# [The Best Programming Languages for Each Situation](https://tomassetti.me/best-programming-languages/" \o "Permanent Link: The Best Programming Languages for Each Situation)

June 8, 2017/in [Software Development](https://tomassetti.me/category/development/) /by [Gabriele Tomassetti](https://tomassetti.me/author/gtomassetti/)



There is a question that many people take as a sign that the questioner does not understand the subject at all. Some people even find it enraging. The question is usually in the form *What is the best X?* What is the best car? What is the best programming language? But at the same time it is a question that we ask ourselves every time we start a project or we pick a car.

The difference, of course, is that we ask this question in our specific context. We want to know the best programming language for us, for the situation we are in.

We obviously cannot know your situation, but with this article we hope to provide useful information to whoever is thinking about entering a new field or just want to know the current state-of-the-art.

**The Best Programming Languages**

Bottom of Form

**Table Of Contents**

1. [Changing Programming Language Has Costs](https://tomassetti.me/best-programming-languages/#changingCosts)
2. [When To Change Programming Language](https://tomassetti.me/best-programming-languages/#whenToChange)
3. [A Huge Amount Of Money Can Change The Language Itself](https://tomassetti.me/best-programming-languages/#moneyHelps)
4. [A Few Criteria For Choosing A Language](https://tomassetti.me/best-programming-languages/#criteria)
   * [Good Enough Technical Qualities](https://tomassetti.me/best-programming-languages/#technical)
   * [Popularity](https://tomassetti.me/best-programming-languages/#popularity)
   * [Community Fit](https://tomassetti.me/best-programming-languages/#community)
5. [The Best Programming Languages For Some Specific Contexts](https://tomassetti.me/best-programming-languages/#listBest)
   * [For Enterprise And Industry-Minded Academia](https://tomassetti.me/best-programming-languages/#enterprise)
     + [Java](https://tomassetti.me/best-programming-languages/#java)
     + [Kotlin](https://tomassetti.me/best-programming-languages/#kotlin)
     + [C++](https://tomassetti.me/best-programming-languages/#cplusplus)
   * [Theoretical Academia](https://tomassetti.me/best-programming-languages/#interesting)
     + [Python](https://tomassetti.me/best-programming-languages/#python)
     + [Julia](https://tomassetti.me/best-programming-languages/#julia)
     + [Many Other Small Languages](https://tomassetti.me/best-programming-languages/#small)
   * [Concurrency And Reliable Software](https://tomassetti.me/best-programming-languages/#concurrency)
     + [Erlang](https://tomassetti.me/best-programming-languages/#erlang)
     + [Elixir](https://tomassetti.me/best-programming-languages/#elixir)
     + [Go](https://tomassetti.me/best-programming-languages/#go)
   * [System Software](https://tomassetti.me/best-programming-languages/#system)
     + [C](https://tomassetti.me/best-programming-languages/#c)
     + [C++](https://tomassetti.me/best-programming-languages/#cplusplus2)
     + [Rust](https://tomassetti.me/best-programming-languages/#rust)
   * [Game Development](https://tomassetti.me/best-programming-languages/#game)
     + [C++](https://tomassetti.me/best-programming-languages/#cplusplus3)
     + [C#](https://tomassetti.me/best-programming-languages/#csharp2)
   * [Rapid And Productive Web Development](https://tomassetti.me/best-programming-languages/#webDevelopment)
     + [Javascript](https://tomassetti.me/best-programming-languages/%22#javascript)
     + [TypeScript](https://tomassetti.me/best-programming-languages/#typescript)
     + [PHP](https://tomassetti.me/best-programming-languages/#php)
   * [Apple Software](https://tomassetti.me/best-programming-languages/#apple)
     + [Swift](https://tomassetti.me/best-programming-languages/#swift)
     + [Objective-C](https://tomassetti.me/best-programming-languages/#objective-c)
   * [You Have To Build Many Things With One Technology](https://tomassetti.me/best-programming-languages/#boh)
     + [C#](https://tomassetti.me/best-programming-languages/#csharp)
     + [JavaScript](https://tomassetti.me/best-programming-languages/#javascript2)
6. [Two Interesting Languages](https://tomassetti.me/best-programming-languages/#two)
   * [Prolog](https://tomassetti.me/best-programming-languages/#prolog)
   * [Haskell](https://tomassetti.me/best-programming-languages/#haskell)
   * [What Does All This Mean](https://tomassetti.me/best-programming-languages/#meaning)
7. [Good Languages That We Could Not Fit Anywhere](https://tomassetti.me/best-programming-languages/#good)
8. [Summary](https://tomassetti.me/best-programming-languages/#summary)

**Changing Programming Language Has Costs**

If you have a large codebase, you probably do not really choose a language for a new project. You just use the one you have used so far. The main factor is the cost of adding a new language. Hiring new developers, changing your infrastructure, learning the best practices, is simply too much for many companies.

Another problem is that while it is reasonably possible to learn a new language in a month or two, it takes much longer to become proficient at that language. So both developers and their employers prefer to take advantage of the knowledge they already have.

In such cases the answer to the best programming language question usually becomes what you are already using, which is a boring answer. Although this not necessarily mean that is the wrong answer.

