Project: UnifiedDevOps Sorting (codes)v5

All (25) codes

AUTOMATION

- **automated application life-cycle management**

> *This code refers to the automation of some of the processes of the application life cycle (from development and deployment pipelines to monitoring tasks) and the tools adoption or tool providing for supporting these processes. The level of automation may range FROM high-level of build automation (continuous integration, CI); testing automation (continuous testing or quality assurance automation); delivery automation (continuous delivery, CD); deployment automation (continuous deployment, CD); operational tasks automation (continuous measurement/feedback/monitoring from operations to development); and recovery automation TO non-automation. Not to be confused with platform servicing; the code "automated application life-cycle management" refers to pipelines and tools for automated delivery/deployment and operation, whereas the code "platform servicing" refers to the offered services to product teams. Automated recovery refers to the ability to either replace a component that is not working or rollback a failed deployment without human intervention, this means, the capability that applications and infrastructure have to recover itself in case of failures (resilience). There are two typical cases of recovery automation: (1) in cases of instability in the execution environment of an application (a container, for example) an automatic restart of that environment will occur; and (2) in cases of new version deployment; if the new version does not work properly, the previous one must be restored. This auto restore of a previous version decreases the chances of downtimes due to errors in specific versions, which is the concept of zero down-time. [D7] 07/03/2022 1:48:19, merged with automated (continuous) delivery 07/03/2022 1:48:19, merged with automated (continuous) deployment 07/03/2022 1:48:19, merged with automated (continuous) integration 07/03/2022 1:48:19, merged with automated (continuous) operation 07/03/2022 1:48:19, merged with automated (continuous) testing 12/03/2022 18:04:09, merged with tools adoption/providing 17/05/2022 18:17:59, merged with automated recovery*

- **automated infrastructure management**

> *This code refers to the automation of the infrastructure management and the use of configuration management tools. The level of automation may range from low to high levels of automated infrastructure (aka. Infrastructure as Code, IaC) and configuration management to non-automation. Not to be confused with platform servicing; the code "automated infrastructure management" refers to techniques, technologies, and tools for automated infrastructure management, whereas the code "platform servicing" refers to the offered services to product teams.*

- **platform builder**

> *This code refers to the platform that an entity (e.g., a team, an external consultant, etc.) builds to provide automated CI/CD pipelines and managing infrastructure (physical and/or virtual environments).*

- **platform servicing**

> *This code refers to the services that an entity (e.g., a team, an external consultant, etc.) offers/provides to product teams (developers and/or operators) to assist them on DevOps platform, such as: infrastructure management (e.g., containerization, cloud, etc.), infrastructure automation (from low to high levels of automated infrastructure services aka. Infrastructure as Code, IaC), pipeline automation (CI/CD and release tools), IT operation, etc.*

CULTURE

- **blame**

  > *From blameless contexts to blame contexts. In blameless contexts, the development team must not say that a given issue is a problem in the infrastructure, this is, operations team responsibility. Likewise, the operations team must not say that a failure was motivated by a problem in the application, this is, development team responsibility. The teams need to focus on solving problems, not on laying the blame on others and running away from the responsibility.*

- **collaboration**

  > *From the lack or eventual collaboration, which may generate conflicts and disagreements on decisions, to daily collaboration (working together regularly on a daily basis).*

- **communication**

  > *From poor/rare communication (and standardization) to frequent communication.*

- **continuous improvement**

  > *This code refers to the ongoing improvement of products, services or processes through incremental and breakthrough improvements. Continuous improvement seeks to improve every process in your company by focusing on enhancing the activities that generate the most value for your customer while removing as many waste activities as possible.*

- **cultural silos/conflicts**

  > *From non-cultural barriers to existing cultural barriers. Not to be confused with organizational/silos conflicts. While the cultural silos/conflicts focus on practices and culture that leads/break to barriers, organization silos/conflicts focus on team structures.*

- **culture, values & best practices**

  > *This code refers to a generic concept to be used when a quotation explicitly refers to this generality such as DevOps best/good practices or mentions a large subset of these values and practices (continuous integration, continuous testing, continuous delivery and deployment, infrastructure as code, continuous monitoring, etc.), cultural values (collaboration, communication, transparency, etc.), and/or principles (customer-centric action, create with the end in mind, end-to-end responsibility, cross-functional autonomous teams, continuous improvement, automate everything you can, among others). If a quotation refers to a specific practice, value, and principle, specific codes should be used.*

MANAGEMENT

- **change management**

  From managing small and frequent changes (small batch) to large and rare changes. The most effective change management is achieved by organizations that allow employees more scope to influence change management, high degree of automations, less rigid and much less manual approval processes, among others.

- **metrics, visibility & feedback**

  Metrics (process, value, cost, and technical metrics) provide fast and continuous feedback from users, reduce the risk and cost of deployments, get better visibility into the delivery process itself, and manage the risks of software delivery more effectively. Organizations are realizing the value of going beyond basic measurements for service levels and setting goals (service-level objective, SLO) that are based on meaningful metrics for the business. 26/05/2022 20:25:56, merged with service-level objective (SLO) management Service-level objectives (SLOs) as a measure of success

- **product management**

  Product management (related to a product or even service model) versus traditional project management. Each product or service has a team associated with it, and that team is completely responsible for the product/service (from scoping out the functionality, to architecting it, to building it, and operating it). In this way, teams can innovate quickly with a strong customer focus.

