**Software Release Management**

Table of Contents

[Software Releases 3](#_Toc83393941)

[Release Management 4](#_Toc83393942)

[Objectives: 4](#_Toc83393943)

[Release Management Cycle 5](#_Toc83393944)

[Key Terms 6](#_Toc83393945)

[Process 8](#_Toc83393946)

[Plan Release 9](#_Toc83393947)

[Build Release 9](#_Toc83393948)

[User Acceptance Testing 9](#_Toc83393949)

[Prepare Release 10](#_Toc83393950)

[Deploy Release 10](#_Toc83393951)

[Release Management in Agile + DevOPs 11](#_Toc83393952)

[Release Manager 12](#_Toc83393953)

[Best Practices 13](#_Toc83393954)

[Release Package 14](#_Toc83393955)

[Release Note 15](#_Toc83393956)

[Format 15](#_Toc83393957)

[Release Versioning 16](#_Toc83393958)

[Major Release 16](#_Toc83393959)

[Minor Release 17](#_Toc83393960)

[Revision Release 18](#_Toc83393961)

[Build Release 18](#_Toc83393962)

[Semantic Versioning 2.0.0 19](#_Toc83393963)

[Continuous Integration 20](#_Toc83393964)

[Continuous Deployment and Continuous Delivery 21](#_Toc83393965)

