Medientechnik

Programming Languages for the Web 2011

Second Bachelor Thesis

Completed by

David Matthias Stöckl
mt081096

Completed with the aim of graduating with a
Bachelor of Science in Engineering

From the St. Pölten University of Applied Sciences
Media Technology degree course

Under supervision of
DI Markus Seidl

Day _____ Undersign _____

Fachhochschule St. Pölten GmbH, Matthias Corvinus-Straße 15, 3100 St. Pölten, T: +43 (2742) 313 228, F: +43 (2742) 313 228-339, E: office@fhstp.ac.at, I: www.fhstp.ac.at

Declaration

- The attached research paper is my own, original work undertaken in partial fulfillment of my degree.
- I have made no use of sources, materials or assistance other than those which have been openly and fully acknowledged in the text. If any part of another person's work has been quoted, this either appears in inverted commas or (if beyond a few lines) is indented.
- Any direct quotation or source of ideas has been identified in the text by author, date, and page number(s) immediately after such an item, and full details are provided in a reference list at the end of the text.
- I understand that any breach of the fair practice regulations may result in a mark of zero for this research paper and that it could also involve other repercussions.
- I understand also that too great a reliance on the work of others may lead to a low mark.

Day

Undersign

Abstract

This work shows the differences between and the varieties of programming languages for the World Wide Web at present times and also helps understanding these. To reach this aim, the differences between programming languages are analyzed, and the affordances and circumstances of programming for the WWW are determined as well. Furthermore, an analysis of usage statistics is made to show what certain programming languages are in use today.

As a result the most important programming languages for the Web 2011 are determined. Furthermore, a detailed programming language comparison sheet is presented, which can be used to compare the suitability of common and uncommon programming languages for Web use in general.

Table of Contents

List of tables

List of figures

Introduction

While writing this work, the World Wide Web celebrates kind of its 20th birthday: 20 years ago on August 6th 1991 Tim Berners-Lee, a scientist working at CERN, released the Web to the public by inviting all internet users to visit the first WWW-server.

Within the last two decades, the Web, based on the Internet, went through an evolutionary process which is unequaled in human history. Scientists, developers, working groups and programmers motivated by interest or rather economic reasons were creating technologies and solutions (notably the Web browser) to improve and evolve the World Wide Web. As a consequence the Web of today is an incredibly big, interlinked multimedia system for exchanging information, and definitely the most well-known application of a computer network on our planet.

The motivation for writing this work is not to analyze the Web, but the technical base it is built upon. We are not really talking about how interconnected networks are working, but about the possibilities for serving, processing, generating and modifying static and dynamic contents on the Web. This job is done by computer programs, which are written in certain programming languages.

Programming languages are not programming languages. Programming languages from the top view are at least as different among each other as human dialects are, though a lot of them are intended to do similar things. So what are the differences between programming languages? How can they be used to write programs for the Internet? What are the limits and possibilities regarding Web programming? What are the most popular programming languages for the Web, why are they that popular and what is the Web mostly built on?

The evolution of the Web was involving an evolution in programming and programming language history, too. This work is intended to show the results of this evolution: The programming languages which are used on the WWW today, the character of these programming languages, the implementation of these programming languages, the technologies built upon these programming languages and a detailed comparison of these – but also limits and disadvantages of these programming languages and the WWW will be demonstrated. It will be shown furthermore, how new programming languages can evolve and that the current circumstances are not the final ones inside a dynamic and rapidly changing Internet and Web world.

1. The differences between programming languages – advantages, disadvantages and more

A lot of programming languages have been developed during history. While some of these are well-known ones today, others get lost within some time. A lot of people try to design and implement new ones. Familiar or totally new and innovative concepts can be used in programming language design, as long as it these can be implemented.

To reach the aim of this work showing what programming languages to use for the Web and what aspects and features matter, we need to describe how programming languages differ from each other and how to categorize them. Afterwards the requirements for Web use must be compared to the requirements for classic programming, as these are certainly not the same.

1.1 What is a programming language?

“Like any language, programming languages are simply a communication tool for expressing and conveying ideas. In this case we are translating ideas of how software should work into a structured and methodical form that computers can read and understand” (Feminella 2009, “Stackoverflow - What is a computer programming language?”). Things written in a programming language are intended to eventually be transformed into something that is executed; this is why pseudo-code or UML for example are usually not considered to be programming languages.

The term “programming language” is not always used with the same meaning. HTML is sometimes called a programming language, which is considered to be wrong, as HTML does not do anything - it is a markup language to structure documents. An often used term in the context of defining the word “programming language” is *Turing completeness*.

That means the following: “A given programming language is said to be Turing-complete if it can be shown that it is computationally equivalent to a Turing machine. That is, any problem that can be solved on a Turing machine using a finite amount of resources (i.e., time and tape), can be solved with the other language using a finite amount of its resources“ (c2.com 2008, “Turing Complete”).

The TIOBE Index which is trying to collect information about the usage of programming languages only accepts programming languages which are considered to be Turing complete, so SQL for instance is excluded because it is impossible to build an infinite loop with it, for example (Tiobe Software BV 2011, “TIOBE Programming Community Index Definition”).

For this work we will have to make a determination:

- There are the “Turing complete” programming languages which are applicable to solve all common programming problems. Each of these can theoretically be replaced by each other (Feminella 2009, “Stackoverflow - What is a computer programming language?”).
- There are languages which are not necessarily “Turing complete”, usually called “domain-specific languages”. These are often standards or languages used for special tasks (HTML, XML, SQL, CSS …). We will treat these ones in a special way describing their intended usage and role on the Internet without comparing them directly to other languages.

1.2 The classification of programming languages

There are different ways to categorize or group programming languages. Programming languages can be differentiated by various characteristics, like their model of computation, their abstractness, or certain features like garbage collection, automatic type casting, and much more. We need to have a closer look on these differences, so we have a foundation on what we can compare programming languages afterwards.

Note: A lot of programming languages, especially ones with a high hardware abstraction level, cannot be classified exactly or sometimes fit into more than one category.

1.2.1 Classification by computational model

According to Scott (2009, p.11f.) programming languages can be classified into families based on their model of computation. A common way to categorize programming languages differentiates between declarative (**what** should be done by the computer?) and imperative (**how** should the computer do something?) programming languages. We can assume that declarative languages have a tendency to require a less detailed mode of operation during development times, but lower performance and fewer possibilities to change the final execution details are the disadvantages. Compromises between these things have to be made during language design.

Declarative languages (more abstract)

Functional languages	“In essence a program is considered a function from inputs and outputs, defined in terms of simpler functions through a process of refinement” (e.g. Lisp/Scheme, ML, Haskell)
Dataflow languages	“Dataflow languages [...] model computation as the flow of information (tokens) among primitive functional nodes. [...] Nodes

	are triggered by the arrival of input tokens and can operate concurrently" (e.g. Id, Val).
Logic- or constraint based languages	Logic- or constraint based languages provide "[...] goal-directed search through a list of logical rules." The goal is to find values that satisfy certain specified relationships. The term definition fits to Prolog, but SQL, XSLT and Excel Spreadsheets also fit into this category, for example.

Table 1: Declarative programming languages (according to Scott 2009, p.11)

Imperative languages (more concrete)

von Neumann languages	The most familiar, successful and traditional languages in computer science and history. "These languages are based on statements (assignments in particular) that influence subsequent computation [...]" (e.g. C, Ada, Fortran).
Scripting languages	These are considered to be a "subset" of the von Neumann languages. They are mostly built from a lot of different independent components/programs, to provide a fast and easy to use collection as a single programming language. The execution time is lower, as a consequence (e.g. PHP, JavaScript, Perl, Python).
Object-oriented languages	Object-oriented languages, although they are related to the "von Neumann" languages have a much more structured and distributed model than the von Neumann languages. They are described as "interactions among semi-independent objects with own internal state and subroutines that manage that state" (e.g. Smalltalk, C++, Java).

Table 2: Imperative programming languages (according to Scott 2009, p.11f)

1.2.2 Classification by hardware abstraction level and details

Another possibility to categorize programming languages is by their level of hardware abstraction. The meaning of the term "higher-level language" is scientifically used for all programming languages which "allow the specification of a problem solution in terms closer to those used by human beings" (King 1999, "High Level Languages"). This means all programming languages which are not built for a specific machine e.g. more abstract than Assembly language (and machine language). Machine independence in this context means that a programming language should not rely on the features of any particular set

of instructions to implement it efficiently (though real machine independence still is a problem with lots of languages) (Scott 2009, p. 111). COBOL, FORTRAN, Pascal, C, C++, Prolog and Java, are all considered to be “higher level languages” according to this use of the term.

Nowadays the terms low-, middle- and high level language are sometimes used differently, but the meaning of these is vaguely defined (no name 2010, “stackoverflow.com – is C a middle-level language”). Languages have attributes which indicate a “higher” or “lower” hardware abstraction level. According to these attributes it is possible to determine if a language is more abstract from the hardware than another. According to the model of classification by computational model, declarative languages tend to be higher-level than imperative languages, and scripting languages tend to be higher level than von Neumann languages - these are “higher-level” languages, but sometimes considered to be low level (= closer to the hardware than others) by a different meaning of the term (Spiewak 2008, “Defining High, Mid and Low-Level Languages” and Morgan 2010, “stackoverflow.com – low, mid, high level language, what’s the difference“).

Indicators

As it is not clearly possible to categorize languages by levels in a useful scientific way, we created this figure of indicators. These indicators can help to find out the level of abstraction of a programming language. As the title says, these are only indicators.

Lower Level	**Higher Level**
Direct memory management	Automatic memory management
Little (or no) hardware abstraction	Distant from the hardware
Register access	No register access
Good performance	Bad performance
Static typing	Dynamic typing
More compiled	More interpreted
Imperative	Declarative
High complexity	Lower complexity
No virtual machines	Virtual machines
No garbage collection	Garbage collection
Experts exchange	Huge community and support
More development time	Less development time
More details, less “limits”	Less details, more “limits”
Difficult error handling	Better error handling

Table 3: Classification by abstractness of level (Using Scott 2009, p. 11f. and Graham 2002, “Revenge of the Nerds” and Spiewak 2008, “Defining High, Mid and Low-Level Languages” and Morgan 2010, “stackoverflow.com – low, mid, high level language, what’s the difference“)

Examples

Lower	Machine language
	Assembly language (close to the hardware)
	C (an abstraction level higher than assembly)
level	C++ (possibility to abstract things away into classes)
	Java/C# (garbage collection)
	Python / Ruby (remove a lot of details, dynamically typed for example)
Higher	SQL (declarative)

Table 4: Examples for hardware abstraction level (Morgan 2010, "stackoverflow.com – low, mid, high level language, what's the difference")

Summary

A scientific classification by hardware abstraction levels is hardly possible, as the term is often used in a subjective manner. It is possible however to categorize programming languages by certain indicators which indicate a higher or lower hardware abstraction level.

1.2.3 Classification by programming language paradigms

Programming languages are often classified using so called paradigms. A programming paradigm is a fundamental style of computer programming, also called a high level model of what computation is about. This model is also widely used to categorize ("high level") programming languages. As the term "paradigm" is abstract, the categorization possibilities are not clearly defined by the term. A higher level programming language usually embodies features of more than one paradigm.

The main paradigms in computer science education

functional	Expression of computations as the evaluation of mathematical functions
imperative / structured / procedural	The language provides statements, which explicitly change the state of the memory of a computer system. Central features are variables, assignment statements and iteration form of repetition. Based on the von Neumann architecture.
object-oriented	Behavior is associated with data-structures called "objects" which belong to classes which are usually structured in a hierarchy.
logical	Computation is expressed exclusively in terms of mathematical logic.

Table 5: Main paradigms in computer science education (according to Zander no date, "Language paradigms" and University of Birmingham, no date, "Lecture 1: What are Programming Paradigms?")

Real world usage grouped by the major programming paradigms (according to Tiobe index)
According to the TIOBE index (Tiobe Software BV 2011, "TIOBE Programming Community Index Definition") the following statistics indicate the percentage of real world usage of programming languages grouped by the major programming paradigms (July 2011):

Object-Oriented Languages	55.9% +
Procedural Languages	38.1% -
Functional Languages	4.4% +
Logical Languages	1.6% +

+ indicates a rising tendency (previous year) // - indicates a downside trend (previous year)

Object-oriented, statically-typed languages have been the most popular ones for more than 5 years.

1.2.4 Advanced classification by programming paradigms

There are much more programming paradigms than the basic ones and sometimes the term paradigm is used in a much more specific way. For a fundamental categorization of programming languages the four main ones will be good enough, but there are much more, like "concurrent", "generic", "reflective" and so on.

Because of its recent importance a short excursion on the concurrency paradigm is shown below:

The "paradigm" of concurrency
Support and features for concurrent programming and threading can be affordable and important features of modern programming languages, when it comes to certain projects. Concurrency and parallelism have become ubiquitous in modern computer systems (Scott 2009, p.638).

Motivations for concurrency (Scott 2009, p. 575):

- *Capturing the logical structure of a problem*: Many programs (servers, graphical applications) have to handle largely more than one important task at the same time. Multithreading is the simplest and most logical way to handle this.
- *Exploit extra processors for extra speed*: High-end servers, supercomputers, and multiple processors have become ubiquitous. Programs should be written with concurrency in mind.

- *Cope with separate physical devices*: Applications that run across the Internet or a more local group of machines are inherently concurrent.

Not every language can be used for concurrent programming easily, as libraries have to be written and implemented.

1.2.5 Summary

We figured out three common ways to categorize programming languages in this chapter: Classification by computational model, classification by hardware abstraction level and classification by basic programming paradigms. Each of these ways will be used when evaluating programming languages for the Web in chapter five.

1.3 Technically comparable distinctive features of programming languages

When comparing programming languages it is not just important to categorize these by their character, model or paradigm, but to compare implicitly technical attributes and features. The advantages of comparing lots of attributes and features are that they are provable and measurable for every programming language as they are part of the language design. It is possible to evaluate how much a language is interpreted and how much it is compiled and it is possible to evaluate how a programming language handles typing, for example.

It would outrun the capabilities of this work to compare the details in programming language design for specific programming languages, but neither this is important. For comparing adequacy of programming languages for Web use, it is important to focus on elementary differences which matter. The aim of this chapter is to find, describe and explain connections between such distinctive features, so we can use and understand these.

1.3.1 The notion of binding time

A binding connects two things together - a name and the thing it names, for example. Binding time is the time at which a binding is created, the time at which any implementation decision is made, generally spoken (Scott 2009, p. 112f). To understand differences between programming languages it is important to know about binding time, as the times when bindings are made essentially define the character and performance of a programming language. Decisions may be bound at various times (Scott 2009, p.113):

Language design time	control flow constructs, set of fundamental (primitive) types, available constructors for creating complex types, lots of language semantics in most languages
Language implementation time	typically precision (number of bits) of the fundamental types, coupling of I/O to the operating systems' notion of files, organization/maximum sizes of stack/heap, handling of runtime exceptions
Program writing time	algorithms, data structures, names, …
Compile time	mapping of high-level constructs to machine code (including layout of statically defined data in memory)
Link time	Common practice of separate compilation: Various program-modules are joined together by a linker. It resolves inter-module references. Binding between two modules sometimes is not finalized until link time.
Load time	The operating system loads program into memory, choice of physical or virtual machine addresses, while virtual machine addresses are mapped to physical addresses during runtime
Run time	whole execution time, bindings of values to variables, bindings do a host of other language dependent decisions

Table 6: Times at which bindings may be created (Scott 2009, p. 113)

Why binding time matters

The notion of binding time leads to a lot of important facts and indicators, which help to understand the difference between programming languages (Scott 2009, p.113f):

- While early binding times lead to greater efficiency, later bindings lead to greater flexibility
- The terms static and dynamic are usually used to refer to things bound before runtime and at run time
- Compiler-based language implementations tend to be more efficient than interpreter-based language implementations, as they make earlier decisions
- Some languages require lots of important decisions to be postponed until run-time (like type checking): The advantages of this procedure are dynamic references and variables, but with huge recursive loops, huge performance differences may appear

These differences depending on binding time lead to meaningful and more explicit distinctive features we can compare: Compilation versus interpretation and static versus dynamic typing.

1.3.2 Compilation versus interpretation

Every programming language designer must decide how much his or her programming language is interpreted and how much it is compiled. While compiled programs basically provide faster performance, interpreted languages provide better controls (during runtime), more flexibility and a better error handling.

The model of compilation (Scott 2009, p. 17f.):
The code (high-level) of a program is written and afterwards it is compiled by the compiler to become a ready program (in machine language basically). The ready program takes an input and generates an output.

The model of interpretation (Scott 2009, p. 17f.):
An interpreter implements a virtual machine, which understands a high level programming language and takes the input as well as the program code more or less at one time, executing these directly.

The mixture of both, which is used generally (Scott 2009, p. 18):
The source program is translated into an intermediate program, where more or less parts of the program are ready compiled. The more complex this part of the process is, the more the language is called “compiled”. Afterwards the virtual machine (interpreter) is invoked to do its work and generate the output.

Interpreted programs do trivial transformation of the program code, to make interpretation more effective, while compiled programs are mostly analyzed. They are translated into machine code before the program can be executed (but usually still some subroutines are needed, to be used at runtime).

How compilation works (Scott 2009, p. 26)

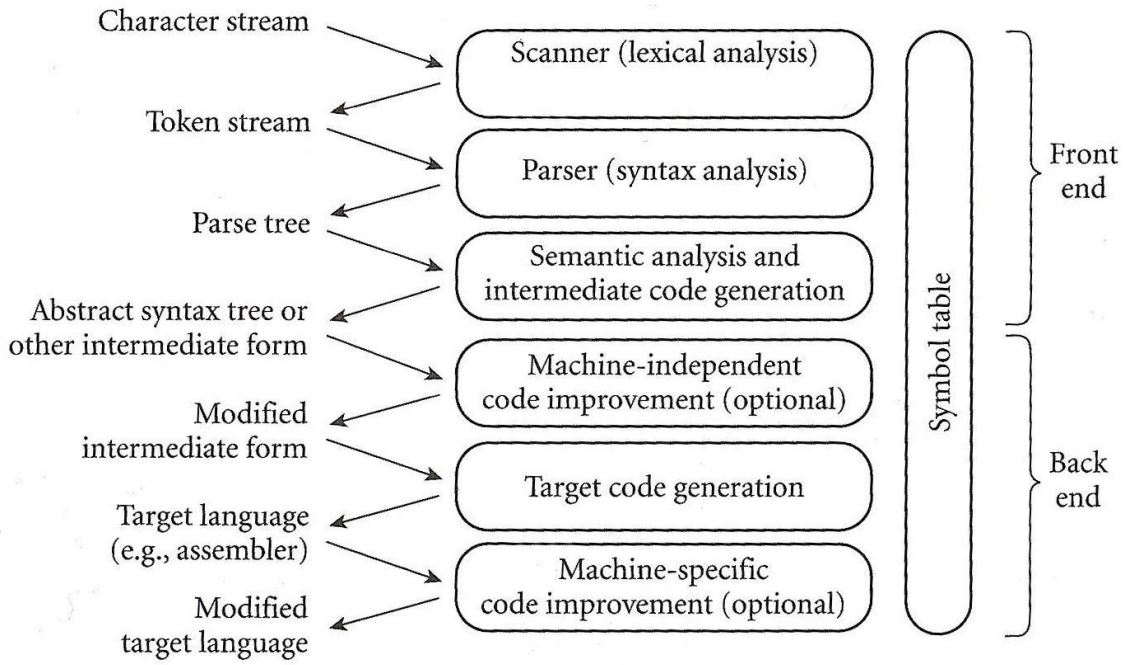

Figure 1: Phases of compilation (Scott 2009, p.26)

Explanation: Phases of compilation are listed on the right; on the left the intermediate form of the information, which is passed between the phases, is shown. The symbol table shows throughout compilation as a repository for information about identifiers.

