CSE 4513

Lec – 14

Open Source Compliance

Open-source Software (OSS) License

- What is it????
  - ✓ OSS license is a type of license for computer software and other products that allows the source code, blueprint or design to be used, modified and/or shared under defined terms and conditions
  - ✓ OSS license is a legal and binding contract between the author and the user, declaring that the software can be used in commercial applications under certain conditions
  - ✓ In general, OSS licenses makes the source code available under terms that allow for modification and redistribution without having to pay the original author.

Impact of FOSS Licenses

FOSS licenses may impact:

- Use of the software

- Modification of the software

- Maintenance of the software

- Distribution of the software and its derivatives

- Intellectual property rights (IPR)

PERMISSIONS GRANTED

FOSS licenses may permit:

- Modification of the source code
- Recompilation of the software
- Redistribution of the original source code, modified source code and/or binaries
- Integration of the software with proprietary software

Open Source Licenses : Permissive

in many cases, many open source developers felt the best way to make software "free", is to make the licenses the least restrictive as possible.

- ✓ We call these licenses "**permissive**" licenses.
- ✓ Permissive licenses have very little restrictions and obligations, letting you effectively do whatever you want at no cost (e.g., copy it, distribute it, sell it, build your own products on top of it, etc.)

E.g., **MIT License:**

"Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions…"

Open Source Licenses : Copyleft (Protective)

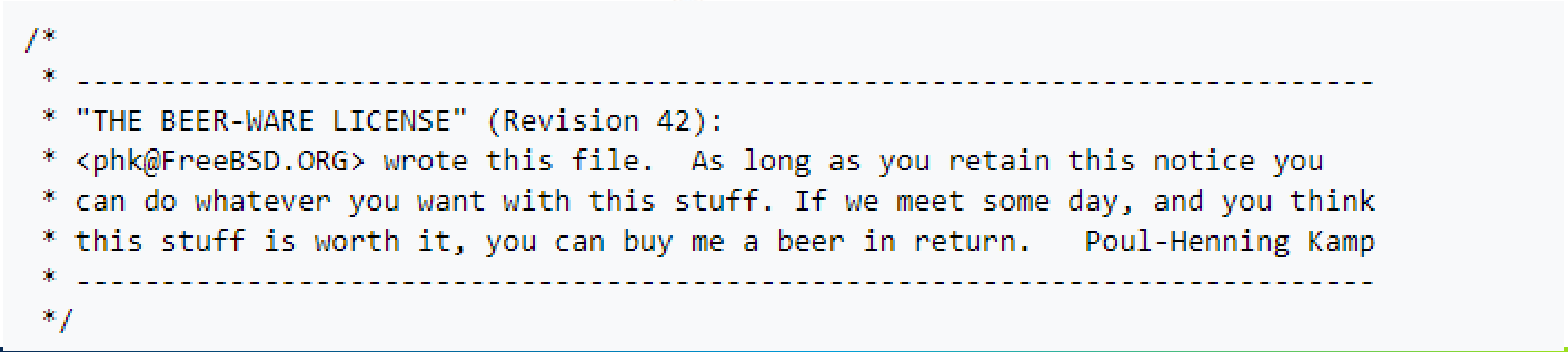
In other cases, some open source developers felt the best way to make software "free" is to include a "viral" element to ensure software always remains "free" and infects other software to make other software "free" too..

- ✓ We call these licenses "**viral**" or "**copyleft**" licenses.
  - (1) If you distribute the "copyleft" software, often you must disclose source code.
  - (2) If you build software that is based on the "**copyleft**" software, often you must also disclose your proprietary source code.

E.g., General Public License (GPL), Affero General Public License(AGPL)

BEERWARE LICENSE.

Many licenses are written by programmers, and they have a sense of humor…



```
/*
 * ----------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <phk@FreeBSD.ORG> wrote this file.  As long as you retain this notice you
 * can do whatever you want with this stuff. If we meet some day, and you think
 * this stuff is worth it, you can buy me a beer in return.   Poul-Henning Kamp
 * ----------------------------------------------------------------------------
 */
```

THE FACT

- ✓ There are over 25 million repositories on GitHub, over 430,000 projects on SourceForge
- ✓ These days a developer will do a Google search, find five open-source products that fit his[/her] need and the next thing you know one of them is in a product.
- ✓ Many software developers work under the following false misconception:
  - "OSS is freely available, so I can use it without any kind of restriction".
- ✓ This is completely wrong: many OSS, even when freely available, are in fact governed by their own licensing conditions, which imply, in most cases and depending on the use of the OSS, strict contractual license restrictions.

Mitigating Risks Through Compliance Practices

- Identification of the origin and license of used software
- Identification of license obligations
- Fulfillment of license obligations when product ships

What is FOSS Compliance

- ✓Compliance means that users of FOSS must observe all the copyright notices and satisfy all the license obligations for the FOSS they use.
- ✓In addition, companies using FOSS in commercial products, while complying with the terms of FOSS licenses, want to protect their intellectual property and that of third party suppliers from unintended disclosure.

What is FOSS Compliance

Open Source Compliance refers to the aggregate of

- policies
- processes
- training
- tools

that enables an organization to effectively use open source software and contribute to open communities while

- respecting copyrights,
- complying with license obligations, and
- protecting the organization's intellectual property and that of its customers and suppliers.

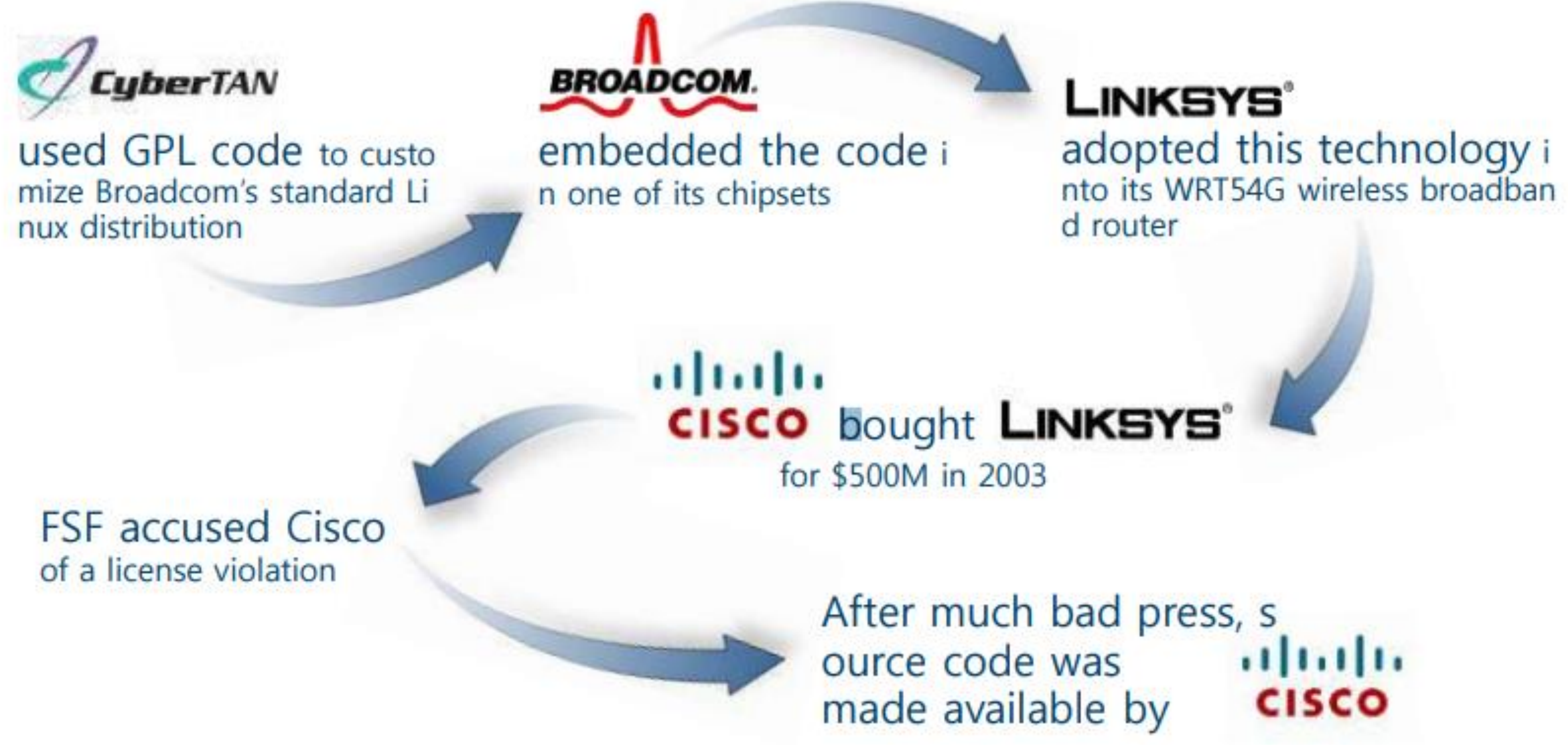
COMPLIANCE OBJECTIVES

- ✓ Comply with third party software supplier contractual obligations in light of FOSS licensing obligations
- ✓ Facilitate effective usage of FOSS in commercial products
- ✓ Protect commercial product differentiation while complying with FOSS contractual obligations

Compliance Benefits

- ✓ Increased understanding of the benefits of FOSS and how it impacts your organization
- ✓ Increased understanding of the costs and risks associated with using FOSS
- ✓ Better relations with the FOSS community and FOSS organizations
- ✓ Increased knowledge of available FOSS solutions
- ✓ Be prepared for possible acquisition, sale, new product or service release, where compliance assurance is mandatory before the completion of any of these transaction
- ✓ Improve your overall FOSS strategy using the results from your compliance program

The Trail of the CISCO Compliance Problem



CyberTAN used GPL code to customize Broadcom's standard Linux distribution

BROADCOM embedded the code in one of its chipsets

LINKSYS adopted this technology into its WRT54G wireless broadband router

CISCO bought LINKSYS for $500M in 2003

FSF accused Cisco of a license violation

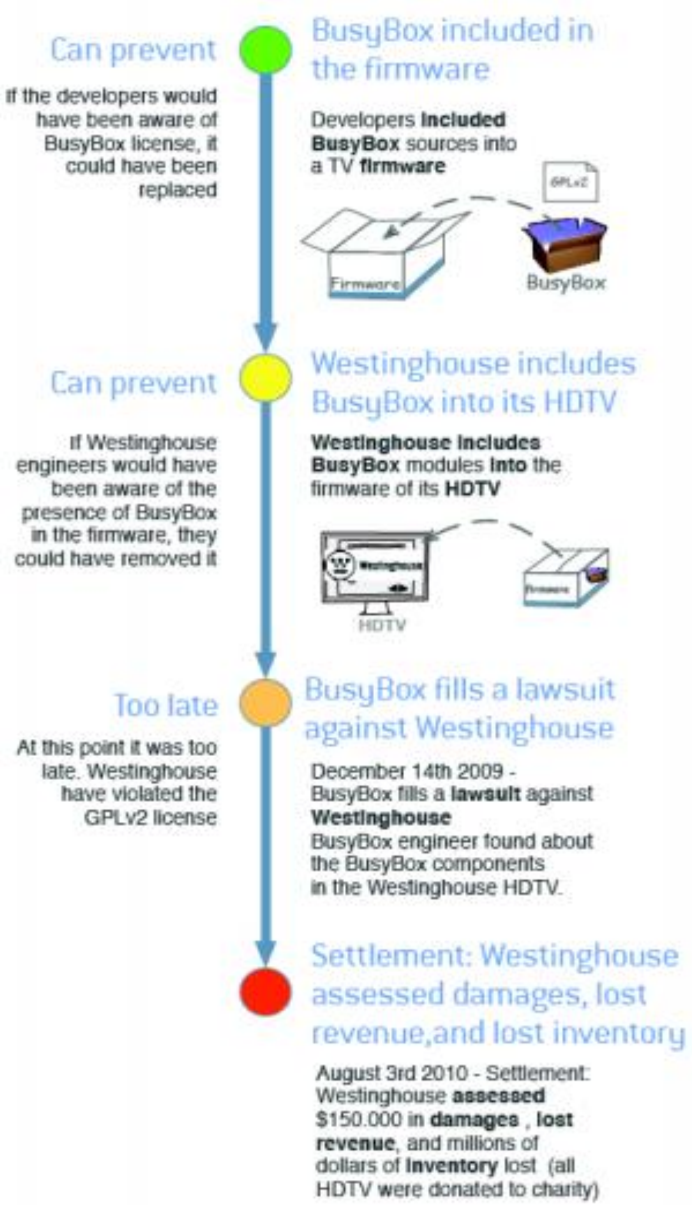
After much bad press, source code was made available by CISCO

How did this story end?

