Abstract:

In this fast changing world, the software industry is flourishing day by day; and with these people prefer learning programming languages. But nowadays there are more than 300 programming languages which are available for people to learn, of which only a few of them are such that they can be applied comparatively widely. As languages increasingly support generics, it is important that language designers understand the features necessary to provide powerful generics and that their absence causes serious difficulties for programmers.

In this paper I present a comparative study on eight programming languages with respect to the following parameters like: Reusability, Portability, Reliability, Readability, Efficiency, Availability of compilers and tools, Familiarity, Expressiveness, Run-time checks and verbose code. Here we do the comparative study of - C, C++, C#, Java, PHP, ASP, JavaScript & VisualBasic. The technical features of these eight programming languages are summarized and compared with each other; thus the comparative study would help people to choose which language to select to learn for the purpose they want to complete. This experiment was done to make the work of a programmer easy by giving him the important information about the features necessary to provide powerful generics.

Keywords: generics, generic programming, C, C++, C#, Java, PHP, ASP, JavaScript, VisualBasic, Reusability, Portability, Reliability, Readability, Efficiency, Familiarity, Expressiveness, Run-time checks, verbose code

I. INTRODUCTION

The first high-level programming languages were designed during the 1950s. Ever since then, programming languages have been a fascinating and productive area of study. Programmers endlessly debate the relative merits of their favorite programming languages, sometimes with almost religious zeal. On a more academic level, computer scientists search for ways to design programming languages that combine expressive power with simplicity and efficiency.

The complexity of engineering software has increased dramatically in the past decade. In the early years most engineering applications were concerned solely with solving difficult numerical problems, and little attention was paid to man-machine interaction, data management, or integrated software systems. Now, computers are expected to solve a much wider variety of problems, particularly those in which numerical computations are less predominant. With the continuing increase in the variety, functionality, and complexity of engineering software, with its more widespread use, and with its increasing importance, more attention must be paid to programming language suitability so that rational decisions regarding language selection may be made. A programming language should also be reasonably natural for solving problems, at least problems within its intended application area. For example, a programming language whose only data types are numbers and arrays might be natural for solving numerical problems, but would be less natural for solving problems in commerce or artificial intelligence.

Conversely, a programming language whose only data types are strings and lists would be an unnatural choice for solving numerical problems.

II. CRITERIA FOR A GOOD LANGUAGE

To begin the language selection process, it is important to establish some criteria for what makes a language good. A good language choice should provide a path into the future in a number of important ways:

- Its definition should be independent of any particular hardware or operating system.
- Its definition should be standardized, and compiler implementations should comply with this standard.
- It should support software engineering technology, discouraging or prohibiting poor practices, and promoting or supporting maintenance activities.
- It should effectively support the application domain(s) of interest.
- It should support the required level of system reliability and safety.
• Its compiler implementations should be commensurate with the current state of technology.
• Appropriate software engineering-based supporting tools and environments should be available.

Effectively satisfying the above criteria is not easy, and it may require using different languages in different situations. However, as these points are violated, additional risk is involved with near-term development, as well as future technology changes. Key risks encountered as each of these criteria is violated are:
• If a language is not independent of a particular platform, portability is severely compromised. Hardware and software options are also limited, both for the original system and for future upgrades.
• If compiler implementations do not comply with a standard language definition, compiler-unique solutions are created. This also severely compromises portability, as well as options for future upgrades.
• To the extent that poor practices are used in software development, both development and debugging times will be extended, and poor code characteristics will make both testing and maintenance a nightmare.
• Poor support for the application domain will compromise the ease of development, as well as performance and readability characteristics of the code.
• If reliability is compromised, the system will not only perform below expectations, but it will also become much more costly across its lifetime. If safety is compromised, life and property will be endangered.
• An out-of-date compiler is inferior and difficult to use, producing substandard code which is difficult to create and maintain. It can also prohibit the use of key language features.
• The lack of appropriate automated development support compromises developer productivity and system quality.