While a purely compiled language only would pass through this procedure when it is deployed, an interpreted language has to go through this procedure on every new request, which leads to comparatively poorer performance.

Decision: Compiled versus interpreted language

According to the public library of the IBM Corporation (no date, “Compiled versus interpreted languages”) the choice should depend on the following:

Interpreted	Compiled
+ development time restricted + ease of future changes to a program - higher execution costs - higher overhead +- better for ad hoc requests, than	+ very efficient code which can be executed any number of times + overhead for the translation is only incurred on time, when the source is compiled

predefined requests ➔ use it for less-intensive parts of an application (interfaces, prototype or just less-intensive parts)	- more development time - changes are expensive ➔ use it for the intensive parts of an application (heave resource usage)

Table 7: Compiled versus interpreted languages

“One of the jobs of a designer is to weigh the strengths and weaknesses of each language and then decide which part of an application is best served by a particular language.” (IBM Corporation no date, “Compiled versus interpreted languages”)

Summary
Every language is more or less compiled and therefore more or less interpreted. While the advantages of compilation are performance and gentle consumption of resources, the advantages of interpretation are flexibility and ease of development.

1.3.3 Storage management

In any discussion of names and bindings there has to be a distinction between names and the objects which they refer to. Several key events can be identified:

- Creation of objects
- Creation of bindings
- References to variables, subroutines, types …
- Deactivation and reactivation of bindings (possibly temporarily unusable)
- Destruction of bindings
- Destruction of objects

Between creation and destruction, objects and bindings have certain lifetimes.

There are three principal storage allocation mechanisms, used to manage an objects’ space. Static allocation (absolute addresses throughout program execution), stack-based allocation (allocation in last-in, first-out order, in conjunction with subroutine calls and returns), and heap-based allocation (allocation / de-location at arbitrary times, which means a more general and expensive storage management algorithm).

Garbage Collection
Many languages specify objects to be deallocated implicitly when they cannot be reached any more from any program variable. To make this possible, the run-time library for a language has to provide a garbage-collection mechanism to find and remove unreachable

objects. Garbage collection is a typical feature of scripting languages, but also a lot of the newer imperative languages (Java, C#, Modula-3) (Scott 2009, p.120f).

The disadvantages of garbage collection are complexity in language implementation and the nontrivial amount of time consumption in certain programs. But arguments in favor of garbage collection are error safety (so called manual deallocation errors are the most common and costly bugs in real-world programs, as they are difficult to identify and fix) and less work for the programmer. With improved garbage collection algorithms and increasing complexity of applications in general, automatic garbage collection has become an essential feature for many languages.

Summary
Heap based allocation is more consuming, but more flexible than classic storage mechanisms and many modern languages use all of these. An essential issue in comparing programming languages is garbage collection (and its quality). While manual garbage collection can lead to a naturally better performance, automatic garbage collection can lead to error safety and ease of the programmers' work (in contrast to the implementers' work). Programming languages can be compared based on their memory consumption.

1.3.4 Type systems

A type system informally consists of a mechanism to define types and associate them with certain language constructs, a set of rules for type equivalence, type compatibility, and type interference (Scott 2009, p. 290). Subroutines (also called procedure, function, method, routine or subprogram) have (return) types in some languages, but not in others. Types are used to classify values and determine the valid operations for a given type (the type "int" for example can represent and integer and allow the operations "+" and "-"). Types can become complex especially in object-oriented programming (Tratt 2009, "Dynamically Typed Languages").

Type checking (Scott 2009, p. 291)

- A language is said to be *strongly typed* if any operation on any object that is not intended to support that operation is prohibited.
- A language is said to be *statically typed* if it is strongly typed and type checking is performed at compile time (or at least most type checking can be performed at compile time, in practice). C is an example for that.
- A language is said to be *dynamically typed* if type checking is delayed until run-time (and is often found in languages which delay other issues until runtime). A

language can be dynamically, though strongly typed (Lisp, Smalltalk, Python, Ruby).

Advantages / disadvantages

Static typing	Dynamic typing
■ Each error detected at compile-time prevents a run-time error ■ Better performance ■ Better debugging ■ Types are a form of documentation / comment ■ Code completion (functions and attributes of static typed languages can be automatically displayed)	■ Simplicity ■ extensive meta-programming abilities (reflection, eval, compile-time meta-programming, continuations) ■ refactoring ■ libraries (highly optimized libraries to minimize performance issues) ■ portability (*usually* less direct reliance to underlying platform) ■ high-level features (*usually* rich set of built-in data types and goes along with automatic memory management) ■ unanticipated reuse (less code) ■ interactivity (execution of commands on running system, interactive computations) ■ Compile-link-run cycle tends to be shorter ■ Run-time updates (arbitrary manipulation and emulation of data, type mismatch causes run-time error instead of low-level crash)

Table 8: Advantages of static typing vs. advantages of dynamic typing (Tratt 2009, "Dynamically Typed Languages")

Real-world usage (Tiobe Software BV 2011, "TIOBE Programming Community Index for July 2011")

According to the TIOBE Index (explanation in the section “statistics and trend analysis”) the popularity of the two basic type systems (static vs. dynamic) is shown in the following graphics:

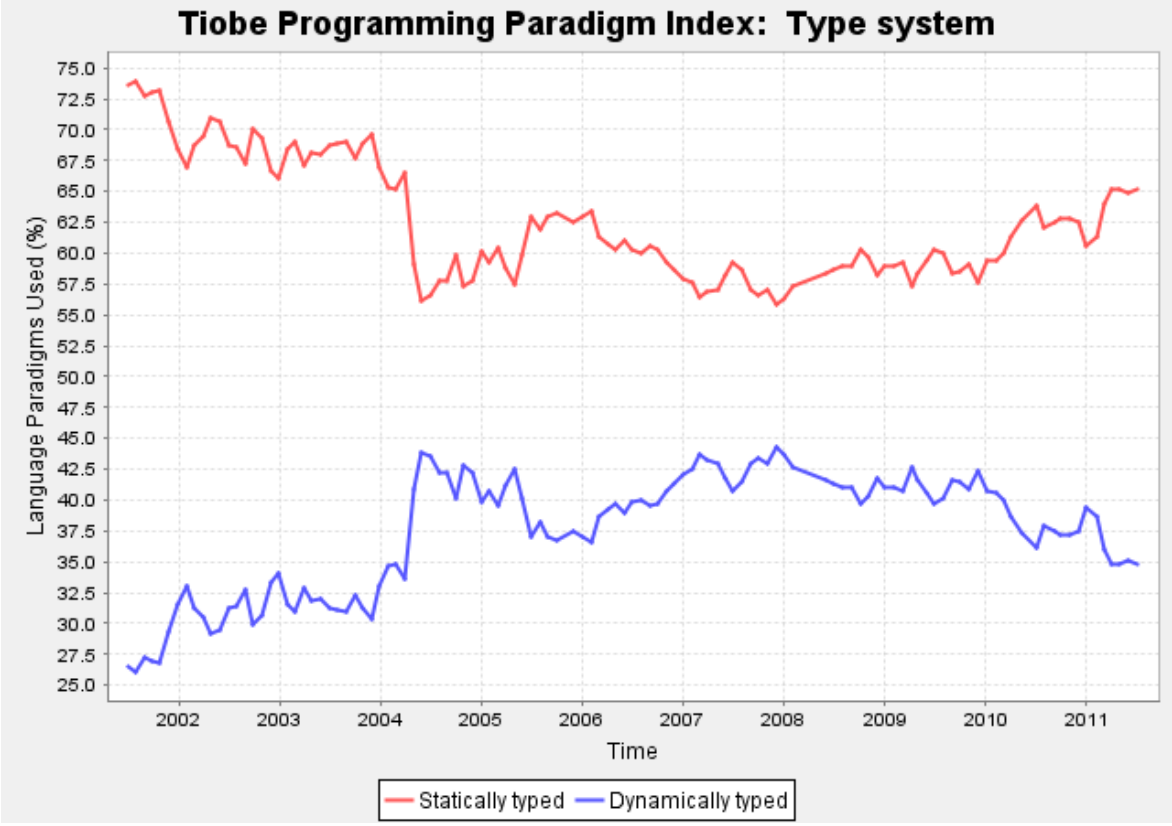

Figure 2: Type system evolvement (Tiobe Software BV 2011, “TIOBE Programming Community Index for July 2011”)

Statically Typed Languages	65.2% (+1.3% last year)
Dynamically Typed Languages	34.8% (-1.3% last year)

Summary

Type systems / type checking are an essential issue when it comes to comparing programming languages. There is a huge difference between statically and dynamically typed programming languages (although there are hybrids), while each concept has various advantages and disadvantages. The selection of the type system should depend on the intended use of the program code.

1.3.5 A closer look on the dissimilarity of scripting languages

Scripting languages are not intended to write new applications from scratch, but first of all to combine components, by using collections of useful components (Ousterhout 1998,

p.24). Ousterhout has predicted that programmers will rely increasingly on the usage of scripting languages for the top-level structure of their systems, which at least partly became true nowadays. The term “scripting language” is defined vaguely, as it is used explicitly for “glue languages” to coordinate multiple programs sometimes. In a diversified way also “macros” of Microsoft Office and XSLT could be seen as scripting languages, for example (Scott 2009, p. 652). Scripting languages are considered to be responsible for much of the most rapid change in programming languages today (Scott 2009, p. 655).

History

A lot of general purpose scripting languages has been evolved during history (examples are Perl, Tcl, Python, Ruby, VBScript (Windows) and AppleScript (Mac)) (Scott 2009, p.650). With the growth of the World Wide Web, the usage of scripting languages (especially Perl) for “server-side” Web scripting grew rapidly (this means a Web server executes a program to generate the content of a page). Soon PHP was developed (originally written in Perl) to be the most popular platform for server-side Web scripting. Well-known competitors are JSP, Ruby on Rails, and VBScript (Microsoft Platforms).

For scripting on client computers, the language JavaScript has been developed by Netscape. It has been standardized by the ECMA insitution in 1999.

Characteristics of scripting languages (Scott 2009, p. 652ff)

<table>
<tr><td>Both batch and interactive use</td><td>

Rare: Compiler reads entire source program before any output (Perl)
Often: Compilation / interpretation of output line-by-line, some accept keyboard commands during execution

</td></tr>
<tr><td>Economy of expression</td><td>

Trying to minimize the amount of code characters
Avoidance of extensive declaration:

<table>
<tr><th>Java</th><th>Perl / Python / Ruby</th></tr>
<tr><td><pre>class Hello {
public static void
main(String[] args) {
System.out.println(“Hello
world!”);
}
}</pre></td><td><pre>print “Hello world!\n”;</pre></td></tr>
</table>
</td></tr>
</table>

Lack of declaration / simple scoping rules	■ simple rules to govern scope names ■ optional declaration to change default scopes
Flexible dynamic typing	■ most scripting languages are dynamically typed, in some variables are checked immediately before use (e.g. PHP, Python, Ruby, Scheme) ■ in others, variables are interpreted differently in different contexts (e.g. Rexx, Perl, Tcl)
Easy access to system facilities	■ Fundamental requests, directly supported ■ huge amount of built-in commands to access operating system functions ■ usually easier to use than corresponding libraries in classic languages like C
Sophisticated pattern-matching and string manipulation	■ extraordinarily rich facilities for pattern-matching / search / string manipulation, typically based on extended regular expressions
High-level data types	■ high-level data types get built into the syntax and semantics of the language itself ■ storage is invariably garbage collected

Table 9: Characteristics of scripting languages (Scott 2009, p. 652ff)

Differences in efficiency and expressiveness

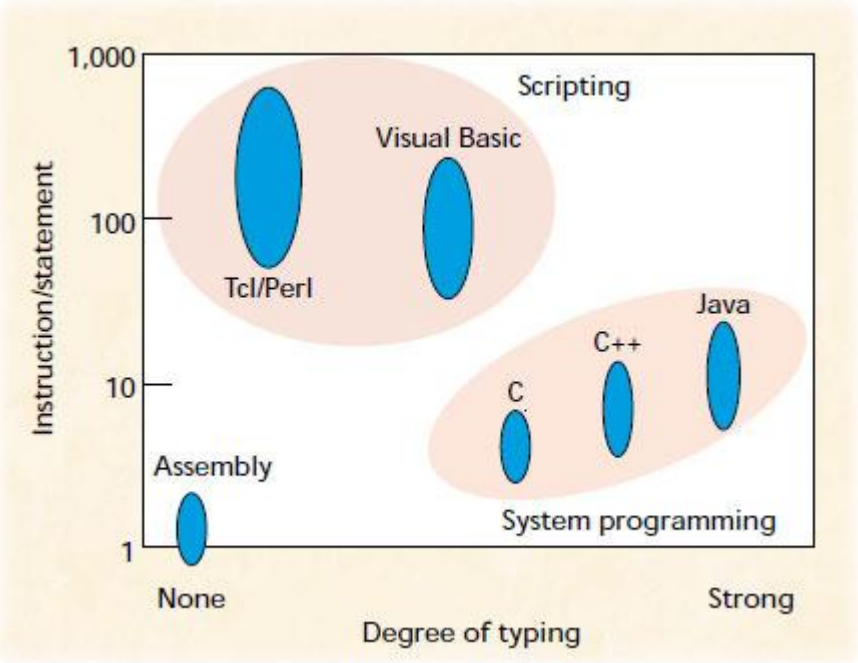

Figure 3: Programming language comparison based on their kind and level (Ousterhout 1998, p. 25)

According to Ousterhout (1998, p. 25) one instruction in a (weakly typed) scripting language like Perl generates up to 1000 processing instructions, while strongly typed system programming languages generally create a maximum of 50 instructions per statement. Code can be developed 5 to 10 times faster in a scripting language, but will run 10 to 20 times faster in a traditional language (Ousterhout 1998, p.27).

Intended use of scripting languages

According to Scott (2009, p. 655ff), we can basically differentiate between three kinds of scripting languages in relation to their intended use:

Well defined problem domains (most)	Examples: Shell (command) languages, text processing / report generation, mathematics / statistics and extension languages
General scripting to support "programming in the large"	modules, separate compilation, reflection, program development environments, … (e.g. Perl, Python, Ruby)
General scripting language	widely used for scripting (e.g. Scheme, Visual Basic)

Table 10: The differences between scripting languages

Summary
Scripting languages have a lot of clearly distinctive features in comparison to other programming languages and are very common in the World Wide Web. In contrast to the implementation process (building a compiler / interpreter), development with scripting languages is easy, fast and very comfortable - though not fail-safe. They are expressive, but suffer from low-performance. Some of them are featured with extraordinary good and pioneer features and libraries.

1.3.6 Comparing performance (execution time)

The performance or speed of execution of programming languages is an important criterion when it comes to comparing programming languages. We already figured out a lot of ways which improve or downgrade performance so far.

Measuring the actual performance of programming languages
The performance of programming languages can be compared by benchmarks, which cannot be done using trivial techniques. The “Computer Language Benchmarks Game”-project (Fulgham 2011, “Help”) is comparing benchmarks for different programming languages in various ways. The details about how programming languages can be measured using benchmarks can also be found on Fulgham (2011, “Help”). We decided to include the benchmarks of the “Computer Language Benchmarks Game” in the programming language comparison to get an idea of the language performances, as the results vary widely – the results cannot be proven, but they can be used as an indicator of how fast a programming language is.

1.3.7 Comparing expressive power (amount of code)

The literature about programming languages contains a lot of informal claims on the relative expressiveness of programming languages. In the work “On the relative expressiveness of programming languages” by Felleisen (2010), a formal notion of expressiveness is developed, some ideas about expressiveness are captured and shown and some widely held beliefs are analyzed. The study focuses on functional programming languages.

Results of the study (Felleisen 2010, p. 40f)
The following results can be deducted:

- The key to programming language comparisons is a restriction on the set of admissible translations between programming languages

- The translations between programming languages should preserve as much of a programs structure as possible
- Formal expressiveness results are close to the intuitive ideas of literature
- Increase of expressive power can destroy semantic properties of the core language
- Increasing expressiveness facilitates the programming process by making programs more concise and abstract (Conciseness Conjecture)

➔ An increase in expressive power is related to a decrease of the set of "natural" (mathematically appealing) operational equivalences

Comparing the expressive power of programming languages by measuring

There is no trivial way to measure the expressive power of programming languages scientifically. The "Computer Language Benchmarks Game"-project (Fulgham 2011, "Help") however provides data about the estimated expressive power of programming languages. This data can be used for comparing programming languages, though it should not be used for drawing any scientific conclusions, as the results are not clear enough.

1.4 Other distinctive features of programming languages

When comparing programming languages, all important aspects should be considered. There are a lot of important issues that should be considered that cannot be compared in a technical measurable way. How popular is a programming language? What kind of are its community and support? Is the language documented well? How much time of learning does it take to master the language? Is its design and operating mode mainstream or exotic? Is it created to be used for specific purposes? How about expressiveness and performance? What about built-in support or library support of certain features?

1.4.1 Popularity of programming languages and correlating factors (tools, support / updates, community, sources / libraries / frameworks, documentation)

The popularity of programming languages matters (Welton 2005, "The Economics of Programming Languages"). The so called "network effects" mean, that the more individuals use something, the more valuable it is to everyone – this sentence can be applied on programming languages partly: the number of books, open source libraries, examples, discussions/support groups, and jobs, is strongly dependent of the popularity of a programming language. Also community issues regarding internationalization and frequent updates/further developments are partly depending on the popularity factor. That does not mean that good work cannot be done with uncommon languages sometimes, too.

Languages have a tendency to stay popular

There are two main reasons why established programming languages do not disappear and are not replaced by other or better ones fast:

- Rewriting large pieces of software from scratch is expensive, time consuming and complex
- Learning a new programming language is usually hard and time consuming for individuals (especially not expert programmers), so they defend their choice for a certain language, after it has been made

How new programming languages can establish

New languages can be better and more innovative than common ones, especially if they have useful characteristics (easier to write, more efficient, higher quality, or more productive for example). Compatibility to common languages and “niche-targeting” can help establishing these (Welton 2005, “The Economics of Programming Languages”), but also other factors can influence the popularity of a language.

The role of programming environments and tools

Programmers need tools to work more efficiently, as compilers and interpreters alone do not guarantee a fast and easy development process. Editors which show dynamic information about the current program, pretty-printers for proper formatting, style checkers that remind of syntactic or semantic conventions, configuration management tools, and profilers are just some examples of how the work of developers can and should be assisted (Scott 2009, p.24).

Most modern programming environments are huge tool collections, to automate and simplify a lot of the steps in testing and creating code. Debugging, editing the source code, rebuilding a program and remote ftp (file transfer protocol) for example can possibly be done within the same window, for example.