**When to Change Programming Language**

**

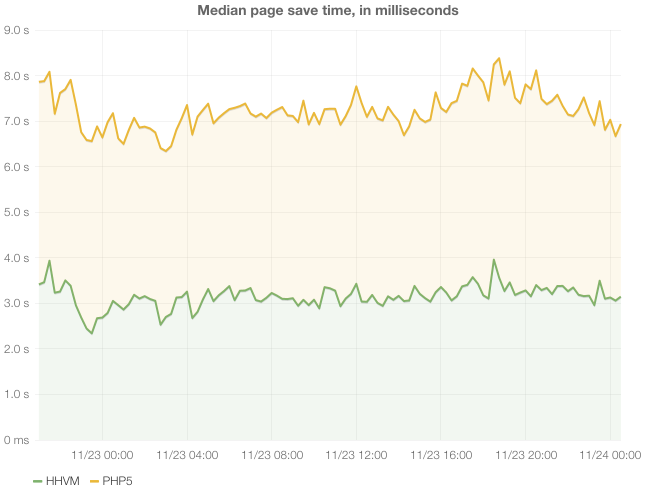
*Twitter outages were so frequent that the error page became a mascot: Fail Whale*

The necessities of their business may force some companies to change programming language, like when Twitter changed from Ruby to Java and Scala. The performance of Ruby was simply not good enough and was impacting their ability to deliver their service. The change was costly, but necessary. Other than performance there were [other technical factors behind their choice](https://www.infoq.com/articles/twitter-java-use/).

No language is perfect, in fact there seems to be new problems with Scala, [such as the learning curve](https://en.wikipedia.org/wiki/Scala_(programming_language)#Criticism). But it is fair to say that, in some situations, it is not true that all language are equals. Which is a common refrain of some people.

When you have significant and peculiar needs, some languages are really better than others.

**A Huge Amount Of Money Can Change The Language Itself**

**

*Graph of the performance improvement brought by*[*HHVM to Wikipedia*](https://code.facebook.com/posts/1645476812348639/hhvm-adoption-news/)

There is another famous case that shows that money can help you in some cases: Facebook and PHP. PHP has been accused of many things, many of them true, especially when it comes to security. But few people would say that PHP is particularly slow. It is just that Facebook was, and is, so big that any improvement in performance can save them a significant amount of money.

On one hand PHP was not the language with the best performance, but it was easy to work with and they had a huge codebase in that language. You might say that they had outgrown the language. But instead of changing their development language they changed the language itself and the way it worked. They created not one, but two technologies to improve its performance: first [HipHop for PHP](https://en.wikipedia.org/wiki/HipHop_for_PHP), a PHP-to-C++ transpiler, and then the more famous [HHVM](https://en.wikipedia.org/wiki/HHVM), a virtual machine for PHP and Hack. In fact they also invented a dialect of PHP called Hack.

They could keep PHP because its performance was not so bad, nor they required it to be that good. Twitter simply could not work with Ruby, while Facebook could work even with normal PHP, it would have simply cost more.

**A Few Criteria For Choosing A Language**

While we all agree that there is not a best programming language in absolute terms, we think that there are *preferable programming languages for specific tasks*. We think that it is possible to set some reasonable criteria to guide professional developers and business alike. These criteria can help you choose the best programming language for your situation.

**Good Enough Technical Qualities**

Aside from the obvious performance requirements, the language must have good technical qualities for your needs. For instance, if your software is heavily concurrent you need a language that has first-rate support for that.

Another problem that Twitter had was that LAMP model of Ruby deployment did not support encapsulation well. So it was difficult to build a separate and independent storage or search service. This was not a case of *Ruby is bad*, but more *Ruby is designed for something else*. Remember that technical qualities does not just mean the things you can measure and see, like performance or the syntax. It also means how the language works behind the scenes.

So you do need to choose a language that fit well your use case, a language that checks all the boxes. Although you do not necessarily need to pick the one that best fit your use case. That is because it is not always possible to rank different needs or, even if it were, you might not know which one matter the most to you.

For example, imagine that you can determine that language X would be better if you achieve 5 million users or more, while language Y would make easier and cheaper to reach 5 million users. How can you know if you will ever reach 5 million users? Maybe language X would be too costly to use at the start and so your company will fail before reaching that many users.

Consider technical requirements as a filter, your language must pass them, but it does not need to be the best possible language for that.

**Popularity**

The language you choose should be popular enough. This can help you saving money and time, especially because of open source development. For instance, if your company use PHP, now you can take advantage of the hard work of Facebook developers. You can use HHVM to improve the performance of your software. It can also make more probable to find good ready-to-use libraries to speedup your development.

There are several ways to [measure the popularity of a programming language](https://en.wikipedia.org/wiki/Measuring_programming_language_popularity):

* the quantity of jobs available
* the quantity of search engine searches
* the quantity of GitHub projects
* …

All of them are useful, none of them is perfect. For instance, there are many JavaScript projects on GitHub that are composed of one file and are currently inactive. That is simply because the language and the tools encourage you to create very small libraries. How can you understand the popularity of a language called Go (there is also another one called Go!) looking at search engine results?

