# Tools: Ecosystem Overview

Module 2.3

# Software Development

Software development tools help with the process of programming, documenting, testing, and fixing bugs in applications and services.

**Local Development Environment**

A consistent local development environment is critical to quickly get employees started contributing to your product.

- Minimal requirements may vary in your environment depending on individual preferences, ranging from multiple displays for increased collaboration or high resolution displays for more comfortable long-term viewing sessions, to specific keyboards, mice, and other input devices.
- Qualifying the current standard of your local development environment includes determining whether there is a consistent framework that teams share internally and across teams.

# Software Development

**Local Development Environment**

- There is a balance to strike between ensuring a consistent experience and allowing employees to customize their workstation and work habits to best suit their own individual needs.
- Too much customization can lead to isolation of knowledge or the additional overhead of time and effort spent fussing with specialized environment setups.
- Identify a shared area for documenting the local development environment. Documentation can be stored within a version control repository, an internal wiki, or even a Google

# Software Development

**Version Control**

- Having the ability to commit, compare, merge, and restore past revisions of objects to the repository allows for richer cooperation and collaboration within and between teams.
- It minimizes risks by establishing a way to revert objects in production to previous versions.
- Every organization should implement, use, train, and measure adoption of version control.
- Version control enables teams to deal with conflicts that result from having multiple people working on the same file or project at the same time, and provides a safe way to make changes and roll them back if necessary.

# Software Development

**Version Control**

When choosing the appropriate version control system (VCS) for your environment, look for one that encourages the sort of collaboration in your organization that you want to see. Qualities that encourage collaboration include:

- opening and forking repositories;
- contributing back to repositories;
- curating contributions to your own repositories;
- defining processes for contributing; and
- sharing commit rights.

# Software Development

**Version Control**

Terminology related to version control includes:

**Commit:** A commit is a collection of actions comprising total number of changes made to files under version control.

**Conflicts:** A conflict results when two changes are too similar in nature, and the version control system can’t determine which is the correct one to accept. In most cases, the version control system will provide a way to view and select the desired change to resolve the conflicts.

**Pull request:** A pull request is a mechanism that allows an individual to signal that a contribution is ready to be reviewed and merged into the main branch.

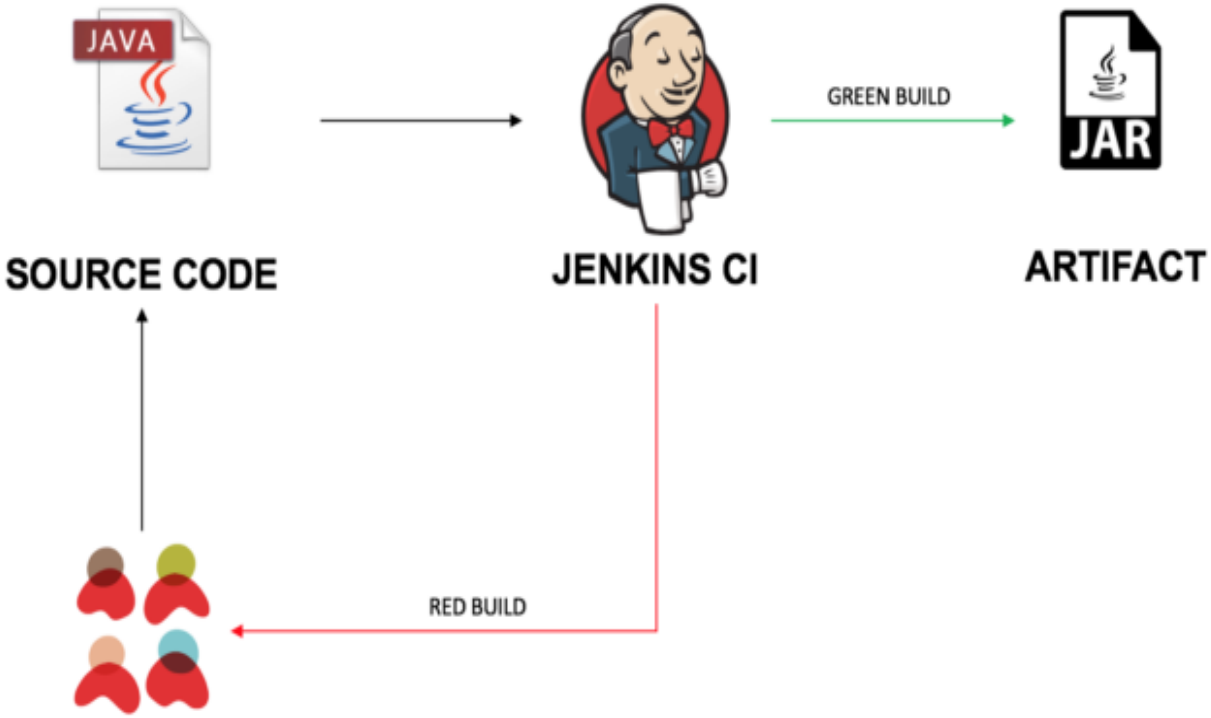
**Cherry picking:** Cherry picking is a way to choose specific commits from one branch and apply them to another. This is useful when you want to select specific changes from a pull request.

