Software Configuration Management

JD Kilgallin

CPSC:480

11/09/22

Pressman Ch 22, 27

Learning Objectives

- Software build processes
- Software versioning
- Software release engineering
- Software deployment and installation processes
- Software Configuration Management (SCM) concepts

Completing a Software Development Cycle

- Starts with finishing the construction phase (implementation and testing concurrently).
- Software is built, validated, packaged, released, installed, operated, supported, and maintained. This whole process, as well as the resulting artifacts, may be called a "release".
- Release preparations may begin as soon as the cycle is *started*, but the bulk of the work will come at the end of the cycle.
- May take up to 90 days after the last development sprint is complete.
- Involves more coordination with the rest of the business (sales, marketing, support) and other stakeholders.

Code Complete

- Means all net-new code (meeting new requirements or otherwise planned for this release) has been implemented and tested.
- Aka "pencils down", and usually at the end of the last scheduled sprint.
- The first concrete milestone in approaching a release.
- The primary deadline for "crunch time" where developers may be expected to put in more work ahead of this to deliver functionality on time.
- The only changes that occur after this point are bug fixes and minor, low-risk changes to polish or clarify behavior (e.g. changing a string for a user prompt or moving a button to a more prominent place).
- A related milestone is the "Zero Bug Bounce", representing the point where all bugs reported by QA have been *triaged. D*oes *not* mean the software-is bug free, but that any new bugs can be handled promptly.
- Eventually, the team signs off that the code is ready for release.

Builds

- The process of converting source code and other input into operable software (executable program and associated artifacts). Different systems (CPU instruction set, OS, etc) have different executable formats and require different builds.
- Thousands of (partial or full) builds occur throughout each development cycle, on developers' local workstations while making changes, at code checkin for approval, and when automated tests are run (usually after every work day). Most are only intended to be seen or used within the team.
- Special builds start happening near the end of the cycle called *Release Candidates*. These are builds that may be used outside the dev team. The first rc build rarely makes it to end customers as bugs are found and fixed, resulting in newer rc builds.
- Since internal builds ("Debug") happen so often, and rc builds ("Release") happen less frequently but have very high impact, it's important that both build processes are well-defined, documented, and automated.
- There are tools and frameworks designed for scripted builds (e.g. make, Jenkins & other CI/CD pipelines) and most projects will have Debug and Release processes for each system architecture the software might run on.

Build Steps

- Release builds include *much* more than "gcc *.c -o program.exe && mv program.exe /website/downloadPortal/releasedSoftware".
- Starts with "pre-flight" sanity checks – static analysis, linting tools, IDE/SCM warnings and alerts (e.g. GitHub dependabot looks for dependencies that are known to have security issues).
- Tools may run that operate on the source code to modify or add code:
  - Obfuscator makes it harder to de-compile or reverse engineer program.
  - Database creation/modification scripts or an installer may be auto-generated from application code where mapping between classes and SQL tables are set up.
- Dependencies are downloaded and verified, and full build occurs with defined compiler flags (e.g. enable all warnings).
- Compiler output is analyzed (e.g. warnings may result in failure) and smoke test is done on the built software, followed by post-build steps like binary signing, then full suite of automated tests.

Build Integrity

- It is important for organizations and other users to clearly understand all software running in their environment and where it came from.
- Software of unknown provenance or software that has been tampered with (maliciously or not) are major security issues.
- A "low-assurance" process to confirm that the acquired software matches the vendor's intended release binary is for the vendor to publish a secure (collision-resistant) *hash* of the software. Consumers can compute the hash of the bits they got and confirm it matches the published hash. SHA-256 is the current standard.
- The standard way to *rigorously* confirm that software is published from a trusted vendor without tampering by any other party is to enforce *signature verification*. An IT department may configure all machines to automatically enforce this – it may be a legal requirement.
- Similar principles apply to library code, namespaces, official documents, et