- **team self-organization & autonomy**

  From external and/or bureaucratic dependencies and approvals to high levels of team self-organization and autonomy (i.e., the team has freedom and autonomy to organize its tasks, budget, infrastructure, etc.). Alignment with the Amazon motto ‘‘You built it, you run it’’. 07/03/2022 0:24:35, merged with autonomy 07/03/2022 1:57:10, merged with external dependencies

- **transfer of work between teams**

  Transfer of work and responsibilities between teams (e.g., between development and infrastructure/operation teams). Hence, the definition of development "done" may go from committing a change to running the change in production-like environments. When there is transfer of work, depending on the high/poor trust/confidence on the other team, it may generate stress in the teams.

ORGANIZATIONAL STRUCTURE

- **devops (bridge) team**

  DevOps teams collaborate, assist, support, help development and operation teams, mainly by deploying and hosting applications in the platforms they build (platform builders), monitoring and providing support. The engineers of these (bridge) DevOps teams are the DevOps practices facilitators; hence, they usually create, deploy, and manage both the infrastructure (environments) and deployment (CI/CD) pipelines. They may be also involved in other tasks, such as requirements (user stories) analysis and coding. They are usually the bridge interface between developers and IT Operations to drive the DevOps values and practices.

● enabler (platform) team

This code refers to specific structures/teams that organizations create to satisfy some needs of product teams. Sometimes it is not necessary to create new structures/teams because the operations team/department takes over these needs and assists product teams. These needs can be platform servicing and tools (mainly for infrastructure and deployment pipelines), consulting, training, evangelization, mentoring, human resources, etc. Thus, they behave as enabler teams by providing these capabilities. These structures/teams are named in different ways, e.g., DevOps Centers of Excellence, chapters, guilds, platform/SRE teams, etc. This code should be used when a quotation explicitly mentions these new structures/teams or when a quotation describes the capabilities (at least three) of these enabler teams. 03/03/2022 23:59:06, merged with DevOps Center of Excellence (CoE) 03/03/2022 23:59:06, merged with DevOps Chapter 03/03/2022 23:59:06, merged with DevOps Platform Team

● organizational silos/conflicts

From non-organizational silos/barriers (e.g., co-locating teams/departments) to siloed departments and existing organizational barriers (segregated departments; frictions, mistrust, conflicts, and disagreements among silos; silos that become bottlenecks; minimal or no awareness of what is happening on the other side of the wall).

SHARING

● alignment of dev & ops goals

From misalignment among teams that have their own interests and goals (local optimization) to alignment with business goals and global ones (product thinking). A typical example of misalignment is when "developers want to deliver as much as possible, whereas operations target stability". A typical example of alignment is when product teams and DevOps/SRE teams agree service-level objectives (SLO).

● responsibility/ownership sharing

From shared responsibility of the products, output artifacts (e.g., databases), and tasks (e.g., NFR shared responsibility, infrastructure management shared responsibility, monitoring shared responsibility, and incident handling shared responsibility, etc.) to separate responsibilities and tasks (each team member has different responsibilities and tasks). Thus, if there is no shared responsibility, there is necessarily a transfer of work development to production and operation (and vice versa). However, ownership sharing is related to a new definition of "done" (e.g., developers' work doesn't finish with coding, but they support deployment in production). * e.g., developers perform automated infrastructure management (write infrastructure code, IaC). 04/03/2022 0:08:18, merged with NFR shared responsibility*

● skills/knowledge sharing

From high to low levels of knowledge sharing (e.g., developers may have knowledge about infrastructure/platform, or minimal or no awareness of what is happening on the other side of the wall, aka. wall of confusion or siloization of knowledge). Knowledge sharing may include a shared repository (the source of truth) and administrative rights to different environments (testing, pre-production, production). Knowledge sharing helps build trust between Devs & Ops personnel so that the former can share/delegate tasks to the latter second, and vice versa. However, knowledge sharing also increases the cognitive load (knowledge and expertise) that some members of the team acquire since sharing promotes that these members are responsible for everything.

● **stack & tools sharing**

> *From a holistic view of the tools and stack—the frontend, backend, libraries, storage, kernels, and physical machine—to non-shared stack.*

SKILLS & ROLES

● **cross-functionality/skills**

> *From multidisciplinary/poly-skilled teams (i.e., teams with all the necessary skills such as development, infrastructure, etc.) to teams with a lack of skills/knowledge/background. This can be addressed by product teams with their own infrastructure staff/engineers or senior developers that are responsible of infrastructure (with knowledge on automated infrastructure, and thus, DevOps facilitators). 13/03/2022 0:13:47, merged with need for dedicated infra engineers*

● **role definition/attributions**

> *From "skills over roles" and T-shape/DevOps engineers (aka. full stack engineers) to well-defined and differentiated roles. Approaches with well-defined and differentiated roles may decrease collaboration and promote a transfer of responsibilities; or there can be collaboration and avoid conflicts over who is responsible for each task. 02/03/2022 12:48:04, merged with well-defined and differentiated roles 07/03/2022 2:08:43, merged with T-shape engineers*

● **training, evangelization and mentoring**

> *DevOps evangelization and mentoring*