**Community Fit**

The language must also be popular in the community you belong to. A good community fit has many advantages: developers think the way you want them to think and they also usually have non-programming skills or knowledge that you need. Which means that you have to spend less time in training them and you are less at risk of hiring the wrong developer.

PHP has improved a lot in the technical department, but it is still chosen by people that want programming to be easy. Which means that there are many not-that-good PHP programmers and it might be hard to find the good ones. Even worse, you may not find out if they are good until you make them work for your project and you see the results.

On the other hand, maybe not all people that works professionally with C and C++ have a computer science degree, but probably they all think like the people that do have one. Simply because if you do not think, plan and write C/C++ code in the proper way, your code will make you suffer for your mistakes soon enough.

Also, some languages are typically embraced by certain communities, so there are more libraries to cover that area. Think about the amount of libraries that are being developed to analyze data in Python. This is a consequence of the fact that many data scientists have embraced Python. You can benefit from this by adopting Python if you need to develop applications in that area.

**The Best Programming Languages For Some Specific Contexts**



We have made this list for pragmatic purposes. We did no try to find the best programming language for each possible niche. We simply listed the sectors for which we could find at least two programming languages which fit reasonably well.

Let’s see our list of best programming languages, so we can all start quarrel about it.

**For Enterprise And Industry-Minded Academia**

The worlds of enterprise and academia are both full of unmovable objects.

While there is innovation, there is also a proper way of doing things and there must be a good reason to change. By industry-minded academia we mean academia that is linked directly with real products and businesses. For instance when we talked about [parsing in Java](https://tomassetti.me/parsing-in-java/) we found out that many of the best tools came directly from academia. Indeed research in all fields of academia may end up in real products, but usually the path is more circuitous.

**Java**

It could be argued that C# is a slightly better language than Java and the CLI is more flexible than the JVM. But Java is a good language and the JVM is a good platform. More importantly everybody else use it, so you should do that too. In other words, Java is good enough and it is very popular which is especially relevant in the enterprise world. Java is also taught in many computer science courses, but the enterprise presence it is the reason is used in industry-minded academia.

In the enterprise world changing a platform does not just mean changing the codebase, but finding a whole new set of experts and solutions for all the technical, regulatory and business problems. It is frequently not worth the effort. And for Java there are many libraries and products already available which are non-existent for everything else. There are even special [Java platforms for the enterprise](https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition) that add new features specifically for that world.

**Kotlin**

If you are in a Java company, the only way to get partially out of Java might be Kotlin. Kotlin is a language designed to be safer and more productive than Java, while being easy to use and 100% compatible with Java. It is developed by the famous software development company JetBrains. Recently it became the second language (after Java) to be [supported by Android](https://android-developers.googleblog.com/2017/05/android-announces-support-for-kotlin.html). It is also used by [Corda](https://github.com/corda/corda), a distributed ledger developed by a consortium of well-known banks.

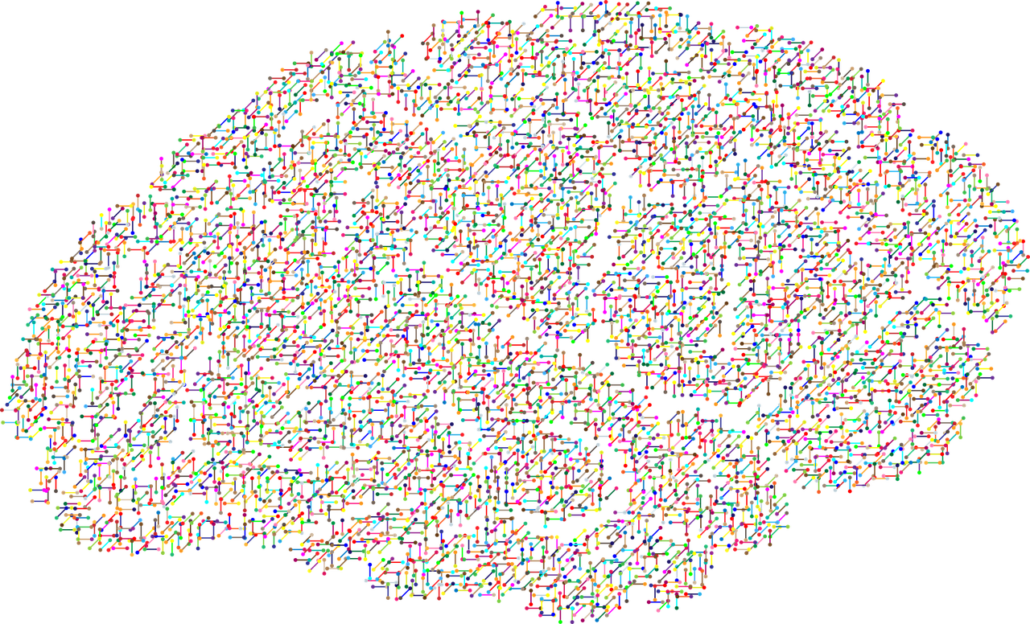
**C++**

C++ was born as an improvement of C and it largely succeeded, at least in the eyes of the public. [Linus Torvalds might not like it](https://lwn.net/Articles/249460/), but most people do. If your company need a language with great performance and closeness to hardware it is a great choice. Although compatibility with C is still important, it is not easily obtained now. If nothing else because both languages have kept evolving.