III. CRITERIA OF LANGUAGE COMPARISON

We consider the following feature criteria:
• Reusability: Does the language support effective reuse of program units? If so, the project can be accelerated by reusing tried-and-tested program units; it might also develop new program units suitable for future reuse. Relevant concepts here are packages, abstract types, classes, and particularly generic units.
• Portability: Does the language help or hinder writing of portable code? In other words, can the code be moved from one platform to a dissimilar platform without major changes?
• Reliability: Is the language designed in such a way that programming errors can be detected and eliminated as quickly as possible? Errors detected by compile-time checks are guaranteed absent in the running program, which is ideal. Errors detected by run-time checks are guaranteed to cause no harm other than throwing an exception (or at worst terminating the program), which is second-best. Errors not detected at all can cause unlimited harm (such as corrupting data) before the program crashes. While reliability is always important, it is absolutely essential in safety-critical systems.
• Efficiency: Is the language capable of being implemented efficiently? Some aspects of object-oriented programming entail runtime overheads, such as class tags and dynamic dispatch. Run-time checks are costly (although some compilers are willing to suppress them, at the programmer’s own risk). Garbage collection is also costly, slowing the program down at unpredictable times. Interpretive code is about ten times slower than native machine code. If critical parts of the program must be highly efficient, does the language allow them to be tuned by resort to low-level coding, or by calls to procedures written in a lower-level language?
• Readability: Does the language help or hinder good programming practice? A language that enforces cryptic syntax, very short identifiers, default declarations, and an absence of type information makes it difficult to write readable code. The significant point is that code is read (by its author and other programmers) more often than it is written.
• Availability of compilers and tools: Are good-quality compilers available for the language? A good-quality compiler enforces the language’s syntax and type rules, generates correct and efficient object code, generates run-time checks (at least as an option) to trap any errors that cannot be detected at compiletime, and reports all errors clearly and accurately. Also, is a good-quality integrated development environment (IDE) available for the language? An IDE enhances productivity by combining a program editor, compiler, linker, debugger, and related tools into a single integrated system.
• Familiarity: Are the available programmers already familiar with the language? If not, is high-quality training available, and will the investment in training justify itself in future projects?
• Expressiveness: This factor reflects the ability of a language to express complex computations or complex data structures in appealing, intuitive ways.

IV. EVALUATION OF PROGRAMMING LANGUAGES

IV-A. C++

Its definition should be independent of any particular hardware or operating system: C++ is completely independent of any particular hardware or operating system.
• Its definition should be standardized, and compiler implementations should comply with this standard: C++ has followed the standardization path of most languages. First, the language was created and used. As its popularity grew, it began to spawn a number of different dialects. Then, C++ started a standardization process, with the main core of the language being standardized. Because C++ is a relatively new language, its standardization process is not yet complete. It is expected that the C++ standard will be used much as the C standard, and it will be common for compiler implementations to support standard C++ with additional, system-dependent features. This will continue to result in the creation of much non-standard


C++ code.

It should support software engineering technology, discouraging or prohibiting poor practices, and promoting or supporting maintenance activities.

Unlike C, the structures and object-oriented features of C++ provide support for the concepts of encapsulation and data abstraction.

It should effectively support the application domain(s) of interest.

As with C++, has proven to be a very versatile language, supporting any domain in which it has been tried.

It should support the required level of system reliability and safety.

Reliability is supported by the object-oriented features of C++. Safety-critical systems, those on which human life may depend, are not effectively supported by C++ because of its lack of support for software engineering technology in its C subset.

Its compiler implementations should be commensurate with the current state of technology.

Because of the immense popularity of C++, its compilers continue to be improved using current technology.

Appropriate software engineering-based supporting tools and environments should be available.

Again, because of its popularity, a wide variety of supporting tools and environments is available for C++ development.

