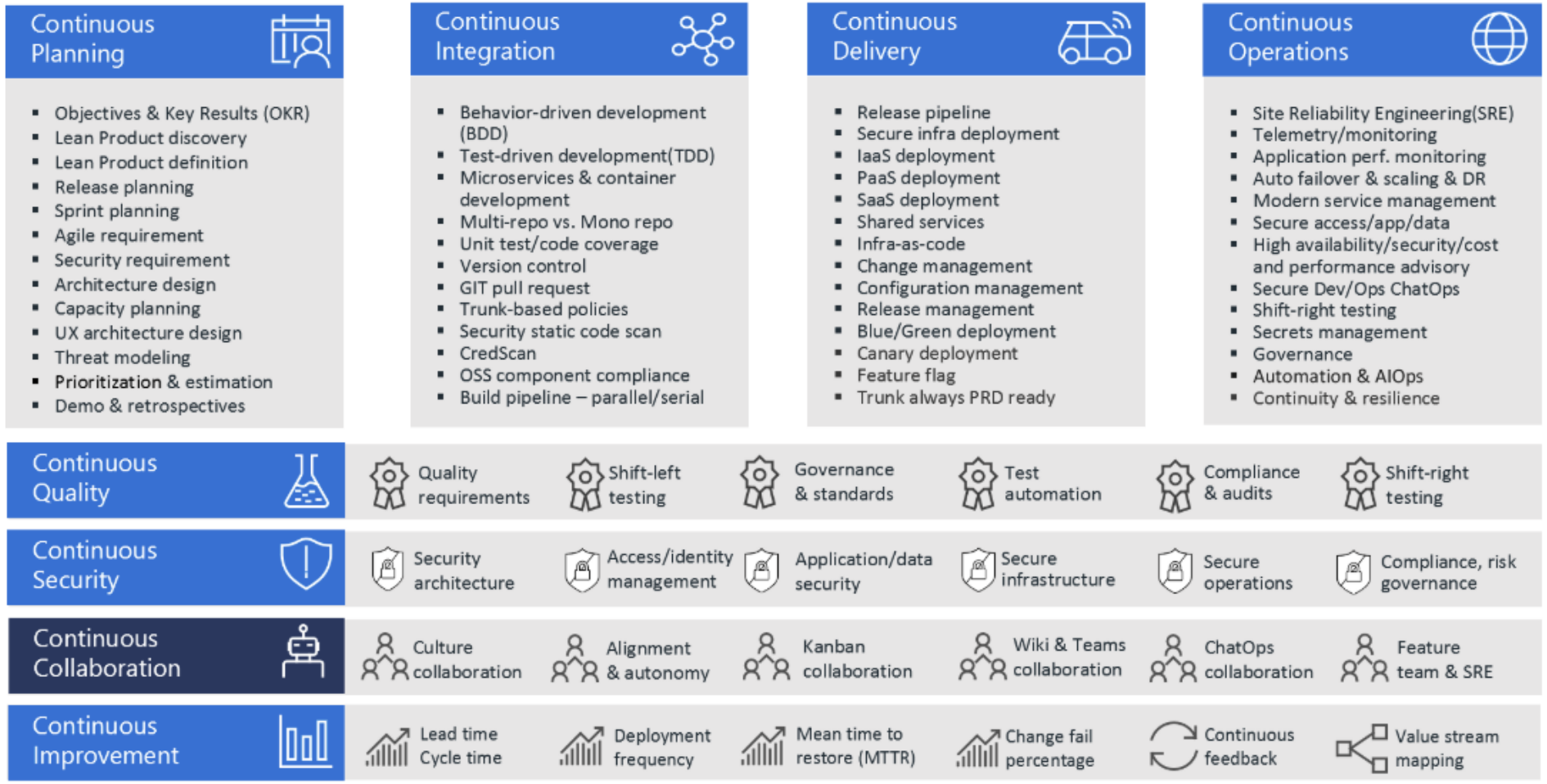
Setting Timelines for Goals

- Measurable goals should have timelines that challenging yet achievable
- Timelines should be a constant series of short-term goals – each clear and measurable
- Shorter timelines have advantages:
  - Easier to change plans or priorities when necessary
  - Reduced delay between doing the work and getting feedback
  - Easier to keep organizational support when positive outcomes are apparent