[Benefits of Continuous Delivery 22](#_Toc83393966)

## Software Releases

A **software release** is a launch of new software or the release of a group of features. Releases help the development team **plan** their work and set deadlines for when they need to **launch**.

A release is not just the date on which new technical functionality is introduced. It is the point at which the software is **fully prepared** to delivery a **new customer experience** and support **every customer interaction** associated with it.

## Release Management

**Release management** is the process of planning, building, testing and deploying hardware and software and the version control and storage of software. Its purpose is to ensure a **consistent method of deployment** is followed.

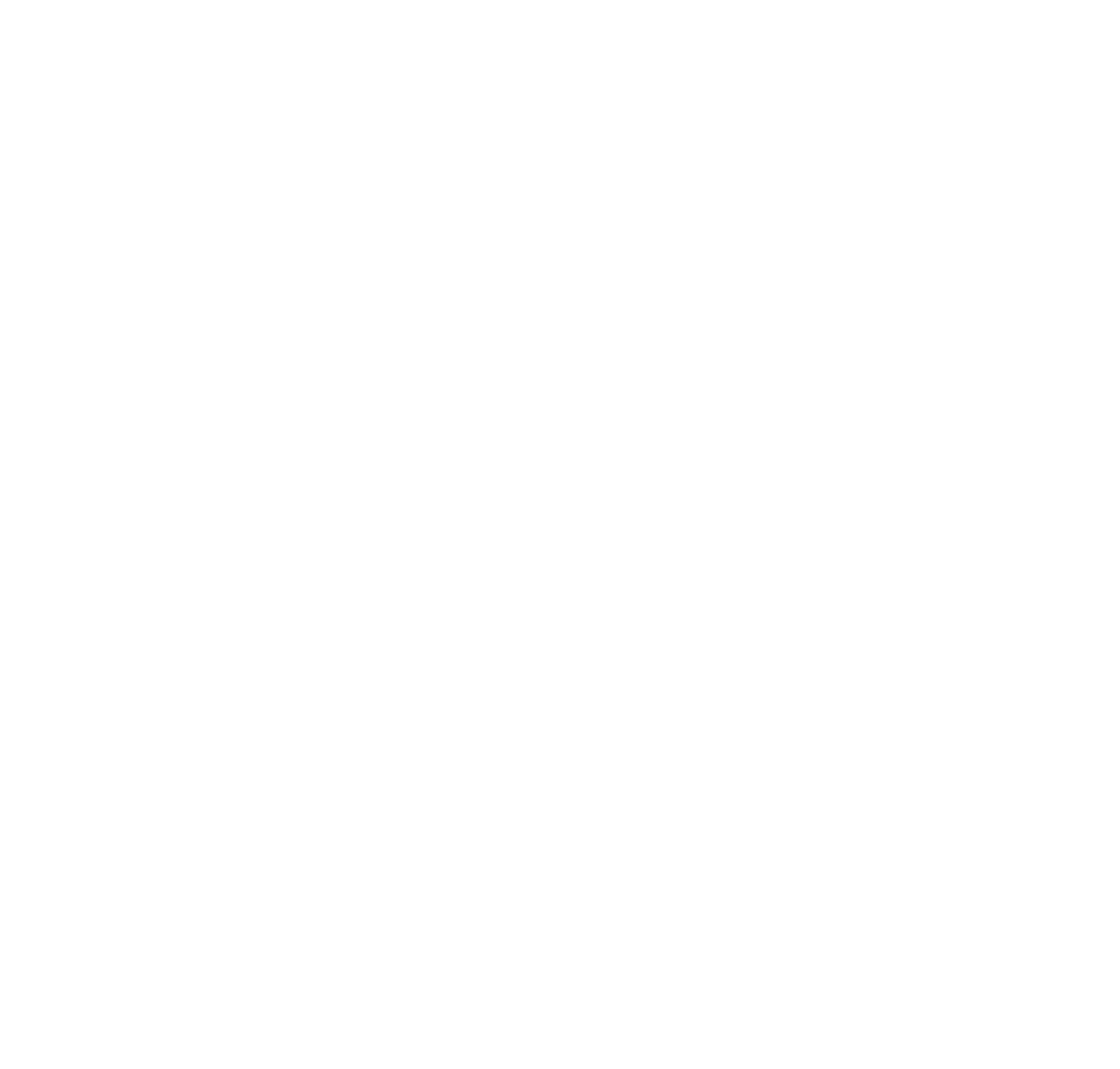
Release management reduces the likelihood of incidents as a result of rollouts and ensures that only tested and accepted versions of hardware and software are installed at any time.