This long history has given similar advantages to Java for the enterprise, namely that there is a lot of code already written in it. It has also great performance. One reason is because the language is lower level than Java and most languages used by the average programmer. Another reason is that some of the best programmers in the world have dedicated their life to optimize its compilers. They have not just worked on the code for the compiler themselves, but they have studied and developed many techniques to improve them.

The history has also given some disadvantages: there are some many features that it does not really exist a standard way of using C++. You will probably have to enforce your way of using it, to avoid many maintenance problems down the line.

**Theoretical Academia**



By theoretical academia we intend to contrast it with the industry-minded academia that we have talked about before. We mean to indicate the kind of work in academia that people find really academic. The stuff that is more experimental and farther away from real applications. That stuff that will be *eventually* prove to be useful.

**Python**

The only good general answer for this category is Python. It is not the only language that is widely used, but it is the only language that is used in many different fields. It has some important technical features, such as the fact that is easy to extend and embed. But it is so popular because it is designed to be beautiful and fun to use.

For instance, Python uses whitespace instead of curly brackets to delimit blocks of code. The language and the community support a culture that lead to code that reject clever tricks, but prefer instead code that is readable, simple and explicit. So it is easy to use and it is thus great for prototyping. In many ways, it is the best programming language to underline the importance of culture in programming.

A good example of what we mean by theoretical academia is artificial intelligence. And in fact Python is very popular in several areas of artificial intelligence: from machine learning, with libraries like [scikit-learn](https://en.wikipedia.org/wiki/Scikit-learn) and [TensorFlow](https://en.wikipedia.org/wiki/TensorFlow), to natural language processing, with [NLTK](https://en.wikipedia.org/wiki/Natural_Language_Toolkit). Python has also great support for scientific computing, with [SciPy](https://en.wikipedia.org/wiki/SciPy), and a whole lot of mathematical needs, with [NumPy](https://en.wikipedia.org/wiki/NumPy).

**The Bad Part**

Unfortunately, there are downsides to the use of Python: many practical aspects are not taken care of. For one, being an interpreted language performance is not great compared to C++, although it is not terrible either. This is one of the reasons that has led to the creation of several different implementations. There are also implementations made to support JVM or .NET or even make Python run on microcontrollers.

Another problem is that there is not yet a standard and easy way to deploy Python applications, which means that [it is surprisingly complex](https://www.nylas.com/blog/packaging-deploying-python/).

The migration to version 3, released in 2008, is problematic and still ongoing. Version 3 is incompatible with the previous one and this has resulted in the longest migration time in recent memory.

**Julia**

Julia is a language that is made to address the need of high-performance scientific computing, although it can also be used for more mundane web development. In very short terms: it is a more performing and less popular version of Python. Indeed, it has also other technical advantages, like good support for concurrency and parallelism and the ability to call directly C, Fortran and Python code. Given the respective communities and intended usage it is often used together with Python.

**Many Other Small Languages**

There are many other small languages that are used in some part of academia or industry when complex mathematics is needed. One such language is **R,** a programming language that is widely used in statistics and data analysis. R has some useful features for such sectors, like its own documentation format and the fact that many standard functions are written in R itself, thus they are easy to analyze for their users. But the best advantage is the great availability of libraries and expertise for statistics and data analysis.

**MATLAB** is both a language and a computing environment. Which means that it can be readily and easily used by people that are not programmers by trade. This category includes most scientists. It is greatly used for technical computing in industry and academia.  Being proprietary software it has literally a price tag. A famous competitor of MATLAB is **Wolfram Mathematica**, which is behind the notable [Wolfram Alpha](http://www.wolframalpha.com/), a *computational knowledge engine*.

Both R and MATLAB are technically programming languages, but for the most part they are not used as such or by programmers. They are used by scientist and industry experts for research and development.

**Concurrency And Reliable Software**

The fundamental issues related to concurrency affect any software. For example, how to coordinate the access to shared resources it is a problem that could affect any program. In practical terms, though, most developers tend to ignore it. This is because the basic solutions provided by most programming languages are good enough. Where it could really start affecting them, like web development and access to databases, very smart people have done a lot of work for them.

When you start to have same real strong requirements in terms of concurrency, you might adopt a functional programming language.

However, if your business depends on it, you want something more. A language that is explicitly geared to develop highly reliable concurrent applications.

**Erlang**

Any sufficiently complicated concurrent program in another language contains an ad hoc informally-specified bug-ridden slow implementation of half of Erlang.

*–*[*Robert Virding*](https://rvirding.blogspot.it/2008/01/virdings-first-rule-of-programming.html), one of the creators of Erlang

When you absolutely need to develop distributed, fault-tolerant, reliable, soft real-time, concurrent applications you choose Erlang. It is an open-source language developed by Ericsson, a telecommunications company, exactly for such needs. The language itself obviously has proper technical features like being a functional programming language and implementing an [Actor-model](https://en.wikipedia.org/wiki/Actor_model).

However, the real advantage of Erlang it is that the whole ecosystem is developed with that objective in mind. For instance, it has some interesting features relative to the [supervision of the processes and their organization](http://erlang.org/doc/design_principles/des_princ.html). In fact, Erlang it is not just a language, but a runtime system and some ready-to-uses components like a distributed database. It has tools to check the quality of the code or the execution of the program. The whole system is called Open Telecom Platform (OTP).