- ✓Major loss of Cisco's Intellectual Property rights and competitive advantage.
- ✓Loss of revenue est. $50 M

Lessons Learned from Compliance Disputes

- ✓ Company to publish licensing notice on their website
- ✓ Company to provide additional notices in product publications
- ✓ Company to make available the complete and corresponding source code used in their product freely available on its website
- ✓ Company to cease binary distribution of the FOSS in question until it has published complete corresponding source code on its web site
- ✓ Company to pay an undisclosed amount of financial consideration to the plaintiffs
- ✓ Company to make available the complete and corresponding source code used in their product, releasing code that contains their product differentiation as open source under the GPL.



Can prevent

If the developers would have been aware of BusyBox license, it could have been replaced

BusyBox included in the firmware

Developers **included** **BusyBox** sources into a TV **firmware**

Can prevent

If Westinghouse engineers would have been aware of the presence of BusyBox in the firmware, they could have removed it

Westinghouse includes BusyBox into its HDTV

Westinghouse includes **BusyBox** modules **into** the firmware of its **HDTV**

Too late

At this point it was too late. Westinghouse have violated the GPLv2 license

BusyBox fills a lawsuit against Westinghouse

December 14th 2009 - BusyBox fills a **lawsuit** against **Westinghouse**
BusyBox engineer found about the BusyBox components in the Westinghouse HDTV.

Settlement: Westinghouse assessed damages, lost revenue,and lost inventory

August 3rd 2010 - Settlement: Westinghouse **assessed** $150.000 in **damages** , **lost** **revenue**, and millions of dollars of **inventory** lost (all HDTV were donated to charity)

Case was settled early August 2010 and included:

- ~ 150,000 USD in damages, lost revenue and lawyer fees
- Millions $ in inventory lost (HDTV using busy box were donated to charity)

Lessons Learned from Compliance Disputes

✓ In almost all cases, the failure to comply with the FOSS license obligations has also resulted in:

- Public embarrassment
- Negative press
- Damaged relationships with some of their customers, suppliers and most notably the FOSS community

Ensure Compliance Prior to Product Shipment

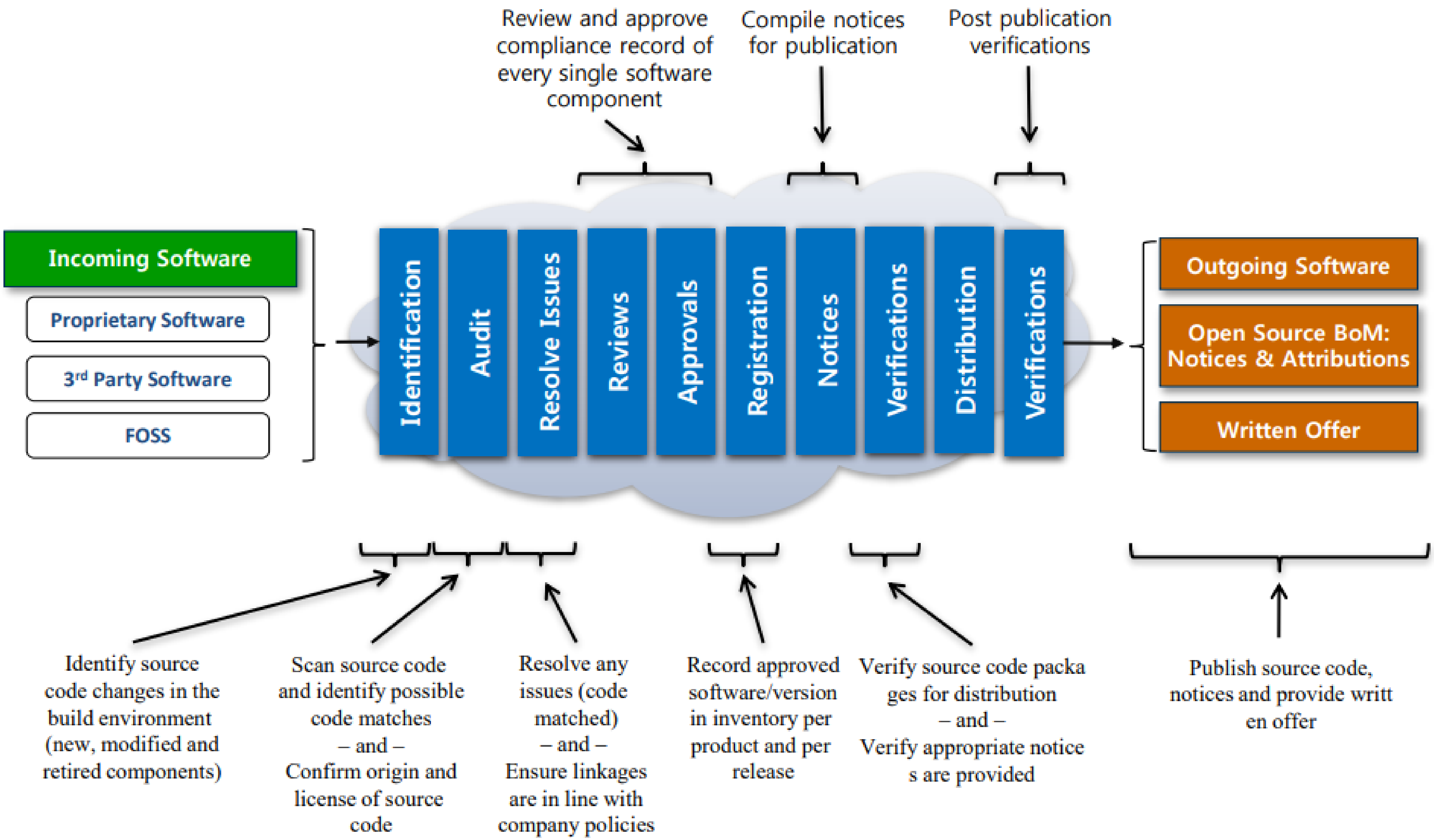
- ✓ To avoid being successfully challenged with regard to FOSS compliance, companies must make compliance a priority before product ship.

- ✓ Companies must establish and maintain consistent compliance policies and procedures and ensure that FOSS licenses, proprietary and 3rd party licenses co-existence well before shipment.

What need to be ensured

- companies need to implement an end-to-end FOSS management infrastructure that will ensure:
  - ✓ Identify all FOSS it is using in its products
  - ✓ Collect the applicable FOSS licenses for review by the legal department
  - ✓ Develop FOSS use and distribution policies
  - ✓ Institutionalize FOSS and compliance training to ensure that all employees are aware of the legal risks involved with using FOSS and aware of company policies
  - ✓ Ensure that your software vendors, suppliers and subcontractors are adhering to FOSS license requirements
  - ✓ Furthermore, companies need to know not only which FOSS they are using, but also how they are using them.

Compliance Management End-to-End



Review and approve compliance record of every single software component

Compile notices for publication

Post publication verifications

Incoming Software

Proprietary Software

3rd Party Software

FOSS

Identification

Audit

Resolve Issues

Reviews

Approvals

Registration

Notices

Verifications

Distribution

Verification

Outgoing Software

Open Source BoM: Notices & Attributions

Written Offer

Identify source code changes in the build environment (new, modified and retired components)

Scan source code and identify possible code matches
– and –
Confirm origin and license of source code

Resolve any issues (code matched)
– and –
Ensure linkages are in line with company policies

Record approved software/version in inventory per product and per release

Verify source code packa ges for distribution
– and –
Verify appropriate notice s are provided

Publish source code, notices and provide writt en offer

Compliance End-to-End Process

Compliance due diligence involves the following:

- ✓ FOSS used in the product has been identified, reviewed and approved
- ✓ The product implementation includes only the approved FOSS
- ✓ FOSS used in the product have been registered in the FOSS inventory system
- ✓ All obligations related to the use of licensed material have been identified
- ✓ Appropriate notices have been provided in the product documentation: these include a written offer to provide source code, attributions and copyright notices
- ✓ Source code including modifications (when applicable) have been prepared and ready to be made available once the product ships
- ✓ Verifications of all the steps in the process

Elements of FOSS Compliance Management

✓Identification of FOSS

✓Auditing source code

✓Resolving any issues uncovered by the audit

✓Completing reviews

✓Receiving approval to use FOSS

✓Updating software inventory

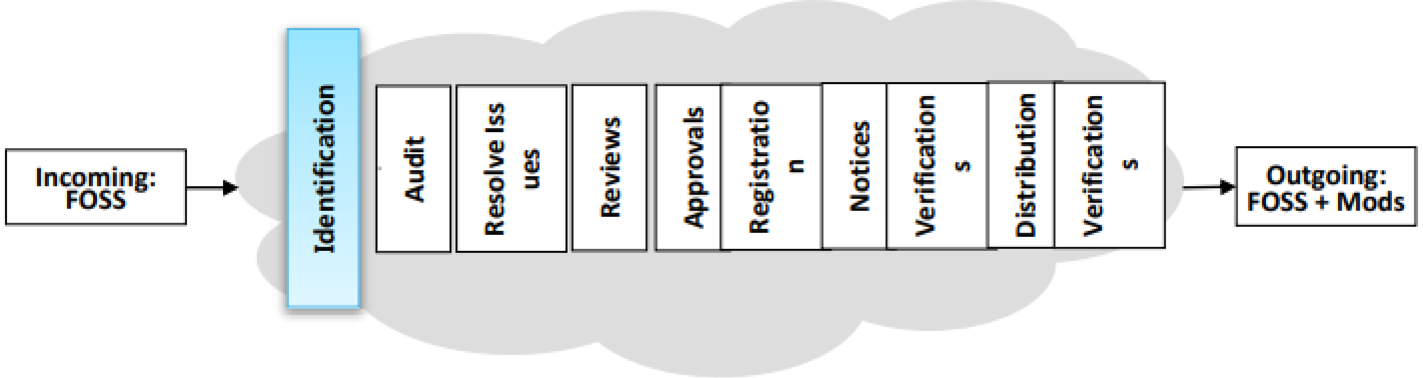
✓Updating end user documentation

✓Performing verification of all previous steps prior to distribution

✓Distributing FOSS including any modifications (if any) when Applicable

✓Performing final verifications in relation to distribution

IDENTIFICATION OF FOSS



Pre-requisites:

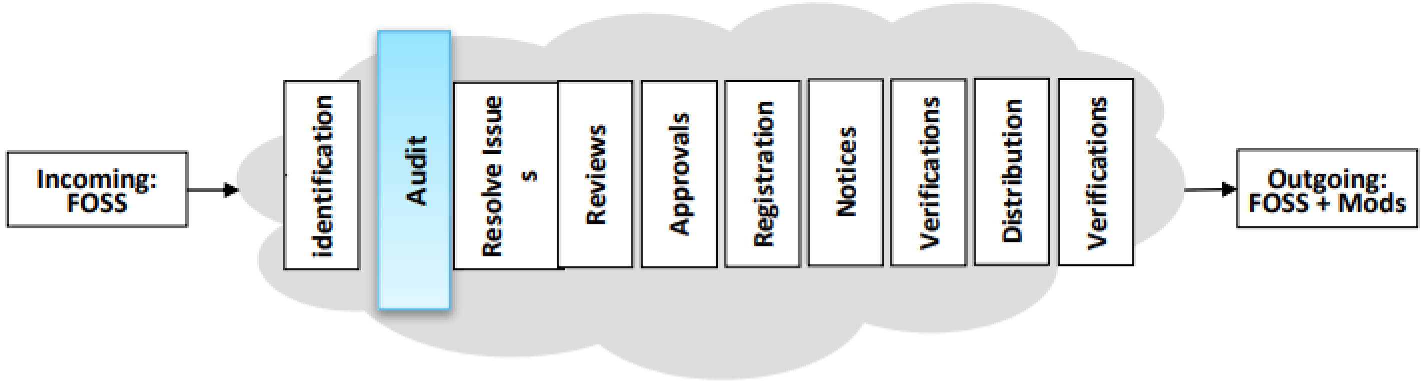
One of the following conditions is met:

- A discovery of a FOSS being used via a platform scan
- A discovery of a FOSS being used as part of third party software

Outcome:

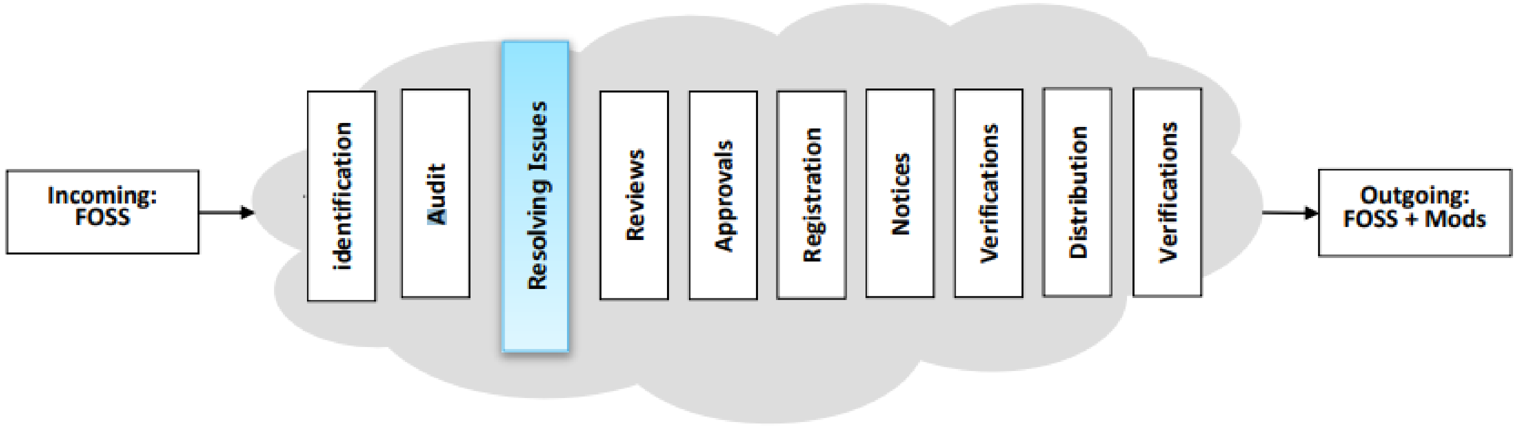
- A compliance record is created (or updated) for the FOSS
- An audit is requested to scan the source code

AUDITING SOURCE CODE



- ✓ It consists of scanning the source code using automated source code analysis tools to discover source code matching FOSS source code
- ✓ The goals with the audit are to:
  - Identify the bill of material (BoM) of the component in question
  - Confirm the origin(s) of the source code
  - Flag dependencies, code matches and licensing conflicts
  - Understand the licenses that govern its use, modification and distribution
  - Identify the obligations of the various licenses

Resolving Issues



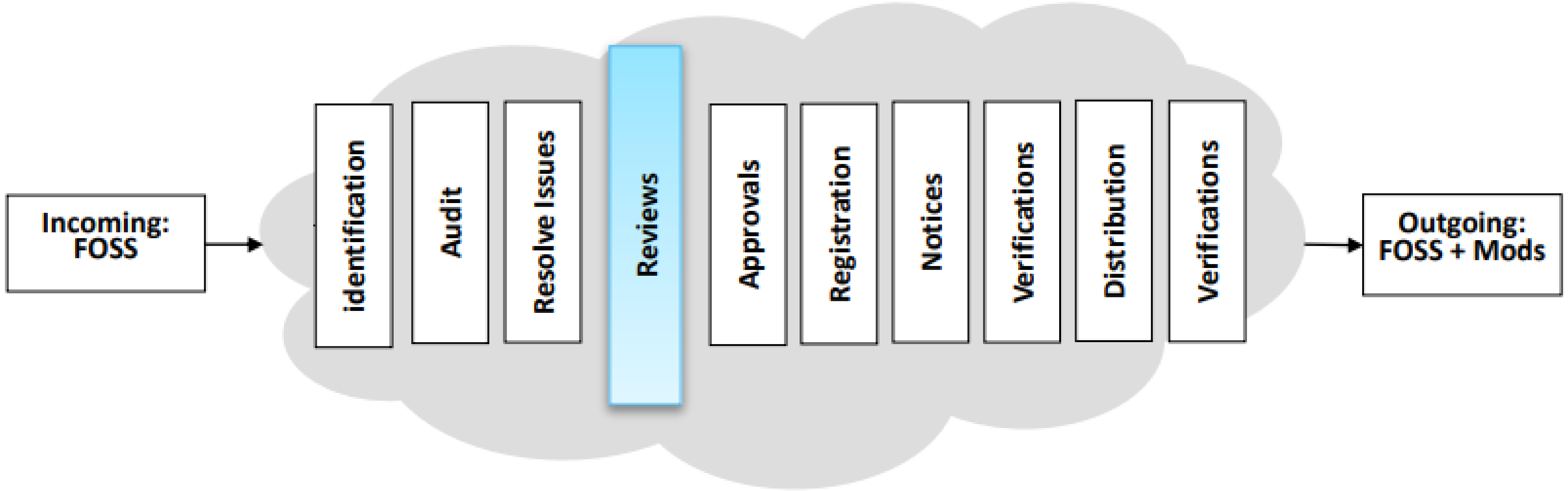
Pre-requisites:

- A source code scan has been completed
- An audit report is generated identifying the origins and licenses of the source code and flagging source code files that were not identified and that need further investigation

Outcome:

- A resolution for each of the flagged files in the report and a resolution for any flagged license conflict

REVIEWS



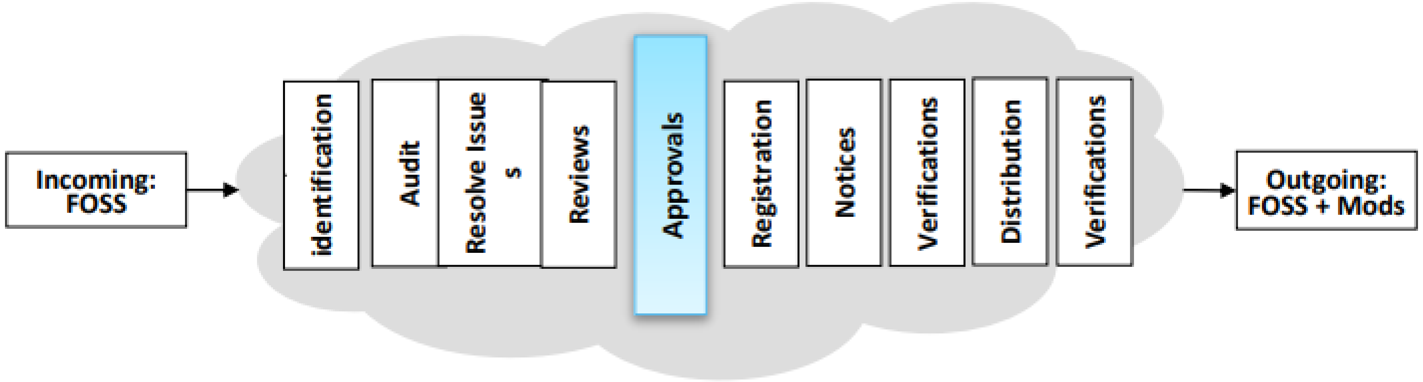
Pre-requisites:

- Source code has been audited
- All identified issues have been resolved

Outcome:

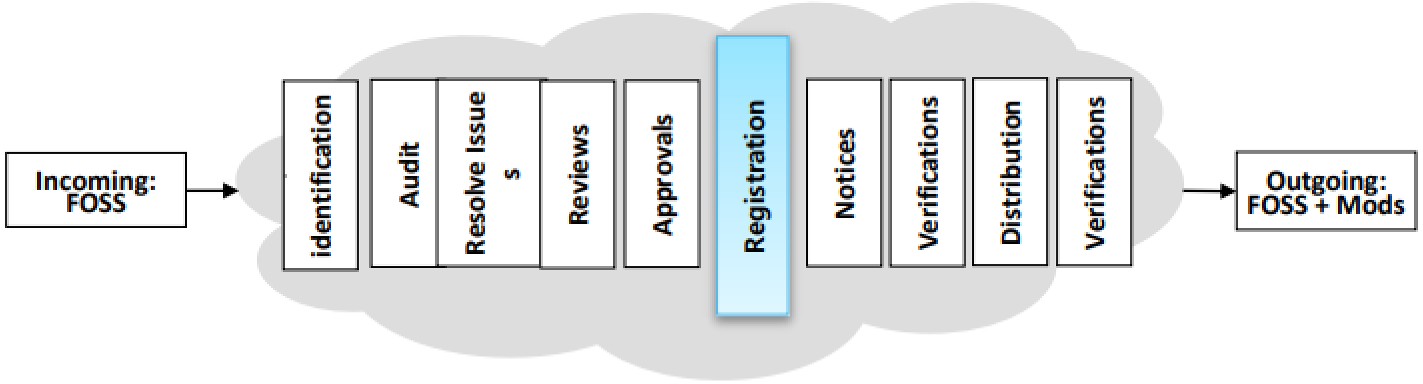
- Open Source Review Board(OSRB) members perform an architecture review and a linkage analysis for the specific component and mark it as ready for the next step (i.e. Approval) if no issues were uncovered

APPROVALS



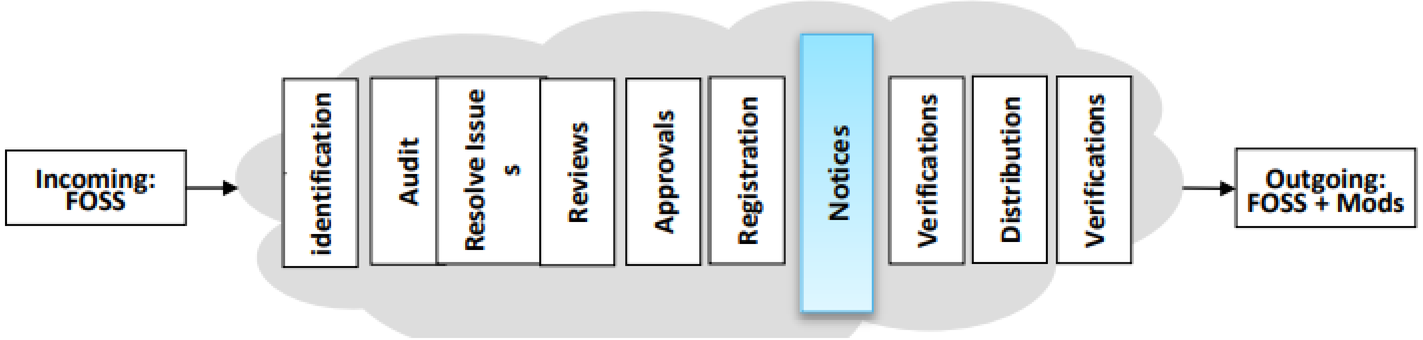
- In this step, the software component is either approved for usage in the product or not.
- The approval comes from the OSRB.