### Objectives:

We want to deploy **applications changes** into **production** without disrupting business. To do this successfully, we must:

* Deploy on time
* Deploy on budget
* Have no or negligible impact on existing customers
* Satisfy the requirements of new customers, competitive pressure and/or technological advances.

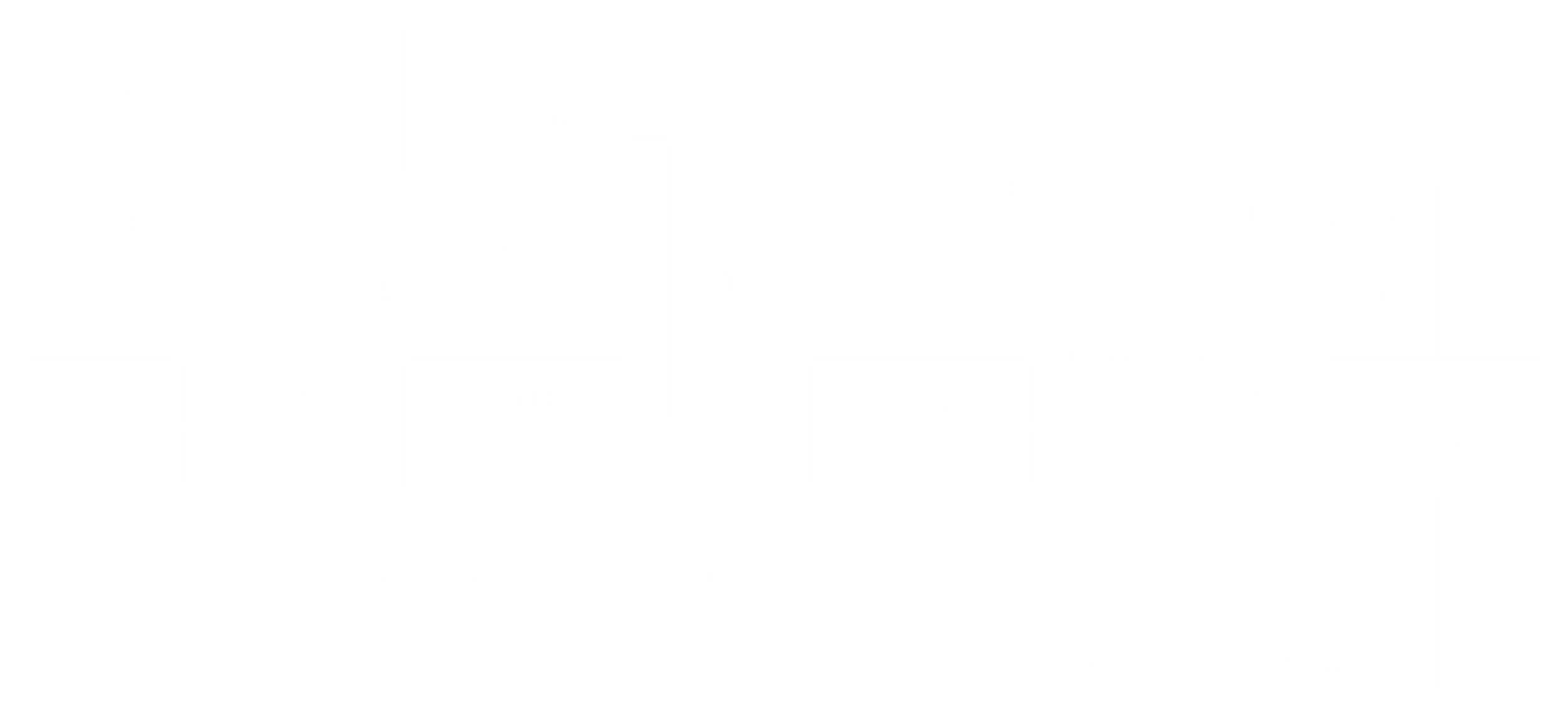
## Release Management Cycle



## Key Terms

* **Development Work Order:**This is a work order for the development or modification of a software application or system.
* **DevOps Team:**They increase coordination between the development and operations functions, creating a separate team. Usually refers to key team members on both the development and operations sides who work to coordinate the two functions.
* **Installation Work Order:**Similar to a development work order, an installation work order is for the installation of a software application, system, or infrastructure component.
* **Product Owner:**The product owner on a development project is the main stakeholder. They usually represent the business or the product’s (eventual) users, and they define the vision for a product.
* **Project Manager:**A project manager takes charge of the direction for a single product. They’re responsible for defining the product roadmap and vision and for negotiating deliverables. Typically, a project manager’s core responsibilities do not extend beyond a single product or closely related family of products.
* **Release Manager:**The role of the release manager is to plan, coordinate, manage, and schedule all the items that comprise a release.
* **Release Policy:**This is a set of rules for how to deploy releases to the live operational environment. Different release policies apply to different releases, depending on such factors as impact and urgency.
* **Release Record:**A release record documents the history of a release, from the planning to the closure of the development process.
* **Release Unit:** This term refers to a set of configuration items that a team simultaneously tests and releases into the live environment to implement approved changes. A configuration item, in turn, is a component of an infrastructure that’s under configuration management, which is the process of making sure that a product’s attributes and performance are consistent with its design, requirements, and operational information.
* **Service Owner:**A service owner takes on high-level accountability for a specific IT service. They’re less concerned with the daily operations needed to deliver the service; that is the role of a service manager.
* **Quality Manager:**A quality manager ensures that a release meets stipulated standards. They may have release managers reporting to them.

