ISSN: 2473-4901
v11 n6451

General Programming Languages for Information Systems (IS) Students

Euijin Kim
e.kim@moreheadstate.edu

Morehead State University
Morehead, KY 40351

Abstract

This study presents a systematic approach for selecting the most suitable programming languages for information systems (IS) major students. A review of past practice reveals that programming languages for IS students have often been chosen arbitrarily or based on popularity, typically adopted from other disciplines such as computer science without regard to their specific needs. This study introduces a comprehensive framework that enables educators and practitioners to make informed decisions when selecting programming languages for IS education.

Keywords: Programming languages, information systems, compilers, interpreters, hybrid programming languages, adaptive compilers.

ISSN: 2473-4901
v11 n6451

General Programming Languages for Information Systems (IS) Students

Euijin Kim

1. INTRODUCTION

Programming languages are a vital component of information systems (IS) education because they enable students to better understand computers, leading to more efficient use. Moreover, possessing programming skills can enhance job opportunities in areas such as web development, mobile programming, database management, and business analytics.

In information systems education, however, programming has sometimes been overlooked. For instance, the 2010 Curriculum Guidelines for Undergraduate Degree Programs in Information Systems (Topi et al., 2010) relegated programming to a secondary capability, making it an elective course, Application Development. This approach was contested by proponents of programming as a core course for IS students (Janicki & Cummings, 2022). Fortunately, the 2020 curriculum guidelines (Leidig & Salmela, 2020) recognized the importance of programming and reinstated it as a core course, Application Development/Programming. This shift is justified because programming knowledge and skills will benefit IS students in various ways, including helping them understand computers better, thereby enabling them to use computers more efficiently.

A subsequent important question is: what programming languages should be taught to IS students? While a few studies have investigated programming language choice in the IS field (e.g., Parker et al., 2006), most research has focused on which programming languages are used in IS education (e.g., Smith & Jones, 2021). It is time to consider programming language choice for IS students in a more systematic manner.

2. LITERATURE REVIEW

Computer science has a long history of considering the selection of programming languages, predating other fields (e.g., Tarp, 1982). The selection process is, however, more focused on efficiency in programming languages.

In IS education, research on selecting programming languages is either scarce or passive. One notable exception is Parker et al.'s (2006) study, which identified 23 selection criteria based on a literature review and later condensed them into 11 higher-order selection criteria. While their work provides valuable insights, some of these criteria may be either irrelevant or one-dimensional for today's environment. For instance, software cost is now a relatively minor concern since most programming language tools are available free of charge (e.g., Java). The items listed under these criteria are presented in a one-dimensional manner, assuming they serve as direct predictors of the selection process. Unfortunately, this oversimplification seems problematic, as it appears to overlook the complexity involved in the selection process.

Research specifically focused on information systems is also passive. For instance, Smith and Jones' (2021) survey summarized the programming languages used among US colleges for IS or business analytics courses. They found that Python, Java, and Visual Basic were the most popular choices. Notably, these programming languages seem to have been chosen passively without careful consideration by many US colleges. The study also underscores the need for further exploration into programming languages suitable for IS students.

Considering these problems, this proposal presents a new framework for selecting appropriate programming languages for IS major students. It begins by summarizing the fundamentals of information systems and employs a systematic framework to help identify the most suitable programming languages for IS students.

3. INFORMATION SYSTEMS

Information Systems (IS) leverages information technologies to support business processes and decision-making, encompassing various business functions such as production and operations management, marketing, finance, human resource management, and accounting (Laudon & Laudon, 2020). These functions may involve sub-processes, such as production and operations

management containing supply chain management, or be integrated into higher-level processes, including strategic management that encompasses finance, human resource management, and other relevant functions. This complexity makes business processes more intricate. Information systems play a crucial role in integrating these business functions to facilitate more efficient processing, thereby generating business value.

These unique features of information systems are distinct from those found in other related disciplines, such as computer science, computer engineering, and information technology. For instance, computer science or software engineering emphasizes software optimization, which demands strong mathematical and logical foundations. Computer engineering focuses on hardware optimization, often involving software development, whereas information technology is more focused on software/hardware installation and maintenance.

In contrast, the domain of information systems revolves around business value, necessitating distinct processes that integrate technological and organizational aspects. In other words, the uniqueness of IS lies in its focus on business, whereas other disciplines prioritize technology. These differences are summarized in Figure 1, which highlights the distinct nature of information systems' focus on business value. To generate business value, information systems undergo frequent changes, influencing the choice of programming languages for IS development and maintenance.