# Software Development

**Artifact Management**

An artifact is the output of any step in the software development process.

For example:



# Software Development

**Artifact Management**

When choosing between a simple repository and more complex feature-full repository, understand the cost of supporting additional services as well as inherent security concerns. An artifact repository should be:

- Secure;
- Trusted;
- Stable;
- Accessible; and
- Versioned.
- Artifact repositories allow you to store artifacts the way you use them.
- Ideally, your local development environment has the same access to your internal artifact repository as the other build and deploy mechanisms in your environment. This helps minimize the “it works on my laptop” syndrome because the same packages and dependencies used in production are now used in the local development environment.

# Software Development

**Artifact Management**

Terminology related to artifact management includes:

**Dependency management:**

Dependency management is the manner and degree of interdependence a software project has on another software project. Different mechanisms exist to expose this dependency. From an artifact level, storing the dependent artifacts for your software is helpful.

**Pinning:**

Pinning is the method of locking down an explicit version of an artifact for an environment. In dependency management, this can be helpful to define the explicit version of a dependent software artifact that works with your project.

**Promotion:**

Promotion is the method of selecting a specific version of software to move toward delivery, generally keyed off the successful passing of tests.

# Automation

Automation tools reduce labor, energy, and/or materials used with a goal of improving quality, precision, and accuracy of outcomes.

**Server Installation**

Server installation is the automation of configuring and setting up individual servers.

Hardware manufacturers like HP and Dell provide a tool that works for their brand of hardware. Some Linux distributions provide operating system–specific tooling as well. As an example, Cobbler and Kickstart can be used to automate the server installation of Red Hat Enterprise Linux or CentOS.

Operations staff can write Kickstart files that can specify hard disk partitioning, network configuration, which software packages to install, and more.

The hardware lifecycle begins with planning and purchasing (or leasing); continues with installation, maintenance, and repair; and ends with trading in, returning, or recycling.

# Automation

**Infrastructure Automation**

Fundamentally, infrastructure automation is provisioning elements of infrastructure through code—treating this code just like the rest of your software, with the ability to recover business through data backups, code repository, and compute resources.

Terminology related to infrastructure management includes:

**Configuration drift**

Configuration drift is the phenomenon where servers will change or drift away from their desired configuration over time.

**MTBF**

MTBF is mean time between failure: the uptime between two failure states.

**MTTR**

MTTR is mean time to repair: the length of time that it takes to restore operation to a system.

# Automation

**Infrastructure Automation**

**Availability**

Availability is a commonly used measurement of how often a system or service is available compared to total time it should be usable.

Availability = MTBF / (MTBF + MTTR).

**Capacity management**

Capacity management is a process used to ensure that infrastructure and other resources are correctly sized to meet current and future business needs in a cost effective manner.

**Snowflake server**

A snowflake server is a server that has gotten to its current, desired configuration by way of many manual changes, often a combination of command-line sorcery, configuration files, hand-applied patches, and even GUI configurations and installations.