## Process



1. First, someone decides **changes need to be made** to an application. This need is **documented** and results in a release.
2. A **developer** sees the changes to be made and **edits the source code** of the application to remedy the issue(s).
3. Once ready, the **code is aggregated and compiled** into a build.
4. The build is deployed in a **QA environment** to be thoroughly tested.
5. The **build is tested**. The testing can be something as simple as seeing if the page loads, called a “smoke test,” or it can be in depth and executed by a tester.
6. After initial testing, if the build is up to snuff, it enters another **non-production-testing environment** (Staging). If there are bugs or it doesn’t meet the criteria, it will be retested until it can be considered ready.
7. Once an application is ready, it is **deployed**, or “released,” into production.

### Plan Release

The **entire release is structured** in the **planning stage**. A workflow is created that both the team and key stakeholders can refer to.

The workflow should explain how the whole release is staged and what each team member will do. This includes:

* Timelines
* Delivery dates
* Requirements
* The overall scope of the project

### Build Release

Next, we **design and build** the actual product based on the requirements outlined in the release plan.

### User Acceptance Testing

Once everything is addressed, we subject the build to **real-world scenario testing**. This can take several iterations, since issues that are found are sent back to the development team.

### Prepare Release

The **finishing touches** are put on the product, including final **quality review** to ensure the build meets the minimum acceptable standards and business requirements. Then the **functional team** validates the findings and finalizes the release. The build must be approved by the **product owner**.

### Deploy Release

The build is released to the **live production environment**. End users and the company may need to be educated about the product again, such as notifications about changes and how to operate new features.

The development team should also meet up to assess the release’s **performance** and discuss how the deployment went. Lingering issues should be identified and documented to be addressed in the next iteration.

## Release Management in Agile + DevOPs

In **Agile**, release units are delivered in every **2 week sprint** and release packages are deployed to production every **10 to 12 weeks**.

When an enterprise adopts DevOPs, the **release manager** coordinates with **environment managers** for testing and deployment planning for release packages. Release management is responsible for coordinating with the DevOps manager to monitor the continuous integration and delivery in the DevOps pipeline.

## Release Manager

* Understands the business needs and their priorities, and under what circumstances those priorities can change.
* Works with business leaders, product owners, IT project teams, and operations staff to ensure every release contains the correct features.
* Changes the release scope or re-prioritize release features according to information from project and portfolio managers.
* Has a clear picture of development dependencies, and how changes to one part of a product can affect the stability of the whole.
* Schedules release unit dependencies into release packages.
* Understands the bandwidth and work capacity of each team involved in development.
* Understands the availability of resources and environments for testing.
* Schedules builds and testing according to team and resource bandwidth and availability.
* Create release plans, including governance and approval requirements.
* Ensure compliance of new releases with governance requirements.
* Optimizes value creation at every step, from feature check-in to deployment.
* Schedules seamless release deployments.

## Best Practices

* Automate as much as possible, both on DevOPs and testing. This cuts down on human cost, allows rapid iterations and reduces mistakes.
* Have clear requirements, and from these, make testable acceptance criteria. There should not be any ambiguity about whether the software is ready.
* Minimize user impact by minimizing or eliminate downtime and testing for regressions before release.
* Make things immutable. Instead of modifying an existing machine, deploy a complete image with the modifications. This avoids bugs due to an unexpected series of actions.

## Release Package

A **release package** contains information about all of the configuration items, such as documentation, test plans, services, etc., associated with a change, and links to the release that is deploying the change.

The changes are recorded in a **change record**, but a single change record might be associated with multiple release packages.

## Release Note

A **release note** is released with the final build. It contains the new enhancements and new issues.

### Format

* **Header -**Name of the document, which carries product name, release number, release date, release note date and version.
* **Overview -**An overview of the product and changes to the recent software version.
* **Purpose -**An overview of the purpose of the release notes which lists the new feature, enhancements and defects of the current build.
* **Issue Summary -**Provides description about the defect.
* **End-User Impact -**Provides information about the end-users impact due to the defect.
* **Contact -**Support contact information.

## Release Versioning

**Versions** and **releases** are denoted by different markings depending on the company, but a general format is to use a **quadruplet** of integers to indicate Major, Minor, Revision and Build Number.

