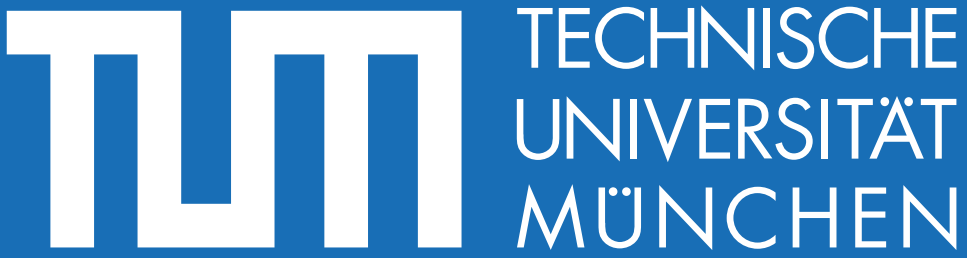
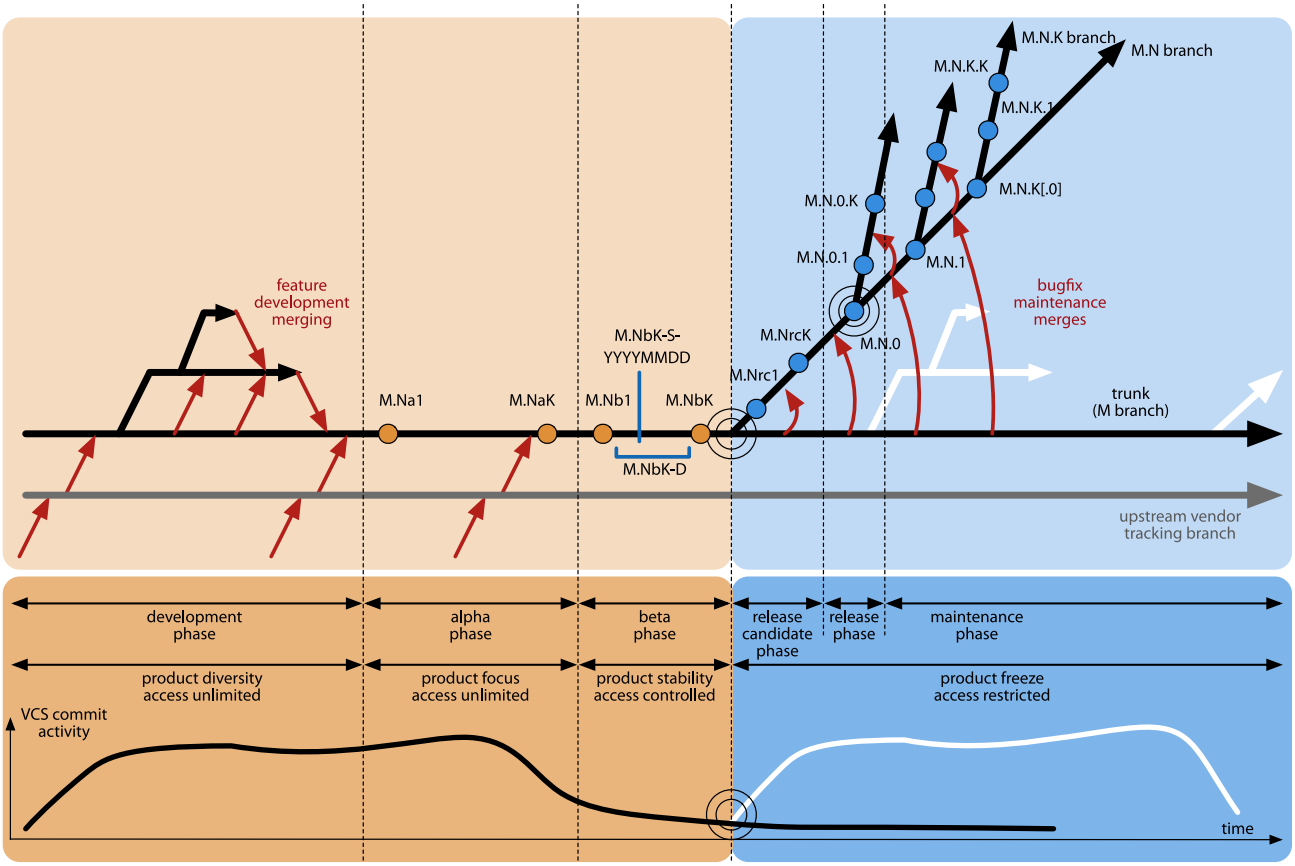