Code Signing

- Vendor obtains a code signing certificate from a publicly-trusted PKI and the certificate + private key are stored securely by the vendor.
- [A secure hash of] a release candidate is signed by the certificate's private key and the signature is attached to the build artifact, along with the signing certificate's public key. This is a major application of PKI.
- Before the software is executed in a restricted environment, the system will compute the hash of the software about to run, use the certificate's public key to verify that the hash matches the expected value, and confirm that the certificate is properly issued by a legitimate authority. Any failure will prevent program execution.
- Keyfactor owns **all four** of the major software products for secure code signing, and most of the relevant patents/intellectual property. These platforms keep the signing cert on a secure server, with restrictions and audit trail to unlock the certificates briefly to sign a release candidate.

Acceptance Testing

- Release candidate produced from release build process is picked up, and the team knows it's passed all the pre-build, build-time, and post-build verification steps, based on source code that's been signed off as (tentatively) release-ready.
- QA staff runs through any pre-defined manual tests and additional general testing. This may be the last step before the software is handed to someone outside the team.
- Internal stakeholders (sales/operations, mostly) begin *alpha testing.*
- After alpha testing, designated external stakeholders begin *beta testing.* With company-hosted software (e.g. Facebook), operations staff may begin A/B-testing by deploying the software to end users in a limited capacity.
- Some testing may be outsourced to e.g. software security consulting firms.
- Bugs uncovered in this process are triaged and new rc builds are published as needed. Eventually no new bugs are found that require an immediate fix, or the clock runs out and a defective release is required anyway.

Software Versioning

- It is important to closely track the built software and the exact inputs (source code, process, dependencies, and other inputs) that were used to make that build.
- Version numbers are used by:
  - The installer to upgrade software
  - Support staff to assist with customer issues
  - QA/PM team to track bug introductions and fixes
  - Engineering team to develop hotfixes and security patches for old releases
  - Release notes and change control systems
  - Product metrics and statistics services
  - Users to find matching documentation and resources
  - Software that uses the product as a dependency
  - Marketing to manage customer expectations
- Some projects may use separate version systems for internal tracking and external publication (e.g. Windows vs NT Kernel version)

Version schemes

- There are many ways to record a software version, such as "Year.Month.Incremental" - e.g. new software today might be posted as "2022.11.0", and if another release came this month, "2022.11.1".
- Probably the most common is "Major.Minor.Revision.Build":
  - Major – The slowest to change. Usually indicates significantly different functionality, possibly incompatible with previous major versions, and new primary features. Accompanied by larger marketing campaigns & solicitations for customers to upgrade. At Keyfactor this happens annually.
  - Minor – Less significant changes, or significant changes to only a limited part of the program, along with bug fixes and selected smaller improvements. Customers are less likely to choose to upgrade from e.g. a 10.4 to a 10.5 release. Monthly at Keyfactor.
  - Revision – Iterations on a release candidate. If 10.5.0 goes out for beta testing and a bug is found and fixed, 10.5.1 would be the next rc (rc builds also usually have an "rc" suffix, so 10.5.1rc would be the actual version). Should correspond to a specific commit.
  - Build – The literal build number from one revision. It is possible for two builds from the same source code to be different (e.g. build settings; obfuscation variations; compiler bugs; changes to build toolchain, dependencies, or dependencies' dependencies). Rev/build may be flipped.

Software Releases

- Last pre-release steps include signoffs from sales, operations, finance, marketing, and legal departments that they are ready for the release.
- Once a release candidate has been built, published, tested, and accepted, it goes through the final steps to be released.
- This varies depending primarily on the distribution and consumption model (e.g. webapp, desktop executable installed from CD-ROM, app store).
- May be multiple processes for different release targets, or extended processes where the released software is a component of a larger product (e.g. Keyfactor sells a "software appliance", which is a Virtual Machine with an operating system, database, and Keyfactor software pre-installed and configured).

