

## /trends &amp; hypes

# Babylonian language confusion

**The abacus is a counting frame known to have been in use in Egypt 2500 years ago. This arithmetical aid may even date back to 3000 BC. The trail leads to Babylon – the original source of the computer, but also of language confusion.**

New computer languages regularly become available. The question for every company is what to do with them. Keeping abreast of everything is impossible; there is neither the time nor the money. It would not be sensible either, because many new languages will not survive for long. Evaluating and comparing languages requires understanding and knowledge of application, software and vocabulary. Strong and weak typing are not the same as static and dynamic typing. But what exactly are they? And what do they tell us about a language, about the productivity of programmers and about the flexibility, quality and shelf life of the end product? This article aims to answer those questions, in the hope of making the task of choosing the right language easier. Because renewal is essential in order to be able to compete in the eternal race towards ever bigger and more complex systems. Consolidation/standardisation simply is not an option.

## Static and dynamic typing

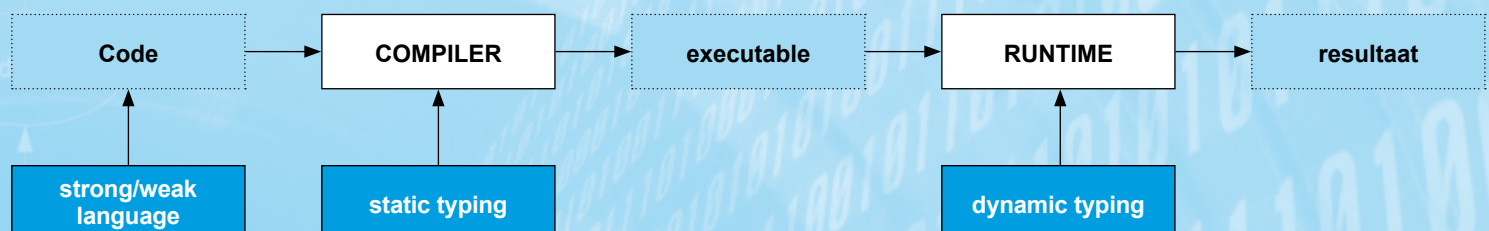
EA language is a bridge between the human being and the computer. The human being uses it to tell the computer what to do. The basis of every language is the same: the user writes a programme as a chunk of text (source code). That is done by the compiler, who translates the text into computer language (the executable). In some languages, it is primarily the compiler who checks for errors in the code. This is called static typing. In other languages, the compiler does less checking, but we come across the errors primarily when the code is run. That is a runtime error. A language that largely leaves error checking to the runtime is a dynamically typed language.

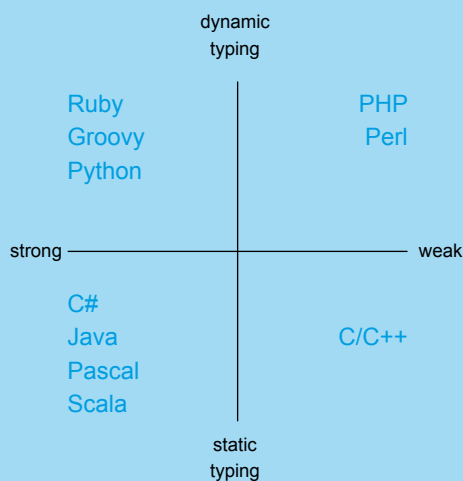
## What is a type?

The term 'type' refers to the kind of data a programme processes. A programmer cannot simply apply any operation to any kind of data. How strictly a language deals with types is indicated by strong or weak typing. So static and dynamic typing is separate from the terms strong and weak typing. That sometimes causes language confusion of Babylonian proportions: you can have strong and weak dynamic typing, and strong and weak static typing. An analogy may be useful to clarify the differences, lest we end up in our own Babylonian language confusion.