# Automation

**Infrastructure Automation**

When people refer to configuration management in operations roles, often the intended meaning is infrastructure automation.

- Depending on the system, different parts of it might be automated, including aspects such as which software packages are installed on a server, what versions of those various packages should be used, the presence or absence of various files on the system, or which services are supposed to be running.
- “Treat infrastructure code like the rest of your software” means the code is developed using a common local development environment, versioned in version control, versioned in artifacts in an artifact repository, tested, and verified before being put into production.

# Automation

**Infrastructure Automation**

Infrastructure automation should minimally provide:

- Management of configuration drift
- Elimination of snowflake servers
- Versioned artifacts of infrastructure code
- Minimizing complexity

Infrastructure automation leads to repeatable, consistent, documented, auditable, and resilient processes.

This frees up time, improves efficiency of staff, allows for more flexibility, and facilitates risk measurement.

Infrastructure automation also increases the degree of confidence that the machine setup and deployment are identical, reducing the amount of time spent debugging problems based on system differences.

# Automation

**System Provisioning**

Buy, and provision hardware in data centers, now they have the option to invest in cloud infrastructure. With on-demand computing, companies can purchase only what they need and scale up and down as necessary. This infrastructure can be purchased and provisioned much faster than physical hardware, and is often more cost-effective for organizations.

**Test and Build Automation**

Build automation tools usually specify both how the software is to be built (what steps need to be done and in what order) and what dependencies are required (what other software needs to be present in order for the build to succeed).

Some tools are best suited to projects in specific programming languages, such as Apache's Maven and Ant, are most often used with Java projects.

Others, such as Hudson or Jenkins, can be used more broadly with a wider range of projects.

# Automation

## Test and Build Automation

These tools usually fall into one of three use cases:

**On-demand automation**

This tool is run by the user, often on the command line, at the user's discretion. For example, a developer might run a Make script by hand during local development to make sure she can build the software locally before checking it into version control.

**Scheduled automation**

This process runs on a predefined schedule, such as a nightly build. Nightly builds are created every night, usually at times when nobody is working so that no new changes are taking place while the software is building (though this is becoming less doable as teams become more globally distributed).

**Triggered automation**

This tool runs as specified events happen, such as a continuous integration server that kicks off a new build every time a commit is checked into the code.

# Automation

## Test and Build Automation

Terminology related to testing includes:

**Smoke testing**

Smoke tests originated with hardware testing to determine whether powering on a device would cause it to start smoking, an indication of a major problem. In software, a smoke test is a very basic, quick test to determine whether the output is possible and reasonable.

**Regression testing**

Regression tests verify that changes to software do not introduce new bugs or failures.

**Usability testing**

Usability tests involve testing a product on users to measure its capacity to meet its intended purpose by testing it on users.

# Automation

## Test and Build Automation

Terminology related to testing includes:

**A/B testing**

A/B tests are an experimental approach in which two different versions of a web page or an application are compared to determine which one performs better.

**Blue-green deployment**

Blue-green deployment is a release process of maintaining two identical production environments. One environment is considered live and serves all traffic. The final stage of testing a new release is done in the second environment. When tests pass, traffic is routed to the second environment. This deployment process can reduce risk; if something happens with the new release, you can immediately roll back to the previous production environment.

# Automation

## Test and Build Automation

Terminology related to testing includes:

**Canary process**

In the past, coal miners used canaries as an early warning system to detect toxic gases. When the canaries started to suffer symptoms of exposure, miners knew that the conditions of the mine were unsafe. In software, the canary process involves running new code on a small subset of production systems to see how performance of the new code compares to the old code.

# Automation

**Monitoring**

Monitoring is the process of tracking the current state of your systems and environment, usually with the goal of checking whether or not they meet some predefined conditions of what constitutes the desired state.

The methods of information collection include metrics and logs.

Monitoring includes gathering basic system-level metrics such as

- if a server is up or down,
- how much memory and CPU are being used, and
- how full each disk is,

As well as higher-level application monitoring, which can range from

- how many user requests a web server is handling,
- how many items are queued in a queuing system,
- how long a given web page takes to load, and
- what are the longest-running queries going into a database.