Software Engineering in Industrial Practice (SEIP)

Dr. Ralf S. Engelschall

Version Control Architecture



The **Version Control** is about the consequent versioning of all source code artifacts in a **Version Control System (VCS)**, like Subversion or Git. One uses five different kinds of **Branches** in a VCS: **Vendor-Branch**, **Trunk** (or **M Branch**), **Feature-Branch**, **Release-Branch** (or **M.N Branch**) and **Hotfix Branch** (or **M.N.K Branch**).

The **Vendor Branch** (or **Upstream Vendor Tracking Branch**) holds the unmodified copies from third-party sources. Its data is regularly integrated into the Trunk to be modified there, if necessary. It exists as long as the product itself. A classic use case consists of third-party libraries that have to be modified.

The **Trunk** or **M Branch** (called `trunk` with Subversion and `master` or `main` with Git) at any time keeps the current integrated state of the next major version ("M") of the product. It exists as long as the product itself.

A **Feature-Branch** is always forked from the Trunk (or even from another Feature-Branch). It contains the changes during the development of a new feature, in case it is more extensive, or has to be developed over a longer period of time, or needs to be developed by more than one person. It is regularly updated with the changes of the Trunk (in Git by "merging" or "rebasing"), integrated into the Trunk at the end of the feature development, and then usually deleted again.

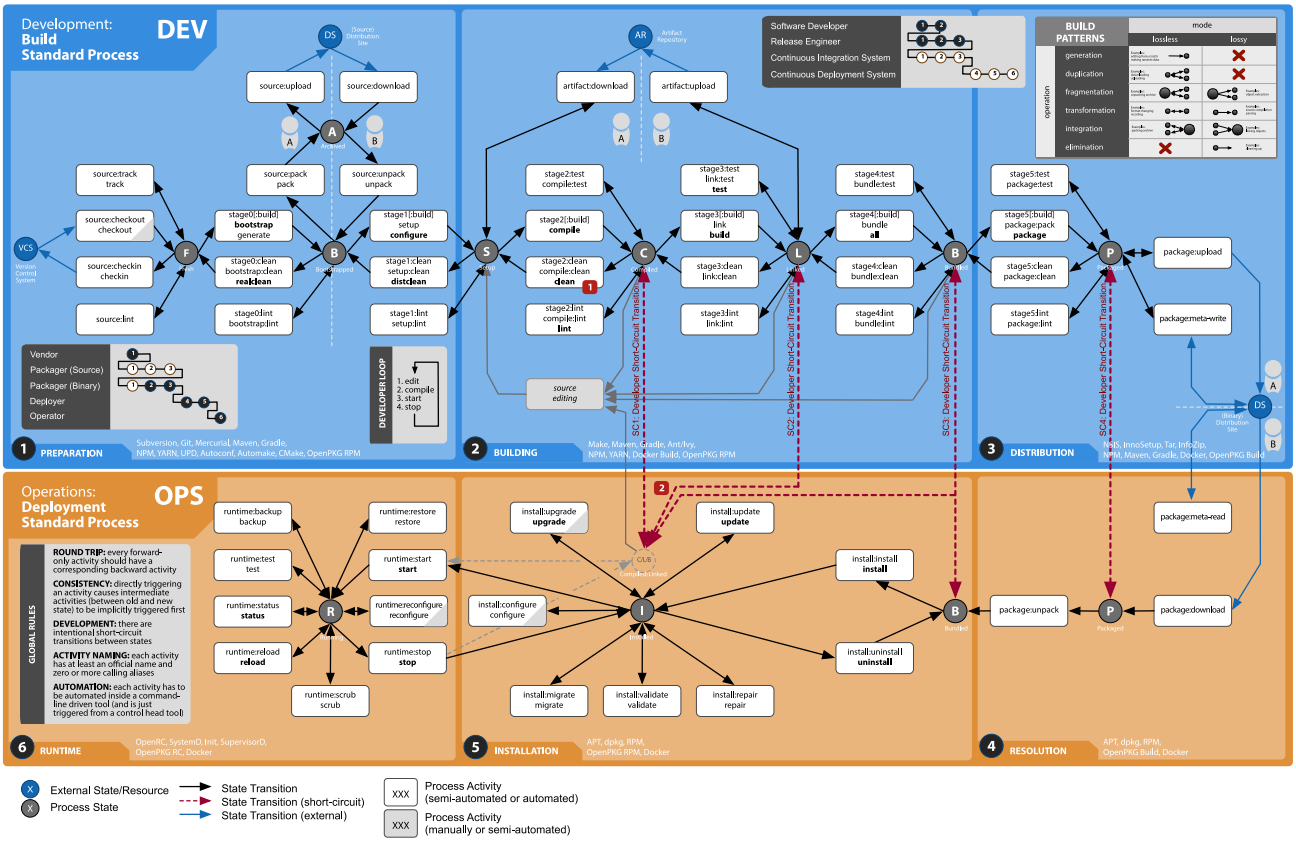
A **Release-Branch** or **M.N Branch** is forked from the Trunk, usually after the Beta phase ("M.NbK") and before the Release Candidate phase ("M.Nrc1"). It regularly receives from the Trunk bugfixes through so-called "cherry picking" of changes. On it, the release versions "M.N.K" are created. The "M.N Branch" exists as long as the "M.N" version of the product is not yet "end-of-life."