In short, Erlang is a battle-tested language and platform to create reliable and concurrent applications.

**Elixir**

Erlang has been developed for the enterprise which means that it might be technically flawless, but not that pretty or very productive. To address this flaw José Valim created Elixir, a language built on top of the Erlang platform. For instance, Elixir added support for extensibility. You could think of Elixir as Erlang for people that are used to Ruby or Python.

It is gaining increasing popularity and is used by companies like Pinterest and Moz, but it has not yet a very large number of users.

**Go**

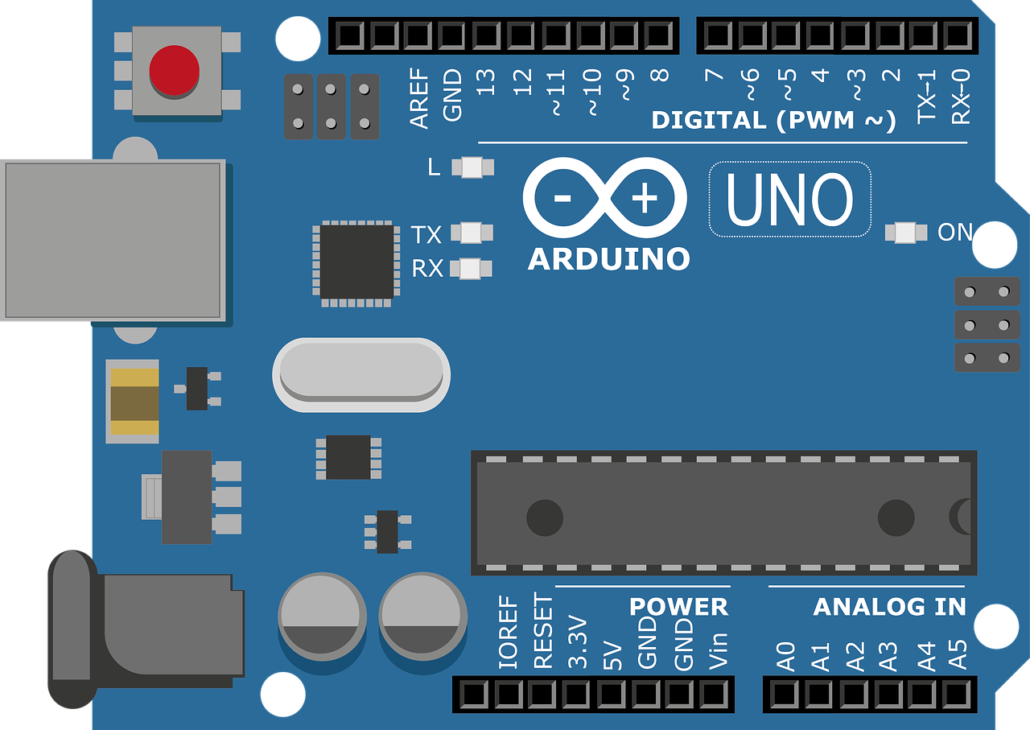
Another company that has a need to develop concurrent applications is certainly Google. Because of that is not surprising that people that works there created Go. It started as an experiment by some engineers, to develop a language that improved on C++ and Java. More than an evolution is a re-imagination of these languages, with some added benefits.

It was designed to be scalable to large systems and usable without an IDE, but also productive and being especially good at networking and concurrency. Other than a well-thought design, it has some specific features for concurrency like a type of light-weight processes called *goroutines*.

All its authors expressed a dislike for C++ complexity. So, in some way, it is a language designed to persuade C people to enter the new century.

While being developed at Google it would be incorrect to say that is backed by Google in the same way Microsoft backed C#. People working at Google have developed other languages, too. So it is more a language that many people like, including some people that work at Google. In fact, it is used also by Docker and Dropbox.

**System Software**



System software is a software meant to be used primarily by other programs. It has some specific needs: performance and closeness to hardware. If you are writing system software, you are the one that have to deal with all the hardware differences so that everybody else can work more productively. You have also have to offer the best possible performance, because everything else depends on you. Not all programs need a 1% more efficiency, but some do and you are writing your software also for them.

**C**

C was a revolutionary language that essentially eliminated all of its then contemporary competitors, except for a few niches. Despite being still loved by many of its users, for everybody else it has some issues. Some of them, like manual memory management, and not being very productive might be necessity for performance. But others, like not having any feature to support large scale programs or the arcane preprocessor, are mostly legacy of its age, that nobody would put in a contemporary language.

Nonetheless there is simply no alternative that has the same large number of libraries and optimization that C has. No other language has that many expert developers in system programming. And its users must be expert, because the language does not help you in any way to develop good and large applications.

The compilers for C language are so efficient that the language has also a second-life has an intermediate language. For example, implementations of Python and PHP are written in C.

**C++**

C++ is also widely used for system programming. It has the advantage of better supporting large applications, for instance with object-oriented programming. But some people find this and other additions simply unnecessary for system programming.