# Automation

**Monitoring**

**Metrics**

Metrics are the collection of qualitative and quantitative measurements. Generally they are compared against some benchmark or established standard, tracked for analytics, or tracked for historical purposes.

Metrics are one of the key parts of monitoring; data can be gathered and stored for nearly any part of even the most complex web software, and different teams can have different metrics that they keep track of and use in their work.

StatsD and Graphite are very commonly used and a powerful combination for tracking, storing, and viewing metrics.

# Automation

**Monitoring**

**Logging**

Logging is the generation, filtering, recording, and analysis of events that occur on a system due to operating system or software messages.

- When tracking down the source of a software issue, one of the first things that engineers often do is to check the logs for any relevant error messages.
- Logs can be treasure troves of useful information, and with storage getting cheaper and cheaper, just about any log you might want can be saved and stored for later use.
- Logs can come from the applications you develop, from third-party tools you use, and even from the operating system itself.

# Automation

**Monitoring**

**Alerting**

Monitoring and alerting are important not only from a performance perspective, but also from a preventative one, in that they help you find out about potential issues before they become actual issues for your customers. For example, when the US HealthCare.gov site was first launched in October 2013, they originally had no monitoring or alerting to let them know whether or not the website, which had been two years in the making, was up or not.

When reasoning about alerting, you need to consider several factors:

- Impact
- Urgency
- Interested party
- Resources
- Cost

# Automation

**Monitoring**

**Alerting**

Alerting is the process of creating events from the data gathered through monitoring, with the goal usually being to direct human attention to something that might need human intervention.

**Events**

Event management is the element of monitoring that acts on existing knowledge around impacts to systems and services.

For 24/7 services, this generally reflects the need for real-time information about the status of all of the different components of infrastructure. A system is configured to monitor a specific metric or log based on a defined event and to signal or alert if a threshold is crossed or an alert condition has been met.

# Automation

**Monitoring**

**Events**

- With much software development now being done on web software that is expected to be available 24/7, more consideration is being given to handling alerts that occur when engineers are at home instead of in the office. One way of dealing with this is to set up as much automated event handling as possible.
- Many alerting and monitoring systems have built-in ways to automatically respond to a given event. The Nagios monitoring system, for example, has “event handlers” that can be configured for different alert conditions. These handlers can do a variety of things, from automatically restarting a service that had crashed to creating a ticket for a technician to replace a failed hard disk.
- Automated event handlers can cut down on the amount of work that operations staff have to do.

# Automation

**Monitoring**

**Events**

- It’s important to ensure that your failure conditions are clearly defined, that the event handler process is understood well enough to be automated, and that there are necessary safeguards in place to keep the automation from causing more problems than it solves.
- No alerting system is 100 percent accurate 100 percent of the time.
- A false positive is when an event is generated when there isn’t an issue.
- A false negative is when an incident occurs without generating an alert for it, which could lead to a longer time before the issue is detected and resolved.

# Evolution of the Ecosystem

Over time, from server installation automation to infrastructure configuration and automation, there is a trend to simplify and remove the repetitive tasks that can be subject to human error.

- With the introduction of containers, the pipeline from the laptop to production has simplified more.
- As automation is added to the different parts of the environment, new patterns are discovered.
- With infrastructure automation, adhering to one version of an operating system is less important, as quickly spinning up a new instance with updated packages on a new system may be more useful from a security standpoint.
- Continuous delivery and continuous deployment have freed humans up to focus on what matters.
- Automated shortened feedback cycles through build automation with tests give us additional confidence and insight into our systems.

# Evolution of the Ecosystem

The ecosystem will continue to evolve as application development adopts increased operability.

When the operable requirements are standardized and automated, employees have the freedom of choice of language and framework. These trends bring into focus the tools that stress the “we” over “me,” building understanding across teams and encouraging time spent on valuable outcomes.

# The Right Tools for Real Problems

For a tool to be widely adopted and implement successful change, it needs to solve real problems.

People need to be involved throughout the decision-making processes for a given tool, so their issues, frustrations, and concerns will be discovered, understood, and addressed.