How important companies and technologies can influence the popularity of programming languages

The Microsoft Corporation is a company which utilizes a lot of internally developed technology for programming tasks, which is usually not standardized and almost exclusively built for usage with Microsoft software (Scott 2009, p.651). The programming language Visual Basic, and its descendant for scripting tasks, VBScript, which are widely used for programming purposes on the Internet are examples for that.

According to Noack (2005, “Ausarbeitung: Objective C”) the history of the programming language “Objective C” is strongly connected to the companies NeXTStep and Apple and

the operating systems they published. Due to the success of the Apple Inc. company and its products (McLaughlin 2008, “Apple's Jobs Gushes Over App Store Success”), whose technology is largely based on Objective C and the Cocoa Frameworks (Apple Inc. 2011, “Cocoa”), Objective C gained a large boost in popularity according to the TIOBE index (explanation in the section “statistics and trend analysis”).

These are just two examples, but lots of popular programming languages are related to a strong commercial company. The companies bring develop these programming languages or at least bring them forward. The advantages of this process are usually great support and tools for these languages. The disadvantages in contrast are closed source and the liability to pay costs on lots of tools or applications. An example for this is the Microsoft Visual Studio (Microsoft Corporation 2011, “Microsoft Visual Studio”). For open source and free programming languages support and tools are important, too. A good and useful programming environment (e.g. Eclipse, Netbeans), for example.

Factors for a programming language to become or to stay popular

According to Paul Graham (2001, “Being Popular”), a programming language notably becomes popular, if expert hackers like it, though also legacy software (COBOL) and Hype (Ada, Java) have to be considered. He names a bunch of criteria, which can be responsible for a language to become popular and which he thinks expert hackers like:

- external factors (scripting language of a popular system, books, online documentation)
- brevity (no useless characters, an argument for weakly typed languages)
- hackability (clean and dirty, with no limits what the programmer can do)
- easy creation of “throwaway programs”
- libraries (good and many)
- syntax (compact and useful)
- efficiency (fast results)
- time (languages must be around for some time before they even become considered by many)

Summary

Many factors are to be considered when it comes to the popularity of programming languages, and by far not always the “best” programming languages become established. The factors why a programming language is or is not popular actually should be considered when programming languages are compared. Important questions are:

- Is there a huge company which empowers the development of the programming language or which is using it widely?

- Are there adequate development tools for developing with this programming language and are they suitable for the intended purpose?
- Is the programming language mostly popular because of legacy issues?
- How about documentation, material, and programming comfort?

In the “statistics chapter” we will learn a lot about the actual popularity of programming languages and about the ones which are most common nowadays. But the factors for popularity should always be considered when choosing a programming language, as popularity and things relating to popularity certainly can lead to bad decisions, when choosing the best language for a programming task.

1.4.2 Syntax, semantics and complexity

In the previous section about popularity, we already determined the roles of complexity, syntax and semantics as factors with influence. In this section we will have a closer look on how these things are different for many programming languages. Syntax and semantics of a programming language must be clearly defined. The differentiation between the two is useful, as there are lots of programming languages with very different syntax, but similar semantics (Scott 2009, p. 42f).

Syntax

The syntax of a programming language defines it structural rules, and how the compiler identifies the structure of a given input program. The first one is important for the programmer and relies on regular expressions and context-free grammars, while the second one is important for compilers to analyze the program and relies on scanners and parsers. The syntax does not say anything about the meaning of a program. Syntax with the same meaning can be written in different forms, as the following example of a for-loop for three different programming languages shows:

Fortran 90 / 95 / 2003	C++	Python

do i=0,9
  statements...
  if (condition1) then
  cycle
  else if (condition2) then
  exit
  endif
  statements...
enddo
	for (i=0; i<10; i++) {
	  statements...
	  if (condition1) {
	  continue;
	  } else if (condition2) {
	  break;
	  }
	  statements...
	}
		for i in range(1,10):
		  statements...
		  if condition1:
		  continue
		  elif condition2:
		  break
		  endif
		  statements...

Table 11: For-loop in Fortran 90, C++ and Python

The creators of a language specify its syntax, and decide how to have the best of it. “Esoteric programming languages” show how different syntax can be (listverse.com 2011, “Top 10 Truly Bizarre Programming Languages”): The programming language “Whitespace”, for example, uses only whitespace characters as syntax, while others are ignored. The programming language “brainfuck” consists only of eight commands: > < + – . , [ ]. Other possible characters are ignored. The following program prints “Hello World” in “brainfuck”:

```
>+++++++++[<++++++++>-]<.>+++++++[<++++>-]<+.+++++++..+++.>>>++++++++[<++++>-]
<.>>>++++++++++[<++++++++++>-]<---.<<<<.+++.------.--------.>>+.
```

Semantics

While syntax concerns the form of a valid program, semantics concern its meanings. That means, a program can be syntactically correct (can be described by context-free grammars), but have semantic errors. Programming languages vary dramatically in their choice of semantic rules (Scott 2009, p. 176ff). There are a lot of approaches to formal semantics belonging to the three major classes of semantics (Mosses 2001, “Programming Language Semantics”):

- Operational semantics: The execution of the language is described directly (correspond loosely to interpretation)
- Denotational semantics**:** Translating each phrase in the language into denotation (correspond loosely to compilation)
- Axiomatic semantics: Giving a meaning to phrases by describing the logical axioms

Some languages allow intermixing of operands of many types in expressions, while others do not. Some only do minimal checks (C), while some try to check as many rules as possible (Java).

Complexity of programming languages

The complexity of a programming language depends on the design of its syntax and semantics. How hard a programming language is to learn for an individual depends on various factors (Graham 2001, “Being Popular”), so it cannot be measured easily.

Summary

The syntax of a programming language is responsible for its structural rules and how the compiler identifies its structure. There are a lot of different forms how syntax can appear, as it can be chosen arbitrary. There is a huge bunch of possibilities for implementing and enforcing programming language semantics (meaning of a program). Comparing the details for each language is difficult, but it is possible to determine if semantic rules are

rather strongly checked or not. The complexity of programming languages cannot be measured easily.

1.4.3 Intended use

Many programming languages are not built for general purpose use. As we learned in the chapter about scripting languages, some are even built for very specific problem domains, like text processing or mathematics (Scott 2009, p. 655ff). Lots of well-known programming languages are almost exclusively used for specific purposes, like SQL for database communications and XSLT for transformations of XML documents. As long as a programming language is “Turing complete”, it can theoretically be used for every programming purpose. This is true for XSLT for example, as a script by the Unidex company proves (Unidex, Inc. 2008, “Universal Turing Machine in XSLT”). But some programming languages are better for appointed tasks than others.

How good a programming language qualifies for a concrete task must be determined in each single case and cannot be shown across-the-broad. However, most language creators and distributors write a lot of things according to the intended use of “their” programming language on the project Websites. What is written on the Websites does not say much about the qualification and effectiveness of a programming language in reality, but much about its “intended use”, according to the creators/distributors.

1.5 Summary

In this chapter we figured out the main differences between programming languages. We showed what a programming language is and some important things about its architecture and it is intended to work.

We will use the results of this chapter to compare and analyze the most important Web programming languages later in this work. Before we can do this, we have to figure out the affordances for Web programming languages and how they can be used on the Web. We also have to figure out what the most important languages are, before we can compare these.

2. Analysis of the Internet: History, technical principals, the World Wide Web and domain-specific languages

Computer systems connected on the Internet are based on a huge amount of different technologies (different displays, different browsers, PCs and Workstations or handheld devices …). There are different hardware components, different operating systems and

more. The software and interfaces of computer systems connected to the Internet are programmed in completely different ways and programming languages and also the way they are intended to work. There are only standardized interfaces how these systems can connect to the Internet, the Web and each other, and so there must also be standardized ways about what content these systems can read and understand.

This not only affects content formatting (e.g. HTML or XML) or design definitions (e.g. CSS or XSLT), but also all client side scripting and programming technologies. These are required for really dynamic Web contents. There is a possibility to install programs (browser Plugins) for example, to display dynamic content. But this requires a client to run certain software which is supported by a version of this Plug-in (see Adobe Flash for example), which is not standardized and can be boycotted easily by notable companies, for example (Jobs 2010, “Thoughts on Flash”).

In spite of that, for the client-server-based communication on the Internet it does not matter how servers work internally. They get an input via a standardized protocol on a certain port and generate an output as answer to the client.

There are Web projects which only provide static contents. The server system only returns ready content and no extra programming is required. But there is a tendency on the Internet towards dynamic contents. With trends like RIAs, Cloud Computing and Web services a lot of the important work must be done inside the server system (Scott Alan Miller February 18th 2010, “Trends in Thin Client Computing”). The percentage of server-side works and logics has been growing and is still growing rapidly, which indicates a trend toward thin clients. This fact vindicates our procedure here. The explanation on thin and fat clients can be found on Vangie Beal 2006 (“The Differences Between Thick & Thin Client Hardware”).

Although it theoretically does not matter how a server works internally, there are a lot of aspects and features which make a server, its’ software and the used programming languages on a server more desirable for usage in the Internet than others. We will figure out these to prove and to demonstrate, why the current servers and programming languages are used on the Internet currently and what things could be done better.

In the 1990s the Internet grew rapidly with the number of Websites rising. Any computer connected to the Internet (which is just the name for global most famous and biggest network of computers) can basically act as a server, to provide its information to another computer (Bates 2006, p.11f).

2.1 Standardization and protocols

It is obviously desirable that Web servers understand the requests of clients and client systems should understand the answer of Web servers, so they can process and present it. Because the more different ways of communication exist, the harder it gets to handle compatibilities and the more work programmers and developers have to do to spread their contents. So there are always efforts to standardize the ways of communication and content preparation on the Internet to avoid incompatibilities.

One of the most important institutions in this context is the W3C, as it is working out standards for “long-term growth of the Web” (w3.org no date, “W3C Mission”). It is working on the further development of HTML, CSS and many more (w3.org no date, “All standards and drafts”). There are other important organizations for standardization on the Internet, too (Maher 1998, “An Analysis of Internet Standardization”):

- traditional (long-term involvement in standardization): ANSI, IEEE, IEC, ISO, JTC, ITU
- modern (arouse specifically around the Internet): IETF, W3C, IAB, FNC, IANA, IESG, IRTF, CNRI

When it comes to research about standards and widely used Internet technologies, these organizations provide a good starting point.

Protocols

The Internet is a worldwide network of connected computer systems (and the most well-known on our planet). It has been growing for a long time and there are thousands of ways to transport data through this network. Standard-network-protocols like HTTP (for the actual delivery of Websites), SMTP (for sending emails), FTP (for transferring files) and many more have evolved.

The TCP/IP protocol family contains more than 500 network protocols, which are considered to be the core of communication on the Internet (searchwindevelopment.techtarget.com October 2000, ”Definition: Internet”). But also other protocols like UDP or SCTP are in use in this huge network, as they may have advantages like connection oriented communication. A list of all registered ports for network communication can be found on meineipadresse.de (Juli 7th 2011, “TCP/IP Ports“).

HTTP

The Hypertext Transfer Protocol (usually version 1.1) is ubiquitous nowadays (Gray 2001, p. 35ff). It is the most important Protocol to power the World Wide Web as it specifies how

servers and clients on the Internet usually communicate (see the specification of the W3C for further details: http://www.w3.org/Protocols/).

When a system sends a request to a port of another system, this system can send an answer back to the first system. On the Internet the first system is called a client, and the second system is called a server. On the Internet a client (often using a graphical browser) sends a request header to a server via HTTP and the server sends back a document, like an HTML-page via HTTP which can be processed and displayed in the clients' browser.

Form of a HTTP Request	Form of a HTTP Response
Request = Request-Line *((general-header | request-header| entity-header) CRLF) CRLF [message-body]	Response = Status-Line *((general-header | response-header | entity-header) CRLF) CRLF [message-body]

Table 12: HTTP-Request/response form (Gray 2001, p.35ff)

When it comes to the topic of server programming on the Web, we have to keep in mind that the form of the HTTP protocol defines how information is usually interchanged between clients and servers.

2.2 Web browsers

A Web browser is a program that accesses and displays files and other data available on the Internet (Houghton Mifflin Harcourt Publishing Company 2010, "browser science definition"). Users who surf on the Internet usually use one of a handful of common and popular Web browsers:

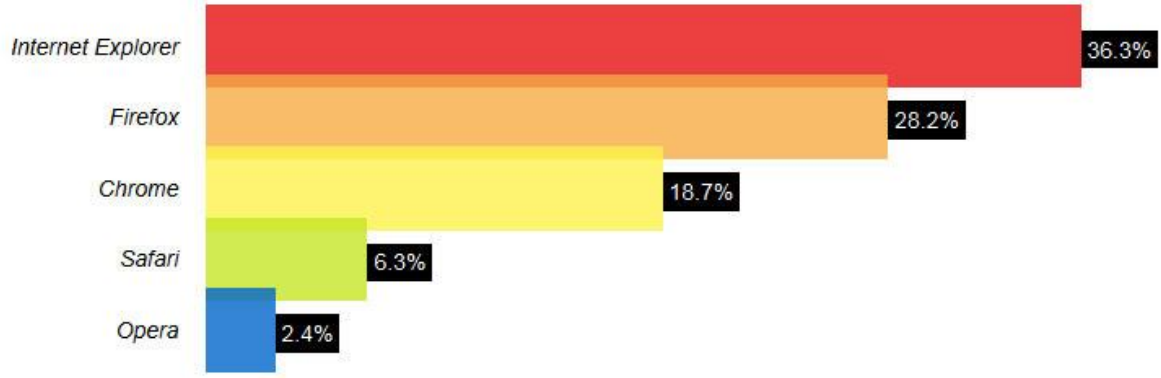

Figure 4: Web browser stats in June 2011 (Awio Web Services LLC 2011, "Global Stats – June 2011")

According to the graph, 92.2% of all Internet surfers are using one of the five main important browsers in the browser business currently. The graph was generated on June

30th 2011, while the data is based on the last 15000 page views of each of 48712 random Websites which take part in the statistics gathering program.

According to these results, at least 9 out of 10 Internet users are able to use dynamic Websites and Web applications which have been programmatically optimized for these five major Web browsers.

2.3 Domain-specific languages for the World Wide Web

This section is dedicated to domain specific languages for the World Wide Web.

"A *domain-specific language* (DSL) is a small, usually declarative, language that offers expressive power focused on a particular problem domain. In many cases, DSL programs are translated to calls to a common subroutine library and the DSL can be viewed as a means to hide the details of that library" (Deursen 1998, "Domain-Specific Languages: An Annotated Bibliography").

According to the categorization by computational model of programming languages, all of the domain-specific languages to be important for this work fit into the group of declarative programming languages, which are very high level.

These languages are only partly considered to be programming languages, as most of these are not "Turing complete". They can hardly, if ever be used for general purpose programming. Because of this, there is no sense in comparing these in the "programming language comparison". Anyhow the listing and explanation of these languages is essential for understanding programming language affordances on the Internet. Some of these languages can be used for "client-side programming", which is at least true for XSLT, a Turing complete. But because of its domain-specific purpose we will list it here as well.

2.3.1 Content formatting

The separation of presentation and content is one of the most important capabilities of Markup Languages, especially HTML. Websites can be viewed from any kind of screens, like computer, mobile device, and television and more, also printing HTML pages is sometimes required. HTML cannot provide the facilities to cope with this diversity, but with stylesheets for the presentation part, these can be provided.

HTML and XML are considered to be applications of SGML, the Standardized General Markup Language, which roots go back to the late 1960s, where a standard for text processing languages was required (Bates 2006 p.2ff and Hofmann, Raithelhuber 1998, "SGML/XML").

(X)HTML (Hypertext Markup Language)

"... HTML documents should work well across different browsers and platforms. Achieving interoperability lowers costs to content providers since they must develop only one version of a document. If the effort is not made, there is a much greater risk that the We will devolve into a proprietary world of incompatible formats, which will ultimately reduce the Web's commercial potential for all participants." (Bates 2006, p.5)

Hypertext Markup Language is a way to format documents, to be viewed on a computer screen. It is *the global publishing format of the Web*. Web browsers render HTML, but many programs can read or display it. It is not only for formatting text and defining its semantic structure, but also for including images, sound, videos and more. It is a continuously developed standard which is evolving since its creation by Tim Berners-Lee at CERN (European center for particle physics) in the earliest years of the Web (Bates 2006, p. 2ff).

The success of HTML started due the development of the Mosaic browser, which was funded by the US government and distributed free of charge. A lot of its functionality was picked up by the Netscape Navigator browser, which is still a well-known one.

What HTML actually is

HTML is not a programming language, as it is not possible to write any programs with it – and neither is it a data description language – because it is not possible to structure data in a distinct way, as this is possible with XML. There have been a lot of efforts to push XHTML, a variant of HTML closer to XML, which would add this capability to HTML. And in fact, XHTML makes it possible to create the cleaner and more fail-save ways Web pages (Bates 2006, p.77ff), but is not as fault-tolerant than HTML.

HTML is a de-facto standard for content formatting, which is understood by Web browsers. XML is also a widely supported alternative.

XML (Extensible Markup Language)

XML is a Markup Language for formatting content in documents, but the rules are stricter and there are a lot of more possibilities. While HTML limits the number of allowed tags and attributes, XML provides the possibility to define own ones and structure documents in a completely customized way. Because of its strict rules data can be stored in a distinct way, which makes it possible, to read, write and process structured data using XML in programming languages and applications (into arrays for example). There are a lot of existing parsers (openjs.com no date, "XML Parser for JavaScript – xml2array()") for this task, partly built into (scripting) languages (php.net no 2011, "XML Parser"). This is not

possible with HTML. Because of this, XML was sometimes considered to be “the future of the Web” (Bates 2006, p.6).

2.3.2 Presentation of Websites

A “stylesheet” interprets the content of a document to generate a view to be displayed. The content may not be changed by a stylesheet. Stylesheet languages are computer languages to express the presentation of structured documents, while different stylesheets can be used to customize the presentation of Websites for certain systems (Lie 2005, “Cascasding Style Sheets”).

CSS

“CSS has some unique and innovative features, including cascading, pseudo-elements and pseudo-classes, forward-compatible parsing, and media types. By now, CSS has established itself as one of the fundamental specifications on the Web and most Web sites are using it. As such, the efforts to create a stylesheet language for the Web have been successful. Also, CSS has partially fulfilled its ambition of maintaining HTML as a structured markup language and ensuring that documents can be styled by users. Alas, due to limited support for CSS in the dominant browser on the Web, CSS cannot yet be used in full “(Lie 2005, “Cascading Style Sheets”).

Some key design definitions were made when CSS was released:

- CSS was a new language intended for Web use (designers took some ideas of a standard called DSSSL, but did not use it as base)
- a non-Turing-complete declarative language (for reasons of ease and security)
- support of progressive rendering: this is a main difference to XSL-languages (e.g. XSLT) – parts of a document can be displayed even, if it is not fully downloaded, while reordering of elements, for example, is not possible with CSS (as it is possible with transformation languages)
- CSS should work with structured documents
- CSS should work with any Markup Language (so it is possible to use XML with CSS, for example, which is good for the popularity of both)

Note: CSS is getting more powerful with newer releases, called levels (CSS level 2 and CSS level 3) (Etemad 2011, “Cascading Style Sheets (CSS) Snapshot 2010”).

XSL and XSLT

„Transformation-based stylesheet languages do not adorn a tree; instead they transform the logical structure into a presentational structure. DSSSL and XSL are stylesheet languages that fall into this category. Often, these languages are referred to as

transformation languages rather than stylesheet languages. In the case of XSL, the transformation language has been given its own name, XSLT (where T stands for transformation) “(Lie 2005, “Cascading Style Sheets”).