REGISTRATION



- Once is software component has been approved for usage in a product, its compliance ticket will be update to reflect the approval and it will be added to the software inventory that tracks FOSS that used in products.

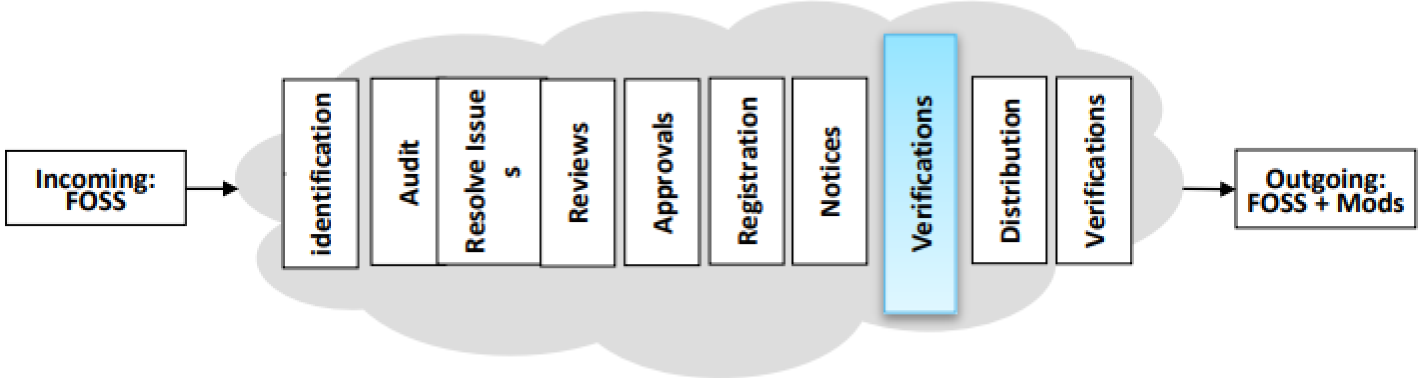
NOTICES



Companies using FOSS in a commercial product must:

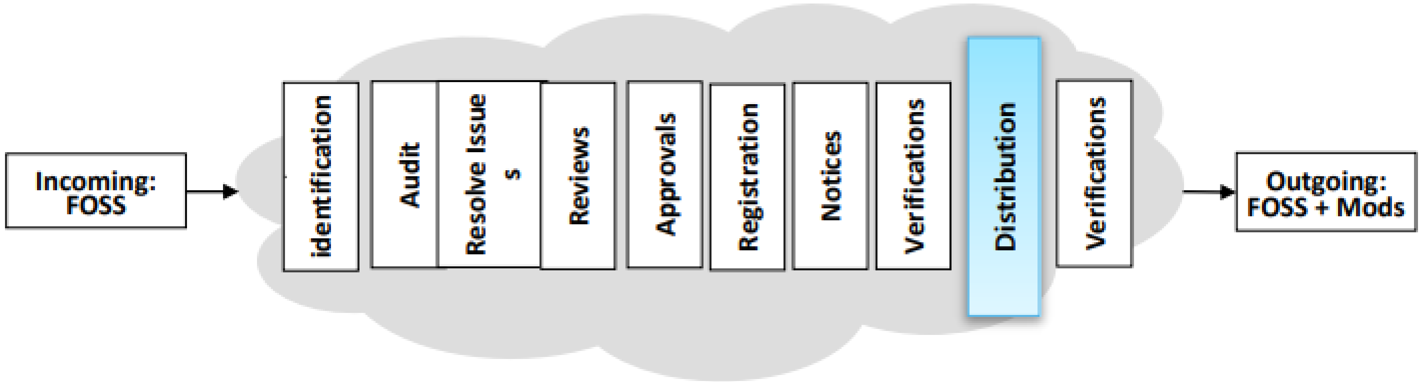
- ✓ Acknowledge the use of FOSS by providing full copyright and attribution notices
- ✓ Inform the end user of their product on how to obtain a copy of the FOSS source code (when applicable, for example in the case of GPL and LGPL)
- ✓ Reproduce the entire text of the license agreements for the FOSS code included in the product.

Pre-Distribution Verifications



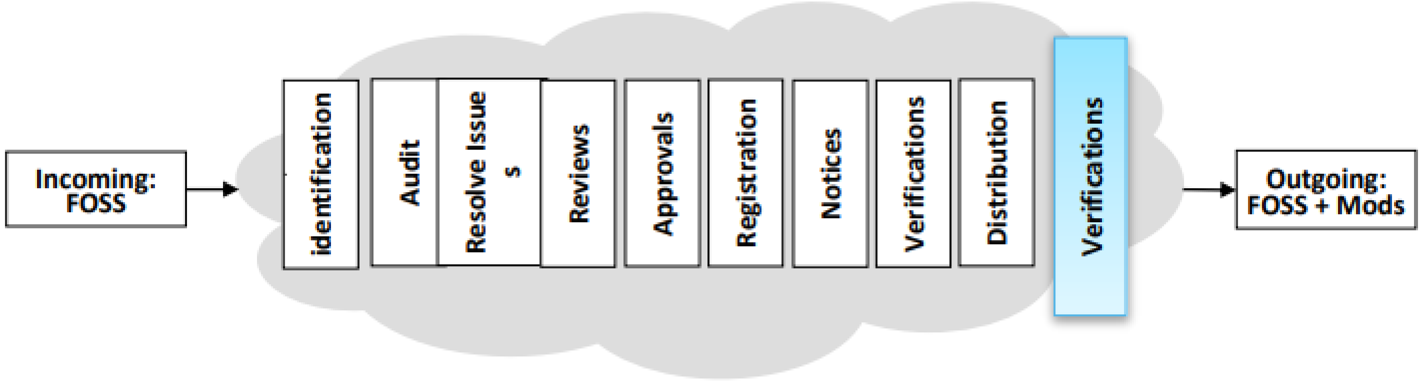
- Part of the pre-distribution verifications is to ensure that:
  - ✓ FOSS packages destined for distribution have been identified and approved
  - ✓ The source code packages (including modifications) have been verified to match the binary equivalence shipping in the product
  - ✓ All appropriate notices have been included in the product documentation to inform end-users of their right to request source code for identified FOSS

DISTRIBUTION



- Once all pre-distribution verifications have been completed, the next step is to upload the FOSS packages to the distribution web site, identified with labels as to which product and version it corresponds to.

FINAL VERIFICATIONS



Pre-requisites:

- The source code is published on the web site

Outcome:

- Verifications that the source code is:
  - ✓ Uploaded correctly
  - ✓ Corresponds to the same version that was approved
  - ✓ Accessible for download for the public

GENERAL GUIDELINES

- Request formal approval for each open source software you are using in product or in SDK (refer to your company’s Usage Policy)
- Save the web site from which you downloaded the open source package and save a mint copy of the package you downloaded
- Consult with your manager when you upgrade your open source software version. License changes can occur between versions.
- Don’t change or eliminate existing comments in headers
- Do not re-name open source modules
- Do not discuss coding or compliance practices with persons outside the company

General Guidelines

- Do not copy/paste FOSS code into proprietary or third party source code or vice versa without OSRB approval.
- Mixing of different FOSS licenses in a derivative work must be avoided.
- When in doubt, always refer to the FSF resource page on license compatibility available at http://www.fsf.org/licensing/licenses/index_html.
- Do not remove or in any way disturb existing FOSS licensing copyrights or other licensing information from any FOSS components that you use.
- All copyright and licensing information is to remain intact in all FOSS components.

SW RELEASE MANAGEMENT..

What is a SW Release

- ✓ A SW release is the launch of a new SW or a combination of features that will provide value to customers or users.
- ✓ For your customers, it's a promise of new value they can look forward to embedding in their everyday work and activities.
- ✓ For your internal teams, a release helps them plan their work and when they'll be needed to launch great products.

A release is not just the act of providing access to new technical functionality. Instead, think of a release as the date when the company is ready to deliver a new customer experience and support every customer interaction point associated with it.

What is 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 that a consistent method of deployment is followed.

- ✓ It 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.

THE OBJECTIVES RELEASE MANAGEMENT

- ✓ Deploy application changes into production without disrupting the business.

- ✓ To be classified as successful, the release must meet the following objectives:

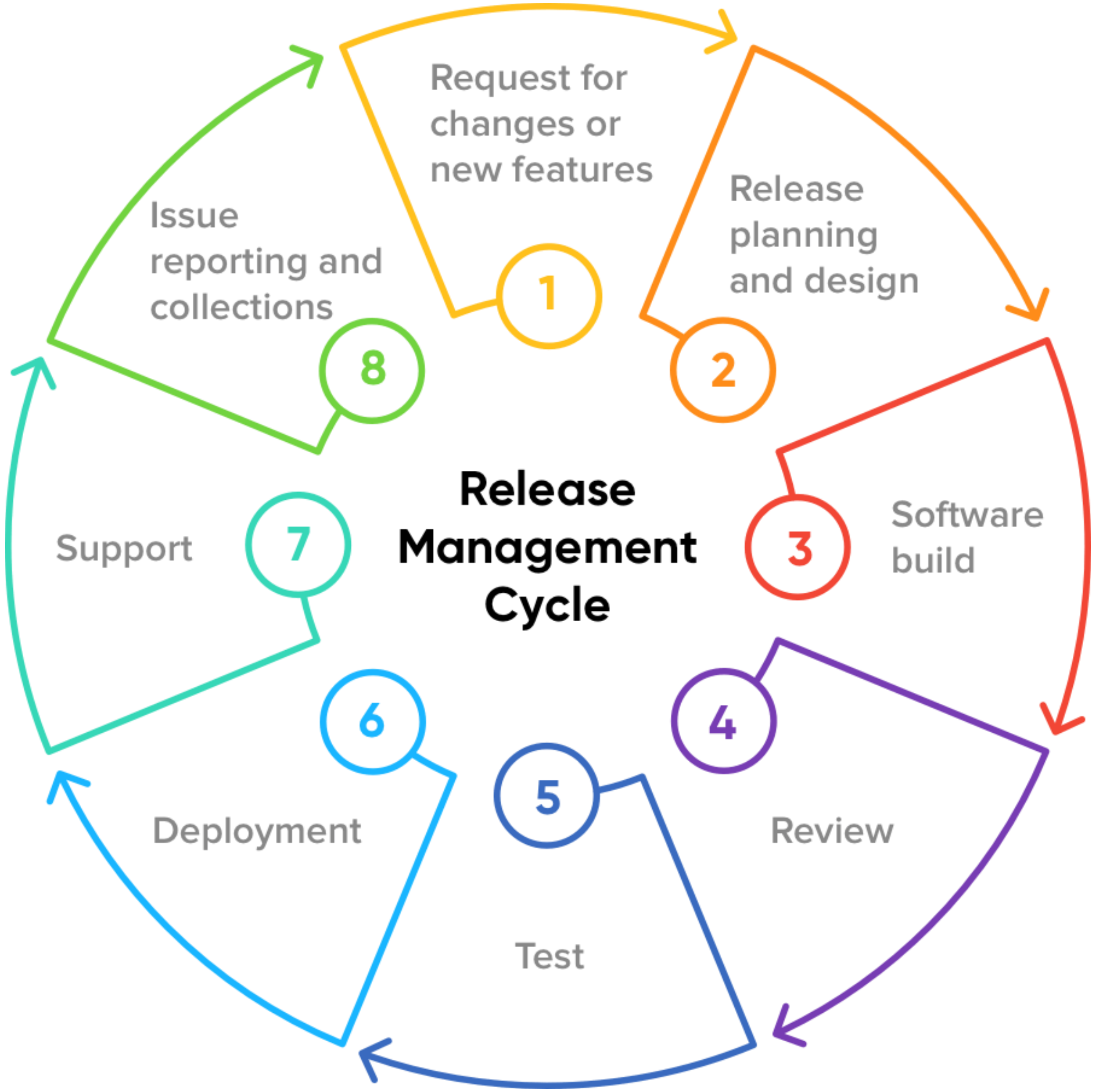
  (a) Deployed on time

  (b) Deployed on budget

  (c) Have no or negligible unwanted impact on existing customers

  (d) Satisfy the requirements of new customers, competitive pressure and/ or technological advances.

Release Management Cycle



1 Request for changes or new features

2 Release planning and design

3 Software build

4 Review

5 Test

6 Deployment

7 Support

8 Issue reporting and collections

Release Management Cycle

Key Terms in Release Management

To master release management, you’ll need to understand commonly used terms.

Development Work Order: This is a work order for the development or modification of a software application or system.

DevOps Team: is to 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.

Key Terms in Release Management

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 various factors as impact and urgency.

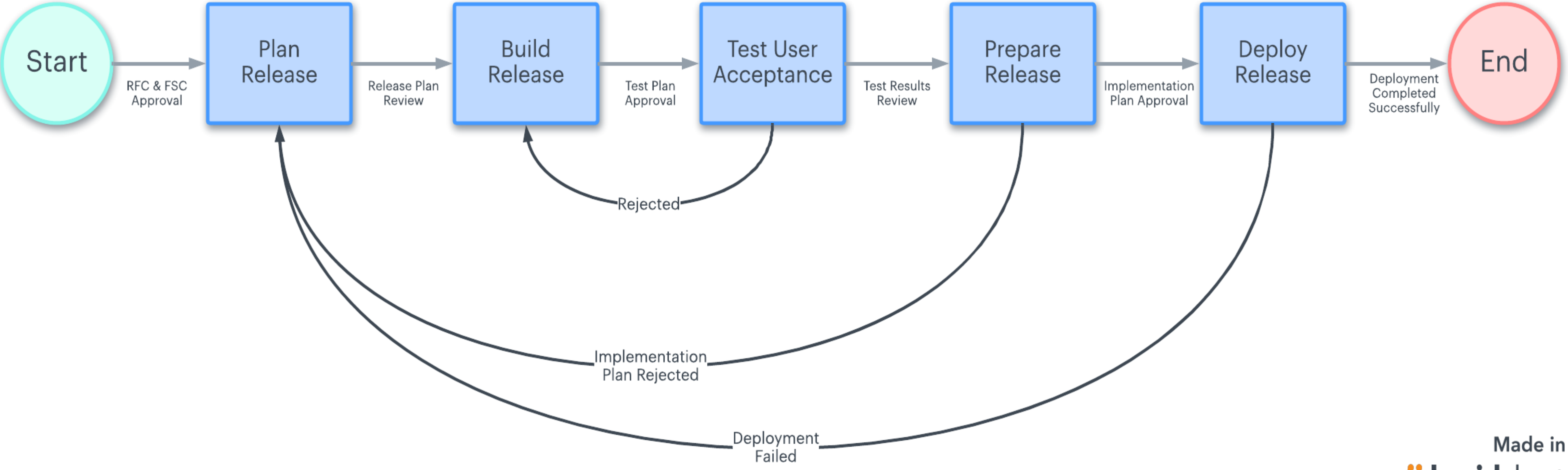
Release Record: A release record documents the history of a release, from the planning to the closure of the development process.

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.

Quality Manager: A quality manager ensures that a release meets required standards. They may have release managers reporting to them.

The Process

Release Management Process



Made in Lucidchart

The Process

- First, someone decides changes need to be made to an application. This need is documented and results in a release.
- A developer sees the changes to be made and edits the source code of the application to remedy the issue(s).
- Once ready, the code is aggregated and compiled into a build.
- The build is deployed in a QA environment to be thoroughly tested.
- 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.
- 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.
- Once an application is ready, it is deployed, or “released,” into production.

The Process – Plan Release

- ✓ The planning stage may be the most time intensive as this is where your entire release is structured from start to finish.
- ✓ During this stage, create a workflow that both your team and key stakeholders can refer to throughout a release.
- ✓ The later in the development cycle an error is found, the more expensive it is to fix.
- ✓ The workflow should explain at a glance how the whole release is staged and how each team member plays a part. Your release plan should include:
  - Timelines
  - Delivery dates
  - Requirements
  - The overall scope of the project

The Process – Build Release

- ✓ With the release plan finalized, you can start designing and building the product for release.
- ✓ This is the actual “development” of the product based on the requirements outlined in the release plan.
- ✓ Once all the issues that may have come up are addressed, it’s time to subject the build to real-world scenario testing.
- ✓ This could take several iterations and allows the team to identify any bugs or issues that may arise in a real-world environment.
- ✓ As issues are identified, the build is sent back for development at stage two. In other words, within the iterative release management process, the work may flow from stage two to stage three and back again until the release is approved.

The Process – UAT

Already Discussed

The Process – Prepare Release

- ✓ This step is to put the finishing touches on the product, taking into account everything that was learned in UAT.
- ✓ Release preparation also includes a final quality review by the QA team.
- ✓ During the review, the QA team will conduct final checks to ensure the build meets the minimum acceptable standards and business requirements outlined in the release plan.
- ✓ Once the review is completed, the functional team will validate the findings and finalize the release for deployment.
- ✓ Before the build can deploy into a live environment, it must be approved by the product owner.

The Process – Deploy Release

- ✓ It’s time to release your product into the wilds of the live production environment.
- ✓ Besides simply sending the build out into production, the deployment stage also includes messaging and education on the product to both the end user and your company at large.
- ✓ For instance, users should be notified of changes with the release and how to operate within the new features. Depending on how significant the changes were, you may need to provide robust and ongoing training to get everyone up to speed.
- ✓ Finally, during the deployment stage, the development team should meet to assess the release’s performance and discuss how the deployment went.
- ✓ If there are any lingering issues, those should be identified and documented for the team to address in the next iteration..

Release Management in Agile+DevOps

- ✓ In Agile methodologies, agile release trains align to value streams to deploy releases.
- ✓ Release units are delivered every 2-week sprint and release packages are deployed to production every 10-12 weeks..
- ✓ As an enterprise adopts DevOps, the release manager also 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.

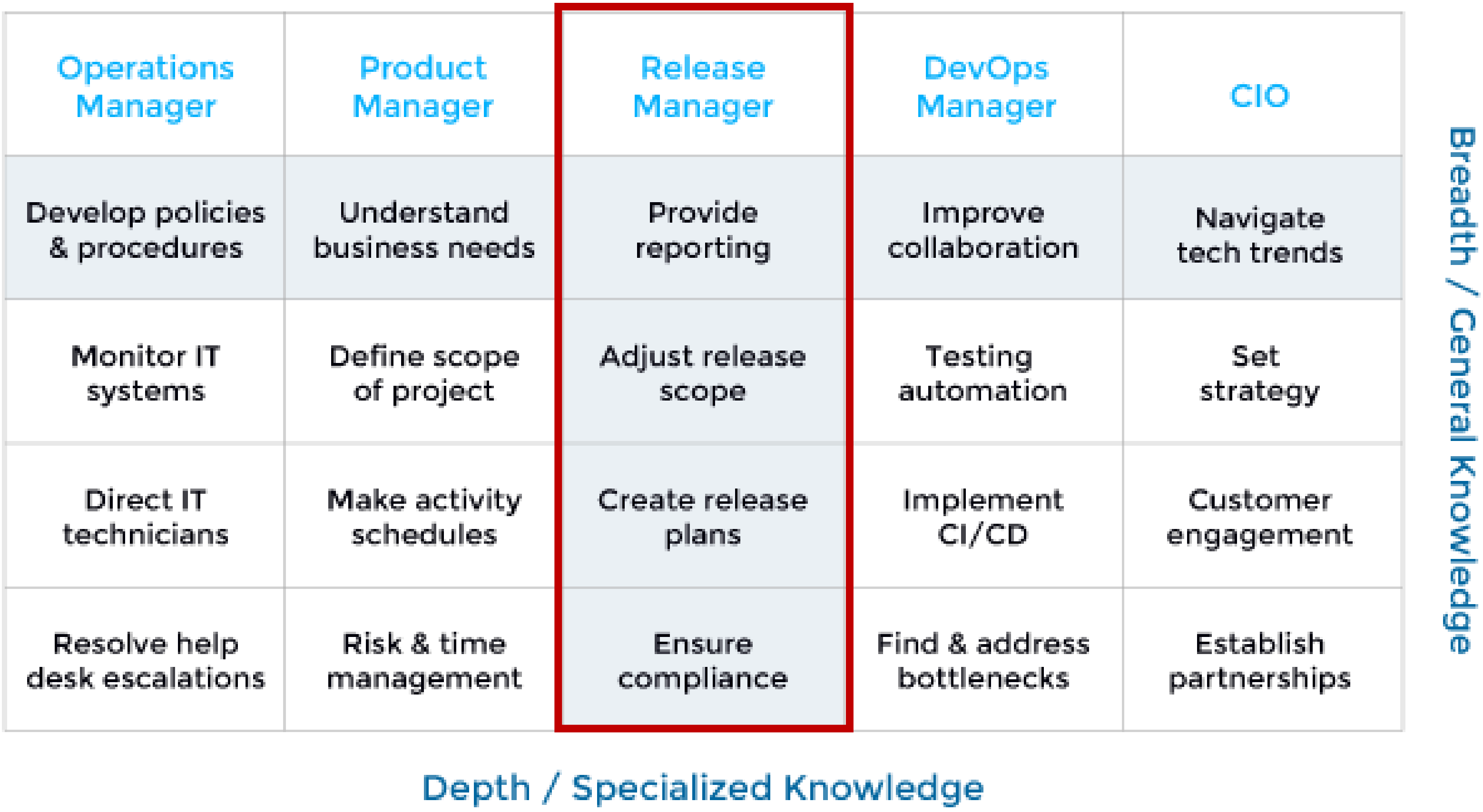
What does a Release Manager do?

- 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.

What does a Release Manager do?

- 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.

