1.4 Deployment

Development, Release, and Deployment Concepts

Version Control

A version control system records changes to files or sets of files stored within the system. This can be source code, assets, and other documents that may be part of a software development project. Developers make changes in groups called commits or revisions. Each revision, along with metadata such as who made the change and when, is stored within the system in one way or another.

- Having the ability to commit, compare, merge, and restore past revisions to objects to the repository allows for richer cooperation and collaboration within and between teams. It minimizes risks by establishing a way to revert objects in production to previous versions.

Development, Release, and Deployment Concepts

Test-Driven Development

In test-driven development, the code developer starts by writing a failing test for the new code functionality, then writes the code itself, and finally ensures that the test passes when the code is complete. The test is a way of defining the new functionality clearly, making more explicit what the code should be doing.

Having developers write these tests themselves not only greatly shortens feedback loops but also encourages developers to take more responsibility for the quality of the code they are writing.

This sharing of responsibility and shorter development cycle time are themes that continue to be important parts of a devops culture.

Development, Release, and Deployment Concepts

Application Deployment

Application deployment is the process of planning, maintaining, and executing on the delivery of a software release. In the general sense, the craft of application deployment needs to consider the changes that are taking place underneath the system.

Having infrastructure automation build the dependencies required to run a specific application—whether they be compute, operating system, or other dependencies— minimizes the impact of inconsistencies on the released software. Depending on the application type, different engineering concerns may be important.

For example, databases may have strict guarantees in terms of consistency. If a transaction occurs, it must be reflected in the data.

Application deployment is a critical aspect to engineering quality software.

Development, Release, and Deployment Concepts

Continuous Integration

Continuous integration (CI) is the process of integrating new code written by developers with a mainline or “master” branch frequently throughout the day. This is in contrast to having developers working on independent feature branches for weeks or months at a time, merging their code back to the master branch only when it is completely finished.

- Long periods of time in between merges means that much more has been changed, increasing the likelihood of some of those changes being breaking ones.
- With bigger changesets, it is much more difficult to isolate and identify what caused something to break. With small, frequently merged changesets, finding the specific change that caused a regression is much easier.
- The goal is to avoid the kinds of integration problems that come from large, infrequent merges. In order to make sure that the integrations were successful, CI systems will usually run a series of tests automatically upon merging in new changes. When these changes are committed and merged, the tests automatically start running to avoid the overhead of people having to remember to run them—the more overhead an activity requires, the less likely it is that it will get done, especially when people are in a hurry.
- The outcome of these tests is often visualized, where “green” means the tests passed and the newly integrated build is considered clean, and failing or “red” tests means the build is broken and needs to be fixed. With this kind of workflow, problems can be identified and fixed much more quickly.

Development, Release, and Deployment Concepts

Continuous Delivery

Continuous delivery (CD) is a set of general software engineering principles that allow for frequent releases of new software through the use of automated testing and continuous integration.

It is closely related to CI, and is often thought of as taking CI one step further, that beyond simply making sure that new changes can be integrated without causing regressions to automated tests, continuous delivery means that these changes can be deployed.

Development, Release, and Deployment Concepts

Continuous Deployment

Continuous deployment (also referred to as CD) is the process of deploying changes to production by defining tests and validations to minimize risk.

- While continuous delivery makes sure that new changes can be deployed, continuous deployment means that they get deployed into production.
- The more quickly software changes make it into production, the sooner individuals see their work in effect. Visibility of work impact increases job satisfaction, and overall happiness with work, leading to higher performance. It also provides opportunities to learn more quickly. If something is fundamentally wrong with a design or feature, the context of work is more recent and easier to reason about and change.
- Continuous deployment also gets the product out to the customer faster, which can mean increased customer satisfaction (though it should be noted that this is not a panacea—customers won't appreciate getting an updated product if that update doesn't solve any of their problems, so you have to make sure through other methods that you are building the right thing). This can mean validating its success or failure faster as well, allowing teams and organizations to iterate and change more rapidly as needed.