## Apples and pears

Let us imagine a computer programme as a fruit stall. In a weak language, the fruit can all be mixed up together. In a strong language, all the apples must be in the apple crate and all the pears in the pear crate. In this example, a crate is the equivalent of an array, which in the weak case you can fill with anything and in the strong case only with real numbers, say. In a weak language, problems can arise when all the fruit in a mixed fruit crate has to undergo the same operation. That cannot happen in a strong language like Java. This first requires all the types of fruit to be defined, before assigning the operation 'peeling' only to bananas and 'cutting' only to apples and pears. In a weak language (such as C), the programmer can easily mix all the different types together, but this in turn can have all kinds of unintended effects and eventually result in a programme crash.





Language	Web framework
C#	ASP.NET
Java	JavaEE, Tomcat, Struts2, Tapestry, Wicket
Python	Django
Ruby	Rails
Groovy	Grails
Scala	Lift
Perl	Catalyst
PHP	Zend Framework

### Hello world

One way of getting a first impression of how user-friendly a language is to see how much code is needed to put the text “hello world” on the screen. In one language, one line is enough, whereas other languages require pretty much a full page of code. It is an indication of the ease of entry. Can I go straight to work, or do I have to learn all manner of things first? Dynamically typed languages have a low entry threshold and are much more flexible in use. Statically typed languages offer a better guarantee of error-free software because the compiler has already tracked down most of the errors. But they are also complex and laborious. The programmer first has to type everything (“this is a crate of apples, those are bananas”), and that takes a lot of work. It demands precise knowledge of the grammar, which raises the entry threshold. So the choice is all about weighing up confidence, flexibility and productivity. A predominantly dynamically typed language allows you to build applications quickly, but the results are more sensitive to errors, so you will need to do much more testing of the end product. So much of the work shifts from coding to testing.

### Tools

Every professional needs good tools. That starts with a good working environment. What the statute book and jurisprudence are to the lawyer, libraries and frameworks are to the programmer. In order to write code, the developer uses an advanced editor similar to Word. It automatically completes text, checks syntax as you type and compiles directly. This type of editor is known as an Integrated Development Environment (IDE). It contains help functions, support and built-in debuggers. In short, an indispensable tool.

### Library and framework

A language usually also has a standard library, in which frequently-used functions are already completely worked out. The programmer can easily add these functions to his work using the Application Programmers Interface (API). A standard library is a part of the language. Alongside this, there are frameworks. These are libraries developed for a special purpose. Again, they contain complete bits of functionality to add a

particular concept at a stroke. Some languages have enormous frameworks, such as Java EE (Java Enterprise Edition).

The skill of a programmer lies above all in his familiarity with that mountain of functionality and knowing where to find what. A good framework, good documentation and good support from an IDE are highly conducive to productivity.

### Popularity and frontrunner drag

The choice of a language determines which frameworks and which development environments you have at your disposal. Conversely, a good IDE and a large framework can make a language popular. But the popularity of a language or an enormous framework can also act as a drag. The result can be that the language barely gets improved or a framework has a very steep ‘learning curve’ and ultimately acts as a drag on productivity. New languages take advantage of that. They can start from scratch and throw all the ballast overboard. Often a new language is more or less a copy of an existing, successful language. A mix of the good things from other languages, just a bit slicker. On the other hand, that does mean that experience and history are lost, including a framework.

### Standardise or renew?

The driving force behind the innovation and evolution of languages is the drive for higher productivity and quality. Sometimes that can be achieved with a good and extensive framework. Other times, it is better achieved with a new language. Pushing a language to make it a standard does not work, or at best it works only temporarily. Languages and frameworks age. We call this ‘legacy’. They cannot keep up with constant innovation in hardware, software and operating systems. The maintenance costs of legacy systems will always tend to increase. It is an eternal onward progression in which no one can permit themselves to stand still. When it comes to games computers, it is accepted that a game for the PlayStation 1 will not work on a PlayStation 3. Similarly, a company that is entirely dependent on its IT systems for its core business will now and then be forced to make a drastic decision and take a big step forwards.