A **Hotfix Branch** or **M.N.K Branch** is forked from the "M.N Branch" as needed, usually directly from the "M.N Branch," usually directly before the need for a first **Hotfix** "M.N.K.1". It exists as long as **Hotfixes** for the release version, "M.N.K" have to be provided.

Questions

- Which five **Types of Branches** are known in a **Version Control System**"?
- Which **Type of Branch** in a **Version Control System** is usually deleted after successful integration?
- Between which two temporal **Phases** of the Release Management is the **Release Branch** in a **Version Control System** usually forked off from the **Trunk**?

Assembly Process Architecture



In the **Assembly Process Architecture**, a **DevOps Pipeline** is used to automatically transition a version of a software product from the sources in the **Version Control System** to the running instance in operation.

The Dev(elopment) part of the **DevOps Pipeline**, the so-called **Build Standard Process**, is usually automated via **Continuous Integration (CI)**. The Op(eration)s part of the **DevOps Pipeline**, the so-called **Deployment Standard Process**, is usually automated via **Continuous Integration (CI)**. **Deployment Standard Process**, is usually automated via **Continuous Deployment (CD)**. All activities of the **DevOps Pipeline** are automated via specialized build and deployment tools and these activities are automatically executed in a CI/CD system after each change in the **Version Control System**.

In particular, the **Assembly Process** should enable a meaningful **Round Trip**, in that the process is understood as a state machine and for all (forward) activities there are meaningful associated backward activities are existing. If a certain target state is externally requested, all intermediate activities between the source and the target state are implicitly executed.

For productivity or the most efficient "Developer Loop" possible in software development, up to four **Short Circuit Transitions** are supported, with which a deliberate "Shortcut" from the **Build Standard Process** to the **Deployment Standard Process** can be made.