Readability:

Although it is possible to write C++ code that is understandable, and the object-oriented nature of C++ is supported with understandable syntax, it is not common practice to use a verbose, understandable style for C++ any more than for C. C++ still provides the cryptic C shortcuts that run counter to clarity, and they are commonly used.

Maintainability:

When C++ is being used to create object-oriented code, the programmer has good object-oriented features to facilitate maintainability. However, the C problem of little inherent support for maintainability still remains in other C++ language features.

Mixed language support:

C++ will readily use object files produced by any language compiler as it composes an application. This is easy because C++ requires no consistency checking among these separate files. While that makes the object files easy to use, it does not provide specific support for properly interfacing the languages or for verifying correct exchange of data across the established interface. C++ improves on C with better language constructs for facilitating language interfacing.

Portability:

C++ does not yet have an existing standard, but, when it does, it will probably not alter the C characteristics in this respect. Common practice will not necessarily adhere to the standard. However, C++ does encourage the encapsulation of dependencies, a feature which facilitates portability. C++ tools and tool sets are also widely available on many platforms.

Reliability:

C++ improves considerably on the language characteristics of C for supporting reliability with features such as encapsulation, as well as improved expression.

Reusability:

Support for reusability requires support for code clarity, encapsulation, maintainability, and portability. C++ provides much more inherent support for these characteristics.

Safety:

C does not provide good support for any safety features. C++ improves somewhat on these characteristics of C, since it facilitates better expression of abstractions.

Standardization:

C++ is in the process of being standardized by both ANSI and ISO. However, once completed, there is no reasonable expectation that a C++ compiler will follow the standard without including additional features.

Support for modern engineering methods:

C++ was created to support object-oriented programming, which provides support for encapsulation and data abstraction. This makes its software engineering support rather one-dimensional, but still substantial.

IV-B. JAVA

Its definition should be independent of any particular hardware or operating system.

Java is completely independent of any particular hardware or operating system.

Its definition should be standardized, and compiler implementations should comply with this standard.

Java has had an unusual road to standardization. As with most languages, Java was developed and used before any attempt to get it officially standardized. However, many of the purported benefits of Java are lost unless a standard is strictly followed. Sun Microsystems, who developed Java, has dealt with this very effectively by making Java and its accompanying development tool kit freely available on the Web. Although Java has just recently begun the process of getting standardized through ISO [Sun 97], virtually all Java implementations follow the Sun de facto standard.

It should support software engineering technology, discouraging or prohibiting poor practices, and promoting or supporting maintenance activities.

Java was developed largely because C++ could not effectively meet this criterion. Hence, although Java uses C++ syntax and it is object oriented, it is not similar to C++ in other substantial ways. Java both discourages and prohibits poor practices.

Maintenance is supported by these characteristics.

It should effectively support the application domain(s) of interest.

Java was developed specifically to support WWW applications. However, it is also a general-purpose language. Although it is still a very young language, it has proven to provide good support for any domain in which it has been tried.

It should support the required level of system reliability and safety.

Java provides many features which support system reliability. For safety-critical systems, those on which human life may
depend, no current language is entirely satisfactory. Java has not been around long enough to be studied for suitability for safety-critical systems. However, since it does not provide formal analysis features, at best it would require a combination with mathematical specification, rigorous analysis, and formal proofs of correctness before it could be appropriate for use in safety-critical systems.

Its compiler implementations should be commensurate with the current state of technology.

Java is very sophisticated for such a new language, and it is driving some of the current technology trends. Its Sun compiler implementation is commensurate with current technology.

Appropriate software engineering-based supporting tools and environments should be available.

Current Java tool kits contain primarily tools to support code creation, although some software engineering-based tool kits are beginning to appear.

Readability:
Java is strictly object oriented, so its form is very well defined. The code suffers somewhat from the cryptic C syntax forms. C

Maintainability:
Many features of Java support maintainability, such as those which support code clarity, encapsulation, and object orientation. Object-oriented capabilities can have both good and bad effects on maintainability, but, if used properly, object-oriented programming will improve maintainability.