**Rust**

Rust is a new language sponsored by Mozilla for system programming. It is open-source and thus open to the contributions of the community. But the design of the language evolved with the development of Servo, an experimental layout engine for a browser. It is loved by many of its users, and in fact is the *most loved programming language* according to a [survey of developers by StackOverlow](https://insights.stackoverflow.com/survey/2017#most-loved-dreaded-and-wanted).

Rust is designed to have also great support for concurrency, memory safety and large scale applications. For instance, you cannot have null or dangling pointers, which can cause memory-related bugs in C or C++. It has classes and a performance comparable to C++.

It is much less popular than C or C++ and it has not been used in widespread applications. Mozilla plan to eventually use it for Firefox (although not directly with Servo). At the moment the most important software that uses it is probably Tor, an anonimity network for which security is literally the *raison d’etre*.

**Game Development**

Game development might seem the most orthogonal thing to system programming, since you really do not want [software that plays games](https://en.wikipedia.org/wiki/WarGames). Nonetheless there is a similar need for performance. This is useful for some demanding games, but it is mandatory for building a game engine, a software that will be used to simplify the development of games.

Game development has also a diametrical need: productivity. This is a requested feature for many businesses, but it is a life-or-death reality in game development. No other software industry is so dependent on a good launch for its revenue. A game developer could simply fail before even launching the game.

The combinations of these requirements lead to an industry that is heavily reliant on specific libraries, game engines, SDKs and tools.

**C++**

The most used programming language for professional game development is C++, which marries good productivity with performance. It has also the most experienced developers. Some software may be written in C, but [most developers prefer using C++](https://developer.nvidia.com/open-source-vulkan-c-api). Notable examples of software made for C++ are DirectX technologies and the Unreal Engine.

**C#**

If you do not need optimal performance, or prefer easier development and cross-platform support, C# is a good choice. It is really a matter of libraries and SDK. Initially Microsoft created XNA to make game development more accessible on Windows and Xbox, but was later supplanted by the cross-platform MonoGame. It is also the preferred language to use with the widespread game engine Unity.

**Rapid And Productive Web Development**

**

*Try as you might, you cannot escape WordPress*

There is a lot of web development going on. It is probably the most widespread field of development. Everybody need to develop web applications or at least applications that interact with the internet. So the capabilities are there, both in the sense of available libraries and available programmers. Almost any popular language power some large websites, proving that it can be done professionally. Heck, there are even [C++ frameworks for rapid web development](http://cppcms.com/wikipp/en/page/main).

Then the rationale for using one language rather than the other mostly relies on the community you need and/or the language you know. For example, your application might be related to technical computing and then you choose Python.

If you have no preference though, we can present some default choices.

**JavaScript**

JavaScript is obviously the only choice for client-side web development, but it is used also on the server-side. There are countless libraries for JavaScript, in every possible style of development. There are web frameworks for all kind of applications, from [ambitious ones](https://emberjs.com/) to [simple ones](http://riotjs.com/). So you can also find everything that suits your needs and your style.

There are fundamentally two reasons for the spread of JavaScript to the server: optimization of the runtime environments and the competence of developers.

Multitude of developers have spent a lot of time to optimize the performance and quality of the JavaScript engines and the tools used with the language. Probably only C and C++ can claim a similar level of investment. Thus it makes sense to take advantage of such efforts and use it everywhere.

JavaScript can offer the best of the best developers, that have learned both to write code fast and write it optimized for performance. You want to use their skills everywhere you can. And the other side of the web, that is to say the server, need very similar expertise.

On the other hand, JavaScript might be the only language that is used, for the most part, by people with no formal training or education in it. People that have not even read a manual of JavaScript, but they have sort-of learned it by copy and paste. People that have started using jQuery before they started using JavaScript. They also contribute to the spread of JavaScript, because it is the only thing they know.

This means that you might have to take some time to find the right developers for your project.

**TypeScript**

New web frameworks have made easier to build large and robust applications. Despite that some people think JavaScript itself is not enough. For these people TypeScript might be the ideal compromise. It is a superset of JavaScript with static typing and class support that transpiles to JavaScript itself. Which means that you can reuse the JavaScript infrastructure in deployment.

It is developed primarily by Microsoft, but it is also used by Google for [Angular](https://en.wikipedia.org/wiki/Angular_(application_platform)) (not to be confused with AngularJs).

**PHP**

PHP was designed as a set of tools to simplify and speedup server-side web development. You could accuse PHP of being a badly-designed language. But you would be wrong: **it was not designed at all**, it was not even intended to be a language.

Rasmus Leedorf about the creation of PHP:

I really don’t like programming. I built this tool to program less so that I could just reuse code. [..] I don’t know how to stop it, there was never any intent to write a programming language […] I have absolutely no idea how to write a programming language, I just kept adding the next logical step on the way

On the other hand, this is a testament of the power of the philosophy of just getting things done. Because PHP did work, not beautifully or securely, but it did work where nothing else could. Indeed, until few years ago it was the only easy way of doing server-side web development. It is easy to deploy and it has a huge range of libraries for all the web development needs. For these reasons, even today, CMS are a PHP stronghold.