Lesson 02: Continous Collaboration

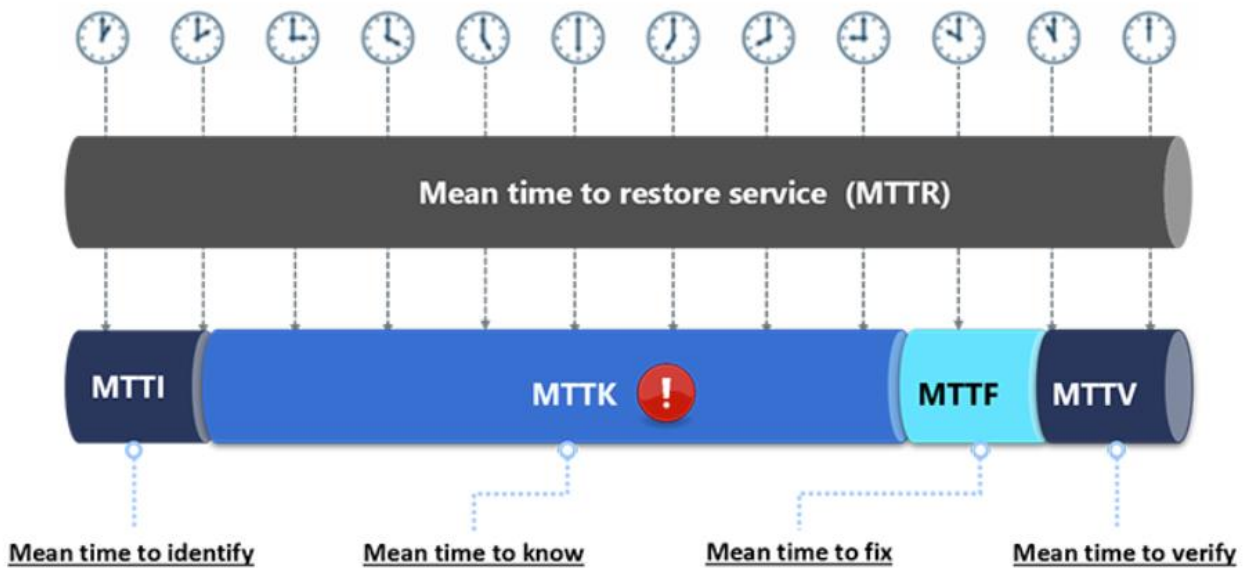
Continous Collaboration

- Continuous Collaboration is one of the eight capabilities in the DevOps taxonomy.



Mean time to resolution

- A measurement of the average amount of time from when an issue is detected until it has been completely fixed.
- Shortening MTTR by attacking MTTK (mean time to know) is a good operations KPI



The 5 Dysfunctions of a Team

- Absence of trust - Members of great teams trust one another on a fundamental, emotional level, and they are comfortable being vulnerable with each other about their weaknesses, mistakes, fears, and behaviors.
- Fear of conflict - Teams that trust one another are not afraid to engage in passionate dialogue around issues and decisions that are key to the organization's success.
- Lack of commitment - Teams that engage in unfiltered conflict can achieve genuine buy-in around important decisions, even when various members of the team initially disagree, because they ensure that all opinions and ideas are put on the table and considered.
- Avoidance of accountability - Teams that commit to decisions and standards of performance do not hesitate to hold one another accountable for adhering to those decisions and standards.
- Inattention to results - Teams that trust one another, engage in conflict, commit to decisions, and hold one another accountable are very likely to set aside their individual needs and agendas and focus almost exclusively on what is best for the team.