As XSLT is a Turing complete programming language, which can be used for generating dynamic Web content on the client-side, you can find a section about XSLT in the section about the implementation of programming languages on the Web.

2.3.3 Other domain-specific languages in Web and Internet programming

There are a lot of more domain specific languages around on the Internet (e.g. cucumber, rails validations, ant …) (Fowler no date, “Domain-specific languages”). There are two important examples to be considered here in short, as they are important languages which will not show up in the programming language comparison.

SQL

SQL (Structured Query Language) is a declarative language for managing data in relational database management systems and is the most widely used language in the context of database programming on the Internet. It is usually used in combination with other programming languages, which are used for Web programming (Wimmer 2002, “SQL Tutorial”).

SQL is traditionally not considered to be a programming language as the standard (i.e. SQL92) is not Turing complete (Bell 2009, “Is SQL or even TSQL Turing Complete?”). The TIOBE index is also not listing SQL. However some newer implementations and variations of SQL are real programming languages, like PL/SQL (Oracle) or T-SQL (SQL Server). According to Whitacre (2009, “SQL is Turing-complete”) the SQL:2008 standard (ISO/ANSI), which is not available for free (Paulley 2008, “SQL:2008 now an approved ISO International Standard”) is Turing complete now. Although pure SQL offers various advanced programming possibilities nowadays, it is still a domain-specific language and should rather not be compared in our comparison for general programming tasks on the Web.

Regular expressions

“Regular expressions are the key to powerful, flexible, and efficient text processing. Regular expressions themselves, with a general pattern notation almost like a mini programming language, allow you to describe and parse text. With additional support provided by the particular tool being used, regular expressions can add, remove, isolate, and generally fold, spindle, and mutilate all kinds of text and data”(Friedl 1997, p.1).

Regular expressions are an essential technique in programming language science in general, and also considered to be a declarative, domain-specific language (Fowler no date, “Domain-specific languages”). They are available in most scripting languages (e.g. PHP, JavaScript …). They are listed here as a second example of domain specific languages on the Web.

2.4 Summary

As the variety of computer systems is huge, there is a need for standardization on the Internet, especially on the World Wide Web. There is a lot of organization for working out standards on the Internet. In the context of “programming languages for the Web”, a lot of important aspects of the Internet have to be considered:

- There are countless different protocols for the Internet, but for the World Wide Web the most important one is HTTP (and its structure), which is used to deliver pages and handle most of the communication.
- There are five major Web browsers for Internet usage with desktop computers currently; these are namely Mozilla Firefox, Microsoft Internet Explorer, Google Chrome, Opera, and Apple Safari and they probably make up more than 90% of usage statistics
- There are some “domain specific languages” which were mostly developed especially for Web use. These are not really programming languages, but standards. These domain specific languages are intrinsically tied to the World Wide Web at present times, including (X)HTML, CSS, XML and possibly XSLT and SQL

3. Web and Internet programming and implementation

There are two possibilities for individuals to communicate using the Internet: One possibility is to use a Web browser, a program which is built for browsing the Internet, especially the World Wide Web and the other possibility is using a standalone desktop program, which has a built-in way to communicate with Internet servers or users (peer-to-peer). Note: Since the rapid growth of the mobile Internet, a hot discussion arouse about what is better for mobile devices depending on the use case – native applications or Web applications.

As a logical consequence of this, there are three ways in what context programming languages can be used “for the Internet”, and each of these ways has certain limits:

- Programming for dynamic content generation in a Web browser.
  Limit: The browser must understand the programming language or technology which is served (and security restrictions must be accounted).
- Programming of dynamic Web applications using the server.
  Limits: The server has only the information about the client, which the client sends or can be made to send. The answer of the server must be in a way that the client understands.
- Programming stand-alone applications which communicate with a server or other clients.
  Limit: The program must be compiled for and installed on the clients' operating system.

When the aim is to provide a Web application, the first two technologies can work together in a lot of useful ways (e.g. AJAX). While the first two key points are mostly relevant for the World Wide Web, the third is rather concerning the whole possibilities of the Internet.

The aim of this chapter is

- to analyse the affordances for programming languages on the Web
- to summarize all common and uncommon possibilities for implementation of programming languages on the Internet and on the World Wide Web

To reach this aim, first of all we have to make some clarifications about some common terms for the Web which we will use based on the data we have collected here later on.

Web Programming

A lot of content on the World Wide Web is still static (as for search engines all content is, for example), but most Websites and Web applications today interact with their users. Web pages contain dynamic content and a lot of ways for clients to interact with a server. To make this possible, there was a need for server and client side scripting- and programming technologies – these are what "Web programmers" usually work with today (Bates 2006, p. 1f).

When adding interactivity to the Web programmatically, the possibilities are multifaceted. While simple playback of audio and video is not that exciting from a programming languages point of view, content that is generated on the fly by a program respectively a script associated with an Internet URI is what to focus on. The first thing interesting in this context is, if a program is executed on the client system or the server system (Scott 2009, p. 680). Basically, in terms of Turing completeness, every programming language could be used for programming tasks on the Web - but not every programming language suits

this purpose well enough, as this is a question of adequacy, compatibility and (security) restrictions.

As we already figured out in the previous chapter, the Web is different: Programs must generate output which Web browsers can understand and display to the user. There is a need for technologies which simplify the process of generating rich interactive Web sites and Web applications, as trying to generate dynamic Web pages simply with a classic programming language often is a crude attempt probably.

3.1 The generation of dynamic Web content using the server side

Web server software at least does the following: When a request of a certain URL is sent from a client, the Web server software will look for the requested resource and likely return some kind of answer to the client (usually the resource or an error). A client request usually contains a HTTP-header and possibly some certain parameters. This is everything a server gets to know about a client. If the client requests a resource associated with a certain program or executable file on the server, the server will pass the data to this program (Bates 2006, p. 348).

It depends a lot on the used Web server software (most common are Apache and the Internet Information Server (Netcraft 2007, “August 2007 Web Server Survey”)), how requests are processed, how data is provided to applications, executable files and how it is returned (or what to return in case of error). The traditional way to delegate the generation of (dynamic) Web pages to an application is a standard called CGI. Recently new technologies have evolved, usually plugins or modules are built for the server software itself to support a certain programming language or even special server software is built to power a desired programming language or technology for handling the server-side job.

3.1.1 CGI-Scripts

CGI is a standard that defines how Web server software can delegate the generation of Web pages to a stand-alone application or an executable file (Bates 2006, p.346). When a client requests an URI associated to a CGI-program, the program is executed by the server and the answer (something that the browser understands, typically HTML) is sent back (Scott 2009, p.680).

CGI-Scripts may be written in any programming language available on the servers’ machine, which justifies our course of action to consider every Turing complete programming language to be a possible “programming language for the Web”. There can be found tutorials on the Internet for CGI programming in C, for example (Korpela 2010,

"Getting Started with CGI Programming in C"), though C is not considered to be a typical Web programming language. According to Bates 2006, most CGI applications tend "to be fairly trivial", so large compiled languages are considered to be overkill in *most* cases of use.

How CGI basically works (Python could be any language here):

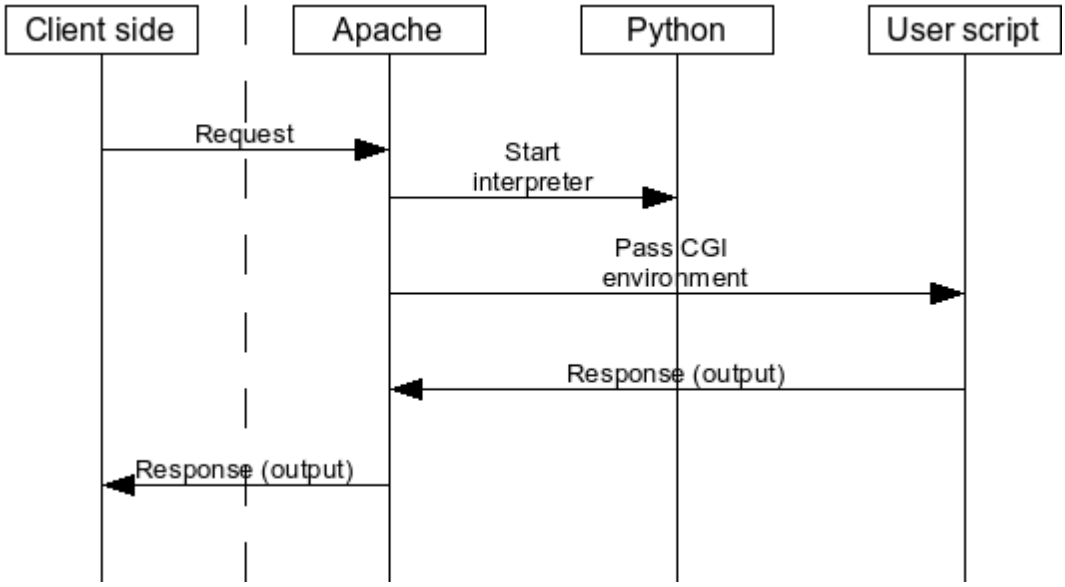

Figure 5: Request flow for CGI (Boender 2009, "Apache, FastCGI and Python")

CGI is considered to be slow sometimes, as the interpreter for a language has to be started for each single request. This is a minor problem with compiled languages (Podgoretsky 1997, "Common Gateway Interface (CGI)"). However there are newer variations of CGI which are considered to be faster and more effective, depending on the context in which they are used.

FastCGI and similar descendants

FastCGI can reduce the overhead for interpreted languages (e.g. Perl), as the interpreter of a programming language has only to be started once, when the Web server is started. It also provides a lot of more advanced possibilities, than CGI does (Brown 1996, "FastCGI Specification").

How FastCGI basically works:

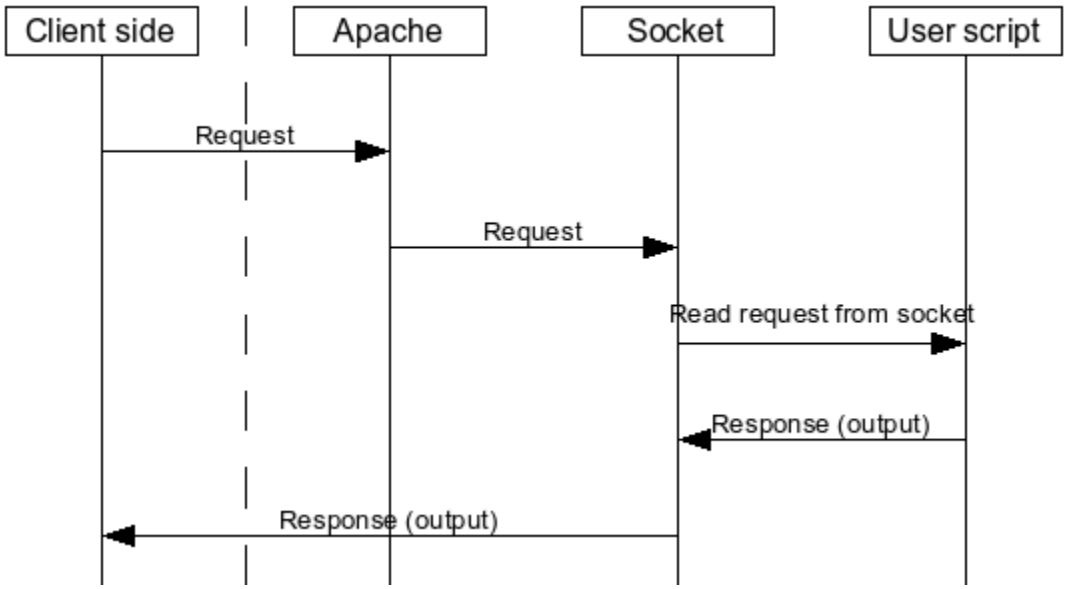

Figure 6: Request handling in FastCGI (Boender 2009, "Apache, FastCGI and Python")

There are a lot of other modifications and branches of the CGI technology (e.g. SCGI, WSGI, PSGI), which each have certain advantages and disadvantages.

3.1.2 "Module" or "plugin"-loading mechanisms – embeddable server-side scripts

According to Scott (2009, p.681ff) (at least standard) CGI Scripts have several disadvantages:

- each script must be launched as a separate program, with potentially significant overhead (compiled to native code they can be very fast once running)
- scripts must be installed into a separate trusted directory (security reasons)
- the name and directory of the script usually appears in the URI
- each script must generate the HTML-tags for formatting and displaying it, which is an additional effort

Because of this, most modern Web servers implement "module" or "plugin" loading mechanism, that allow interpreters for scripting languages to be incorporated inside the server itself, so as a consequence scripts can be embedded in "ordinary" Web pages. Clients have possibly no way to even know that a script exits then.

Embeddable server-side scripting languages include scripting languages like e.g. PHP, Ruby and SSI (Server Side Includes), SSJS (Server Side JavaScript) (Netscape Communications Corporation 1998, "Server Side JavaScript Guide v1.2"), as specialized

solutions like ColdFusion (with ColdFusion), C#/Visual Basic (with ASP and ASP.NET) and Java (with JSP).

3.1.3 A word on well-known, general and specialized solutions for Web programming

There are a lot of huge and massive proprietary and/or specialized solutions and frameworks which are usually used for server-side Web programming. Some of these are:

- ASP (Microsoft proprietary)
- CSP, Server-Side ANSI C
- ColdFusion (Adobe proprietary, formerly Macromedia, formerly Allaire)
- CGI and/or Perl (open source)
- Groovy (programming language) Grails (framework)
- Java, e.g. Java EE or WebObjects
- Lotus Domino
- PHP (open source)
- Python, e.g. Django (Web framework) (open source)
- Real Studio Web Edition
- Ruby, e.g. Ruby on Rails (open source)
- Smalltalk e.g. Seaside, AIDA/Web
- SSJS Server-Side JavaScript, e.g. Aptana Jaxer, Mozilla Rhino
- Websphere (IBM proprietary)
- .NET and .NET MVC Frameworks (Microsoft proprietary)

These are partly scripting languages, partly solutions associated with a certain server technology and/or programming language. The existence of frameworks or other technologies behind a programming language can rapidly influence its qualification for certain Web programming tasks, as a lot of work and time can be saved by using ready-bake functionalities.

3.1.4 Innovative and useful technologies / techniques in Web programming language design and implementation

Embeddable server-side scripting languages (or specialized/proprietary solutions for Web programming) usually provide lots of innovative and/or useful technologies/techniques for the simplification and enhancement of Web programming. We cannot figure out all of these for this chapter, but some important examples with explanation. These technologies/techniques can improve the qualification of a programming language for Web use significantly.

The advantage of separating coding logic and markup logic (separation of concerns)

A lot of proprietary and specialized solutions (PHP, ASP, JSP …) for Web programming tasks have in common that they can largely separate logic coding and HTML display coding (Gray 2004, p. 337ff). This concept is related to the MVC-pattern (Model-View-Controller) which is a common design pattern for advanced programming techniques.

The “servlet style” of Java requires programmers to echo content embedded in code (as it is the traditional way). For example:

```
out.println(“<body><h1>…</h1>…”);
```

This technique limits the adaptability and flexibility for editing page layouts, especially for non-programmers. Because of this, for a lot of remarkable Web programming languages (e.g. the ones accounted above) interpreters have been built, to transform “template-like markup documents” containing “processing instructions” (Scott 2009, p.682) into the actual programming language, before processing the request and generating the output for the server response.

The approach of “template-like” embedding of programming tasks in Web sites also is likely to appear in various forms. For the JSP technology for example, HTML documents can have “directives”, “actions” and “scriptlets” to be interpreted (Gray 2004, p. 338). Server-side includes use a similar approach for embedding programming tasks in HTML-templates, too (SELFHTML e. V., “Allgemeines zu Server Side Includes”).

Examples for fragmented loop scripts embedded into a Web page

The possibility to embed processing instructions into a Web page and even to fragment these instructions is an example for the simplification of generating output in (specialized) programming languages on the WWW. In a classic programming language, every output would have to be generated with an extra “print-like” statement.

PHP Example	JSP Example

<html>
<body>
 	
<?php for($i=0;$i<10;$i++) {
if($i%2) { ?>
<?php echo $i;
?>
 	
<?php } else { ?>
<?php echo $i; ?>
<?php }
} ?>
After the loop…
 	
</body>
</html>
	<%@ page language="java" %>
	<html>
	<body>
	
	<% for (int i = 0; i < 10; i++) {
	if(i%2==0) { %>
	<%= i %>
	<% } else { %>
	<%= i %>
	<% }
	} %>
	After the loop…
	
	</body>
	</html>

Table 13: Examples for fragmented loop scripts embedded into a Web page (Scott 2009, p.684 and Masslight, Inc. no date, "Introduction")

The HTML-output for both snippets in a browser will be as expected: A row of number from 0 to 9 with different formatting of even and odd numbers.

Case Examples: Directives and actions

More examples for innovative techniques and concepts in specialized Web programming language approaches are directives (e.g. JSP, ASP.NET) and actions (JSP) (there are certainly more than these) (Hall no date, "JavaServer Pages (JSP) 1.0" and Microsoft 2011, "Directives for ASP.NET Web Pages").

Directives *(specialized commands)*, in the context of JSP and ASP.NET, can be used to import packages, define error handling pages, define session information, and set page attributes, for example. Usage of JSP directives in JSP-templates:

```
<%@ directive attribute="value" %>
```

Actions, or JSP Action tags, enable a programmer to make fast usage of built-in functions. These include fallback-, include-, forward- or other specialized functions, for example. Actions are XML-tags to be used inside a JSP-template. Usage of JSP-actions in JSP-templates:

```
<jsp:action_name attribute="value" />
```

3.1.5 Summary

There are a lot of possibilities how server-side Web programming can be done and implemented. Usages of the CGI-Standard or custom solutions allow basically every

programming language to be used for server-side programming tasks. During history a lot of technologies evolved to provide faster or easier server-side programming. Embeddable server-side scripts have advantages regarding performance and usually functionality in difference to traditional CGI. There are a lot of huge projects, frameworks and solutions to simplify, speed up or improve Web server programming possibilities. The existence of such projects can boost a programming languages qualification for Web use. Technologies, techniques or concepts which make possible a separation between code and markup became very popular, as various other innovative or specialized approaches. Features for better development comfort are also good arguments to use a programming language for the Web.

3.2 The generation of dynamic Web content using the client side

Client-side content generation is another important part of the World Wide Web. Using the client side for generating dynamic content and displaying Internet content has a lot of advantages and disadvantages. The client side conditions are different from the server side conditions. Interactive Websites usually use a combination of both, client side and server side content generation.

3.2.1 Client-side scripting

Programming scripts for the client side in the World Wide Web, has a lot of advantages and disadvantages:

Advantages	Disadvantages
■ immediate responses to user actions (faster) ■ no page reloads required ■ fall-back mechanisms are possible ■ facilitates usability improvements ■ minimize server load	■ require an installed interpreter on the clients' machine (compatibility) ■ almost exclusive use of the JavaScript scripting language (standardized) for Websites targeting the general public ■ require more quality assurance testing (compatibility)

Table 14: Advantages and disadvantages of client side scripting (Scott 2009, p.686 and Boallert 2004, "Advantages and disadvantages of client-side scripts")

Using client-side scripting is a useful and important feature of the WWW. It is no alternative, but an important correlation for server-side scripting, as usage terms are different.