Release Manager



Operations Manager	Product Manager	Release Manager	DevOps Manager	CIO
Develop policies & procedures	Understand business needs	Provide reporting	Improve collaboration	Navigate tech trends
Monitor IT systems	Define scope of project	Adjust release scope	Testing automation	Set strategy
Direct IT technicians	Make activity schedules	Create release plans	Implement CI/CD	Customer engagement
Resolve help desk escalations	Risk & time management	Ensure compliance	Find & address bottlenecks	Establish partnerships

Breadth / General Knowledge

Depth / Specialized Knowledge

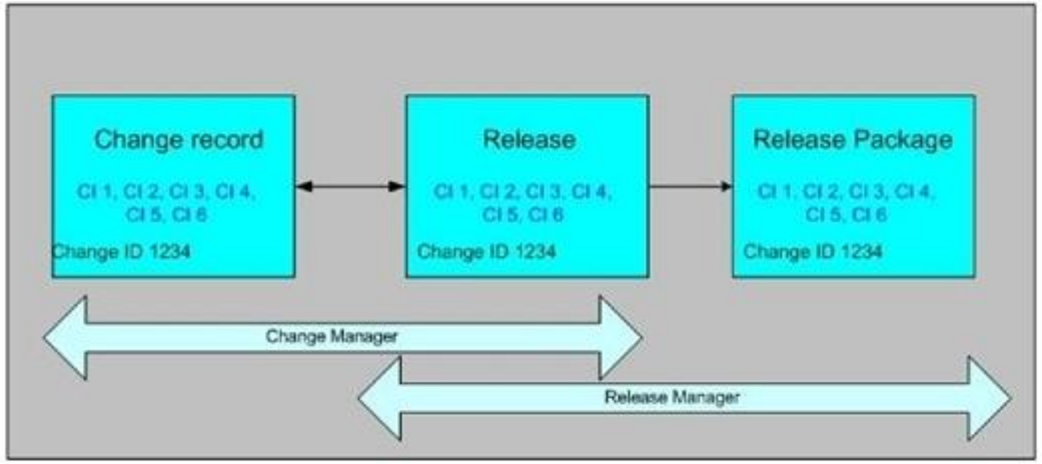
Release Management Best Practices

some natural best practices:

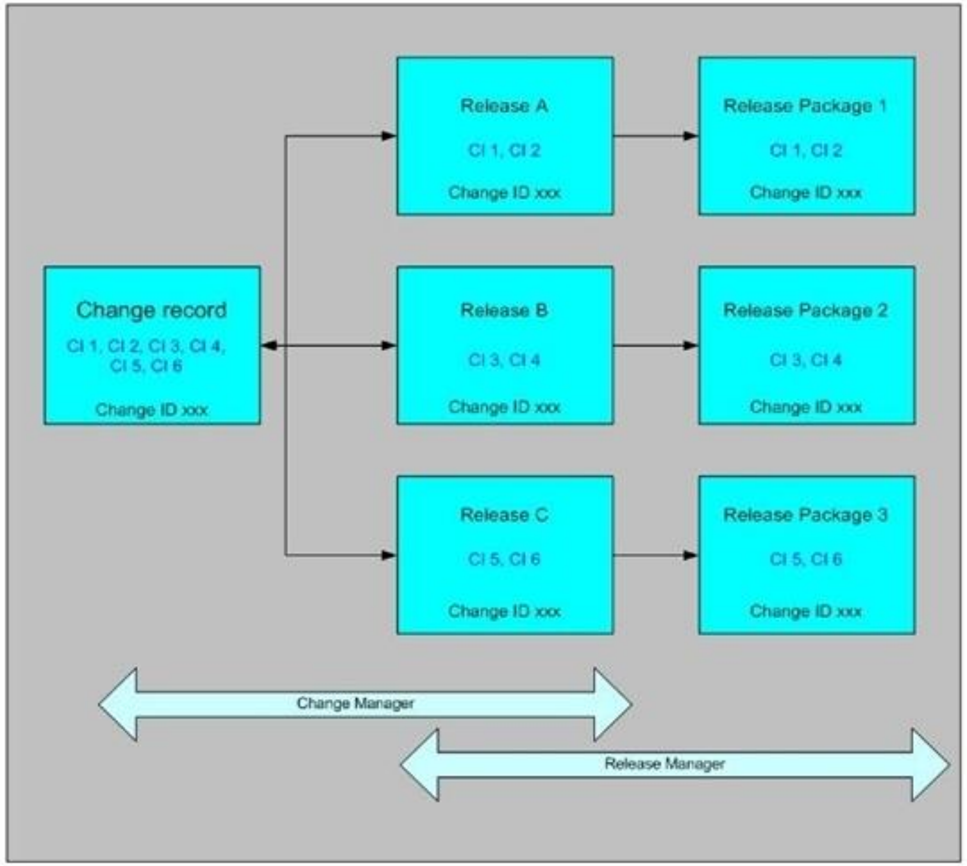
- **Automate** as much as possible, both on the devops and testing side. Automation cuts down on the human cost, allowing more rapid iteration, and reduces the chance for false positives and negatives (computers tend to make less mistakes than humans).
- **Have clear requirements**, and from these make testable acceptance criteria. There should be no ambiguity about whether the software is ready to ship.
- Minimize user impact of a release by **minimizing or eliminating downtime** and testing for regressions before release.
- Make things **immutable wherever possible**. Instead of modifying the configuration of an existing machine, deploy a complete image that contains all of the configuration. This avoids bugs from appearing due to an unexpected series of actions.

WHAT IS A RELEASE PACKAGE

- ✓ A release package contains information about all of the configuration items (for example, documentation, test plans, services, and so on) associated with a change and links to the release that is deploying the change.
- ✓ Typically, a Change Manager might have already linked to a release, or created a request for release.



Depending on the type of change and its scope, a release might or might not have a release package. A single change record can have one or more releases associated with it.



RELEASE NOTE

- ✓ Release notes is a document, which is released as part of the final build that contains new enhancements that went in as part of that release and also the known issues of that build.

Release Notes 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 using a quadruplet of integers to indicate Major, Minor, and Revision (Rev); Build numbers are for internal tracking and verification of the software build process and will not be visible to customers are part of the software version number.

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

Example: “**2.4.7.1333**”

The version above is reference as version 2.4.7, Where: **Major** release is 2, **Minor** release is 4, **Rev release** is 7 and **Build** is not used except for internal tracking/deployments

MAJOR RELEASE

- ✓ A Major Release is a full product release of the software.
- ✓ It generally contains new customer-facing functionality and represents a significant change to the code base comprising the software product or family of products, or is used to represent a significant marketing change or direction in 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 driven

Audience:

New customers.

Existing customers with qualifying contracts

Existing customers desiring to upgrade to the new feature set

MINOR RELEASE

- ✓ A Minor Release of the software may be comprised a rollup of several branched releases, enhancements/extensions to existing features or interfaces driven by internal or external requirements.
- ✓ external requirements could be driven by enhancements to meet new sales area , internal requirements could be enhancements aligned to a new marketing push.

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 (Rev) is a build of all or part of the software that is initially distributed to an internal audience, specifically software quality assurance, for software validation.
- ✓ If the Rev is successfully validated and accepted, this version is “released” to manufacturing. If defects are found that prevent the Rev from successfully being validated and prevent the release to manufacturing, the Rev value will be incremented prior to the next validation cycle.

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

- ✓ A **Build Release** is a build of all or part of the software distributed to an internal audience, these releases should have targeted feature enhancements and issue resolution documented to allow testing in the targeted/specific areas where the changes were implemented.

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

MAJOR.MINOR.PATCH

Check below link for more information.

https://semver.org/

Continuous Integration (CI)

✓Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day.

✓Each check-in is then verified by an automated build, allowing teams to detect problems early.

✓By integrating regularly, you can detect errors quickly, and locate them more easily.

Continuous Integration doesn’t get rid of bugs, but it does make them dramatically easier to find and remove.”

— Martin Fowler, Chief Scientist, ThoughtWorks

Continuous Integration (CI)

Continuous Integration brings multiple benefits to your organization:

✓ Say goodbye to long and tense integrations

✓ Increase visibility enabling greater communication

✓ Catch issues early and fix them regularly

✓ 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

CONTINUOUS INTEGRATION (CI)

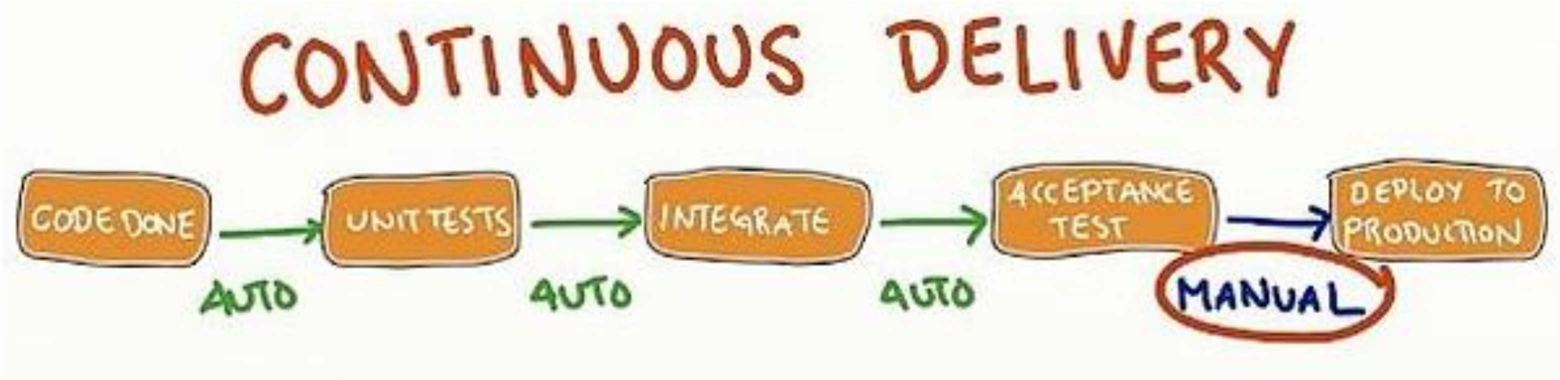
How to achieve:

- ✓ 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 when they occur
- ✓ 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 (CD)

delivery

Continuous ~~delivery~~ is a software development practice where code changes are automatically prepared for a release to production.

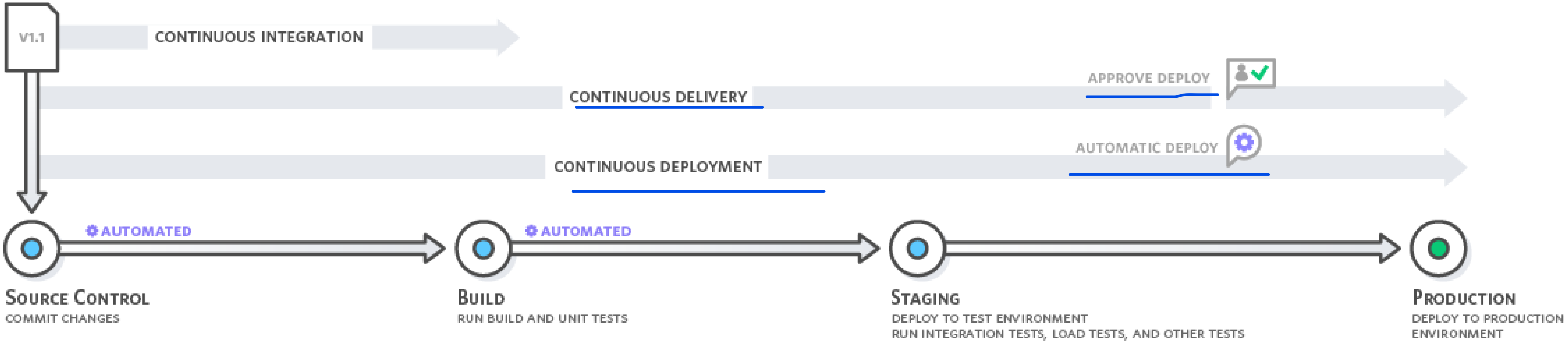


CD expands upon CI by deploying all code changes to a testing environment and/or a production environment after the build stage.