Mixed language support:
Java provides for interfacing with other languages by providing wrappers around the code from the other languages.

Portability:
Java was built for complete portability. Its compiler produces source code in a platform-independent bytecode. The bytecode is then translated at runtime into native machine code for the given platform.

Reliability:
Java requires the specification of information, the omission of which can make a program unreliable, such as type specifications.

Reusability:
Java supports reusability with language features supporting code clarity (making code understandable), encapsulation (making code adaptable), maintainability, and portability.

Safety:
Java was not developed for safety-critical systems, and its capabilities in that area are unproven.

Standardization:
Java is in the process of ISO standardization. Nevertheless, it is currently a very effective de facto standard, and it is reasonable to expect implementations to follow the standard.

Support for modern engineering methods:
Java was developed explicitly to support many software engineering principles, including its support for reliability, maintainability, and portability.

IV-C. C

Its definition should be independent of any particular hardware or operating system:
C was originally defined as a language for system software on Unix platforms. However, it has evolved to a language independent of Unix or any specific platform.

Its definition should be standardized, and compiler implementations should comply with this standard:
C has followed the standardization path of most languages. First, the language was created and used. As its popularity grew, it began to spawn a number of different dialects. Then, C went through a standardization process, with the main core of the language being standardized. It is common for compiler implementations to support standard C with additional, system-dependent features. This results in the creation of much non-standard C code.

It should support software engineering technology, discouraging or prohibiting poor practices, and promoting or supporting maintenance activities:
Because of its original purpose, the easy creation of system software, C has never emphasized support for software engineering technology. As with any language, it is possible to use good software engineering practices with C. However, the language neither prevents nor discourages poor practice, and it, therefore, does not provide good support for maintenance.

It should effectively support the application domain(s) of interest:
Although it was originally developed to support system software, C has proven to be a very versatile language, supporting any domain in which it has been tried.

It should support the required level of system reliability and safety:
Because of its lack of support for software engineering technology, C provides little support for reliability. Safety-critical systems, those on which human life may depend, are also not effectively supported by C because of its lack of support for software engineering technology.

Its compiler implementations should be commensurate with the current state of technology:
Because of the language's immense popularity, C compilers continue to be improved using current technology.

Appropriate software engineering-based supporting tools and environments should be available:
Again because of its popularity, a wide variety of supporting tools and environments is available for C development. Many of these are little more than tools to support code creation, with minimal support for engineering software. However, many also provide appropriate software engineering-based support.

Readability:
Although it is possible to write C code which is understandable, it is not common practice to use a verbose, understandable style. C provides cryptic shortcuts that run counter to clarity, and they are commonly used

Maintainability:
A C programmer must work very carefully to write maintainable code because the language provides little inherent
support.

Mixed language support:
C will readily use object files produced by any language compiler as it composes an application. This is easy because C requires no consistency checking among these separate files. While that makes the object files easy to use, it does not provide specific support for properly interfacing the languages or for verifying correct exchange of data across the established interface.

Portability:
The existence of a standard for C makes portability possible. However, common practice does not necessarily adhere to this standard. There are also no inherent language features that facilitate portability, such as the encapsulation of dependencies. The tremendous popularity of C has spawned tools and tool sets that are widely available on many platforms, enhancing portability.

Safety:
C provides little in the way of inherent language features to support reliability. It readily allows inconsistencies to show up in compiled code.

Reusability:
Support for reusability requires support for code clarity, encapsulation, maintainability, and portability. C provides little inherent support for any of these characteristics. Hence, it does not support development reuse on a large scale. On the other hand, reuse of specialized C libraries, such as graphics libraries, is very effective.

Standardization:
C is standardized by both ANSI and ISO. However, there is no reasonable expectation that a C compiler will follow the standard without including additional features.