Client side scripting possibilities: ECMA Script (JavaScript/Ajax), DHTML and alternatives

"Few programming languages other than Java have been adopted for use in client-side Web applications. Visual Basic from Microsoft is probably the best known but is not widely used for general browser applications. In fact, most programming on the client-side is done in ECMA Script. ECMA Script is an international standard which was developed retrospectively around version 1.1 of JavaScript" (Bates 2006, p.140). ECMA script is well supported by most browsers. DHTML is the combination of HTML-formatted content, CSS, a scripting language and the DOM (document object model, standard), while the scripting language usually is ECMA Script compatible, although this is not obligatory.

There is a possibility for client-server communication without reloading pages, which is a great way for combining client-side and server-side scripts (Wenz 2007, p.13ff). There is a technology which nowadays is commonly called and implemented as AJAX ("Asynchronous Javascript + XML") to do hidden HTTP-requests from the client-side, to avoid page reloads. There is no need to use XML or even JavaScript here (though JavaScript is the most well-known implementation), but it is a technology for increasing dynamics on the WWW.

Frameworks

"In software development, a framework is a defined support structure in which another software project can be organized and developed. A framework may include support programs, code libraries, a scripting language, or other software to help develop and glue together the different components of a software project" (Horwith 2007, "When and why to use a framework"). In the context of client side-scripting (as in the context of server-side scripting) there are a lot of frameworks (usually JavaScript e.g. JQuery, Mootools). These can not only simplify programming tasks significantly, but also compensate disadvantages according to compatibility.

3.2.2 Objects (Browser Add-ons/Plugins)

Another possibility for client side content generation are "objects", which can be embedded into Websites (used HTML-tags are e.g. object, embed, audio, or video). For successful object embedding, the client browser must have an interpreter for (the version of) the programming language that is embedded. The object needs to have an appropriate handler for being displayed.

Advantages	Disadvantages
■ can theoretically be written in any language or be/use any file type ■ the client machines' computing power is required ■ do not produce HTML output, but control the page's real estate themselves	■ the browsers must have a built-in interpreter/handler for the desired technology or a Plugin must be available and installed by the client ■ because of obvious security issues only a handful of technologies is used widely (e.g. Java Applets, Flash, Quicktime Movies …) and these are still limited in their possibilities ■ even "established" proprietary technologies can easily be boycotted, like Adobe Flash by Apple (Jobs 2010, "Thoughts on Flash")

Table 15: Advantages and disadvantages of embedded client-side "objects" (Scott 2009, p.686ff)

As embedded "objects" usually do not have any significant interaction with the browser, they are not considered to be scripting mechanisms. A major advantage for "object" technologies is, when the important browsers implement them by default (e.g. Java virtual machine, built-in video/audio players for certain formats).

Java Applets

Java applets are written in the Java programming language, which provides a huge bunch of possibilities. As most browsers implement a "Java virtual machine" by default, Java programs can be embedded as objects into Websites on the Internet and most clients can handle them (Scott 2009, p.686ff). The technology has been around for a long time, so it is an outstanding candidate here.

Flash

Flash is a multimedia platform used to add animation, video and interactivity to Websites, and is likely used for advertisements, games and animations. According to Hauser / Kappler / Wenz (2009, p.19ff) more than 9 of 10 Internet users have installed a Flash Plugin in their browser (because of recent happenings this number might go down). The roots of Flash go back to 1996 (initial release) and further (Gay no date, "The History of Flash"), so it has been around for more than 13 years now. As it is widely spread (actually the most popular proprietary example for objects on the World Wide Web) and newer

versions allow highly advanced client-side scripting techniques (including outstanding features and object-orientation), we have chosen flash as the second example for generation of dynamic content on the client-side using objects here.

3.2.3 Transformation languages (or transformation-based stylesheets)

Transformation languages, or transformation-based stylesheets, can be used for dynamic Web content generation on the client side (Lie 2005, “Cascading Style Sheets”). For this work we could only figure out XSLT as a commonly supported variant of this group, though there might be others.

XSLT

As all major browsers have support for XML and XSLT nowadays and XSLT is a Turing complete programming language (Unidex, Inc. 2008, “Universal Turing Machine with XSLT”), the right place of XSLT is here in the section about dynamic generation of Web content using the client side.

As CSS has no real ability to perform computations, there was a need for a more powerful language to handle this task, so XSLT has been developed by the W3C (Kay 2001, “What kind of language is XSLT?”). XSLT splits the task of preparing an XML document into 2 steps, *transformation* (converting one XML document into another) and *formatting* (converting the tree structure into a two-dimensional graphical simulation). This proved to be useful for lots of practical terms of use.

An XSLT stylesheet is an XML document, whose basic processing paradigm is pattern matching. XSLT must be transformed by an (interpreter like) XSLT-processor. This processor is what the Web browsers actually have to implement to support XSLT:

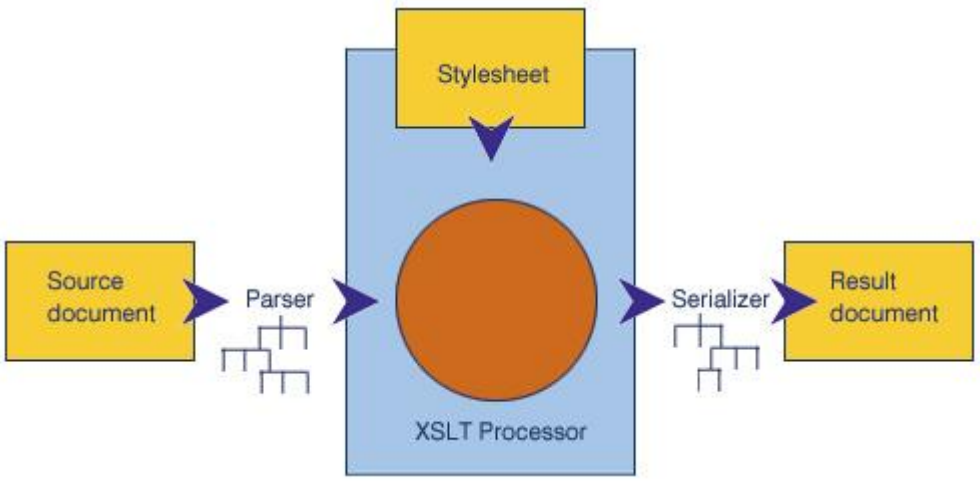

Figure 7: Operation of an XSLT Processor (Kay 2001, “What kind of language is XSLT?”)

Summary
XSLT provides all benefits of a high-level declarative programming language, with a specialization on XML-documents. Some claim it is not a functional programming language, because of the missing ability to treat functions as first-class data type (Kay 2001, “What kind of language is XSLT?”). Others claim it is one (Novatchev 2001, “The Functional Programming Language XSLT - A proof through examples”). However it is much more powerful than CSS and can be furthermore used for a certain kind of static and also theoretically dynamic client-side content generation, as it is a “Turing-complete” programming language.

3.3 Desktop applications versus Web applications (beyond the Web browser)

The Internet can be used and accessed in various ways, using its variety of protocols and possibilities to connect. Programs in any available programming language can be written for creating desktop software which accesses and uses the Internet (browsers themselves actually are programs to do this). If an application should be deployed for desktop use or as a Web application depends on its intended use. As this work is focusing on “programming languages for the Web”, the huge field of desktop applications is only to be covered briefly. Some desktop applications using the Internet are not using the World Wide Web, but certain other possibilities of the Internet.

Desktop vs. browser – when to deploy applications for each
According to Stewart 2007 (“Desktop vs. browser – when to deploy applications for each”), the following things can be considered when making this decision:

Browser Deployment	Desktop Deployment
■ low barrier of entry (browsers are convenient) ■ requirement of a central place for deployment (central update) ■ best cross-platform support (because of standards) ■ Web users have existing “browser knowledge” ■ client machines do not have to be accessed (no installation required)	■ application is front-end and center for users ■ “unlimited” access to system resources ■ possibility to build “widget applications” ■ close integration with the desktop is desired ■ consistent user interface is required

Table 16: Desktop vs. Browser, possible considerations (“Desktop vs. browser – when to deploy applications for each”)

The table above shows examples about what things to consider when a decision for a Web application or a desktop application has to be made. In the context of “programming languages for the Web” it is important to know, that Web applications are bound to the limits of content generation on the client side. Desktop applications, as browser Plugins, have to be installed to be used – this is likely a disadvantage for both – but desktop applications have “unlimited” (access) possibilities afterwards. As desktop applications have to be compiled and to be compatible to each different client operating system (which is hardly possible without limitations), this again is an argument for Web applications.

Rich Internet Applications (RIAs)
Rich Internet Applications are Web Applications which appear in a similar look-and-feel as desktop applications, or can be desktop applications which are using functionalities of the World Wide Web and/or the Internet (Busch, Koch 2009, p.5ff). These applications (the term “Rich Internet Applications” is vaguely defined) are an indicator for the accretion of the Desktop and the Web.

Why Desktop Applications matter for “programming languages for the Web”
Because of various obvious examples and in the context of RIAs we have been approved in our opinion that desktop applications have to be included in this work. There are countless desktop applications nowadays, which access the WWW (and other parts of the Internet) for presenting specialized contents inside of desktop applications (e.g. well-known ones like Google Earth, Eclipse, the Apple App Store …). All of these programs offer specialized built-in browsing functions. So the term “programming languages for the Web” also includes programming languages to be used on client desktops in the broader sense.

3.4 Summary

When talking about “programming languages for the Web”, we have to make a distinction between server-side content processing, client-side content processing for Web browsers and client-side content processing for desktop applications.

Server-side content processing can be implemented using CGI-Scripts, Modules/Plugins for server software or even more specialized open or proprietary software. Modules/Plugins and specialized solutions usually offer a bunch of advantages, including notably the separation of content and presentation via “template-concepts”.

Client-side content processing for Web browsers can be done via client-side scripting techniques, Web-embedded objects or transformation languages. They are essential for

the Web and can work together with server-side programs, but suffer from certain limitations in matters of programming.

Desktop applications which are using the World Wide Web also have to be considered when planning to create certain projects for the Web, especially in the context of RIAs. They must be programmed or compiled in a way compatible to the client system, which also involves new limitations.

4. Statistics and Trend analysis

Programming language popularity and usage statistics are hard to gather and only imprecise data about these can be gathered. There is a hand full of organizations using different examination methods to collect information about usage statistics as good as possible. None of these however focuses on Web programming, but only programming languages in general, so the procedure in this chapter will be as follows.

In the beginning we discover the possibilities to measure programming language popularity. Afterwards we will pick up the most recognized studies, compare their results and make a deduction about the most used programming languages in general. Each of these programming languages will be checked if it is known to be used for programming dynamic Web pages and if so, we will put it onto our list of candidates, whose fitness, features, advantages and disadvantages for Web use will be analyzed more precisely in the next chapter.

4.1 Measuring programming language popularity

The first thing is collecting and evaluating data about the popularity of programming languages. We decided to divide this into two parts – the first part will sum up the results of studies and statistics which deal with the popularity of programming languages in a provable way. The second part will include projects and expert sentiments on programming languages for the Web, as some further surveys. The evaluation and comparison of this data will spawn a list of programming languages to have a closer look on.

4.1.1 Studies and statistics

There are different ways to gather data and make studies about the usage and popularity of programming languages, and none of these is fail-safe, but an indicator. We try to show and summarize four studies with completely different approaches to get a differentiated picture.

A. Counting the hits of the most popular search engines (TIOBE Index) (Tiobe Software BV 2011, “TIOBE Programming Community Index Definition”)
The TIOBE Company has been gathering detailed usage statistics about the usage of programming languages in general for 10 years until now.

Procedure
There are two criteria for a programming language, to be accepted as such for the index:

- „The language should have an own entry on Wikipedia and it should clearly state that it concerns a programming language. This is the reason why ColdFusion, (Ruby on) Rails, Excel, Cocoa, ASP and AJAX are not considered programming languages for the index. “
- „The programming language should be Turing complete. As a consequence, HTML and XML are not considered programming languages. This also holds for the data query language SQL. SQL is not a programming language because it is, for instance, impossible to write an infinite loop in it. On the other hand, SQL extensions PL/SQL and Transact-SQL are programming languages. “

The index is using the following search engines: Google 32%, Blogger 32%, Wikipedia 16%, Youtube 10%, Yahoo! 3%, Bing 3% and Baidu 3%. The counted hits are normalized afterwards for the first 50 languages (=100%). The exact formula and definitions can be found on “TIOBE Programming Community Index Definition” (Tiobe Software BV 2011).

Result Summary (Tiobe Software BV 2011, “TIOBE Programming Community Index for July 2011”)

1. Java (19.3%) +
2. C (17.3%) -
3. C++ (9%) -
4. C# (6.2%) +
5. PHP (6.2%) -
6. Objective-C (5.2%) +
7. (Visual) Basic (5.1%) -
8. Python (3.6%) -
9. Perl (2.3%) -
10. JavaScript (2.2%) -
11. Lua (1.6%) +
12. Ruby (1.3%) -
13. Lisp (0.9%) -
14. Delphi/ObjectPascal (0.9%) -
15. Transact-SQL (0.8%) +
16. Pascal (0.7%) +
17. Assembly (0.6%)
18. RPG (OS/400) (0.6%) +
19. Ada (0.5%) +
20. C shell (0.5%) +

+ indicates a rising tendency (previous month) // - indicates a downside trend (previous month)

Table 17: Tiobe result summary (July 2011) (Tiobe Software BV 2011, “TIOBE Programming Community Index for July 2011”)

20-50 (ordered by position):
PL/SQL, Logo, Scheme, F#, Visual Basic .NET, MATLAB, D, SAS, Q, NXT-G, Scratch, Go, R, Forth, Fortran, ML, Alice, NATURAL, Clean, ActionScript, Icon, ABAP, Haskell, PL/I, OpenEdge ABL, COBOL, APL, Smalltalk, Erlang, Scala

50-100 (alphabetical order):
(Visual) FoxPro, 4th Dimension/4D, ABC, Algol, Arc, ATLAS, Avenue, Awk, Bash, bc, BETA, Boo, Bourne shell, CFML, cg, CL (OS/400), Cobra, cT, Dylan, Eiffel, Factor, Groovy, Inform, Io, J, JScript.NET, Korn shell, LabVIEW, Ladder Logic, MAD, Magic, Maple, Mercury, Monkey, MOO, MUMPS, Oberon, OpenCL, Oz, PILOT, PowerShell, Prolog, Revolution, S, SIGNAL, Standard ML, Tcl, TOM, VBScript, VHDL

Long Term Trends for 10 years:

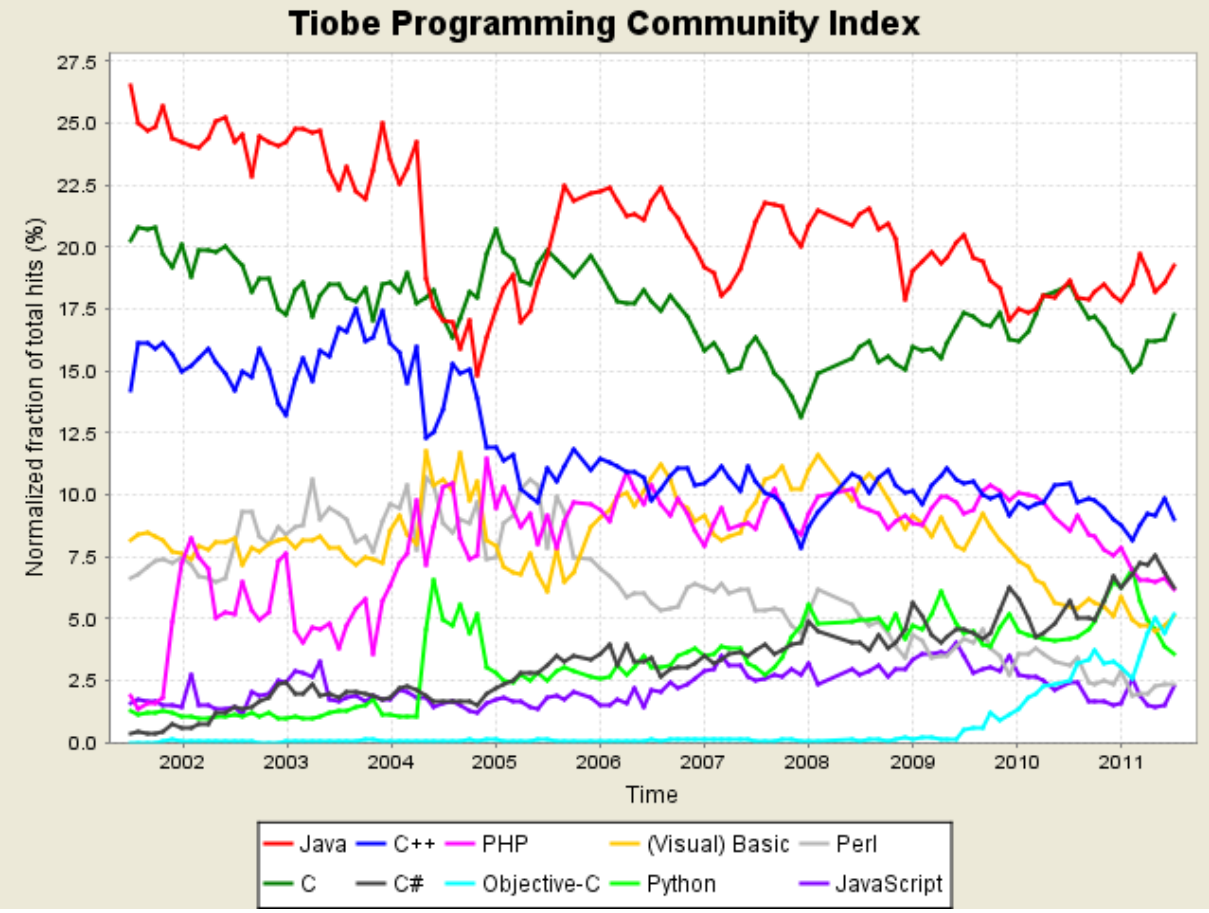

Figure 8: Tiobe Programming community Index (long term trends) (Tiobe Software BV 2011, "TIOBE Programming Community Index for July 2011")

Java: Most popular programming language with a lightly downwards trend

C: Almost as popular as Java with a very lightly downwards trend
C++: 3rd most popular programming language with a downwards trend
C#: Position 4 - outrunning PHP with a continuously rising tendency
PHP: Booming Tendency from 2002-2005, stable until 2010 and lightly downwards trend nowadays
Objective C: strongly booming tendency since beginning of the 3rd quarter of 2009
(Visual) Basic: stable position until 2009, downwards trend currently
Python: (almost) continuously lightly rising tendency until 2011, currently fallback to 3.6% of 2008
Perl: stable until 2005, continuously downwards trend nowadays
JavaScript: relatively stable around 2% for the last ten years

Relevance
The TIOBE Index is a comprehensible study about programming languages and their usage trends. It has got an universal approach and uses long-term gathered data from more than 10 years for calculating its' results. It has limits however: How much a language is talked about on the Internet does not say anything about the lines of code which are actually written and there are factors that could falsify the results. The study is about programming languages in general, which must be considered when drawing conclusions from it for usage of programming languages for the Web.