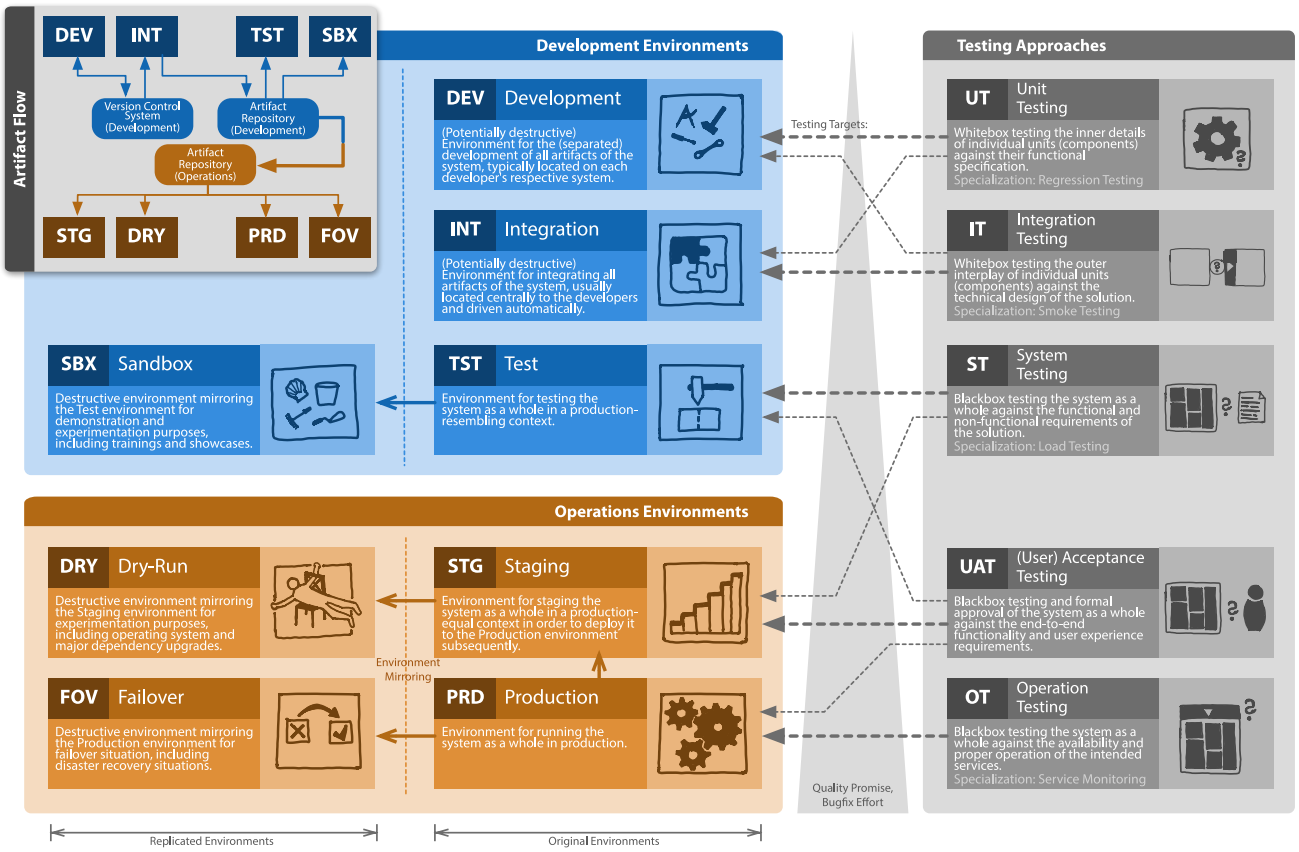
The **Assembly Process Architecture** makes use of four different types of external storage locations: the **Version Control System** stores the "bare" source files, the **(Source) Distribution Site** stores the "source distribution" of the source files prepared for the build process, the **Artifact Repository** stores reused build artifacts (especially libraries) and the **(Binary) Distribution Site** stores the "binary distribution" of the product intended for deployment.

The first three locations are mainly used for passing data between different people. The last one is mainly used for passing data between Dev(elopment) and Op(eration)s.

Questions

- Why should in the **Assembly Process Architecture** up to four so-called **Short-Circuit Transitions** be supported to shortcut from the **Build Standard Process** to the **Deployment Standard Process**?
- What are the four **types of external storage locations** supported by the **Assembly Process Architecture**?

Environments & Quality Assurance



In practice, a distinction is usually made between 4 **Development Environments**, 4 **Operations Environments** and 5 associated **Testing Approaches**.

The **Development Environments** are **Development** (e.g., the computer of the developer), **Integration** (e.g., a central server with a Continuous Integration (CI) system), **Test** (e.g., a central server that resembles the **Production** environment, but is not a copy of it) and possibly **Sandbox** (e.g., a server that is a copy of **Test** for training and show-cases).