In general, understanding the real underlying issues that tools help us solve will enable us to choose the right solutions and fully understand the complexities of our work.

By understanding these complexities, we can work to minimize them and any associated risks, which in turn can help to reduce expenses and ensure that people are focusing on the right areas.

# Embracing Open Source

Open source communities provide a way for individuals to practice collaboration with other individuals.

Employees learn the value of managing contributions from other individuals, and how to create contributions that are useful to others. For example, multiple small, discreet commits that focus on one intended change at a time are easier to manage and accept than one large commit that touches many different pieces of code.

Should companies worry that open sourcing their tooling will take away their competitive advantage? If your company's success depends on your tooling, that should be part of your business model, so that you're making money from either the software itself or, as in many companies these days, the support or services for that software.

# Embracing Open Source

- Contributing to open source is an excellent reflection of company intentions.
- Open sourcing software within companies encourages teams to contribute to each other's projects rather than reinventing the wheel, and it exposes both individual contributors and managers to the benefits of open source collaboration.
- Contributing to open source and using open source often go hand in hand as well. Teams that are used to the open source community are more likely to look for open source solutions that already exist rather than writing their own.
- Many companies look toward the well-known players in the DevOps space and their open source contributions, such as Netflix and Etsy, and feel compelled to start writing and open sourcing many of their own tools as well.

# Embracing Open Source

Tools can and will reinforce behavior, which will affect your culture. It is critical to also examine tools when examining behaviors and cultures, and this is where the true importance of your tooling choices lies.

Using open source can not only open up new solutions in terms of the tools and technologies available, but it can have a noticeable impact on a culture of collaboration, sharing, and openness as well.

One of the benefits of open source software is that you don't have to reinvent the wheel by working on problems that other people have already solved. An open source solution with a strong community contributing to it can make its implementation even more effective.

# Standardization of Tools

Working effectively comes down to developing mutual understandings and negotiating to mitigate around the inevitable miscommunication that occurs when teams are trying to navigate multiple goals at once.

Tools can be used to:

- Improve communication
- Set boundaries
- Repair understanding within the scope of the DevOps compact

Organizations need to standardize tools to balance the challenges and costs of supporting tools that perform the same function.

How do we then balance strengthening the organization through standardization, allowing flexibility at the team level, and empowering the individual to be agile and responsive by choosing her own tools?

# Consistent Processes for Tool Analysis

Standardization of tools bridges old to new as the technologies being used at a company change. With consistent processes for evaluating, choosing, and retiring tools, organizations will:

- decide upon a tool that meets most people's needs;
- ensure necessary features that were present in an old tool are also features of the new one; and
- ensure that employees are properly trained to be able to effectively use a new piece of hardware or software

# Consistent Processes for Tool Analysis

- Without consistent processes to build the necessary bridges, employees will be resistant to new tools or technologies.
- Consistent selection processes can minimize risk by making sure that both current and new needs will be met, as well as simply providing reassurance for people who are more resistant to changes in their environments.
- Consistency can also help you avoid the sorts of logistical nightmares that can crop up without enough process or standardization.
- For example, if each team uses a different issue tracker or ticketing system, you will have less visibility throughout the organization, more duplication of effort, and a great deal of time spent trying to navigate and interoperate these various systems.

# Exceptions to Standardization

There are exceptions to standardization. If a team needs some isolation or unique requirements, there isn't a reason to force them to use the same tooling as everyone else.

- One example is PCI compliance, which requires very strict separation of duties. A team responsible for PCI work is likely to have separate computers running on a separate network from the rest of the organization.
- In a case like this, a segregated environment allows for different tooling without having a detrimental effect on the organization as a whole.
- Decisions need to be made on a case-by-case basis. Even though there are many commonalities, each team and each company has some unique needs and experiences.

# The Impacts of Tools on Culture

**Tools Impacting Communication**

Tools shape behavior, so having tools that reduce the friction involved in communicating with other teams makes it more likely that communication will start to take place.

If a company doesn't even have any chat software, or if what they use has technical limitations that prevent inter team communication, it will be much harder for communication to happen.

