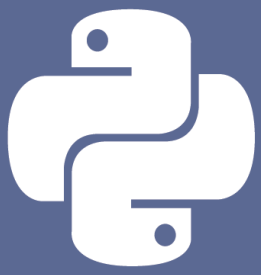
ActiveState®



THE POLYGLOT ENTERPRISE

BEST PRACTICES FOR ADDING A NEW LANGUAGE

THE POLYGLOT ENTERPRISE

BEST PRACTICES FOR ADDING A NEW LANGUAGE

Polyglot is now the norm in most enterprises. The power shift toward developers - those that create the Intellectual Property that differentiates your business - coupled with the technology shift toward services/ microservices has resulted in polyglot: the proliferation of programming languages and technology stacks across the enterprise.

For example, a recent study[1] examined the data from Kaggle's 2018 Machine Learning and Data Science survey[2], which collected more than 23,000 responses from professionals, including their use of 16 different programming languages. The study's analysis of the data revealed which programming languages tend to be used together, such as:

- Python, Bash, Scala
- R, SQL, Visual Basic/VBA, SAS/STATA
- Java, Javascript/Typescript, C#/.NET, PHP
- C/C++, MATLAB
- Julia, Go, Ruby

[1] http://customerthink.com/usage-driven-groupings-of-data-science-and-machine-learning-programming-languages/
[2] https://www.kaggle.com/kaggle/kaggle-survey-2018

THE POLYGLOT ENTERPRISE

However, as most professionals will attest, adding a new language to the enterprise is not a trivial matter. In fact, 67% of respondents that ActiveState polled for our 2018 Developer Survey said they would choose not to add a new language because of the difficulties, which include:

Education. All of your developer, DevOps and other teams that touch code will need to learn a completely new language, as well as any new tooling that language entails.

Tooling. From IDEs to testing top packaging, depending on what you're already using, you may need to either extend or entirely replace your toolchain.

Workflow/Processes. Some languages are more flexible than others, and may well fit within your existing Software Development Lifecycle (SDLC). Others may require you to make extensive changes.

ActiveState has been build engineering open source languages for more than 20 years. We've seen what it takes to introduce, adopt and become proficient with a new language. The following are some best practices any software development team should keep in mind in order to lessen the pain of adopting new languages.

BEST PRACTICES FOR DEVELOPERS

Learning the syntax of a new language is tedious, but straightforward. The hard part is learning how to use a language properly, as well as understanding which libraries/frameworks are available and when they should be called. In all cases, keep in mind the following tips:

- Formal classroom training is usually not sufficient. Start with a small project doing pair programming (employee + consultant) in order to train your trainers, which can they be paired with other employees.

- Find a good linter for the new language (preferably one that's integrated with your IDE) in order to enforce good coding styles and standards, as well as ensure developers avoid common bugs and pitfalls. Start by building shared libraries. After all, you're new to the language, but not new to the business.

- Make sure there's a large and active community contributing to the language since it will make it easier to find code examples, libraries and ensure those libraries are well supported.

- Implement open source governance by implementing tools that will help you track security vulnerabilities/CVEs, identify which licenses can and cannot be used in your corporation, and white/black list those libraries that don't fit your company's policies.

THE POLYGLOT ENTERPRISE

BEST PRACTICES FOR DEVOPS

While developers tend to work with a single language/technology stack per project, DevOps typically work in polyglot environments that feature multiple tech stacks located locally, in the cloud, and/or bridging the two via hybrid cloud implementations. All of which means that any new language introduced will have a cascading effect as it becomes more entrenched in the organization and proliferates across environments. Some things to keep in mind:

- Don't treat the new language as a special, one-off case. Rather, as much as possible, strive to make it fit within your existing best practices (i.e., an automated CI/CD chain and infrastructure as code with version-controlled configuration).

- Making the new language fit your existing processes doesn't necessarily mean using the same tooling/infrastructure. Some of your existing tooling/infrastructure will readily accommodate the new language (with the benefit that there's no need to learn new tooling). However, for other tools, the new language will be a stretch. In these cases, we'd recommend going with dedicated tooling for the the speed, community support and (likely) more frequent updates they offer.

- When it comes to deploying, implementing and configuring the required tooling/infrastructure consider hiring a consultant so you can not only get it right the first time, but also understand the best ways to incrementally improve key metrics (faster build/test/deploy times, reducing # of failures, etc).

- Evaluate existing monitoring tools to understand whether they can accommodate the new services/applications being built, or whether dedicated monitoring tools are necessary.

THE LANGUAGE DISTRIBUTION

It may seem like obvious advice, but ensure everyone standardizes on the same open source language distribution/version in order to minimize inconsistencies across your team. Otherwise, you run the risk of differently configured development environments, which are a key culprit in build failures. When it comes to language distributions, you have three choices:

1 Community distributions are free and ubiquitous, and probably came with your operating system. They're a great way to get started learning the basics, but are limited by the fact that they don't include popular community libraries.

2 Commercial distributions are vendor-supported archives that include the core language plus many of the most popular community-created libraries. These distros can be a good choice for exploring both the language and its ecosystem, and certainly ActiveState has a long history of offering some of the most popular, commercial open source distributions for Python, Perl, Tcl, Go and Ruby.

3 Do-It-Yourself distributions are the best way to create a development environment that requires a specific set of languages, libraries and tools, but can be far too complex when you're just starting out with a new language.

THE POLYGLOT ENTERPRISE

THE FUTURE - POLYGLOT WON'T KILL THE ENTERPRISE

Polyglot is now the norm. As more enterprises increase their use of polyglot projects the associated burden of managing them. Eliminating the complexities of managing dependencies, reducing the build engineering and environment setup time of new languages, and automatically updating libraries as vulnerabilities are found will become business critical. These are some of the key problems that the ActiveState Platform has been designed to solve.

The Platform will automate the building, certifying and resolving of open source languages for any language distribution -- even distributions that include multiple languages. As a result, developers can dedicate time to high-value work, and enterprises can accurately gauge risk, knowing what packages are running in which environments, and what their threat level is.

SEE PREVIEW AT
https://www.activestate.com/products/platform/