Development, Release, and Deployment Concepts

The difference between Continuous Delivery and Continuous Deployment is one that has been discussed a great deal since these topics became more widely used.

Jez Humble, author of Continuous Delivery defines continuous delivery as being a general set of principles that can be applied to **any software development project, including the internet of things (IoT) and embedded software, while continuous deployment is specific to web software.**

Development, Release, and Deployment Concepts

Minimum Viable Product

One theme that has become apparent especially in recent years is the idea of reducing both development costs and waste associated with creating products.

If an organization were to spend years bringing a new product to market only to realize after the fact that this new product didn't meet the needs of either new or existing customers, that would have been an incredible waste of time, energy, and money.

The idea of the minimum viable product (MVP) is to create a prototype of a proposed product with the minimum amount of effort required to determine if the idea is a good one.

Rather than developing something to 100 percent completion before getting it into users' hands, the MVP aims to drastically reduce that amount, so that if significant changes are needed, less time and effort has already been spent. This might mean cutting down on features or advanced settings in order to evaluate the core concept, or focusing on features rather than design or performance.

As with ideas such as Lean and continuous delivery, MVPs allow organizations to iterate and improve more quickly while reducing cost and waste.

Infrastructure Concepts

All computer software runs on infrastructure of some sort, whether that be hardware that an organization owns and manages itself, leased equipment that is managed and maintained by someone else, or on-demand compute resources that can easily scale up or down as needed.

Configuration Management

Configuration management is the process of establishing and maintaining the consistency of something's functional and physical attributes as well as performance throughout its lifecycle. This includes the policies, processes, documentation, and tools required to implement this system of consistent performance, functionality, and attributes.

Configuration management is the process of identifying, managing, monitoring, and auditing a product through its entire lifecycle, including the processes, documentation, people, tools, software, and systems involved.

Infrastructure Concepts

Cloud Computing

Cloud computing, often referred to as just “the cloud,” refers to a type of shared, internet-based computing where customers can purchase and use shared computing resources offered by various cloud providers as needed.

- Cloud computing and storage solutions can spare organizations the overhead of having to purchase, install, and maintain their own hardware.
- The combination of high performance, cost savings, and the flexibility and convenience that many cloud solutions offer has made the cloud an ideal choice for organizations that are looking to both minimize costs and increase the speed at which they can iterate.
- Iteration and decreased development cycle time are key factors in creating a devops culture.

Infrastructure Concepts

Infrastructure Automation

Infrastructure automation is a way of creating systems that reduces the burden on people to manage the systems and their associated services, as well as increasing the quality, accuracy, and precision of a service to its consumers.

- Indeed, automation in general is a way to cut down on repetitious work in order to minimize mistakes and save time and energy for human operators.
- For example, instead of running the same shell commands by hand on every server in an organization’s infrastructure, a system administrator might put those commands into a shell script that can be executed by itself in one step rather than many smaller ones.

Infrastructure Concepts

Artifact Management

An artifact is the output of any step in the software development process. Depending on the language, artifacts can be a number of things, including JARs (Java archive files), WARs (web application archive files), libraries, assets, and applications.

Artifact management can be as simple as a web server with access controls that allow file management internal to your environment, or it can be a more complex managed service with a variety of extended features. Much like early version control for source code, artifact management can be handled in a variety of ways based on your budgetary concerns.

Generally, an artifact repository can serve as:

- a central point for management of binaries and dependencies;
- a configurable proxy between organization and public repositories; and
- an integrated depot for build promotions of internally developed software.

Infrastructure Concepts

Containers

One of the bigger pain points that has traditionally existed between development and operations teams is how to make changes rapidly enough to support effective development but without risking the stability of the production environment and infrastructure.