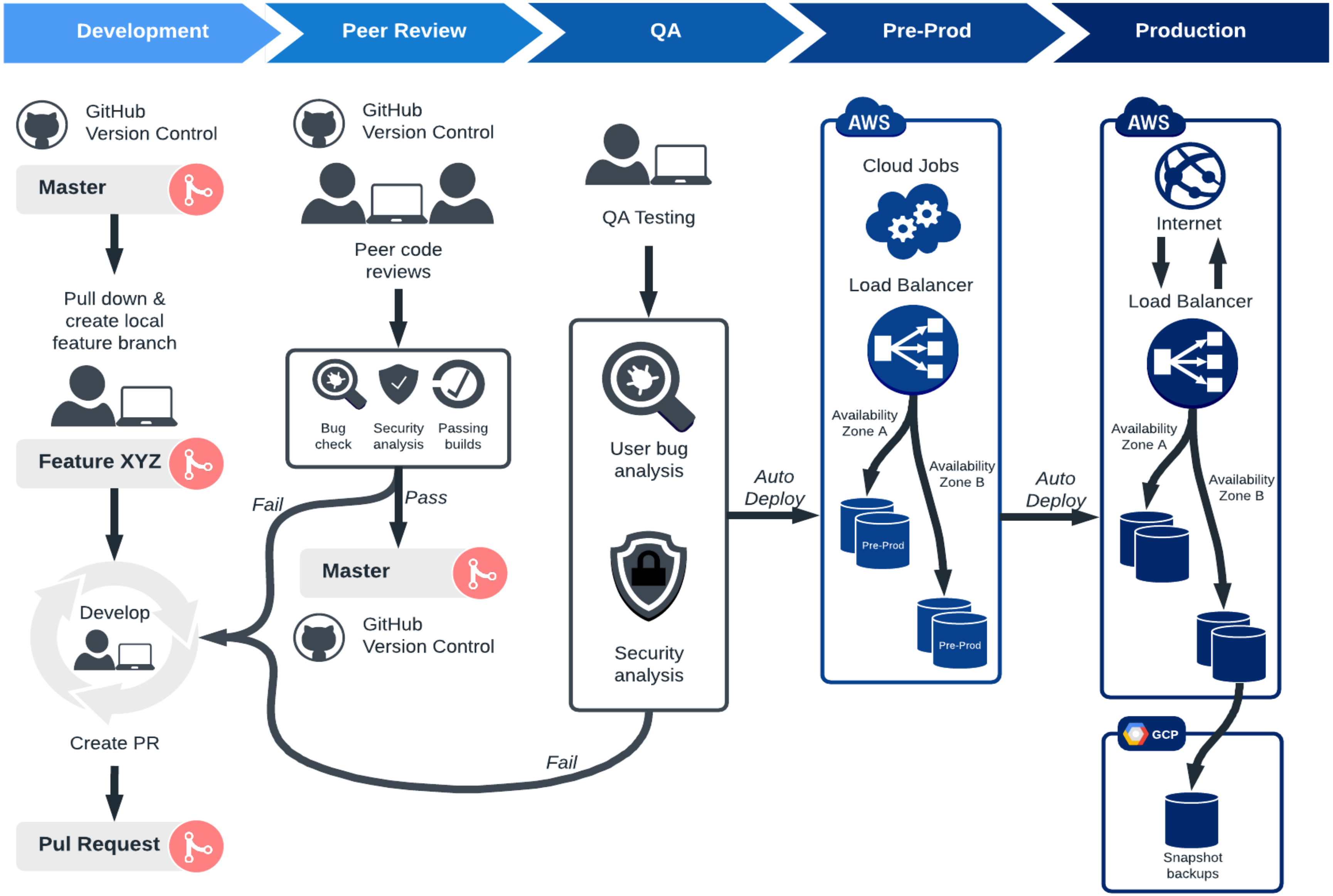
***Continuous delivery** automates the entire software release process. Every revision that is committed triggers an automated flow that builds, tests, and then stages the update.*

Continuous Delivery vs. Continuous Deployment

- ✓ With continuous delivery, every code change is built, tested, and then pushed to a non-production testing or staging environment.
- ✓ There can be multiple, parallel test stages before a production deployment.
- ✓ The difference between continuous delivery and continuous deployment is the presence of a manual approval to update to production. With continuous deployment, production happens automatically without explicit approval.



Simple continuous deployment



Continuous Delivery Benefits

Automate the Software Release Process

Continuous delivery lets your team automatically build, test, and prepare code changes for release to production so that your software delivery is more efficient and rapid.

Improve Developer Productivity

Help your team be more productive by freeing developers from manual tasks and encouraging behaviors that help reduce the number of errors and bugs deployed to customers.

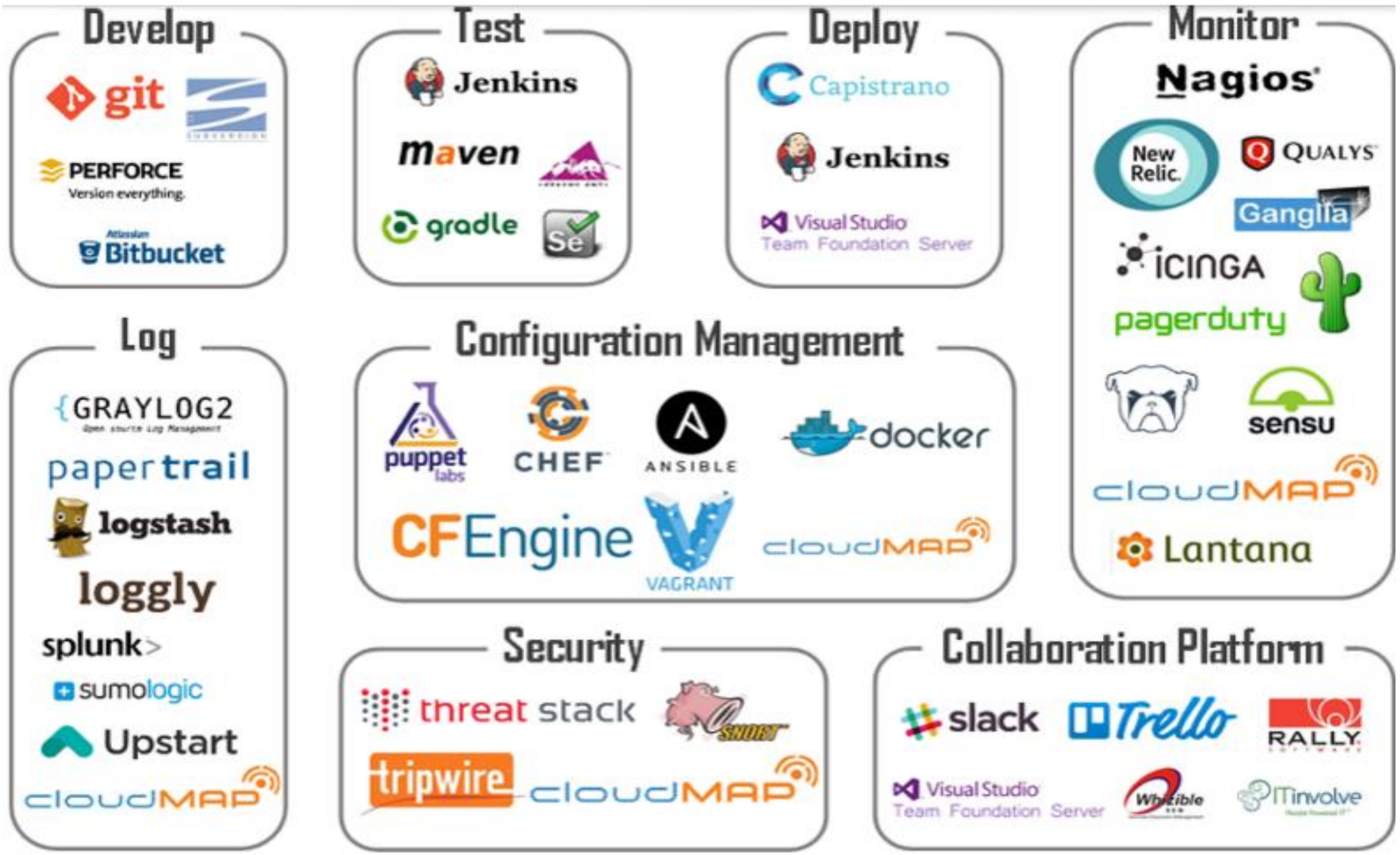
Find and Address Bugs Quicker

team can discover and address bugs earlier before they grow into larger problems later with more frequent and comprehensive testing.

Deliver Updates Faster

helps your team deliver updates to customers faster and more frequently. When CD is implemented properly, you will always have a deployment-ready build artifact that has passed through a standardized test process.

SOFTWARE TOOLS



SW Maintenance…

What is a SW Maintenance

Not just "fixing mistakes"

Any post-delivery modification to an existing system

May be corrective, adaptive, perfective, or preventative

IEEE Definition:

Maintenance is

> The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment.

Types of maintenance

Corrective maintenance

This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.

Adaptive maintenance

This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.

Perfective maintenance

This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.

Preventative maintenance

This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Corrective Maintenance

- ✓ The purpose of corrective maintenance is to restore broken down systems.
- ✓ Corrective maintenance tasks can be both **planned** and **unplanned**.

There are three situations when corrective maintenance occurs:

1. When an issue is detected through condition monitoring
2. When a routine inspection uncovers a potential fault
3. When a piece of equipment breaks down

ADAPTIVE MAINTENANCE

- ✓ The tech environment is constantly changing. New knowledge, hardware, and cybersecurity threats mean that software quickly becomes outdated. Adaptive software maintenance addresses this issue.

- ✓ Adaptive changes focus on the infrastructure of the software. They're made in response to new operating systems, new hardware, and new platforms, to keep the program compatible.

- ✓ Adaptive software changes tend to be low impact for users as they deal with the internal workings of the software. (Making sure it can integrate with new tech.) Users may notice a small improvement in speed or scalability but are otherwise unaffected.

- ✓ In fact, users are more likely to notice when adaptive maintenance isn't completed. For example, it would mean their software stops working on their up-to-date devices.

Perfective Maintenance

- ✓ Perfective software maintenance addresses the **functionality and usability** of the software. Perfective maintenance involves changing existing product functionality by refining, deleting, or adding new features.

- ✓ Any user interface tweaks, redesigns, or in-app user journey changes fall under the perfective maintenance category, too.

- ✓ Perfective changes are highly noticeable. Users won’t notice a bit of refactored back-end code, but they’ll notice any chops and changes visible up-front.

- ✓ So, managing perfective changes means communicating with your users to mitigate any potential negative sentiment.

Change is necessary: but make sure you manage it effectively.

PREVENTIVE MAINTENANCE

- ✓ Preventive maintenance refers to software changes carried out to futureproof your product. So, software maintenance changes are preventive when they prepare for any potential changes ahead.