We stress the importance of communication when it comes to tools because communication can make or break cooperation, collaboration, and affinity within an environment.

**Tools Impacting Broader Behaviors**

Infrastructure automation, chat systems, deployment tools, and any tool that is used by multiple teams within an organization. While it is important to figure out everyone's requirements and try to meet as many of them as possible

# Selection of Tools

Selecting a new tool to add to your environment can be a big decision, and one where many people involved are likely to have strong opinions. Here are a few important factors that should be considered during the selection process:

- Product development
- Community health
- In-house customization

Availability of features, budget, and interoperability with the current tool set or environment are also important considerations.

# Selection of Tools

**Product Development**

An actively developed product will be quicker to get new features, support newer operating systems and platform versions, and deal with any security vulnerabilities.

Using a tool without active development leads to more time spent dealing with bugs or waiting for new features.

How quickly are new features implemented and released? Are these feature requests tracked and regularly evaluated for the product's roadmap? If critical bugs or security vulnerabilities are found, how quickly will they get fixed?

# Selection of Tools

**Product Development**

Look at recent releases for the tools you are considering and what the update process is like.

Also consider how much contact you are able to have with the product developers themselves. Will you have any direct developer or support contacts within the tool vendor's organization?

Having people assigned directly to work with your account can mean better support, as you are able to have an ongoing dialogue, and it can help ensure your issues get dealt with, rather than disappearing into some unknown ticketing system or email inbox somewhere.

# Selection of Tools

**Community Health**

Community health is a measurement of the overall health of a group of individuals who are connected through shared norms, values, and behaviors.

Communities can evolve for a specific tool, set of tools and practices, or role. Keep an eye out for community activity as one of the signs of health including:

- rate of response to pull requests;
- mean time to resolution for issues;
- frequency of release;
- content creation (blog posts, articles, news); and
- rate of forum communication.

# Selection of Tools

**Community Health**

The community and its events should also help foster a safe, respectful, collaborative, and inclusive environment. Pay attention to how members of the community treat each other. Consider the following questions:

- Do projects and community events have codes of conduct?
- What is the tenor of discussions on issues and pull requests?

People who use these various tools in their day-to-day work will likely find themselves interacting with other users, maybe attending local meetups or larger conferences focusing on the tools or how to use them.

# Selection of Tools

**In-House Customization**

A tool that can be easily customized and contributed to will make for a robust solution that is well suited to both the technological and human aspects of an environment.

- This is especially important in organizations with a large number of people working with the tool.
- A tool that deals well with that kind of scale will be able to grow along with your organization, as well as making engineering work easier.

# Selection of Tools

**In-House Customization**

- Open source tools are generally the most customizable, as you have the code available and can much more easily both read through it and modify it as needed. This can aid greatly in doing things like fixing bugs, where instead of having to file a support ticket and wait, you can do much more work to identify the bug and even submit a pull request with a fix yourself.
- Even with closed source tools, pay attention to whether they have something like an API that can be used to develop additional tooling that works with or alongside the tools themselves.
- Being able to customize a tool, fix your own bugs, and even add new features and extensions can go a long way toward making a tool easy to work with over time.

# Elimination of Tools

Regular reviews of current processes and tools should be done to make sure that their usage is still effective. Automation and technical debt tracking can help identify processes and tools that should be eliminated.

- This can help prevent situations where a tool has been causing additional work for those using it, and help identify and eliminate tools that serve the same purpose, reducing both cost and confusion.

These reviews should also involve regular check-ins with employees to discover the underlying stories that are informing and influencing them and their work. Ask questions like:

- What delights/frustrates you?
- What refuels/drains your energy and motivation?
- What value do you perceive from what you currently do?
- What impact do you perceive you have?
- What should we stop doing?
- What should we start doing?

Make sure to ask these questions of the people who spend the most time using these tools in their day-to-day work.

# Elimination of Tools

- Decisions around both selecting and eliminating tools should be done on the ground, so to speak, as the people who use the tools are much more invested as stakeholders than people who don’t.
- These decisions should not be made as top-down mandates from people who haven’t had any direct, recent experience with the tools in question.