Nowadays the field is full of competitors. PHP is still the most widely used server-side language, but it is not the only option anymore. Also the language has improved a lot and PHP 7 is a real designed language. There are many good web development frameworks, like [Laravel](https://laravel.com/) and [Phalcon](https://phalconphp.com/), that encourage robust, secure and well thought development.

At the same time PHP still suffers from its beginnings. You can create good PHP code, but fewer of its users do that. The reason is that PHP still attracts many users that just want to programming to be easy. So you have to pay attention to find the good PHP developers.

**Apple Software**

**

*The glass is compatible only with Apple iWater*

Despite the tendency of describing their software with an overabundance of words like *magic*, Apple write mundane software, like the rest of us. So it may seem strange to dedicate a specific section to Apple software. The reason because it makes sense to do so, is that the company has a strong vision to inspire, which is often also an order to follow.

You can certainly develop for their platforms with normal languages, like C#, and there is also some third-party support for that. But Apple makes more convenient to use their languages. They provide better support and first party tools, like the Xcode IDE. Other than the need to learn a new language the disadvantage is that the languages have some peculiar features. For instance, they use [ARC](https://en.wikipedia.org/wiki/Automatic_Reference_Counting) instead of the more common garbage collection for memory management.

**Swift**

Objective-C without the C

Swift is the newer language and the most productive one. It is already the most used and [most loved](https://insights.stackoverflow.com/survey/2017#most-loved-dreaded-and-wanted)programming language among Apple developers. It uses a more familiar syntax and support more programming paradigms, for instance functional programming. So you should probably use Swift.

**Objective-C**

While Apple might decide any moment to deprecate anything in its stack, it is unlikely that will do so for Objective-C. That is simply because a lot of its codebase and infrastructure is written in Objective-C. So you could choose it if you needed a slightly better performance.

But the language is one of the [most dreaded programming languages](https://insights.stackoverflow.com/survey/2017#most-loved-dreaded-and-wanted). It could be unkindly defined as a worse version of C++. Something made to address similar concerns to the ones that lead to C++, only that it does that in a weirder way.

**You Have To Build Many Things With One Technology**

There are occasions when you have to build something, but you do not know how exactly the final product will look like.

Or you have to build many things, but you are required to pick one technology. It could happen working with or for small businesses. It may be useful working as a freelance developer.

It can also happen when you are not sure how your product will develop, because you cannot anticipate the market. You might start building a desktop application, but your clients demand certain functionalities to be accessible by multiple users, so you add a web application, then…

**C#**

When you have unstable and different requirements you need a language that is adaptable to all situations. Not just in the theoretical sense of being a general purpose programming language, but something that has been used to build all sorts of things. There are tools that can help you in everything. C# is the best language in such cases.

It was not meant to be a revolutionary language, but basically a better version of Java and C++. Since all three languages kept evolving we could argue forever if it has succeeded and find a new answer every year. Technically C# support particular features that no other language has, like [LINQ](https://en.wikipedia.org/wiki/Language_Integrated_Query), that adds support for querying data easily.

A few examples of the real usage of C# for: desktop software (Windows), games (MonoGame, Unity), web development (ASP.NET Core), mobile (Xamarin) and even embedded systems ([.NET Micro Framework](https://en.wikipedia.org/wiki/.NET_Micro_Framework)). Some of this software (e.g. .NET Micro Framework, Unity) is written in C/C++, but it is meant to support the use of C#.

**Platform**

An important difference is that, while Java was always designed with portability in mind, C# was designed by Microsoft for its platforms. This has been both and advantage and a disadvantage. The disadvantage is that it was always mostly used on Microsoft platforms and according to its needs. The advantage is that it was brought and used for everything from desktop software to web development.

This has certainly changed, both in the sense that now [.NET Core](https://en.wikipedia.org/wiki/.NET_Framework#.NET_Core) is designed and actively supported to be cross-platform. Also Microsoft does not use C# for everything anymore. On the other hand, Microsoft keep developing what many consider the best IDE of all: Visual Studio.

The CLI standard is independent from C#, while a Java Virtual Machine (JVM) is very much tied to support Java. This means that an implementation of CLI, like the .NET Framework or Mono, can support many other languages as first-citizens, like Visual Basic.NET and F#. While this is not an advantage of C# *per se* it comes with choosing the language. This is useful because you can keep maintaining one platform and infrastructure, but you can keep C# with another language better suited for a specific subsytem.

**JavaScript**

JavaScript is a language created for client-side web development, so it might seem odd to see it as a language for all things. And indeed it is.  But this is not a suggestion, rather the statement of a fact, or an *act of surrender*. JavaScript will be used for everything. It started to be available for use in server-side web development, with Node.js, then it became possible to use it for mobile, for example with [Apache Cordova](http://cordova.apache.org/), then desktop, with [Electron](https://electron.atom.io/) and [WinJs](http://www.buildwinjs.com/) and games, with WebGL.

There are people that have built [parser generators](https://github.com/pegjs/pegjs) in JavaScript or even [entire operating systems](https://github.com/NodeOS/NodeOS) for JavaScript.

We have already seen the reasons that lead to the spread of JavaScript: optimization and developers’ expertise. While the expertise may be less applicable to everything else outside the web, the will and the support of many companies is there. So we will all have to deal with the fact that some people are willing to use JavaScript for almost everything. At this point any resistance seems futile.