Principals of Continous Collaboration

- Continuous Collaboration encourages you to value:
- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Lesson 03: Project Selection

Lesson 2: Overview

- Greenfield vs Brownfield Projects Defined
- Choosing Greenfield and Brownfield Projects
- Choosing Systems of Record vs Systems of Engagement
- Selecting Groups to Minimize Initial Resistance
- Identifying Project Metrics and KPIs

Greenfield and Brownfield Projects Defined

- Greenfield software projects develop in a totally new environment
- Brownfield software projects develop in the immediate presence of existing software applications/systems

Choosing Greenfield and Brownfield Projects

- Greenfield projects
  - Appears to be an easier starting point
  - A blank slate offers the chance to implement everything the way you want
- Brownfield projects
  - Comes with the baggage of existing code bases, existing teams, and often a great amount of technical debt
  - Spending time maintaining existing Brownfield applications, limits the ability to work on new code

- ✔ There is a common misconception that DevOps suits Greenfield projects better than Brownfield projects, but this is not the case.

Choosing Systems of Record vs Systems of Engagement

- Systems of Record
  - Emphasize accuracy and security
  - Provides the truth about data elements
  - Historically evolves slowly and carefully
- Systems of Engagement
  - More exploratory
  - Uses experimentation to solve new problems
  - Modified regularly
  - Making changes quickly is prioritized over ensuring that the changes are correct

✔ Both types of systems are important

Selecting Groups to Minimize Initial Resistance

- Different types of staff members
  - **Canaries** voluntarily test bleeding edge features
  - **Early adopters** voluntarily preview releases
  - **Users** consume the products after canaries and early adopters
- Ideal DevOps team members
  - They think there is a need to change and have shown an ability to innovate
  - They are well-respected and have broad knowledge of the organization and how it operates
  - Ideally, they already believe that DevOps practices are what is needed
- Ideal Target Improvements
  - Can be used to gain early wins
  - Is small enough to be achievable in a reasonable time-frame
  - Has benefits that are significant enough to be obvious to the organization

Lesson 03: Team Structures

Lesson Overview

- Agile Development Practices Defined
- Principles of Agile Development
- Creating Organizational Structures for Agile Practices
- Mentoring Team Members on Agile Practices
- Enabling In-Team and Cross-Team Collaboration
- Selecting Tools and Processes for Agile Practices

Agile Development Practices Defined

- Waterfall approach
  - Define, analyze, build and test, and deliver
  - Hard to accurately define requirements, which can change over time
  - After delivery requires change requests and additional cost
- Agile approach
  - Emphasizes constantly adaptive planning, and early delivery with continual improvement
  - Development methods are based on releases and iterations
  - At the end of each iteration, there should be tested working code
  - Focused on these shorter-term outcomes

✔ Vertical teams have been shown to provide stronger outcomes in Agile projects

Principles of Agile Development

1. Satisfy the customer

2. Welcome changing requirements

3. Deliver working software frequently

4. Work together throughout the project

5. Build projects around motivated individuals

6. Use face-to-face conversation

7. Measure progress through working software

8. Agile processes promote sustainable development

9. Continuous attention to technical excellence and good design

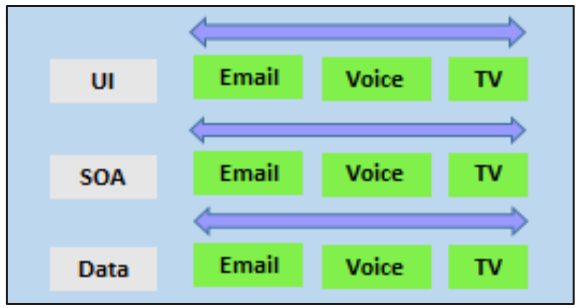
10. Simplicity--the art of maximizing the amount of work not done