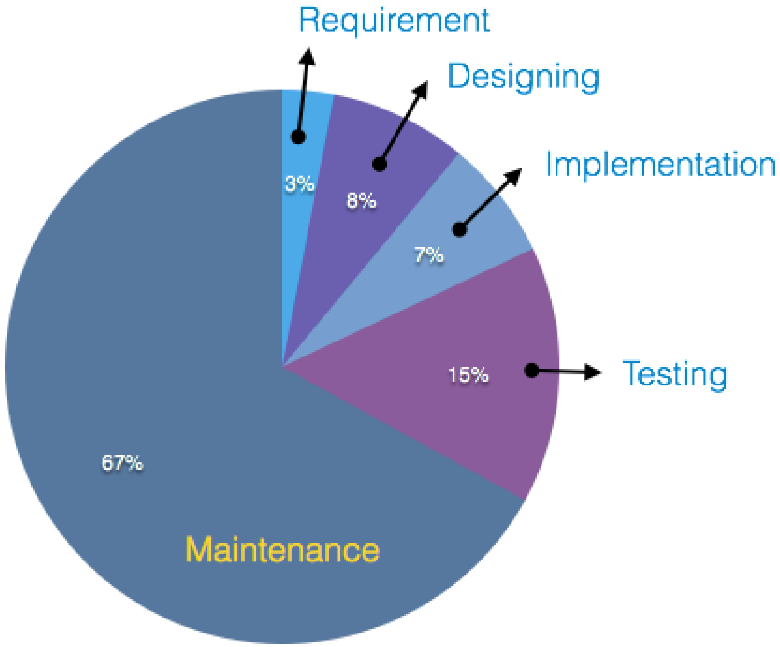
- ✓ This includes making your code easier to scale or maintain and managing your legacy content. It also covers finding and fixing latent faults in your product, before they evolve into operational faults.

- ✓ Preventive software maintenance, then, tends to be behind the scenes. Think tidying and preparation, rather than headline changes.

- ✓ Your users are unlikely to notice preventive software changes – but they still have a positive effect later. This is because preventive maintenance can mean the smoother implementation of bigger changes later down the line. (As well as ongoing stability day to day.)

COST OF MAINTENANCE

- ✓ A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.

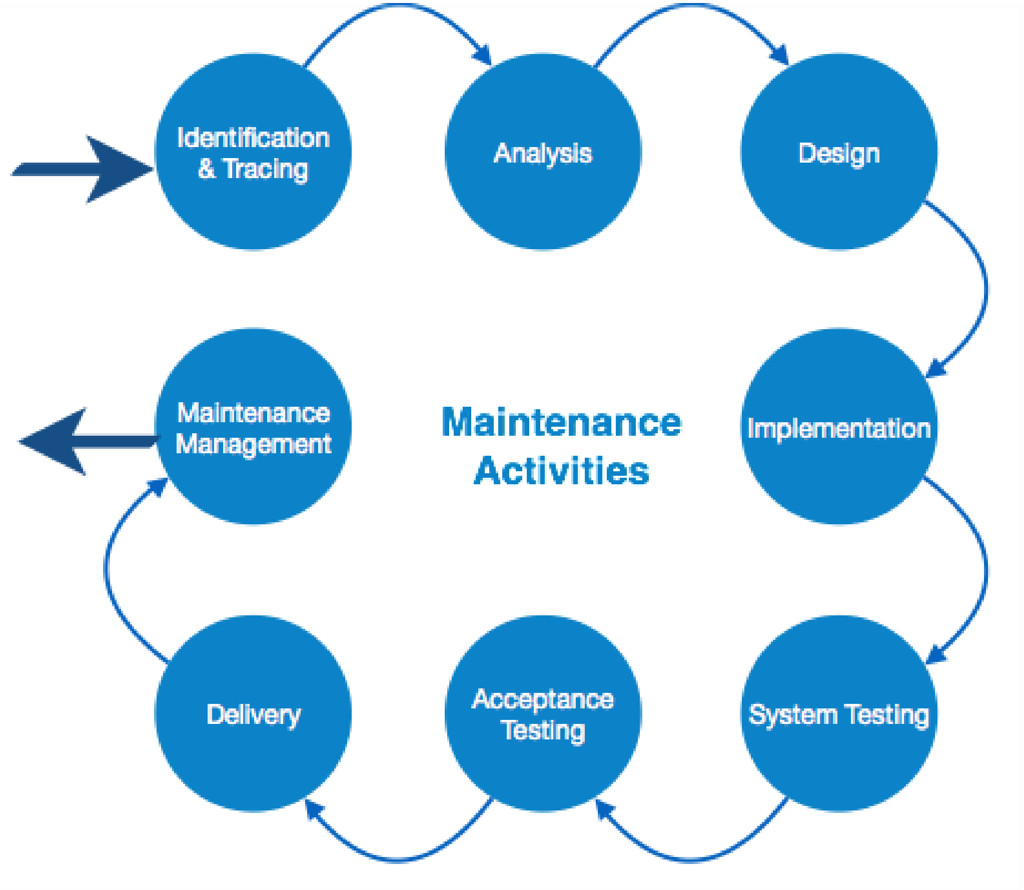


Real-world factors affecting Maintenance Cost

- ✓ Standard age of software is considered up to 10 to 15 yrs.
- ✓ Older software, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced software on modern hardware.
- ✓ As technology advances, it becomes costly to maintain old SW.
- ✓ Most maintenance engineers are newbie and use trial and error method to rectify problem.
- ✓ Changes are often left undocumented which may cause more conflicts in future.

Maintenance Activities

- ✓ IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.



Identification & Tracing - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages.

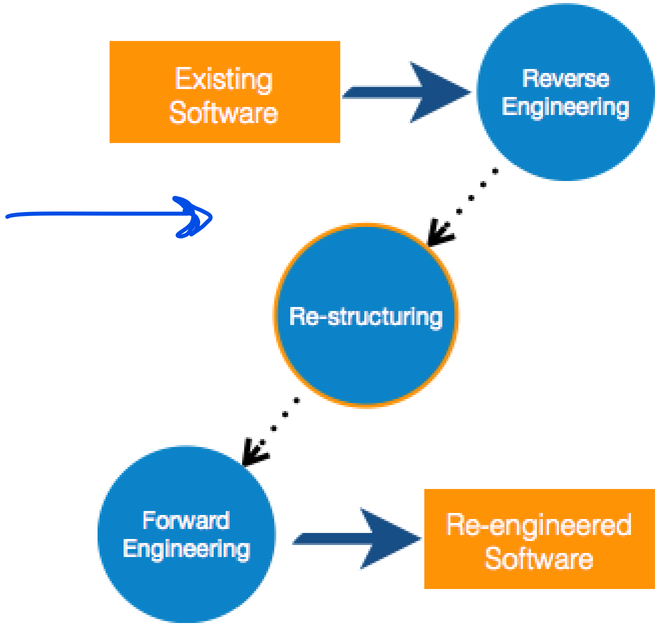
[illegible] implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.

System Testing - Integration testing is done among newly [illegible]

[illegible]

SOFTWARE RE-ENGINEERING

- ✓ When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering.
- ✓ It is a thorough process where the design of software is changed and programs are re-written.

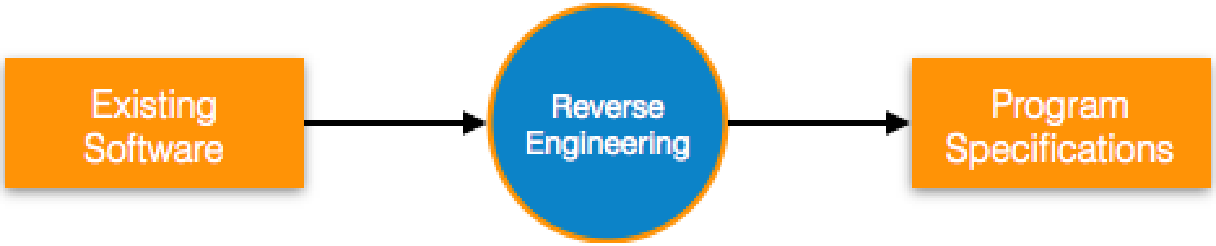


Re-Engineering Process

- ✓ **Decide** what to re-engineer. Is it whole software or a part of it?
- ✓ **Perform** Reverse Engineering, in order to obtain specifications of existing software.
- ✓ **Restructure Program** if required. For example, changing function-oriented programs into object-oriented programs.
- ✓ **Re-structure data** as required.
- ✓ **Apply Forward engineering** concepts in order to get re-engineered software.

Reverse Engineering

- ✓ It is a process to achieve system specification by thoroughly analyzing, understanding the existing system.
- ✓ An existing system is previously implemented design, about which we know nothing. Designers then do reverse engineering by looking at the code and try to get the design.
- ✓ With design in hand, they try to conclude the specifications. Thus, going in reverse from code to system specification.



REUSE

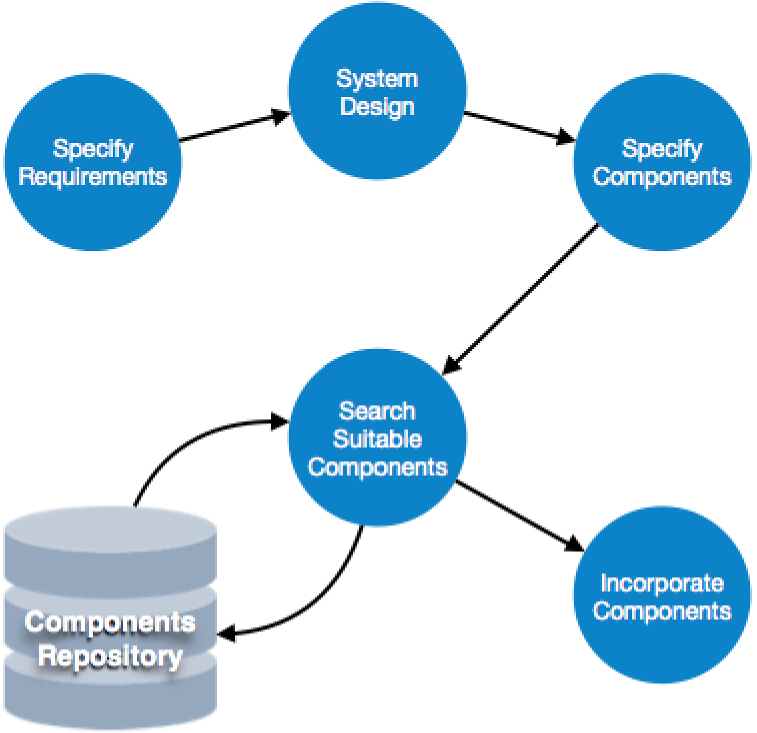
Re-use can be done at various levels

- ✓ **Application level** - Where an entire application is used as sub-system of new software.
- ✓ **Component level** - Where sub-system of an application is used.
- ✓ **Modules level** - Where functional modules are re-used.
- ✓ Software components provide interfaces, which can be used to establish communication among different components.

Reuse Process

Two kinds of method can be adopted: either by keeping requirements same and adjusting components or by keeping components same and modifying requirements.

The Reuse Process



- ✓ **Requirement Specification** - The functional and non-functional requirements are specified, which a software product must comply to, with the help of existing system, user input or both.
- ✓ **Design** - Basic architecture of system as a whole and its sub-systems are created.
- ✓ **Specify Components** - By studying the software design, the designers segregate the entire system into smaller components or sub-systems. One complete software design turns into a collection of a huge set of components working together.
- ✓ **Search Suitable Components** - The software component repository is referred by designers to search for the matching component, on the basis of functionality and intended software requirements..
- ✓ **Incorporate Components** - All matched components are packed together to shape them as complete software.