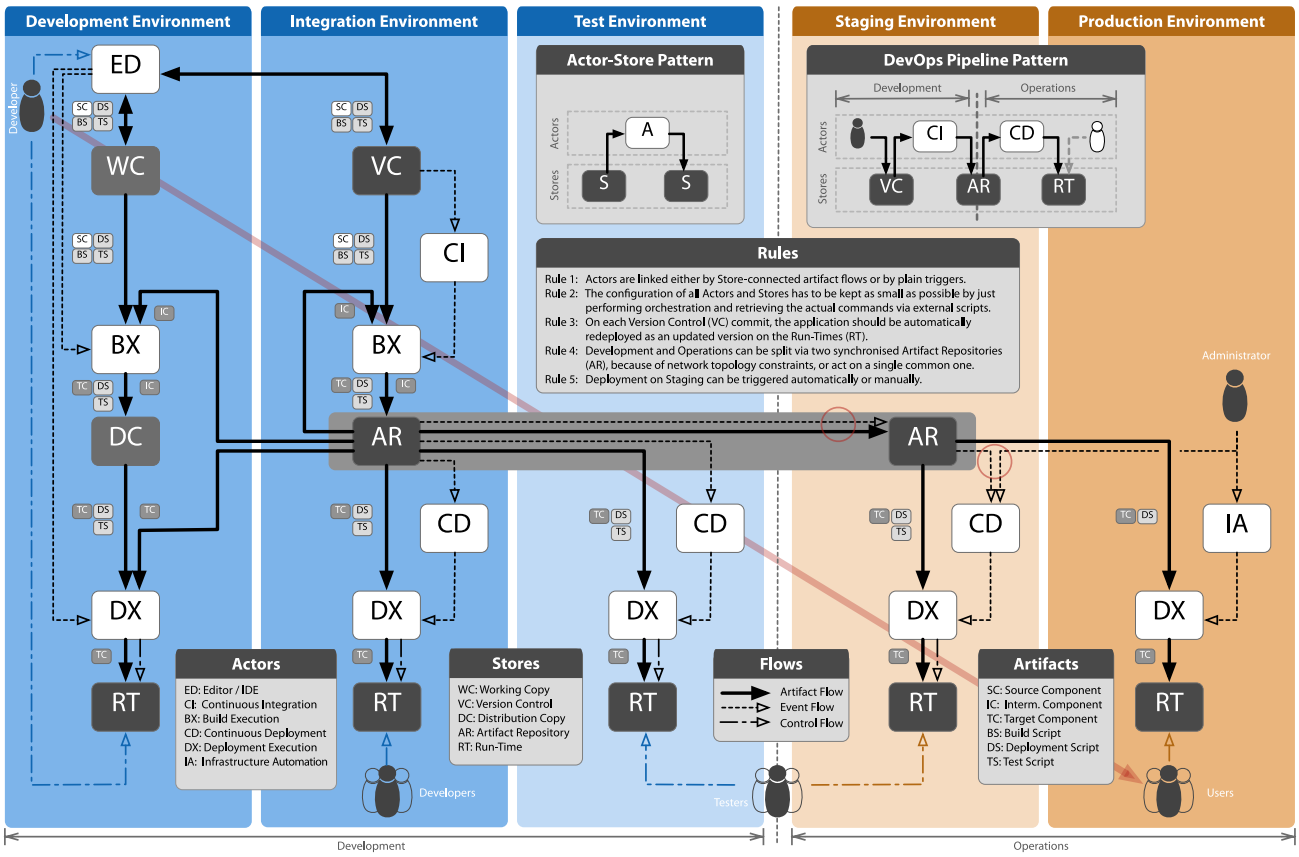
The **Operations Environments** are **Staging** (a copy of **Production**), **Dry-Run** (a 1:1 copy of **Staging**), **Production** (the regular production environment) and **Failover** (a 1:1 copy of **Production**).

The **Testing Approaches** are **Unit Testing** for the functionality of components on the **Development** (and possibly **Integration**) environment, **Integration Testing** for the interaction of components on **Integration** (and if applicable **Development**) environment, **System Testing** for the functional and non-functional properties of the overall system on **Test** (and **Staging** if applicable) environment, **(User) Acceptance Testing** for the "end-to-end" functionality of the overall system on **Staging** (or **Production** if applicable) environment and **Operation Testing** for the availability of the overall system on the overall system on the **Production** environment.