- A relatively new technology that helps alleviate some of this friction is the idea of software containers—isolated structures that can be developed and deployed relatively independently from the underlying operating system or hardware.
- Similar to virtual machines, containers provide a way of sandboxing the code that runs in them, but unlike virtual machines, they generally have less overhead and less dependence on the operating system and hardware that support them.
- This makes it easier for developers to develop an application in a container in their local environment and deploy that same container into production, minimizing risk and development overhead while also cutting down on the amount of deployment effort required of operations engineers

Cultural Concepts

Culture

DevOps is all about **culture**

If culture is not **supportive**, organizations will not be high-performing

Cultural Concepts

Some software development methodologies, such as Agile, define ways in which people will interact while developing software, there are more interactions and related cultural concepts that are important.

Retrospective

A retrospective is a discussion of a project that takes place after it has been completed, where topics such as what went well and what could be improved in future projects are considered. Retrospectives usually take place on a regular (if not necessarily frequent) basis, either after fixed periods of time have elapsed (every quarter, for example) or at the end of projects.

A big goal is local learning—that is, how the successes and failures of this project can be applied to similar projects in the future.

Cultural Concepts

Retrospective

What happened?

- What the scope of the project was and what ended up being completed.

What went well?

- Ways in which the project succeeded, features that the team is especially proud of, and what should be used in future projects.

What went poorly?

- Things that went wrong, bugs that were encountered, deadlines that were missed, and things to be avoided in future projects.

Cultural Concepts

Postmortem

Unlike the planned, regular nature of a retrospective, a postmortem occurs after an unplanned incident or outage, for cases where an event's outcome was surprising to those involved and at least one failure of the system or organization was revealed. Whereas retrospectives occur at the end of projects and are planned in advance.

Postmortems are unexpected before the event they are discussing. Here the goal is organizational learning, and there are benefits to taking a systemic and consistent approach to the postmortem by including topics such as:

What happened?

- A timeline of the incident from start to finish, often including communication or system error logs.

Debrief

- Every person involved in the incident gives their perspective on the incident, including their thinking during the events.

Remediation items

- Things that should be changed to increase system safety and avoid repeating this type of incident.

In the devops community, there is a big emphasis placed on postmortems and retrospectives being blameless. While it is certainly possible to have a blameful postmortem that looks for the person or people "responsible" for an incident in order to call them out, that runs counter to the focus on learning that is central to the devops movement.

Cultural Concepts

Blamelessness

Blamelessness is a concept that arose in contrast to the idea of blame culture. Though it had been discussed for years previously by Sidney Dekker and others, this idea really came to prominence with John Allspaw's post on blameless postmortems, with the idea that incident retrospectives would be more effective if they focused on learning rather than punishment.

A culture of blamelessness exists not as a way of letting people off the hook, but to ensure that people feel comfortable coming forward with details of an incident, even if their actions directly contributed to a negative outcome. Only with all the details of how something happened can learning begin to occur.

Cultural Concepts

Organizational Learning

A learning organization is one that learns continuously and transforms itself…Learning is a continuous, strategically used process—integrated with and running parallel to work.

Organizational learning is the process of collecting, growing, and sharing an organization's body of knowledge.

- A learning organization is one that has made their learning more deliberate, setting it as a specific goal and taking actionable steps to increase their collective learning over time.
- Organizational learning as a goal is part of what separates blameful cultures from blameless ones, as blameful cultures are often much more focused on punishment than on learning, whereas a blameless or learning organization takes value from experiences and looks for lessons learned and knowledge gained, even from negative experiences.
- Learning can happen at many different levels, including individual and group as well as organization, but organizational learning has higher impact to companies as a whole, and companies that practice organizational learning are often more successful than those that don't.

Cultural Concepts

What Does a Good Culture Look Like?