Support for modern engineering methods:
C was not created with support for software engineering in mind, and it provides little inherent support for modern engineering methods.

IV-D. C#

Its definition should be independent of any particular hardware or operating system.:
C# is an object-oriented programming language from Microsoft that aims to combine the computing power of C++ with the programming ease of Visual Basic. C# is based on C++ and contains features similar to those of Java. It requires .Net platform.

Its definition should be standardized, and compiler implementations should comply with this standard.:
Microsoft is collaborating with ECMA, the international standards body, to create a standard for C#. International Standards Organization (ISO) recognition for C# would encourage other companies to develop their own versions of the language.

It should support software engineering technology, discouraging or prohibiting poor practices, and promoting or supporting maintenance activities.:
C# is an object-oriented programming language from Microsoft. It combines the computing power of C++ with the programming ease of Visual Basic. C# is based on C++ and contains features similar to those of Java.

It should effectively support the application domain(s) of interest.:
C# is designed to work with Microsoft's .Net platform. Microsoft's aim is to facilitate the exchange of information and services over the Web, and to enable developers to build highly portable applications.

It should support the required level of system reliability and safety.:
C# is based on .Net Technology. Microsoft addressed and cleared many reliability and stability issues with .NET technology, and now this technology is reliable. With the managed code in C#, the CLR can perform checks for type safety, memory overwrites, memory management and garbage collection. This results in a reduction of memory leaks and related issues. Type safety is another feature that promotes robust, safe programs. Microsoft incorporated several features to promote proper code execution in C#.

Its compiler implementations should be commensurate with the current state of technology.:
C# is very sophisticated for such a new language, and it is driving some of the current technology trends. Its Microsoft compiler implementation is commensurate with current technology.

Appropriate software engineering-based supporting tools and environments should be available.:
Current Visual Studio tool kits contain primarily tools to support code creation, although some software engineering-based tool kits are beginning to appear.

Readability:
C# is strictly object oriented, so its form is very well defined. And also is backed by Microsoft's well structured code editor.

Maintainability:
Many features of C# support maintainability, such as those which support code clarity, encapsulation, and object orientation. Object-oriented capabilities can have both good and bad effects on maintainability, but, if used properly, object-oriented programming will improve maintainability.

Mixed language support:
C# provides for interfacing with other languages to some extent and always requires .Net platform.

Portability:
As such in its defacto standard C# is not portable and always requires .Net and Microsoft platform. But with the help of Mono project one port it to Linux or Unix in restricted manner.

Reliability:
C# is based on .Net Technology. Microsoft addressed and cleared many reliability and stability issues with .NET technology, and now this technology is reliable.

Reusability:
C# simplifies programming through its use of Extensible
Mark up Language (XML) and Simple Object Access Protocol (SOAP) which allow access to a programming object or method without requiring the programmer to write additional code for each step. Because programmers can build on existing code, rather than repeatedly duplicating it, C# is expected to make it faster and less expensive to get new products and services to market.

Safety:
With the managed code in C#, the CLR can perform checks for type safety, memory overwrites, memory management and garbage collection. This results in a reduction of memory leaks and related issues. Type safety is another feature that promotes robust, safe programs. Microsoft incorporated several features to promote proper code execution in C#.

Standardization:
Microsoft is collaborating with ECMA, the international standards body, to create a standard for C#. International Standards Organization (ISO) recognition for C# would encourage other companies to develop their own versions of the language.

Support for modern engineering methods:
C# is developed explicitly to support many software engineering principles, including its support for reliability and maintainability.

IV-E. PHP

Its definition should be independent of any particular hardware or operating system:
PHP is completely independent of any particular hardware or operating system.