As transfer points for the artifacts between the 8 environments serve a **Version Control System** and an **Artifact Repository** on the side of the **Development Environments** and a corresponding **Artifact Repository** on the side of the **Operations Environments**.

Intellectual Content: Version 1.0.6 (2023-11-28), Authored 2018-2023 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.4 (2018-09-15), Copyright © 2018-2019 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

DevOps Toolchain



To support a DevOps approach on the tool-side as well, a **DevOps Toolchain** is advised to be used. At its core, this is based on a pattern in which an **Actor** acts between two **Stores** in each case by taking one or more **Artifacts** as input from a **Store**, processes these and writing one or more **Artifacts** as an output to another **Store**. Additionally, **Actors** can be triggered by **Events** or can be controlled directly by different groups of people through interactions.

This basic pattern is now combined to a **DevOps Pipeline Pattern**, where every "Commit" of a **Developer** in a **Version Control** (VC) system triggers an automatic compilation and integration process of an application in a **Continuous Integration** (CI) system. The results of this process are stored in an **Artifact Repository** (AR), which in turn triggers an automatic installation process in a **Continuous Deployment** (CD) system. The result is the installed application on a **Run-Time** (RT) system, which can be accessed by **Testers** and **Users**.

Logically, the systems VC and CI belong to the area **Development**, while the CD and RT systems belong to the **Operations** area. The AR system, on the other hand, is used by both areas as a common transfer point. Since in practice, the **DevOps Toolchain** is not on a single **Environment**, but is usually distributed on the logically (or even physically) separated **Environments Development**, **Integration**, **Testing**, **Staging** and **Production**, some systems exist multiple times.

We distinguish the **Artifacts** between **Source Component** (`foo.java`), **Intermediate Component** (`foo.jar`) and **Target Component** (`foo.exe`) and, on the other hand, between **Build Script** (`foo.make`), **Deployment Script** (`foo.spec`) and **Test Script** (`foo-test.java`).

Note intentional special cases: The **Development Environment** is different because it allows a fast "edit-build-install-start-stop" loop. The AR system can exist 1 or 2 times, depending on how strongly interweaved Development and Operations are. Deployment in the **Production Environment** should be manually triggered via an **Infrastructure Automation** system.