B. Counting the number of job advertisements that mention certain programming languages (Enticknap 2007, "SSL/Computer Weekly IT salary survey: finance boom growth")

The results shown in this article are based on information from the Computer Weekly/SSL Quarterly Survey of Appointments Data and Trends. It analyses ads for IT professionals on the Web.

Results
The results are ordered by "Skills most in demand on the Web" are the following:

1. SQL +
2. C +
3. Office +
4. Java +
5. SQL Server +
6. C# +
7. .net +
8. Oracle +
14. SAP +
15. HTML +
16. TCP/IP +
17. Linux +
18. J2EE
19. Exchange +
20. JavaScript +
21. Cisco

9. C++ -
10. Unix -
11. ASP +
12. Visual Basic +
13. XML +
22. Windows XP +
23. Access +
24. Focus +
25. Object oriented +

+ indicates a rising tendency (2006 -2007) // - indicates a downside trend (2006-2007)

Table 18: SSL/Computer Weekly IT salary survey: finance boom growth (Enticknap 2007)

Relevance

The survey shows the most common and desired programmer skills in the economic system in 2007. It differs from the aim of this study, as it is not related to Web programming and it also includes technologies, which are not considered to be programming languages by definition.

C. Counting the lines of code in a GNU/Linux distribution (Grupo de Sistemas y Comunicaciones 2009, "SLOCCount Web for Debian Lenny - General Statistics for Debian Lenny Code Counting")

These statistics count the actual number of packages, files and source lines of code (SLOC) of the Linux – Debian release, which release is 5.0 when writing this work. The dumbed result of it is as follows. We stripped out the Linux specific files here.

Result (in SLOC percentage)

C (48.5%)
C++ (30.3%)
Java (4.8%)
Python (3.124%)
Perl (2.9%)
Lisp (2.5%)
PHP (1.3%)
C# (1.1%)
Fortran (0.7%)
Ruby (0.6%)
ML (0.5%)
Tcl (0.5%)
Ada (0.4%)
Objective C (0.4%)
Pascal (0.3%)
SQL (0.2%)
Haskell (0.2%)
Fortran 90 (0.1%)
awk (<0.1%)
JSP (<0.1%)
Modula3 (<0.1%)
COBOL (<0.1%)

Table 19: Counting lines of code in a GNU/Linux distribution (Grupo de Sistemas y Comunicaciones 2009, "SLOCCount Web for Debian Lenny - General Statistics for Debian Lenny Code Counting")

Relevance
Counting the lines of code in a GNU/Linux distribution is real world evidence for the usage of programming languages. It can be used as an indicator for the current real world usage of programming languages, but Linux is not the only operation system so general results might differ strongly. Linux is an operation system furthermore which is not directly related to the data needed for this work, but it is useful as an alternative general overview.

D. Using a number of well-known sites and services to calculate a comparison sheet (DedaSys LLC 2011, "LangPop.com – Programming Language Popularity")
The LangPop.com project uses a lot of well-known services or sites to create sheets about the relative popularity of programming languages. The results are summarized in a normalized comparison sheet.

Procedure
The following services are used and each of these is assessed equally in the normalized sheet finally:

- "Yahoo search results" are gathered by searching for "*language* programming"
- "Craiglist" – using the Yahoo search API with queries like *language* programmer – "job wanted" site:craiglist.org
- "Powell's Books" – searching language names in book titles
- "Freshmeat" platform - search for the utilized programming languages
- "Google Code" - search for the utilized programming languages
- "Del.icio.us" – searches like "language programming"
- "Ohloh" – number of people committing code in a particular language

Results

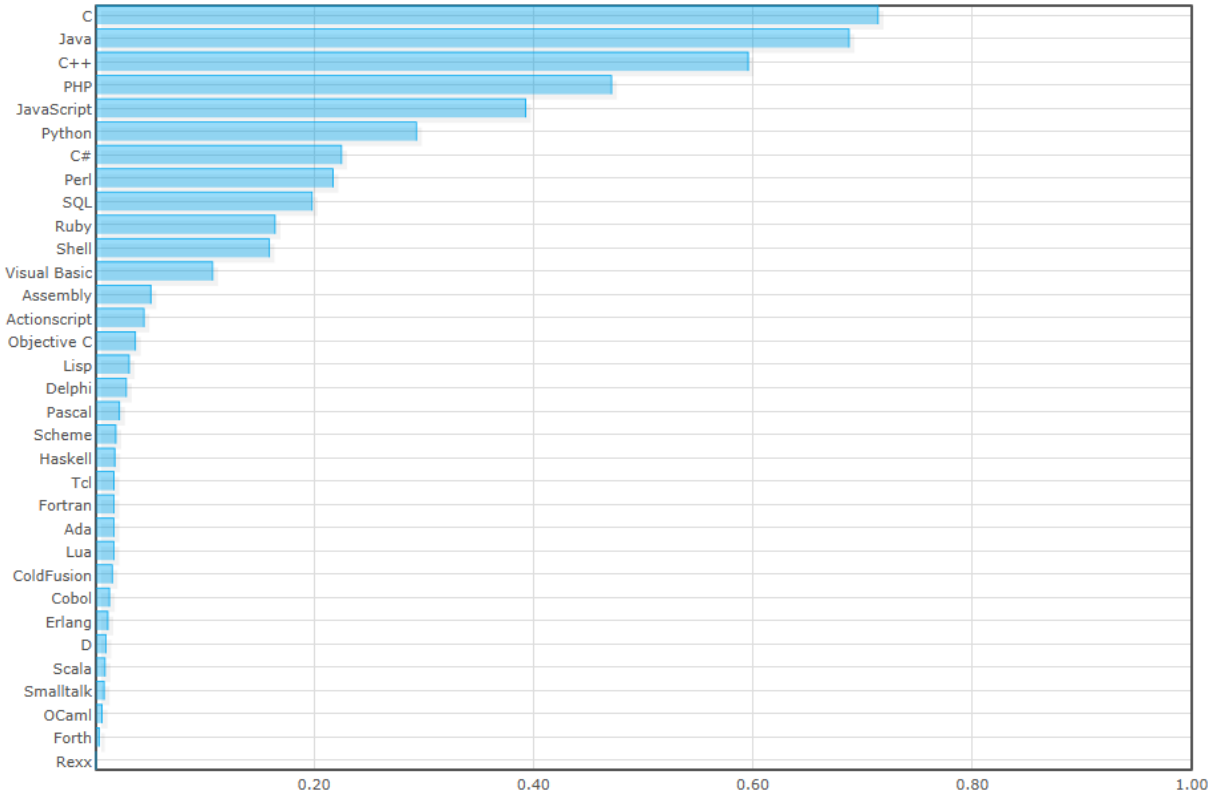

Figure 9: Using a number of well-known sites and services to calculate a comparison sheet (DedaSys LLC 2011, "LangPop.com – Programming Language Popularity")

1. C
2. Java
3. C++
4. PHP
5. JavaScript
6. Python
7. C#
8. Perl
9. SQL
10. Ruby
11. Shell
12. Visual Basic
13. Assembly
14. ActionScript
15. Objective C
16. Lisp
17. Delphi
18. Pascal
19. Scheme
20. Haskell
21. Tcl
22. Fortran
23. Ada
24. Lua
25. ColdFusion
26. Cobol
27. Erlang
28. D
29. Scala
30. Smalltalk
31. OCaml
32. Forth
33. Rexx

Table 20: Using a number of well-known sites and services to calculate a comparison sheet (DedaSys LLC 2011, "LangPop.com – Programming Language Popularity")

Relevance

The project is collecting data about the popularity of programming languages in general, using various different approaches to aim this target. The service claims itself to be not scientific, but it is an attempt to gather as much representative data as possible indicating the popularity of programming languages. So the usage of the completely reproducible data as an indicator in justified. The whole project with explanatory statements concerning the used services can be found on http://langpop.com.

E. Counting the number of book sales for learning a particular programming language (Tim O'Reilly 2006, "Programming language Trends")

Results

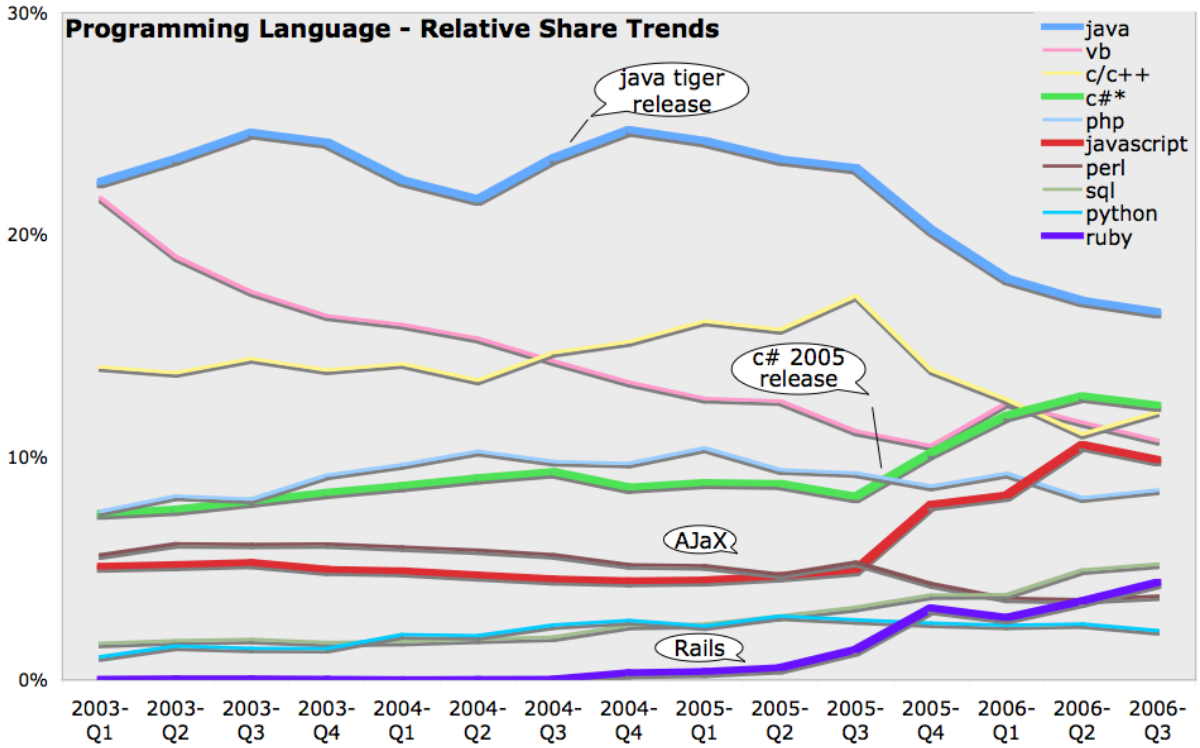

Figure 10: Counting the number of book sales for learning a particular programming language (Tim O'Reilly 2006, "Programming language Trends")

Result Summary

1. Java -	6. PHP
2. C# +	7. SQL +
3. C/C++ -	8. Ruby +
4. VB -	9. Perl -
5. JavaScript +	10. Python

+ indicates a rising tendency (2003 -2007) // - indicates a downside trend (2003-2007)

Table 21: Counting the number of book sales for learning a particular programming language (Tim O'Reilly 2006, "Programming language Trends")

Relevance

The data is an indicator for programming language popularity as the number of book sales shows how much interest for a programming language is on the market. It is especially interesting for this work, as it (almost) only contains programming languages which are known to be used for generating Web pages. Obvious disadvantages are: The data is in the time of writing this work almost 5 years old. Interest and information about a programming language are indicators, but do not give any information about its actual usage.

4.1.2 More statistics, projects, surveys and experts' opinions on the topic

As we could not find any studies and projects which are working or comparing the popularity of programming languages specifically for Web use, this part of the work is for comparing expert sentiments, further surveys and other projects which compare programming languages concerning Web usage.

A. An overview of important Web programming languages by Esfandir Amirrahimi (February 1st 2011)

"The languages mentioned below are attempts to create the 'ideal' Web programming language, which is usually done by extending and restricting existing languages. […] This document attempt to present a short introduction of the most important languages being used in the Web today".

Why this article was chosen

The article is dealing exactly with the topic for programming languages for the Web 2011. It was published on EzineArticles.com, an experts publishing platform. The author has successfully studied in computer science education and is working in the software development business for Web projects.

Programming Languages / technologies estimated to be the most important for Web programming

Language / technology	open source	Reason why this language / technology was chosen
ASP.NET – Active Server Pages	Microsoft (proprietary)	ASP.NET drastically reduces the amount of code required to build large applications, security / safety, disadvantage: bound to the Windows platform
PHP	open source	massive community support, low costs, compatibility, quick adoption, fast updates, no risk of soon disappearance, many programmers
Java / JSP	partly open source (Oracle)	less platform specific than ASP, adoption for Unix / Windows / Mac with little efforts, populariariy
Perl, Perl/Tk	open source	mature, powerful, huge bunch of tools, ongoing development / community, quick / clean / elegant, disadvantages: complexity, different appearance
Python	open source	full featured, object-oriented design but extremely ease of use, cross-platform compatibility, portable GUI libraries
ColdFusion	Adobe (proprietary)	intended for creating dynamic, database powered Web applications, own markup language, very easy creation of dynamic pages
Ruby / Ruby on Rails	open source	dynamic and object-oriented design, clean syntax, simple, productive, growing propagation, MVC-model (Ruby on Rails, highly productive, highly scalable

Table 22: Programming Languages / technologies estimated to be the most important for Web programming

B. Counting the number of job advertisements that mention certain programming languages (Indeed.com 2011, “Job Trends”)

We are using the Indeed.com Job Trends to determine how the job market indicates the popularity of programming languages. We do this using our own keywords, so we decided to list it as “other project”. Keywords (as jobs) might have a limited explanatory power about the real popularity of programming languages, but defining our own keywords has also a big advantage: We can use results of the so far work, to compare especially popular programming languages or associated technologies which focus on Web development. The data is up-to-date when writing this work.

Indeed.com provides a job trend comparison by keywords. We decided to use Web-associated technologies instead of the language (ASP instead of C#, JSP instead of Java), leave out languages like C/C++, which are not considered to be mainly used for Web programming. We think that the results are most interesting using the keywords we have chosen.

Why It is used as an indicator (Indeed.com 2011, “About”)
Indeed claims to be the biggest Job site worldwide currently (50 million unique visitors/month and 1 billion job searches/month, 50 countries, 26 languages).

Results

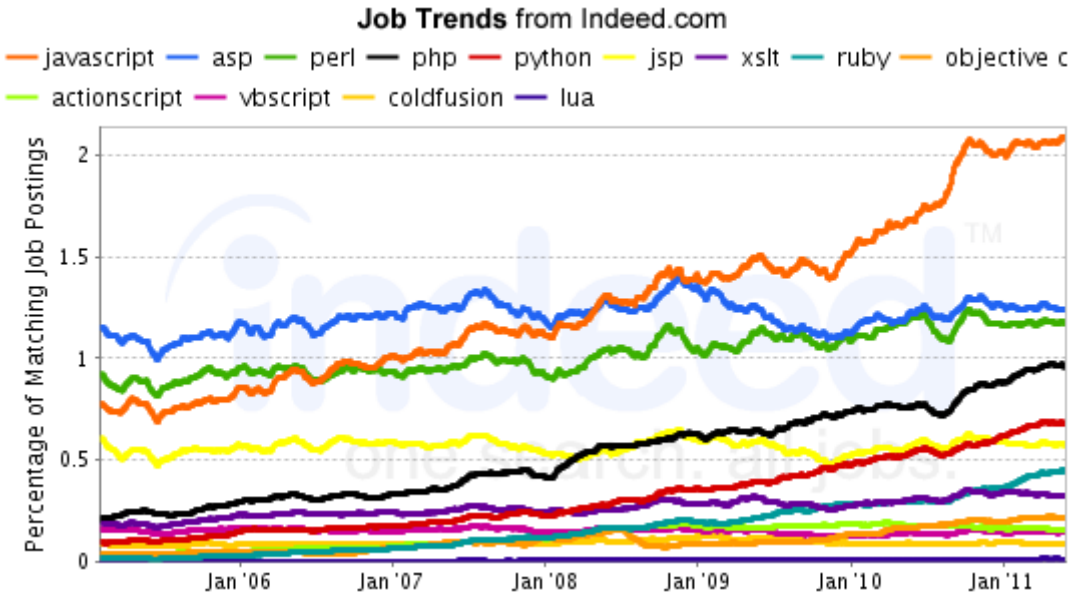

Figure 11: Indeed.com “Job Trends” absolute scale (July 26th 2011, “javascript, asp, perl, php, python, jsp, xslt, ruby, objective c, ActionScript, vbscript, coldfusion, lua Job Trends”)

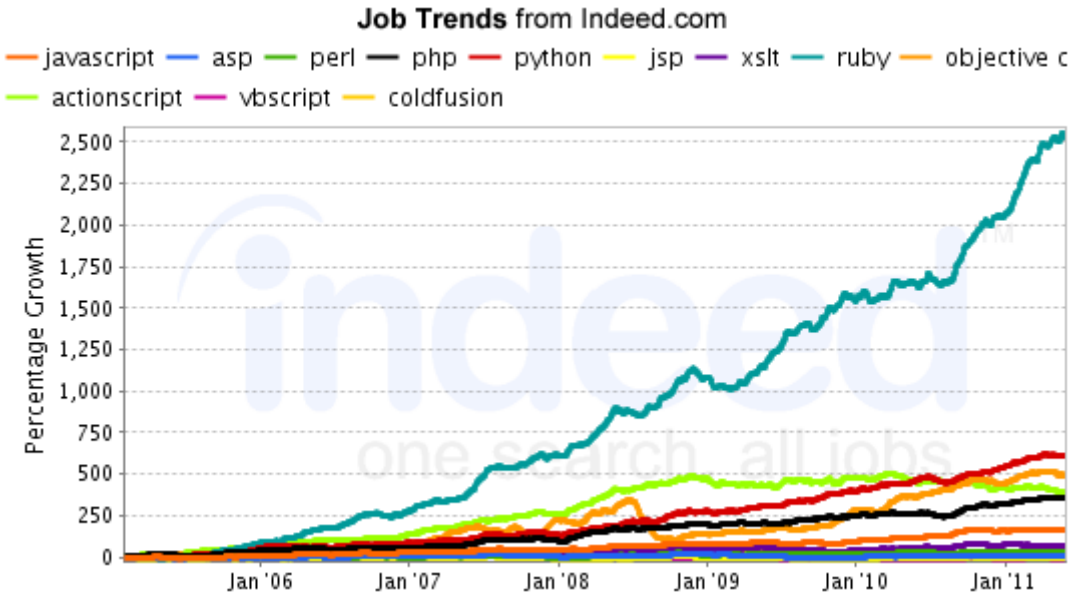

Figure 12: Indeed.com "Job Trends" relative scale (without Lua) (July 26th 2011, "javascript, asp, perl, php, python, jsp, xslt, ruby, objective c, ActionScript, vbscript, coldfusion Job Trends")

Interpretation

1. JavaScript: Very high percentage and constantly rising tendency
2. ASP: High percentage and constant over the last 7 years (note: searching for .NET instead would give a result 4 times higher)
3. Perl: High percentage and lightweight rising tendency
4. PHP: High percentage and remarkable rising tendency
5. Python: Middle-rate percentage and remarkable rising tendency
6. JSP: Middle-rate percentage and constant over the last 7 years (note: searching for j2ee instead would duplicate the result)
7. Ruby: Middle-rate percentage and extraordinary rising tendency
8. XSLT: Middle-rate percentage
9. Objective C: Lower percentage and remarkable rising tendency
10. ActionScript: Lower percentage and constant over the last 3 years
11. VBScript: Lower percentage and constant over the last 7 years (note: searching for "visual basic" instead gives a result more than 4 times higher and is also constant)
12. ColdFusion: Low percentage and constant over the last 7 years
13. Lua: Very low percentage, but enormous rising tendency (we had to remove it from the graph because of readability)

Note: When adding keywords of the domain specific-languages for the Web, the most common are all above the other keywords. Notably SQL, which is even two times more desired than HTML.