Its definition should be standardized, and compiler implementations should comply with this standard:
PHP: Hypertext Preprocessor (a recursive acronym, originally personal home page) is a general-purpose scripting language that was originally designed for web development to produce dynamic web pages. For this purpose, PHP code is embedded into the HTML source document and interpreted by a web server with a PHP processor module, which generates the web page document. As a general-purpose programming language, PHP code is processed by an interpreter application in command-line mode performing desired operating system operations and producing program output on its standard output channel. It may also function as a graphical application. PHP is available as a processor for most modern web servers and as a standalone interpreter on most operating systems and computing platforms.

It should support software engineering technology, discouraging or prohibiting poor practices, and promoting or supporting maintenance activities:
PHP has improved support for object-oriented programming, the PHP Data Objects (PDO) extension (which defines a lightweight and consistent interface for accessing databases), and numerous performance enhancements. PHP currently does not have native support for Unicode or multibyte strings; Unicode support is under development for a future version of PHP and will allow strings as well as class, method, and function names to contain non-ASCII characters. PHP interpreters are available on both 32-bit and 64-bit operating systems, but on Microsoft Windows the only official distribution is a 32-bit implementation, requiring Windows 32-bit compatibility mode while using Internet Information Services (IIS) on a 64-bit Windows platform. As of PHP 5.3.0, experimental 64-bit versions are available for MS Windows.

It should effectively support the application domain(s) of interest:
PHP is an open-source server-side scripting environment that you can use to create and run dynamic, interactive Web server applications. With PHP, you can combine HTML pages, script commands, and components to create interactive Web pages and powerful Web-based applications that are easy to develop and modify.

It should support the required level of system reliability and safety:
PHP improves considerably on the language characteristics for supporting reliability with features such as encapsulation, as well as improved expression. Hosting PHP applications on a server requires careful and constant attention to deal with security risks. There are advanced protection patches such as Suhosin and Hardening-Patch, especially designed for web hosting environments.

Its compiler implementations should be commensurate with the current state of technology:
The PHP language was originally implemented using a PHP interpreter. Several compilers now exist, which decouple the PHP language from the interpreter.

Appropriate software engineering-based supporting tools and environments should be available:
A wide variety of supporting tools, environments, and frameworks is available for PHP development.

Readability:
PHP is strictly object oriented, so its form is very well defined. The code suffers somewhat from the cryptic C syntax forms.

Maintainability:
Many features of PHP support maintainability, such as those which support code clarity, encapsulation, and object orientation. Object-oriented capabilities can have both good and bad effects on maintainability, but, if used properly, object-oriented programming will improve maintainability.

Mixed language support:
PHP allows developers to write extensions in C to add functionality to the PHP language. These can then be compiled into PHP or loaded dynamically at runtime. Extensions have been written to add support for the Windows API, process management on Unix-like operating systems, multibyte strings (Unicode), cURL, and several popular compression formats. Some more unusual features include integration with Internet Relay Chat, dynamic generation of images and Adobe Flash content, and even speech synthesis.

Portability:
PHP was built for complete portability. It is a server side scripting language. PHP is a flexible and portable language and applications programmed in PHP are easy to be implemented or
ported on many operating systems,

Reliability:
PHP improves considerably on the language characteristics for supporting reliability with features such as encapsulation, as well as improved expression.

Reusability:
PHP supports reusability with language features supporting code clarity, encapsulation, maintainability, and portability.

Safety:
Hosting PHP applications on a server requires careful and constant attention to deal with security risks. There are advanced protection patches such as Suhosin and Hardening-Patch, especially designed for web hosting environments.

Standardization:
PHP is in the process of ISO standardization. Nevertheless, it is currently a very effective de facto standard, and it is reasonable to expect implementations to follow the standard.

Support for modern engineering methods:
PHP is developed to support many software engineering principles, including its support for reliability, maintainability, and portability.

IV-F. ASP
Its definition should be independent of any particular hardware or operating system:
ASP is a technology created by Microsoft and is available mainly on Windows servers only.