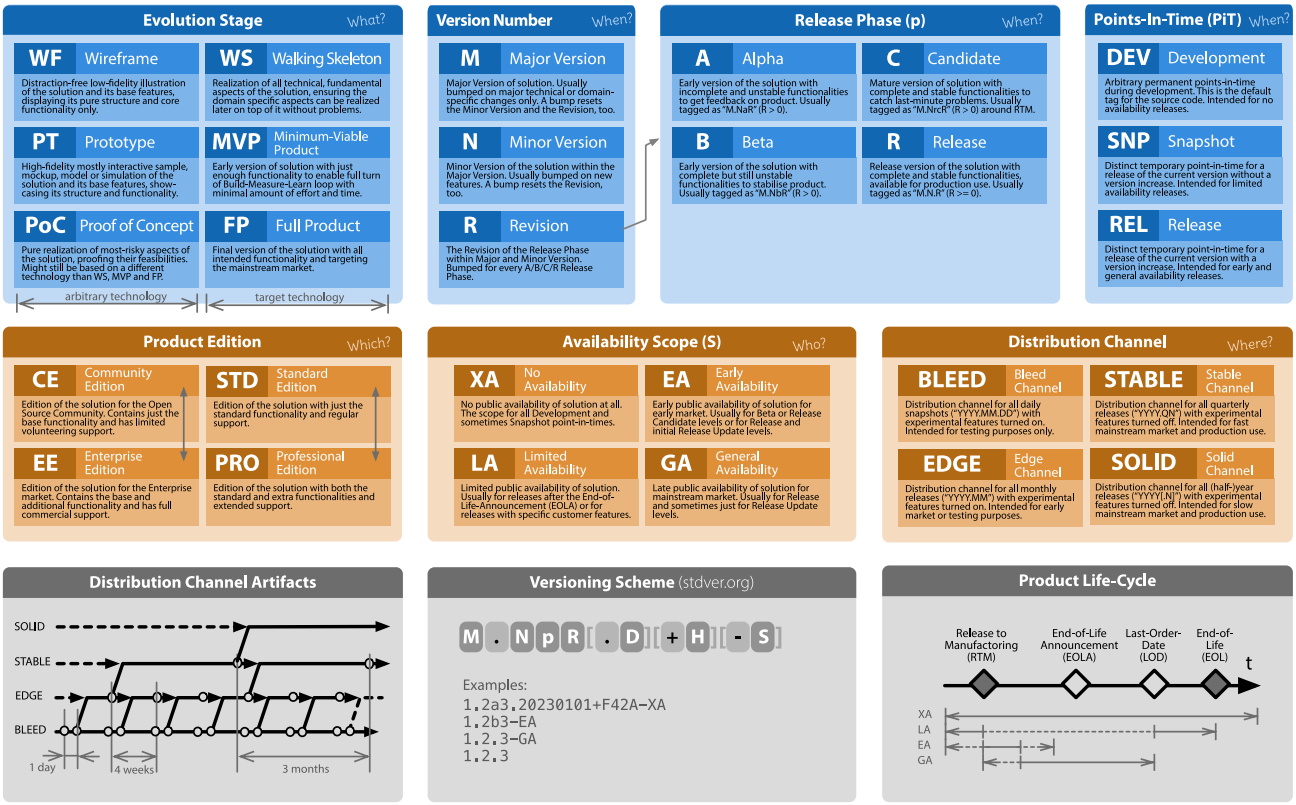
Questions

? Which two **Actor** systems control in the **DevOps Pipeline Pattern** the automated integration and installation process?

ARCHITECTURE FUNDAMENTALS

Software Release Management

TECHNISCHE UNIVERSITÄT MÜNCHEN



In **Software Release Management**, **Releases** of software products are controlled by 7 dimensions, which identify the release via a well-defined **Version Schema**.

The dimension **Evolution Stages** is about the type and development stage of the product. A distinction is made between the (partly not yet based on the later technology) pre-stages **Wireframe**, **Prototype** and **Proof of Concept** and the production-ready stages (which are based on the target technology) **Walking Skeleton**, **Minimum-Viable Product** and **Full Product**.

In the **Version Number** dimension, the product is identified by three numbers: **Major Version**, **Minor Version**, and **Revision**. The first two refer to the content of the product. The latter refers to the respective **Maturity Level** of the product. The **Maturity Levels** dimension itself defines the maturity of the product within the **Major/Minor Version**: **Alpha** (a, incomplete, unstable), **Beta** (b, complete, unstable), **Release Candidate** (rc, complete, stable), **Release** and **Release Update**.

The dimension **Points-In-Time** tells you whether the current release has the status **Development** (as the product is in the VCS), whether it is a special **Snapshot** (e.g., for extremely time-critical hotfixes or early feedbacks) or if it is a normal **Release**.

The dimension **Product Editions** defines the edition or variant of the product: usually **Community/Enterprise Edition** (with focus on the target market) or **Standard/Professional Edition** (with the focus on the range of functions).

The dimension **Availability Scopes** defines for whom the release of the product is available: **No Availability** (only for the manufacturer internally), **Limited Availability** (for special situations), **Early Availability** (for early adopters among customers) and **General Availability** (for all customers). Usually, the Availability Scopes are used for specific **Maturity Levels**, but there is no necessary connection.

The dimension **Distribution Channels** defines over which channels the release is available: **Bleed Channel** (for "Visionaries" and "Die Hards"), **Edge Channel** (for "Early Adopters"), **Stable Channel** and and **Solid Channel** (for all customers). In any case, the **Distribution Channels** should be directly linked to the branches of the VCS.

Questions

- At which **Maturity Level** in **Software Release Management** does one have an incomplete and unstable functionality?
- At what **Maturity Level** in **Software Release Management** does one have complete but usually still unstable functionality?

Intellectual Content: Version 1.0.8 (2015-11-16), Authored 2015-2019 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.1.7 (2023-04-14), Copyright © 2015-2023 Dr. Ralf S. Engelschall <http://engelschall.com>, All rights Reserved.
Unauthorized Reproduction Prohibited. Licenced to Technische Universität München (TUM) for reproduction in Computer Science lecture contents only.