<MAJOR>.<MINOR>.<REVISION>.<BUILD>

Example: 2.4.7.1333

The last of these, the **build number**, is used for **internal tracking** and is made visible to customers.

### Major Release

A **Major Release** is a full product release. It generally contains **new customer-facing functionality** with significant changes to the codebase or marketing or direction of the product.

Scope:

* Any change to the code base that prevents backwards compatibility (e.g. Addition or Removal of features, changes in the DB schema or API commands).
* Any new functionality that is customer facing.
* Any large marketing push that accompanies the product and redirects the product.

Frequency: Market-driver

Audience:

* New customers.
* Existing customers with qualifying contracts
* Existing customers desiring to upgrade to the new feature set

### Minor Release

In a **Minor Release**, we enhance **existing features** due to internal or external requirements.

Scope:

* Minor enhancements and features that do not affect compatibility with its associated major or current minor releases. New features or functionality that does minimally effect the interfaces
* Addition of new features or functions to meet a new sales area that doesn’t affect existing customers of the release. To accompany or communicate a new marketing initiative.
* Error corrections and maintenance.
* A rollup of branched releases.

Frequency: Enhancement or Market driven

Audience:

* New customers.
* Existing customers with qualifying contracts
* Existing customers desiring to upgrade to the new feature set

### Revision Release

A **Revision Release** is sent to the **QA team** for testing. It might contain minor modifications and bug fixes. Once approved, this will be released to customers. If problems are found, another revision must be made.

Scope: Releasing build to SQA for validation

Frequency: As necessary, but typically every 1 to 3 months.

Audience: Existing customers with qualifying contracts.

### Build Release

Every time **anything at all** changes in the code, the **Build Version** number is incremented.

Scope: Resolves a particular defect, typically of a critical Severity level with no viable workaround.

Frequency: Extensively used during internal development.

Audience: Internal developers; Internal verification

### Semantic Versioning 2.0.0

Another popular versioning scheme is [**Semantic Versioning**](https://semver.org/). It follows this pattern:

MAJOR.MINOR.PATCH

The only new part here is the **Patch Version**. This is a **small modification**, such as a memory issue fix or performance increase, which is **released to customers**. Only the part of the code that has been modified is released.

## Continuous Integration

**Continuous Integration** is a development practice that requires developers to **integrate** their code into a **shared repository** several times a day. Each check-in is then **verified** by an **automated build**, allowing teams to detect problems easily. By integrating regularly, problems can be detected quickly and located easily.

Benefits:

* Say goodbye to long and tense integrations
* Increase visibility enabling greater communication
* Catch issues early and fix them early
* Spend less time debugging and more time adding features
* Build a solid foundation
* Stop waiting to find out if your code’s going to work
* Reduce integration problems allowing you to deliver software more rapidly

Steps:

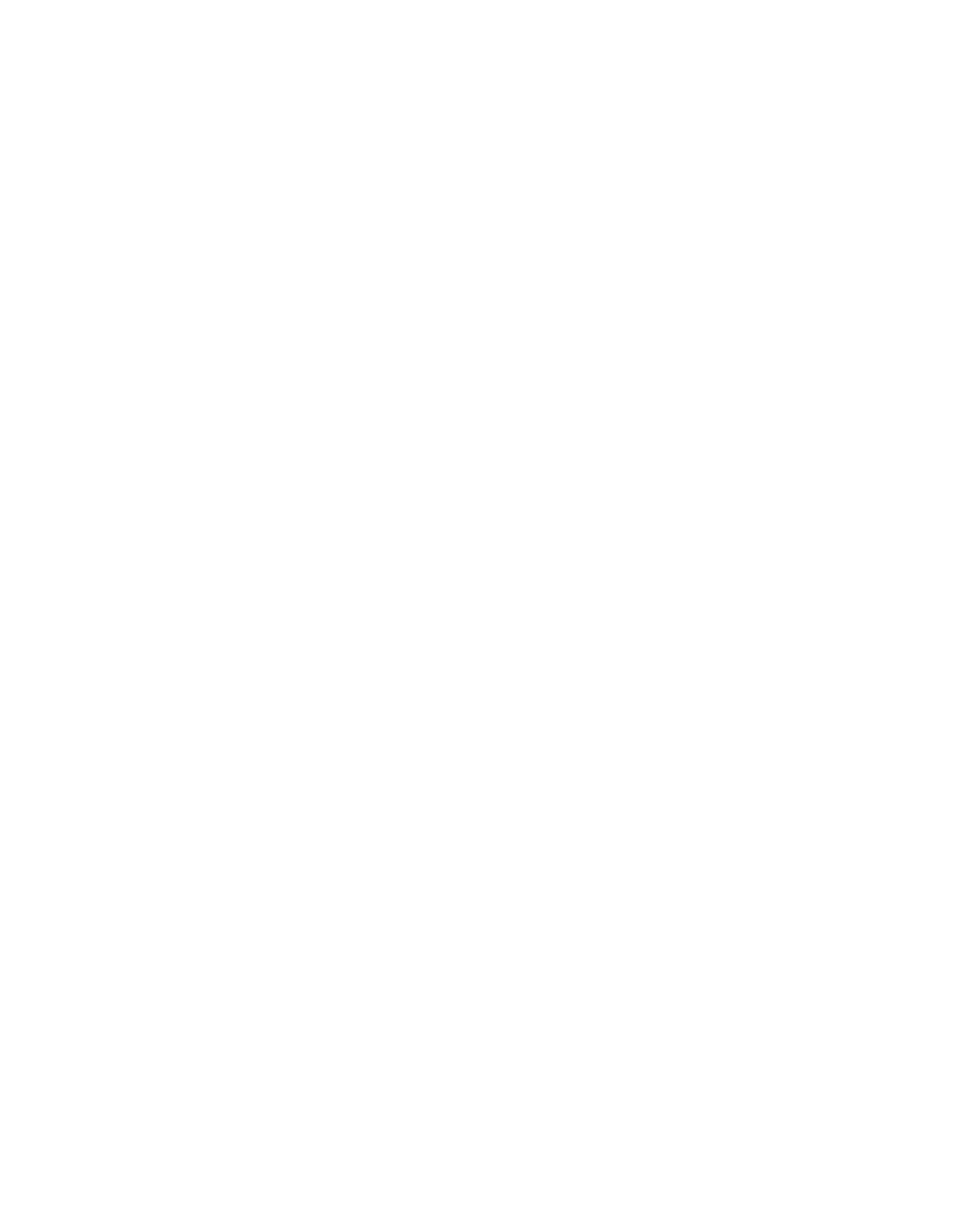
* Developers check out code into their private workspaces
* When done, commit the changes to the repository
* The CI server monitors the repository and checks out changes
* The CI server builds the system and runs unit and integration tests
* The CI server releases deployable artefacts for testing
* The CI server assigns a build label to the version of the code it just built
* The CI server informs the team of the successful build
* If the build or tests fail, the CI server alerts the team
* The team fixes the issue at the earliest opportunity
* Continue to continually integrate and test throughout the project

## Continuous Deployment and Continuous Delivery

**Continuous Deployment** is a software development practice where changes in code are **automatically prepared for release** to production.

Continuous Deployment extends upon Continuous Integration by deploying all code changes to a **testing environment** and/or a **production environment** after the build stage. It **automates** the entire software release process. Every revision triggers an automated flow that builds, tests and then stages the update.

After all the testing is done, the last step just before the software is deployed to production is a **manual acceptance**. This is what happens with **Continuous Delivery**. With **Continuous Deployment** however, even this step is automated, so that the software goes into production automatically.



### Benefits of Continuous Delivery

* Automate the Software Release Process
* Improve Developer Productivity
* Find and Address Bugs Quicker
* Deliver Updates Faster