Trust and **cooperation** across the organization

Indicated **high-quality** decision making

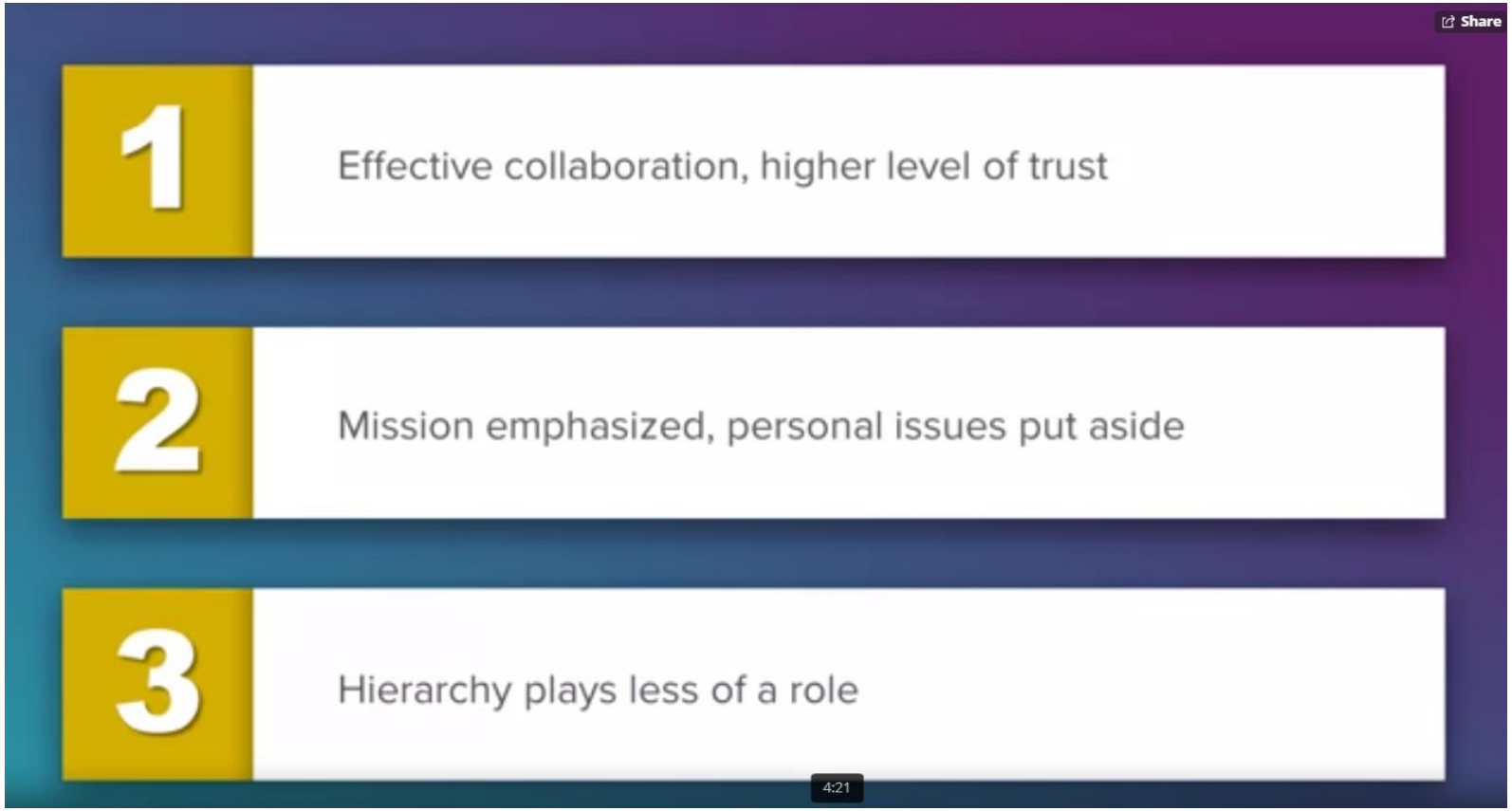
Generative Culture Team

Better information is available to make decisions

Team is **open** and **transparent**

Culture predicts **performance**

The Westrum Model for Improving Organizational Culture – Generative culture Team



References:

Generative cultures: https://www.youtube.com/watch?v=a7gplNBFEcA

https://cloud.google.com/architecture/devops/devops-culture-westrum-organizational-culture

https://www.linkedin.com/pulse/generative-culture-dave-cornelius