Release Artifacts

- Software and associated files built for the release.
- Installer, configuration, and upgrade wizards or utilities.
- Documentation and release notes (changes from previous versions).
- Licenses and notices required for dependencies and compliance reqs.
- Manifest of files & known dependencies ("Software Bill of Materials").
- Packaging & distribution materials (zip file, jar, container, CD+box, etc)
- Infrastructure-as-Code definitions.
- File hosting and canonical download links.
- How-to guides, blog posts, press releases, marketing campaigns.
- Release announcement!

Release Engineering

- Software build, acceptance, release, and deployment processes are involved and complex.
- This process is a legitimate step conducted by the engineering team as part of the larger software development lifecycle process.
- Artifacts critical for distribution, deployment, and operation go well beyond the core software, and may be just as complex (sometimes more!).
- Repeatability and reliability are important quality considerations.
- The stakes for customer satisfaction, business reputation, and ultimately bottom-line revenue are extremely high.
- This makes the conduction of software releases an *engineering* task, called release engineering. Large/mature organizations will have dedicated *release engineers* that work on this. Cloud-hosted software companies (e.g. Facebook) will have DevOps engineers and Site Reliability Engineers (SREs).

Software Deployment

- Once released, users need to be able to acquire and use the software.
- Marketing and announcements convey to customers that the software is available and the manner in which they can acquire it.
- Depending on the product and distribution method, deploying the software may be a simple downloadable executable, or require a team of experts from the vendor's operations team.
- Each user attempting to deploy the software must follow the documented requirements for pre-requisites, backup of any previous versions, installation, and configuration.
- Typically, business software will be deployed in three separate places – testing, pre-production, and production, frequently with backups in each.
- The deployment process is part of the user experience, and a difficult or faulty process reflects negatively on the software product.
- Once deployed, the installation becomes an instance of running, released software. No two deployments are exactly the same.

SCM

- Mature software organizations don't take the raw program bytes and put them through the release process for the first time at the end.
- Changes to *any* input or process that affects release artifacts between versions need to be carefully documented ("change control" or "change management").
- Software Configuration Management is the umbrella activity that occurs throughout the development cycle to manage these changes, along with the combination of people, processes, and tools that participate.
- Project management is responsible for enforcing change control process, and it's a major function of project management and version control software (frequently *called* SCM software).
- The source code repository and commit history are important SCM software components developers interact with regularly, as are CI/CD tools and Infrastructure-as-Code platforms (e.g. Chef, Puppet, Ansible).

SCM Elements

- Already covered source control, issue tracking, build & dependencies.
- Requirements tracing, change control, & audit trails also part of SCM.
- Documentation of these processes (eg change control) is part of SCM.
- Integration and coordination between products and systems also SCM
- Content management is another important piece – tracking product *and* process documents wherever they live (including a team wiki or some such), and recording, publishing, & reporting changes to them.
- Processes usually start as an informal workflow owned by one individual, and gradually become more formalized and disseminated.
- In small teams, these are owned by the project manager and/or release engineer; larger teams have a dedicated *change manager*.

Summary

- Software builds are used to convert source code into executables.
- Software releases start by completing code, creating a release candidate build, and conducting acceptance testing.
- Code signing certificates are a critical tool for ensuring build integrity.
- When a release is approved, it goes through a distribution process.
- Released software is more than just a compiled binary.
- Tracking released software versions and their input is essential.
- Release engineering is the discipline of ensuring quality in this process.
- Deployments can be very complex, especially with multiple instances.
- SCM is the activity of managing inputs to the release process.

References

- Bouncing Zero Bugs, Together. Eric Lippert. Nov 2004. Microsoft Learn.
- Software-Versioning. Sandeep Kapi. Aug 2020. Kav Interactive Single Blog.
- Software Versioning. Wikipedia.
- Qingdao Ruanmei Network Technology Co.,Ltd. 2022. Freefixer.
- What is PKI? Ted Shorter et al. 2022. Keyfactor.
- Software Configuration Management. Priya Pedamkar. May 2020. Educba.
- Software Configuration Management Tools. Swati Tawde. June 2020. Educba.

- *Reading for next lecture: Pressman Ch 19-20 (some content already covered)*