C. Further studies, surveys and expert opinions (brief overview)
There are a lot of more studies, surveys and expert opinions to be found on the Internet, which try to detect the most popular programming languages and all of these provide similar results to what we already have, so here is a brief overview:

Ten Programming Languages for 2011 (conducted by recent survey, experts sentiment) (Taft 2010)	10 Programming Languages You Should Learn Right Now (experts sentiment) (Rothberg 2006)	Computerworld Development Survey gives nod to C# (survey) (Computerworld 2005)
2010	2006	2005
Java C# C/C++ JavaScript Visual Basic PHP Objective-C Perl Python Ruby	PHP C# Ajax JavaScript Perl C Ruby / Ruby on Rails Java Python VB.NET	C# – 72% Java – 66% Visual Basic – 62% C++ – 54% JavaScript – 50% Unix Shell Scripts – 42% Perl – 34% C – 32% PHP – 16% Python – 8% Delphi – 7% TCL – 6% Ruby – 1%

Table 23: Further studies, surveys and expert opinions (brief overview)

4.2 Deductions: What are the (most popular) programming languages for the Web 2011 according to surveys?

Using the surveys we have determined, this section is intended to draw deductions and to discover the most popular programming languages for the Web at present times.

4.2.1 General deductions from analyzing the surveys

When comparing all the surveys, we can make a lot of important deductions about the popularity of programming languages (and/or Web programming languages):

- We can assert that they all have a tendency to bring up the same language names, though in conspicuous different order regarding their importance or actual popularity.

- The difference between actual programming language popularity and Web programming language popularity seems even not as high as expected – system programming languages, remarkably C and C++ go down a little bit and scripting languages are obviously more interesting when comparing programming languages for Web use. We have shown that C and C++ can be used for Web programming tasks, but it is also considered to be “overkill” in most terms of use.
- There are two popular, a bit more hardware-abstracted, *system programming languages*, which are widely used for Web programming purposes, namely C# and Java. Each of these is strongly connected to a specific Web Framework technology, namely ASP (C#) and J2EE (JSP/Servlets) (Java) in terms of Web use. All other languages with huge relevance have at least a lot of characteristics of scripting languages, while Python is partly a hybrid one.
- The usage of preferred programming languages changes year by year. There are a lot of important “newcomers”, which have not been popular for many years and become popular more or less suddenly, because of a certain happening or change (most notably Objective-C, Ruby (on Rails), Lua …).
- “Giants” and programming languages once important do not disappear fast (possibly because of legacy issues), but some of these are losing popularity slow-going, at least in the normalized comparison (e.g. C++, Perl, Visual Basic). Perl for example has still an increasing number of Jobs according to “Indeed”, but has lost about 75% of its high popularity within the last 6 years). The Ada programming language is losing popularity constantly, while Lisp is an example for a programming language to get more popular after almost 20 years of descending (TIOBE long-term trends).
- According to the TIOBE index, C#, Objective-C, Lua, Lisp and Transact-SQL are gaining popularity notably, while most others are losing. According to recent Job Trends (Indeed) Lua, Ruby, Python and Objective-C (in this order) are gaining popularity rapidly.

4.2.2 Candidates for the “Web programming language comparison 2011”

The most important Web programming languages for the Web 2011 have to be chosen based on the collected data so far, and each of it will be listed here with a short explanatory statement. We decided not to set up a differentiated system how to calculate the actual usage results now, as the results from above are vague enough.

To determine the most popular programming languages for the Web, we determine all notable candidates from the studies, surveys, and experts sentiments above.

Mainly server-side scripting languages for the Web

PHP is the most popular scripting language for server-side scripting, and has notable positions in almost any ranking.

Perl has a famous history, and is built for server-side scripting in the first place. Perl has still a constantly good ranking in surveys.

Python has been strong for several years, is known to be used for Web-programming partly and still has a notable upwards trend.

Ruby gained popularity massively in the last 5 years (rails release). It is the youngest notable language scripting language of the five. According to the TIOBE index it is losing some of its popularity now, but most rankings, notably the Job Trends show contrary results.

ColdFusion (with own scripting technologies CFML/CFScript) is kind of the weakest candidate here. It is not listed in the TIOBE index as they do not consider it as a programming language, which is criticized by some people (Gilbert 2008, “TIOBE: Coldfusion not a Programming Language”). But most other rankings list it. It is a (proprietary) solution especially developed for the Web, and is considered to have advantages by experts. Because of these reasons we decided to accept it here.

Mainly client-side programming technologies for the Web

JavaScript is a standard, ranks good in all rankings (especially the job ranking from Indeed) and is the only one to be known implemented by all major browser, so its choice here should be obvious.

ActionScript, the scripting language of Adobe Flash, ranks poorly in the TIOBE index, but still has notable positions in some other rankings. As Flash is a technology for the Web primarily, its position here is justifiable furthermore.

XSLT, though a domain specific language, is considered to be a programming language, as we have proven before. According to the TIOBE index it is ranked below the 100st place, though it is ranked. It is not ranked in other rankings In spite of that, according to the Job trends from Indeed, it is place 8, more desired than Objective-C and ActionScript. As it is significantly associated with the Web furthermore, we have decided to choose it here.

Object-oriented system programming languages for the Web

Java (mainly J2EE, JSP) is listed in every ranking, is the leading language on the TIOBE index currently and even JSP has a notable position in the Job Trends.

C# (mainly ASP.NET), also listed in every ranking, seems still to gain popularity, is holding a respectable fourth place in the TIOBE index currently and even ASP has a constantly great second place in the Job Trends.

Objective-C is the top upward climber of the past few years, although it has been around for a very long time before. It is an application programming language in the first place, but it can be used for Web programming. It also might be one of the best examples for a language, which is used to program desktop applications that use the Web.

C and C++ are probably no programming languages to be used for Web programming in the first place. However they are represented in most of the rankings in key positions and there are lots of C and C++ Tutorials for CGI-programming on the Internet, so they must be considered here.

Visual Basic (and VBScript) – as Visual Basic is not the typical Web programming language (primarily application programming) and VBScript is not that popular and compatible, these candidate(s) do not matter that much. But as they still rank good enough, they have to be considered.

5. Comparison of Web programming languages and related technologies

In this final chapter all results of the previous chapters come together. We have worked out the distinctive features of programming languages, technical and general. We analyzed the affordances and character of the World Wide Web and the Internet. We showed the possibilities of how programming languages can be implemented and used to program the WWW. We also compared a huge number of statistics, surveys end expert sentiments about the usage and usage statistics for programming languages in general and the World Wide Web and found out the most important and popular ones.

The aim of this chapter is to create a profile comparison sheet which can be used to compare the suitability of single programming languages for Web use. Afterwards we will also determine the limits of this sheet, as there will certainly be some. In the third part we will finally compare the most popular Web programming languages using a simplified version of our sheet, as all results must be proved.

5.1 A Web programming language comparison profile

Depending on the results of this work, we can create a programming language comparison sheet, which allows a comparison of programming languages for Web use. For limits of the profile see “limits of the profile”.

5.1.1 The profile

General Information	
Name	*Name of the programming language*
Project Website	*Website-URL for the programming language*
Description	*Brief description of the creators*
Intended Use	*Maybe one of application development, Web, client-side, server-side, embedded system, domain specific, system, scripting, …*
Company relationship	*e.g. open source, proprietary, correlation with well-known company and name of the company/companies*
License	*e.g. MIT, GNU, proprietary, …*
Comparison of Web qualification	
Generation of server-side contents	*Usually one or more of CGI, existing Server Modules/Plugins, or specialized solution*
Features	*e.g. implementation of “template-based” scripting exists, directives, actions …*
Generation of client-side contents	*One of browser scripting (broadly implemented), object embedding (maybe name existing browser Plugins and percentage of browsers implementing), and/or transformation language // Maybe name existing percentage*
Related Web technologies	*Known tools/techniques/technologies/solutions/implementations that are specifically built to simplify usage for Web programming*
Frameworks	*Known existing Web Frameworks (e.g. JQuery, Zend Framework, ASP.NET, …)*
Technical comparison	
Computational Model	*Imperative (von Neumann, scripting language, object-oriented language) or declarative (functional, logical or dataflow language)? Maybe there is more than one possible answer.*
Estimated	*e.g. low, middle, high, very high, maybe some kind of explanation*

Highness of level	
Programming paradigms	*Usually one or more of functional, imperative / structural / procedural, object oriented, or less common: concurrent, distributed, generic, reflective and more*
Compiled or Interpreted	*Mainly compiled or mainly interpreted?*
Garbage collection	*yes or no*
Type system	*Mainly dynamic or mainly static typed? Strong or weak typing?*
Performance ranking	*Note: Results from the "Computer Language Benchmarks Game" may be used for completing the following fields*
Performance	*Insert data for comparison of performance (execution time) here*
Expressive power	*Insert data for comparison of expressive power (code size) here*
Memory consumption	*Insert data for comparison of memory consumption here*
Comparison of other factors	
General popularity	*TIOBE position, possibly more statistics*
Available Tools (known)	*Usually development environments like Eclipse, Netbeans, Visual Studio, …*
Legacy	*Is the programming language popular because of legacy-issues?*
Documentation	*Is there a good documentation?*
Community, Support	*Community size? Free / commercial support? Frequent updates?*
Syntax	*Maybe a short syntax example?*
Semantic rules	*Strong or weak checking of semantic rules?*

Table 24: A Web programming language comparison profile

5.1.2 Limits of the profile

The profile which is for comparing and measuring a programming languages' qualification for Web use is bound to and limited by (at least) the following conditions:

- **Fairness of the comparison:** Because of clearness and technical / logical limits, the comparison is limited in its fairness, as a lot of factors can be affected by certain things.
- **Accuracy of comparison:** The comparison has certain limits according to its accuracy (e.g. comparing performance / expressive power). A lot of highly complicated and differentiated topics of programming language science are compared in a very basic way.

- **Explanatory power of the comparison:** Because of technical / logical limits, the explanatory power of the comparison is limited.
- **Project specific needs:** Each programming project has a lot of project specific needs, which should be considered when choosing a programming language. These needs cannot be documented by the comparison, but it can be used to search for matches.

5.2 Abbreviate comparison of the most important programming languages for the Web

In this last part of the work the most important programming languages for the Web are compared concretely. Popularity is an argument for the use of programming languages, as it sticks together with some other important factors closely (e.g. existence of Frameworks / Solutions, libraries, documentation, tools, community size / support). But for all that there are less popular languages that may be great or even the best ones for certain purposes so these can and should be considered if they are suitable, too.

It is not possible to do a profile comparison for all the major programming languages within this work, as this would go beyond the scope of it. Each programming language would have to be analyzed in detail; the results would have to be proven and after this process, the profile comparison is still limited and uncommon programming languages are not given a chance at all. So regarding on specific needs, the comparison profile can be used for comparing specific candidates and one may chose this candidates inspired by the general descriptions of the languages below. The addiction to Web programming is also described for each candidate. The most important programming languages are the first seven ones to be described.

5.2.1 PHP

The creators: PHP: Hypertext Preprocessor (http://www.php.net/) is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. It is mainly used for server-side scripting, and is licensed open source (php.net 2011, “What is PHP?”).

PHP is usually embedded using server-plugins/modules, has got a massive community support and therefore a huge collection of tools, frameworks, books, tutorials, web features and libraries are existing (Amirrahimi 2011, “An Overview Of Important Web Programming Languages”). As it is spread very widely it will not disappear soon and many web projects are built upon it. New versions of PHP also implement object-orientation.

As it is an interpreted and dynamically typed language, glued together using various existing program parts, the major disadvantage of PHP possibly are performance and security.

5.2.2 Perl

According to its creators, Perl 5 is a highly capable, feature-rich programming language with over 23 years of development (perl.org 2011,"About Perl"). Perl 5 runs on over 100 platforms from portables to mainframes and is suitable for both rapid prototyping and large scale development projects. Perl is an ideal web programming language due to its text manipulation capabilities and rapid development cycle.

Perl is widely used, open source, mainly used for server-side scripting purposes and can be embedded as module into the Apache server, though it is mostly known to be used with CGI. It is a scripting language with a big international community and offers almost every tool to create dynamic websites (Amirrahimi 2011, "An Overview Of Important Web Programming Languages"). Best known disadvantages of Perl are possibly its complexity and individuality.

5.2.3 Python

According to its creators, Python (http://www.python.org) is a programming language that lets you work more quickly and integrate your systems more effectively. Learning it promises almost immediate results in better productivity and lower maintenance costs. It is used in a wide variety of application domains (python.org 2011, "About").

Python is a mostly interpreted, dynamically, but strongly typed, object-oriented open-source programming language that utilizes automatic memory management and is considered to be very suited for Internet programming (Rothberg 2006,"10 Programming Languages You Should Learn Right Now").

5.2.4 Ruby

Ruby (http://www.ruby-lang.org/en/) is promoted as "a dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write" (ruby-lang.org 2011, "Ruby is…").

Ruby is considered to be easy to learn. Ruby on Rails is a highly scalable framework written in Ruby for building Web-applications. It uses the MVC-architecture (Rothberg 2006,"10 Programming Languages You Should Learn Right Now").

5.2.5 C# and ASP (Active Server Pages)

According to its creators, C# is a “type-safe, object-oriented language that is simple yet powerful, allowing programmers to build a breadth of applications. Combined with the .NET Framework, Visual C# 2008 enables the creation of Windows applications, Web services, database tools, components, controls, and more” (Microsoft.com 2011, “Getting Started with Visual C#”).

C#, though standardized, is strongly addicted to the company of Microsoft. ASP.NET (proprietary) runs inside the Internet Information Server from Microsoft and provides the execution of scripts by an internal server (Amirrahimi 2011, “An Overview Of Important Web Programming Languages”). It includes a lot of innovative tools and techniques to simplify Web programming, and drastically reduces the amount of code. (Note: The .NET Framework in which ASP is included includes and uses more programming languages such as Visual Basic.NET).

5.2.6 Java and JSP

Java is a powerful, very popular, object-oriented programming language. It can be used for client and server-side web programming.

Server-side web programming is usually associated to the Java Enterprise Edition (J2EE), and more specifically JSP. According to its promoters, the JavaServer Pages (JSP) technology provides a simplified, fast way to create dynamic web content. They claim, the technology enables rapid development of web-based applications that are server- and platform-independent (oracle.com 2011, “JavaServer Pages Technology”).

The JSP-technology is less platform specific than the ASP-technology and is one of two “non-scripting” languages out of the seven most used web programming languages (Amirrahimi 2011, “An Overview Of Important Web Programming Languages”).

Client-side programming for the Web can be done using Java-Applets (Scott 2009, p.688).

5.2.7 JavaScript

JavaScript is the most important client side scripting language for the Web by far. It is used by “billions of Web pages to add functionality, validate forms, communicate with the server, and much more” (w3schools.com no date, “JavaScript Tutorial”).

JavaScript is the only standardized scripting language to be understood by all major Web browsers. Therefore, countless Frameworks and technologies built upon JavaScript exist, as we figured out in the section about client-side scripting.

JavaScript can also be used for server-side scripting (SSJS). SSJS is “a JavaScript interpreter for running JavaScript programs on the server. It includes a library of objects and functions for accessing databases, sending e-mail and performing other tasks (yourdictionary.com 2010, “SSJS computer definition”).

5.2.8 ColdFusion (with CFML/CFScript)

According to its publisher Adobe Systems Incorporated, ColdFusion (application server) is a proprietary solution which enables developers to build, deploy, and maintain robust Internet applications for the enterprise rapidly (adobe.com 2011, “Products – ColdFusion”).

CFML (ColdFusion Markup Language) and CFScript (ColdFusion Script), similar to JavaScript, are used for server-side scripting purposes on the ColdFusion server (livedocs.adobe.com no date, “About CFScript”). ColdFusion is said to be one of the easiest solutions for building powerful web applications (Amirrahimi 2011, “An Overview Of Important Web Programming Languages”).

5.2.9 ActionScript (Flash)

According to its publisher Adobe, ActionScript (proprietary) is described as the following: “Adobe ActionScript is the programming language of the Adobe Flash Platform. Originally developed as a way for developers to program interactivity, ActionScript enables efficient programming of Flash Platform applications for everything from simple animations to complex, data-rich, interactive interfaces” (adobe.com 2011, “ActionScript Technology Center”).

ActionScript is the scripting language for the most popular object-embedded technology on the Internet, Flash, which must be installed using a browser Plugin. It is great for creating rich multimedia content. Because of the boycott by the Apple company (Jobs 2010, “Thoughts on Flash”), there are chances that its popularity will go down. But according to statistics, we cannot see a significant tendency to fall yet.

5.2.10 XSLT

XSLT is considered to be a domain-specific stylesheet transformation programming language, which is primarily used to process XML content on the client-side in the World

Wide Web. It is designed and developed by the W3C and can be used for various purposes.

As a programming language XSLT has many features, from its use of XML syntax to its basis in functional programming theory (Kay 2001, “What kind of language is XSLT?”). It is said to be and immensely powerful technology. But advanced tasks are considered to be unfamiliar to the average Web programmer today, which probably is its main disadvantage currently.

5.2.11 Objective-C

Objective-C is an intuitive, object-oriented, simple, but powerful application programming language (an extension of the ANSI C language). It is primarily used for the creation of native applications for devices of the Apple company and the Mac system (roseindia.net 2008, “Objective C introduction”). Most of the history of Objective C is addicted to the companies NEXTstep and Apple (Noack 2005, “Ausarbeitung : Objective C p.2”). Originally, Objective C is a very old programming language that was designed and implemented in the early 1980s.

Theoretically it can be used for all kinds of Web programming and as the language has become that popular nowadays, it may be used for several different purposes.

5.2.12 C

C is a traditional, standardized (ANSI C) widely used, general-purpose programming language, which is mainly used for system programming today. It is a structured language, can handle low-level activities, and be compiled for a variety of computers (tutorialspoint.com no date, “C – Basic Introduction”).

C is not a Web programming language in the first place, though lots of software in and around the Web is approximately based on C. It can be and actually is used for CGI programming on the Web (Korpela 2010, “Getting Started with CGI Programming in C”).

5.2.13 C++

C++, similar to C and also standardized, is primarily an application and system programming language. It is very popular, provides object-orientation and is compatible to C. It is more abstracted from the hardware than C but there are few limits in what can be done (cpluplus.com 2011, “A brief description”).

C++, as C is not a programming language for the Web in the first place, but can be can be used and therefore certainly is used for CGI-programming (Rutenberg 2007, “Introduction to C++ CGI”) and application programming for the Web.