Its definition should be standardized, and compiler implementations should comply with this standard:
Active Server Pages (ASP), also known as Classic ASP or ASP Classic, was Microsoft's first server-side script-engine for dynamically-generated web pages. Initially released as an add-on to Internet Information Services (IIS) via the Windows NT 4.0 Option Pack, it was subsequently included as a free component of Windows Server (since the initial release of Windows 2000 Server). ASP.NET has superseded ASP. It is a server-side scripting language designed for web development to produce dynamic web pages. For this purpose, ASP code is embedded into the HTML source document and interpreted by a web server (IIS server), which generates the web page document.

It should support software engineering technology, discouraging or prohibiting poor practices, and promoting or supporting maintenance activities:
ASP provided six built-in objects: Application, ASPError, Request, Response, Server, and Session. Session, for example, represents a cookie-based session that maintains the state of Request, Response, Server, and Session. Session, for example, represents a cookie-based session that maintains the state of Request, Response, Server, and Session. ASP has configuration options that allow you to trade off reliability for performance. You should understand these trade-offs when building and deploying your application. Hosting ASP applications on a server requires careful and constant attention to deal with security risks.

Its compiler implementations should be commensurate with the current state of technology:
Active Server Pages (ASP) technology provides the environment in which you can run scripts on the server. If you can write HTML, you can create an ASP file. It's really that easy. Your ASP file is simply a file that can contain any combination of HTML, scripting, and calls to components. When you make a change to the ASP file on the server, you need only save the changes to the file—the next time the Web page is loaded, the script will automatically be compiled. Of course, to take advantage of ASP, you'll want to add features the script that will run on the server, such as database access or other network-related work.

Appropriate software engineering-based supporting tools and environments should be available:
Current Visual Studio tool kits contain primarily tools to support code creation, although some software engineering-based tool kits are beginning to appear.

Readability:
ASP is strictly object oriented, so its form is very well defined. And also is backed by Microsoft's well structured code editor.

Maintainability:
Many features of ASP support maintainability, such as those which support code clarity, encapsulation, and object orientation. Object-oriented capabilities can have both good and bad effects on maintainability, but, if used properly, object-oriented programming will improve maintainability.

Mixed language support:
ASP provides for interfacing with other languages to some extent on Windows platform.

Portability:
ASP is a technology created by Microsoft and is available mainly on Windows servers only.

Reliability:
ASP has configuration options that allow you to trade off reliability for performance. You should understand these trade-offs when building and deploying your application.

Reusability:
When you identify and build common, reusable routines, particularly code that handles database operations, you can drastically simplify your ASP scripts by compiling the common routine into a Component Object Model (COM) dynamic-link library (DLL) using Microsoft Visual Basic or Microsoft Visual C++. Then, you can create objects from the compiled component and call methods from your ASP page to accomplish with two or three lines of ASP script what may otherwise take 20 to 100 lines
IV-G. VisualBasic

Its definition should be independent of any particular hardware or operating system:

VisualBasic is a technology created by Microsoft and is available mainly on Windows platform only.

Its definition should be standardized, and compiler implementations should comply with this standard:

A programming language and environment developed by Microsoft. Based on the BASIC language, Visual Basic was one of the first products to provide a graphical programming environment and a paint metaphor for developing user interfaces. Instead of worrying about syntax details, the Visual Basic programmer can add a substantial amount of code simply by dragging and dropping controls, such as buttons and dialog boxes, and then defining their appearance and behavior.

It should support software engineering technology, discouraging or prohibiting poor practices, and promoting or supporting maintenance activities:

VB has Poor support for object-oriented programming, inability to create multi-threaded applications, without resorting to Windows API calls, inability to create Windows services, and portability. VB improves considerably on the language characteristics for supporting reliability with features such as encapsulation, as well as improved expression. Visual Basic is designed to enable rapid program development, while not compromising reliability.