**Two Interesting Languages**

There are two languages that we would like to talk about. They are interesting to show what could make a language useful and popular to a diverse community. They are both created in academia and great languages on technical grounds.

**Prolog**

Prolog is a general-purpose logic programming language used for artificial intelligence and computational linguistics. It is still quite popular in that field, but it never really left academia. This is due to its very design. Logic programming is a programming paradigm based on formal logic. So you state some fact, or truth, about an element and then the program itself find the solution. You say thinks like:

* A is true
* B is false only if A is true and C is a cat
* C is a dog

Then you ask the program to tell you what is B.

This is neat, but nobody really found a way to make it work. There are really no artificial intelligence programs useful in real-life with this logical approach. Most of contemporary development in artificial intelligence relies on machine learning or similar non-logic approaches.

Another problem is technical. Prolog is a [fifth-generation programming language](https://en.wikipedia.org/wiki/Fifth-generation_programming_language). In this context, generation does not denote a chronological feature, but a programming language in which the developer states the constraints of the problem and the language itself will find a strategy to solve it. In the case of Prolog the constraints are logical statements.

Well, it turns out that to design a general and efficient algorithm to automatically find a solution is really hard. So the resulting programs are not efficient. This makes impossible to write large scale applications with them.

There are newer languages that attempt to improve on Prolog and make it usable in the real-world, like [Mercury](https://en.wikipedia.org/wiki/Mercury_(programming_language)), but they are not in widespread use.

**Haskell**

Haskell was designed to be the C of functional programming languages: the definitive purely functional programming language. It largely succeeded, but for some time it remained confined to academia and the kind of industry that requires heavy use of advanced mathematics, like finance. Notice that this does not necessarily mean numerical computing.

The functional programming paradigm relies on having *programming functions* that behave as *mathematical functions*. This allows the creation of a function that does not modify outside data or have *observable interactions*with the outside context. Observable interaction refers to things like writing data on a file or raising an exception. Technically this means that a function has no [side-effects](https://en.wikipedia.org/wiki/Side_effect_(computer_science)). This also makes possible to have functions as [first-class citizens](https://en.wikipedia.org/wiki/First-class_citizen). Functions are like any other type and, for example, you can use them as argument of other functions.

Expert programmers have noticed, or probably already know, that these are desirable features for avoiding many problems created by concurrency. And the rising popularity of the internet has increased the need to deal with such problems. So Haskell has increased its popularity. [Facebook and Microsoft](https://wiki.haskell.org/Haskell_in_industry) use it and there are even web frameworks for this language. On the other hand, it is not specifically designed for concurrency, or any industry use, so it is less pragmatic than Erlang or Go.

**What Does All This Mean**

By the standard of academia Prolog and Haskell are both successful languages. Haskell is a usable and popular functional programming language and Prolog in a sense proved that logic programming does not work in practice and it is not easy to use it to solve real problems. That count as success because even a negative answer to an important question is worth the effort.

Of course, by the standard of the real world Prolog it is not that successful. Despite being still the most popular logic programming language. We personally think that Prolog is one of the most interesting languages ever created, but even we could not find a place for it in production code.

This does not just mean that technical features are not enough. That is true, but Prolog and Haskell did have a community and specific uses in mind, in addition to technical features.

It serves to illustrate a strange fact: developers are literally creating the edge of technology, but they benefit from being late adopters. You cannot predict which languages will be successful, nor why. You might spend a lot of time learning and developing with a language, but your community may choose another one. Sometimes the best programming language simply changes. So we cannot guarantee that these will be the best languages forever, only that right now many smart people think they are.

Even the real world success of Haskell might be temporary, because if history teaches us something a language like F#, could easily take its place.

**F#**

From the business perspective, the primary role of F# is to reduce the time-to-deployment for analytical software components in the modern enterprise

Being a purely functional programing language restricts the appeal of Haskell and it makes difficult to understand by the average programmer. It may be easier to adopt a language like F#, that is primarily functional, but support also traditional programming paradigms. F# can run on any platform that implements the CLI-standard, like C#. This also allows it to interact with other languages that run on the same platform and reuse their codebase.

**Good Languages That We Could Not Fit Anywhere**

There are obviously many other programming languages that we could not fit in this analysis. One of the reasons is our approach: some languages are not that popular, some do not have won over a specific community. Sometimes there was simply a better alternative. These reasons had led to the exclusion of languages like Ruby, Perl, Object Pascal and Visual Basic.NET.

There also languages used in very specific niches, for example:

* **ADA**for the US military and related industries
* **Fortran**for numerical computing, especially with supercomputers
* **COBOL** for finance and related industries
* …

We did not mention them because these are very specific, so if you need to work with them you probably already know them.

**Summary**

Talking about the best programming language is risky. It is easy to enrage somebody and hard to say something useful to a large audience. That is why we tried to clearly explain our rationale and provide pragmatic information. We talked about technical features when necessary, but we mostly reported on the community and the best use each language has.

We also tried to make the information accessible to programmers that have no familiarity with a certain field and to non-programmers. So that, if you are working for a small business or thinking about starting one, this might help you in making a decision considering both technical and business aspects.

<https://tomassetti.me/best-programming-languages/>