5.2.14 Visual Basic (and VBScript)

Visual Basic is a proprietary, object-oriented programming language (and also a programming environment) created by the Microsoft Corporation, which is widely used for general-purpose programming within the world of Microsoft (Mabbutt no date, “What is Visual Basic?”).

According to results of this work, it can be used for Web programming purposes at least in the context of desktop applications and CGI-programming.

VBScript (Microsoft Visual Basic Scripting Edition) brings active scripting to a wide variety of environments, including “Web client scripting in Microsoft Internet Explorer and Web server scripting in Microsoft Internet Information Service” (Microsoft Corporation 2011, “What is VBScript?”).

VBScript is a scripting language with all advantages and disadvantages. Because it is only supported by the Microsoft Internet Explorer by default, its applicability for client-side scripting is limited.

Conclusion

Programming languages have huge differences among themselves. They can be classified among their computational model, hardware abstraction level and programming paradigms. Technically comparable distinctive features are basically the level of compilation/interpretation, storage management, type system, effectiveness and expressiveness. Other distinctive features include popularity and a lot of things associated with popularity, syntactic / semantic rules (which are only difficult to compare technically) and the intended use of a programming language.

Affordances for programming languages on the Web are different and limited. There has to be a distinction between server-side programming, client-side programming and web application programming:

- Server-side programming can be done with every programming language basically (usually using CGI), while specific “Module-/ Plugin-loading mechanisms” or special software solutions usually provide significant advantages.

- Client-side programming is limited to the capacity of important Web browsers, so the JavaScript and XSLT-standards as Java Applets are kind of outstanding here, as all other technologies must usually use browser Plugins.
- Web Application Programming can be done with every programming language to be compiled for the client system (but there are advantages/disadvantages to consider)

Especially web specific features, frameworks, solutions, techniques and technologies can improve a programming languages' qualification for the Web significantly.

As a result of the work we can assume, that programming languages for the Web 2011 are mostly scripting languages that have a tendency to high hardware abstraction. Based on their popularity, we figured out the following candidates to be the most important ones currently, according to statistics (note that the information in brackets only describes primary characteristics):

- Java (general purpose language, especially the applications of Java dedicated to Web programming, server-side / client-side)
- C# (general purpose language, especially the applications of C# dedicated to Web programming, server-side, proprietary)
- PHP (scripting language, server-side)
- Perl (scripting language, server-side)
- Python (partly scripting language, server-side)
- Ruby (scripting language, server-side)
- JavaScript (scripting language, client-side)

It stands out that C# is the only "mostly proprietary" language to be widely used for web programming purposes and that 5 out of 7 candidates are scripting languages.

There are some more programming languages, which are almost equally important, but have one of "not as good in statistics" or "primarily not intended for web programming", these are:

- CFML/CFScript for ColdFusion (scripting language, server-side, proprietary)
- ActionScript (scripting language, clident-side, proprietary)
- XSLT (transformation programming language, client-side)
- Objective-C (application programming language, client-side)
- C (system programming language, client-side / server-side)
- C++ (system programming language, client-side / server-side)
- Visual Basic (with descendant VBScript, client-side / server-side, proprietary)

There are a lot of domain specific languages which are not included in this comparison as they are no real programming languages; these are HTML, XML, CSS, and SQL in the first place.

References

List of literature

Bates, C. (2006). Web Programming. Building Internet Applications. Third Edition. West Sussex, UK: John Wiley & Sons Ltd.

Hauser, T. / Kappler, A. / Wenz, C.(2009). Das Praxisbuch ActionScript 3. Bonn, DE: Galileo Press.

Scott, M. L. (2009). Programming Language Pragmatics. Third Edition. Oxford, UK: Elsevier Inc.

Wenz, C. (2007). Ajax. schnell + kompakt. Second Edition. Software & Support Verlag GmbH.

Internet references

adobe.com (2011). “Products – ColdFusion”, [http://www.adobe.com/products/coldfusion/ accessed July 28th 2011]

adobe.com (2011, “ActionScript Technology Center”, [http://www.adobe.com/devnet/actionscript.html accessed July 28th 2011]

Amirrahimi, E. (February 1st 2011). “An Overview Of Important Web Programming Languages”, [http://ezinearticles.com/?An-Overview-Of-Important-Web-Programming-Languages&id=5788027 accessed July 26th 2011]

Apple Inc. (2011). “Cocoa”, [http://developer.apple.com/technologies/mac/cocoa.html accessed July 22nd]

Awio Web Services LLC (June 30th 2011). “Global Stats – June 2011”, [http://www.w3counter.com/globalstats.php?year=2011&month=6 accessed July 23rd 2011]

Bell (May 22nd 2009). “Is SQL or even TSQL Turing Complete?”, [http://stackoverflow.com/questions/900055/is-sql-or-even-tsql-turing-complete accessed July 25th 2011]

Busch, M. / Koch, N. (December 2009). “Rich Internet Applications - State-of-the-Art”, [http://uwe.pst.ifi.lmu.de/publications/maewa_rias_report.pdf accessed July 25th 2011]

Boender (February 12th 2009). “Apache, FastCGI and Python”, [http://www.electricmonk.nl/docs/apache_fastcgi_python/apache_fastcgi_python.html accessed July 24th 2011]

c2.com (no date). “Turing Complete”, [http://c2.com/cgi/wiki?TuringComplete accessed 14th July 2011]

Computerworld (March 28th 2005). “Computerworld Development Survey gives nod to C#”, [http://www.computerworld.com/s/article/100542/Computerworld_Development_Survey_gives_nod_to_C_?taxonomyId=011 accessed July 26th 2011]

cplusplus.com (2011). “A brief description”, [http://www.cplusplus.com/info/description/ accessed July 28th 2011]

DedaSys LLC (2011). “LangPop.com – Programming Language Popularity”, [http://langpop.com/ accessed July 14th 2011]

Deursen, A. (March 2nd 1998). “Domain Specific Languages: An Annotated Bibliography”, [http://homepages.cwi.nl/~arie/papers/dslbib/ accessed July 25th 2011]

Enticknap, N. (September 11th 2007). “SSL/Computer Weekly IT salary survey: finance boom drives IT job growth”, [http://www.computerweekly.com/Articles/2007/09/11/226631/SSLComputer-Weekly-IT-salary-survey-finance-boom-drives-IT-job.htm accessed on 13th July 2011]

Etemad, E. J. (May 12th 2011). “Cascading Style Sheets (CSS) Snapshot 2010 - W3C Working Group Note 12 May 2011”, [http://www.w3.org/TR/CSS/ accessed July 25th 2011]

Fowler, M. (no date). “Domain-specific Languages”, [http://martinfowler.com/dsl.html accessed July 25th 2011]

Friedl, J. E. F. (1997). “Mastering Regular Expressions”, [http://stiiceva.net/book/Mastering%20Regular%20Expressions%20-%20Powerful%20Techniques%20For%20Pearl%20And%20Other%20Tools%20%281997%29.pdf accessed July 25th 2011]

Gay, J. (no date). “The History of Flash”, [http://www.adobe.com/macromedia/events/john_gay/page04.html accessed July 25th 2011]

Gilbert, G. (May 5th 2008). “TIOBE: Coldfusion not a Programming Language”, [http://www.garyrgilbert.com/blog/index.cfm/2008/5/5/TIOBE-Coldfusion-not-a-Programming-Language accessed July 26th 2011]

Graham, Paul (May 2002). “Revenge of the Nerds”, [http://www.paulgraham.com/icad.html accessed July 15th 2011]

Gray, N. (2003). “Web Server Programming”, [http://www.mauriziocozzetto.net/pdf/Web_server_programming.pdf accessed July 24th 2011]

Grupo de Sistemas y Comunicaciones (2009). “SLOCCount Web for Debian Lenny - General Statistics for Debian Lenny Code Counting”, [http://libresoft.es/debian-counting/lenny/index.php?menu=Statistics accessed July 13th 2011]

Hall, M. (no date), “JavaServer Pages (JSP) 1.0”, [http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/Servlet-Tutorial-JSP.html accessed July 26th 2011]

Hofmann, Raithelhuber (October 1st 1998). “SGML/XML”), [http://www.th-o.de/sgml/sgmlv.htm#SGMLV.5 accessed July 25th 2011]

Horwith, S. (2007). “When and Why to use a framework”, [http://www.frameworksconference.com/pages/serveFile.cfm?file=HORWITH_when-whyframework.ppt accessed July 24th 2011]

Houghton Mifflin Harcourt Publishing Company (2010). “browser science definition”, [http://science.yourdictionary.com/browser accessed July 23rd 2011]

Indeed.com (July 26th 2011). “About”, [http://www.indeed.com/intl/en/about.html accessed July 26th 2011]

Indeed.com (July 26th 2011). “javascript, asp, perl, php, python, jsp, xslt, ruby, objective c, ActionScript, vbscript, coldfusion, lua Job Trends”, [http://www.indeed.com/jobtrends?q=javascript%2C+asp%2C+perl%2C+php%2C+python%2C+jsp%2C+xslt%2C+ruby%2C+objective+c%2C+ActionScript%2C+vbscript%2C+coldfusion%2C+lua&l= accessed July 26th 2011]

Indeed.com (July 26th 2011). “javascript, asp, perl, php, python, jsp, xslt, ruby, objective c, ActionScript, vbscript, coldfusion Job Trends”, [http://www.indeed.com/jobtrends?q=javascript%2C+asp%2C+perl%2C+php%2C+python%2C+jsp%2C+xslt%2C+ruby%2C+objective+c%2C+ActionScript%2C+vbscript%2C+coldfusion&l=&relative=1 accessed July 26th 2011]

Jobs, S. (October 2010). “Thoughts on Flash”, [http://www.apple.com/hotnews/thoughts-on-flash/ accessed July 13th 2011]

Kay, M. (February 1st 2001). “What kind of language is XSLT?”), [http://www.ibm.com/developerworks/xml/library/x-xslt/?article=xr accessed July 25th 2011]

King, P. J. B. (August 31st 1999). “High level Languages”, [http://www.macs.hw.ac.uk/~pjbk/pathways/cpp1/node22.html accessed July 16th 2011]

Korpela, J.Y. (June 26th 2010). “Getting Started with CGI Programming in C”, [http://www.cs.tut.fi/~jkorpela/forms/cgic.html accessed July 28th 2011]
Lie, H. W. (March 29th 2005). “Cascading Style Sheets”), [http://people.opera.com/howcome/2006/phd accessed July 25th 2011]
livdocs.adobe.com (no date). “About CFScript”, [http://livedocs.adobe.com/coldfusion/8/htmldocs/help.html?content=CFScript_01.html accesse July 28th 2011]
Mabutt, D. (no date), “What is Visual Basic?”, [http://visualbasic.about.com/od/applications/a/whatisvb.htm accessed July 28th 2011]
Maher, M. (1998). “An Analysis of Internet Standardization”, [http://www.vjolt.net/vol3/issue/vol3_art5.pdf accessed July 25th 2011]
MassLight, Inc. (2001). “Introduction to J2EE”, [http://j2ee.masslight.com/Chapter1.html accessed July 24th 2011]
McLaughlin, K. (August 11th 2008). “Apple's Jobs Gushes Over App Store Success”, [http://www.crn.com/blogs-op-ed/the-channel-wire/210002313/apples-jobs-gushes-over-app-store-success.htm accessed July 22th 2011]
meineipadresse.de (GEOTEK Systemhaus Berlin, July 7th 2011), “TCP/IP Ports”, [http://meineipadresse.de/html/ip-ports.php accessed July 13th 2011]
Microsoft Corporation (2011). “Directives for ASP.NET Web Pages”, [http://msdn.microsoft.com/en-us/library/t8syafc7.aspx accessed July 26th 2011]
Microsoft Corporation (2011). “Getting Started with Visual C#”, [http://msdn.microsoft.com/en-us/vcsharp/dd919145.aspx accessed July 28th 2011]
Microsoft Corporation (2011). “Microsoft Visual Studio”, [http://www.microsoft.com/germany/visualstudio/ accessed July 22nd 2011]
Microsoft Corporation (2011). “What is VBScript?”, [http://msdn.microsoft.com/en-us/library/1kw29xwf%28v=vs.85%29.aspx accessed July 28th 2011]
Miller, Scott Alan (February 18th 2010). “Trends in Thin Client Computing” [http://itmanagement.earthWeb.com/netsys/article.php/3865726/Trends-in-Thin-Client-Computing.htm accessed on July 13th 2011]
Morgan, N. (August 12th 2010). “stackoverflow.com – low, mid, high level language, what’s the difference“, [http://stackoverflow.com/questions/3468068/low-mid-high-level-language-whats-the-difference accessed July 15th 2011]
Mosses, P. D. (2001). “Programming Language Semantics (Summary)”, [http://lib.dnu.dp.ua:8001/l/%D0%9A%D0%BE%D0%BF%D1%8C%D1%8E%D1%82%D0

%B5%D1%80%D1%8B%D0%98%D1%81%D0%B5%D1%82%D0%B8/_Lecture%20Notes%20in%20Computer%20Science/computer%20science/Theoretical%20Computer%20Science%20Proc.conf.2000%20%28LNCS1872,%20Springer,%202000%29%28ISBN%2035406782 39%29%28O%29%28632s%29.pdf#page=626 accessed July 22nd 2011]

Netcraft (August 6th 2007). "August 2007 Web Server Survey", [http://news.netcraft.com/archives/2007/08/06/august_2007_Web_server_survey.html accessed July 24th 2011]

Netscape Communications Corporation (1998). "Server-Side JavaScript Guide v1.2", [http://218.111.200.82:8080/nihonsoft_org/javascript/ServerGuideJS12/index.htm accessed July 24th 2011]

no name (August 12th 2010). "stackoverflow.com – is C a middle-level language"), [http://stackoverflow.com/questions/3468068/low-mid-high-level-language-whats-the-difference?answertab=active accessed July 20th 2011]

Noack, G. (March 18th 2005). "Ausarbeitung : Objective C", [http://www.unix-ag.uni-kl.de/~guenther/objective-c/Ausarbeitung.pdf accessed July 22th 2011]

Novatchev, D. (Novomber 2001, "The Functional Programming Language XSLT - A proof through examples", [http://alamos.math.arizona.edu/~rychlik/CourseDir/589/Assignments/a3/fp.pdf accessed July 26th 2011]

openjs.com (no date, "XML Parser for JavaScript – xml2array()", [http://www.openjs.com/scripts/xml_parser/ accessed July 25th 2011]

oracle.com (no date). "JavaServer Pages Technology", [http://www.oracle.com/technetwork/java/javaee/jsp/index.html accessed July 28th 2011]

Ousterhout, J. K. (March 1998, p. 26), [http://www.stanford.edu/~ouster/cgi-bin/papers/scripting.pdf accessed July 21th 2011]

php.net (July 22nd 2011). "XML Parser", [http://php.net/manual/de/book.xml.php accessed July 24th 2011]

Paulley, G. (July 29th 2008), "SQL:2008 now an approved ISO International Standard", [http://iablog.sybase.com/paulley/2008/07/sql2008-now-an-approved-iso-international-standard/ accessed July 25th 2011]

php.net (2011). "What is PHP?", „Documentation", [http://php.net/ accessed July 28th 2011]

Podogoretsky (1997). “Common Gateway Interface (CGI)”, [http://www.podgoretsky.com/ftp/docs/internet/CGI%20Developer%27s%20Guide/ch1.htm accessed July 24th 2011]
python.org (2011). “About Python”, [http://www.python.org/about/ accessed July 28th 2011]
roseindia.net (no date). “Objective C Introduction”, [http://www.roseindia.net/iphone/objectivec/objective-c-introduction.shtml accessed July 28th 2011]
Rothberg, D. (September 15th 2006), [http://www.eweek.com/c/a/IT-Management/10-Programming-Languages-You-Should-Learn-Right-Now/ accessed July 26th 2011]
ruby-lang.org (2011). “Ruby is…”, [http://www.ruby-lang.org/en/ accessed July 28th 2011]
SELFHTML e.V. (2007). “Allgemeines zu Server Side Includes”, [http://de.selfhtml.org/servercgi/server/ssi.htm accessed July 24th 2011]
Rutenberg, G. (August 10th 2007). “Intruduction to C++ CGI”, [http://www.guyrutenberg.com/2007/08/10/introduction-to-c-cgi/ accessed July 28th 2011]
Spiewak, D. (February 27th 2008). “Defining High, Mid and Low-Level Languages”, [http://www.codecommit.com/blog/java/defining-high-mid-and-low-level-languages accessed July 15th 2011]
Stewart, R. (April 2nd 2007). “Desktop vs. Browser - when to deploy applications for each”, [http://www.zdnet.com/blog/stewart/desktop-vs-browser-when-to-deploy-applications-for-each/329 accessed July 25th 2011]
Taft, D. K. (June 21th 2010). “Application Development: Top 10 Programming Languages to Keep You Employed”, [http://www.eweek.com/c/a/Application-Development/Top-10-Programming-Languages-to-Keep-You-Employed-719257/ accessed July 26th 2011]
Tiobe Software BV (2011). “TIOBE Programming Community Index Definition”, [http://www.tiobe.com/index.php/content/paperinfo/tpci/tpci_definition.htm accessed on 14th July 2011]
Tratt, L. (July 2009). “Dynamically typed languages, Laurence Tratt, Advances in Computing, vol. 77, pages 149-184”, [http://tratt.net/laurie/research/publications/html/tratt__dynamically_typed_languages/ accessed July 21th 2011]
tutorialspoint.com (no date). “C – Basic Introduction”, [http://www.tutorialspoint.com/ansi_c/c_introduction.htm accessed July 28th 2011]

Unidex, Inc. (2008), “Universal Turing Machine in XSLT”, [http://www.unidex.com/turing/utm.htm accessed July 22th 2011]
University of Birmingham (no date). “Lecture 1: What are Programming Paradigms?”, [http://www.cs.bham.ac.uk/research/projects/poplog/paradigms_lectures/lecture1.html accessed July 16th 2011]
Vangie Beal (June 7th 2006). “The Differences Between Thick & Thin Client Hardware”, [http://www.Webopedia.com/DidYouKnow/Hardware_Software/2006/thin_client.asp accessed on 13th July 2011]
w3.org (W3C, no date),”W3C Mission”, [http://www.w3.org/Consortium/mission.html accessed on July 13th 2011]
w3.org (W3C, no date),”All standards and drafts”, [http://www.w3.org/TR/ accessed on July 13th 2011]
w3schools.com (no date). “XSLT Browsers”, [http://www.w3schools.com/xsl/xsl_browsers.asp accessed July 25th 2011]
w3schools.com (no date, “JavaScript Tutorial”, [http://www.w3schools.com/js/default.asp accessed July 28th 2011]
Welton, D. N. (July 18th 2005). “The Economics of Programming Languages”, [http://www.welton.it/articles/programming_language_economics.html accessed July 22nd 2011]
Whitacre, C. W. L. (September 12th 2009). “SQL is Turing-complete”, [http://blag.whit537.org/2009/09/sql-is-turing-complete.html accessed July 25th 2011]
Wimmer (2002). “SQL Tutorial”, [http://sql.idv.edu/ accessed July 25th 2011]
yourdictionary.com (2010). “SSJS computer definition”, [http://computer.yourdictionary.com/ssjs accessed July 28th 2011]
Zander (no date). “Language Paradigms”, [http://courses.washington.edu/css342/zander/css332/paradigm.html accessed July 16th 2011]