11. Use self-organizing teams

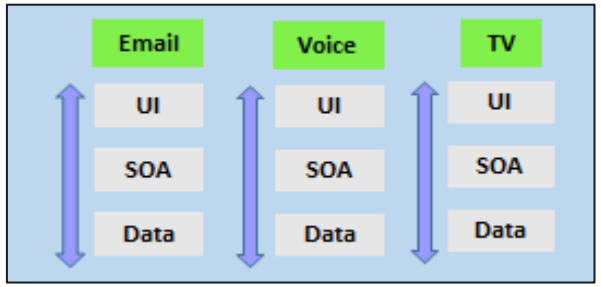
12. Reflect on how to become more effective

Creating Organizational Structures for Agile Practices

Horizontal Teams structures divide teams according to the software architecture



Vertical Teams span the architecture and are aligned with product outcomes and scaling can occur by adding teams



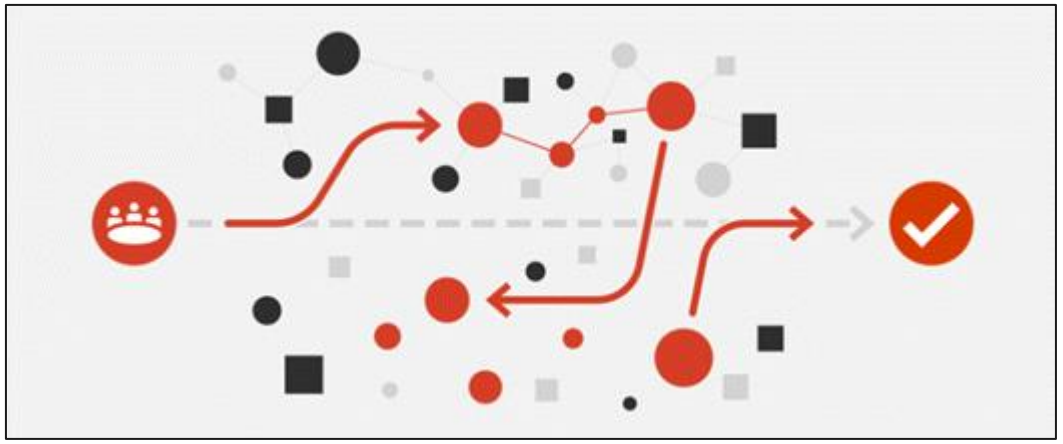
✔ Vertical teams have been shown to provide stronger outcomes in Agile projects

Mentoring Team Members on Agile Practices

- Many teams hire external Agile coaches or mentors
- Agile coaches have teaching and mentoring skills
- Agile coaches tend to be both trainers and consultants
- Some coaches are technical experts
- Some coaches are focused on Agile processes

✔ Team members must learn as they work, and acquire skills from each other

Enabling In-Team and Cross-Team Collaboration



- **Cultural changes** – more open work spaces, meeting etiquette, outsourcing, better communication
- **Cross-functional teams** – collaboration with others, diversity of opinion, rewarding collective behavior
- **Collaboration tooling** – Skype, Slack, Teams, Google Hangouts

Selecting Tools and Processes for Agile Practices

- Tools can often enhance the outcomes achieved
- Physical tools such as white boards, index cards, sticky notes
- Project Management tools for planning, monitoring, and visualization
- Screen Recording tools for recording bugs, building walk-throughs, and demonstrations

Microsoft

AZ-400.7 Module 2: Planning for Quality and Security

Lesson 01: Planning for a Quality Strategy

Measuring and Managing Quality Metrics

- DevOps can help you deliver software both faster and with better quality
- Metrics that directly relate to the quality of both the code being produced, and of the build and deployment processes
  - Failed Builds Percentage
  - Failed Deployments Percentage
  - Ticket Volume
  - Bug Bounce Percentage
  - Unplanned Work Percentage