Reusability:

When you identify and build common, reusable routines, particularly code that handles database operations, you can drastically simplify your VB scripts by compiling the common routine into a Component Object Model (COM) dynamic-link library (DLL) using Microsoft Visual Basic or Microsoft Visual C++. Then, you can create objects from the compiled component and call methods from your VB source to accomplish with two or three lines of VB Code what may otherwise take 20 to 100 lines or more.

Safety:

So far VB compiled code is secure and safe.

Standardization:

There was a standard for Basic that was published in the late 80s or early 90s. It was probably ISO/IEC 10279:1991. But there is no standard for VisualBasic. It is as defined as by Microsoft.

Support for modern engineering methods:

VisualBasic was developed to support many software engineering principles, including its support for reliability, maintainability, and portability. But now VB.NET has superseded VB.

IV-H. Javascript

Its definition should be independent of any particular hardware or operating system:

Javascript is completely independent of any particular hardware or operating system.
hardware or operating system. It is a client side scripting
language

Its definition should be standardized, and compiler
implementations should comply with this standard.

JavaScript is a scripting language (or interpreted language),
which is not as fast as compiled languages but easier to learn and
use. It is only loosely related to Java, and is not a true object-
oriented language. JavaScript can be quickly added to a pure
HTML page to provide dynamic features, such as automatically
calculating the current date or activating an action. The
JavaScript code must be interpreted and executed by a browser as
it reads the Web page or by a Web server before it delivers the
page to the browser.

It should support software engineering technology,
discouraging or prohibiting poor practices, and promoting or
supporting maintenance activities.

Because JavaScript is the only language that the most popular
browsers share support for, it has become a target language for
many frameworks in other languages, even though JavaScript
was never intended to be such a language. Despite the
performance limitations inherent to its dynamic nature, the
increasing speed of JavaScript engines has made the language a
surprisingly feasible compilation target.

It should effectively support the application domain(s) of
interest.

JavaScript is a client-side scripting environment that you can
use to create and run dynamic, interactive Web applications. With
JavaScript, you can combine HTML code and script commands
together to create interactive Web pages and powerful Web-based
applications that are easy to develop and modify.

It should support the required level of system reliability and
safety.

JavaScript and the DOM provide the potential for malicious
authors to deliver scripts to run on a client computer via the web.

Its compiler implementations should be commensurate with
the current state of technology.

A JavaScript engine (also known as JavaScript interpreter or
JavaScript implementation) is an interpreter that interprets
JavaScript source code and executes the script accordingly in
web browser. A web browser is by far the most common host
environment for JavaScript.

Appropriate software engineering-based supporting tools and
environments should be available.

A wide variety of supporting tools, environments, and libraries
is available for JavaScript development.

Readability:

Although it is possible to write Javascript code which is
understandable, it is not common practice to use a verbose,
understandable style. Javascript provides cryptic shortcuts that
run counter to clarity, and they are commonly used

Maintainability:

Many features of Javascript support maintainability, such as
those which support code clarity, encapsulation, and object
orientation. Object-oriented capabilities can have both good and
bad effects on maintainability, but, if used properly, object-
oriented programming will improve maintainability.

Mixed language support:

Mixed language support is not available in Javascript.

Portability:

Javascript was built for complete portability. It is a client side
scripting language. It gets input from user and gives result to
client browser and it works for any OS for any browser where
HTTP protocol works.

Reliability:

JavaScript improves considerably on the language
characteristics for supporting reliability with features such as
encapsulation, as well as improved expression.

Reusability:

Javascript supports reusability with language features
supporting code clarity, encapsulation, maintainability, and
portability.

Safety:

JavaScript and the DOM provide the potential for malicious
authors to deliver scripts to run on a client computer via the web.

Standardization:

Javascript also known as ECMAScript is the scripting
language standardized by Ecma International in the ECMA-262
specification and ISO/IEC 16262.

Support for modern engineering methods:

Javascript supports many modern software engineering
principles.

V. CONCLUSION

PHP with proper use of Javascript is the appropriate option for
Web applications as well as desktop application using localhost
option.