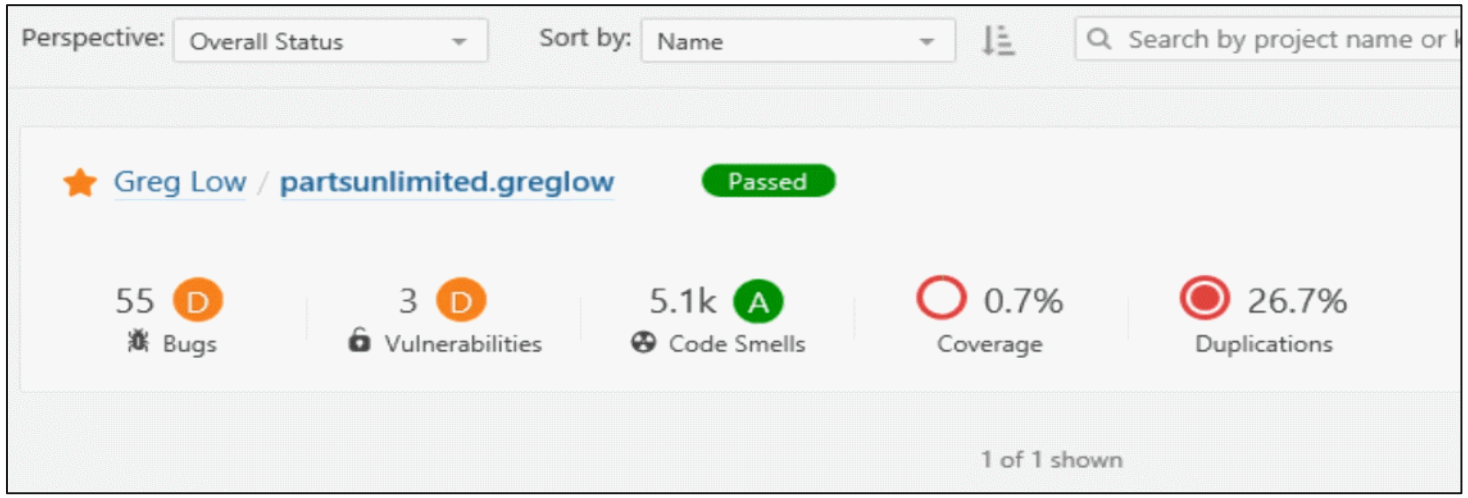
Identifying Project Metrics and KPIs

- **Faster Outcomes** - Deployment frequency, deployment speed, deployment size, and lead time
- **Efficiency** - Server to admin ratio, Staff member to customers ratio, application usage, and application performance
- **Quality and Security** - Deployment Failure Rates, Application Failure Rates, Mean Time to Recover, Bug Report Rates, Test Pass Rates, Defect Escape Rate, Availability, SLA Achievement, and Mean Time to Detection
- Culture – Employee morale and retention rates

✔ Goals must be specific, measurable, and time-bound

Measuring and Managing Technical Debt

- SonarCloud provides bugs, vulnerabilities, and code smells



- You can drill down for remedies and estimates of fix times

Planning Effective Code Reviews

- Code reviews can improve the quality of the applications
- The code review process must be efficient and effective
- Agree that everyone is trying to achieve better code quality
- Should be used as mentoring, not assigning blame
- Small group sessions are best
- Opportunity for all involved to learn, not just as a chore
- Establish and foster an appropriate culture across teams

✔ Improving code quality reduces the overall cost of code

Planning Performance Testing

- **Load testing** is used to find out how a system performs with many concurrent users, large amounts of data, and over a time period
- **Performance testing** looks at how responsive an application is, how efficiently it uses resources, how stable and reliable it is
  - Should be done early and often, so developers can implement the feedback
  - Should be part of your planning, right from the start
  - Often requires tools or suites of tools

✔ Focus not just on features, but also on performance