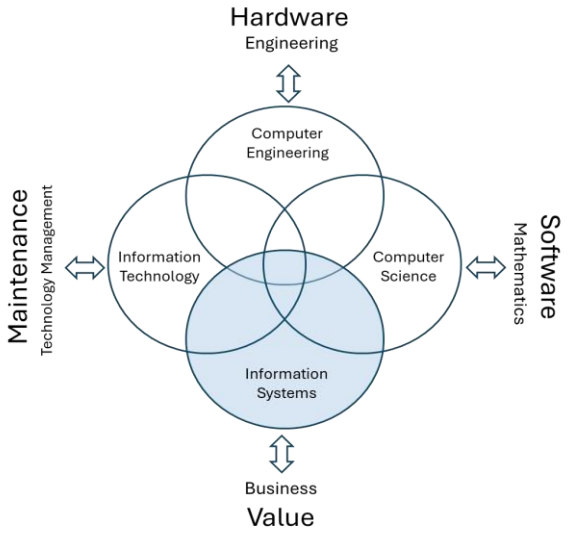

Figure 1: Information Systems and Other Related Disciplines

With the unique features of IS distinguished from those of other disciplines, we will now review relevant factors in selecting programming languages suitable for IS major students.

4. FACTORS FOR SELECTING PROGRAMMING LANGUAGES

To select a general programming language for IS major students, two primary sets of factors can be considered: intrinsic characteristics and external forces.

The intrinsic characteristics encompass technical features, practical usability, and pedagogical value. To put it simply, a programming language must possess these three key intrinsic qualities to be beneficial for IS major students. These characteristics are supported by the Technology Acceptance Model (TAM), proposed by Davis (1989), which suggests that users will accept a technology if it is both useful and easy to use. In our model, the decision-maker is not the user (learner) but rather the educator, who evaluates the technology's usefulness and ease of use in terms of its technical features, practical usability, and pedagogical value. In other words, educators can select programming languages that are suitable for learners' needs and easy enough for them to learn.

The external factors – designed purpose and industry demand – can also be supported by the usefulness construct of TAM. They are separated from the intrinsic characteristics because they are external forces rather than intrinsic characteristics. For instance, when selecting a programming language for web development, one might consider options such as standard tools (HTML, CSS, JavaScript), React Native, or Dart/Flutter. Once these options are identified, the core intrinsic factors can be evaluated to eventually select the final technology.

Industry demand is also processed in a similar manner. By evaluating industry trends and demands, educators can refine their selection of programming languages to ensure that students are equipped with relevant skills for the job market. In this process, the core factors (technical features, practical usability, and pedagogical value) continue to play the primary role.

The external factors are also supported by the Task-Technology Fit model (Goodhue & Thompson, 1995). According to the model, when technology fits a task, performance improves. Designed purpose and industry demand are

ISSN: 2473-4901
v11 n6451

related to tasks which are likely to improve performance. In other words, if a programming language fits a particular purpose or industry demand, learners are likely to improve their performance. Figure 2 demonstrates the factors affecting the selection.

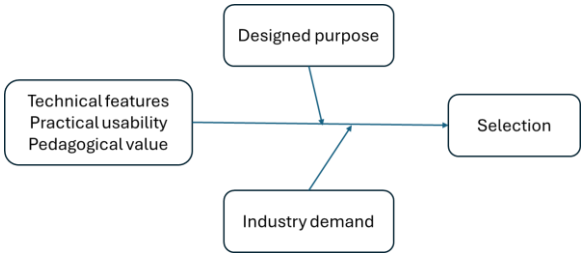

Figure 2: Selection of General Programming Languages for IS Students

Using this model, several programming languages for IS major students will be reviewed in the next section.

5. REVIEW OF PROGRAMMING LANGUAGES FOR IS EDUCATION

In this section, we will apply the framework to review several programming languages that are relevant to IS major students.

Technical Features of Programming languages

Programming languages can be categorized based on their technical features. Specifically, they can be classified into four categories: compilers, interpreters, hybrids (which combine elements of both compilers and interpreters), and adaptive compilers that employ just-in-time and ahead-of-time compilation.

C is a classic example of a compiler-based programming language (Kernighan & Ritchie, 1988). It requires source code files to be compiled into binary code files before distribution (refer to Figure A1, Appendix A). Compiler-based programs are generally faster and more efficient than other types of programs, making them suitable for developing system software, such as operating systems. However, these languages are less suitable for business applications that require frequent changes, as modifying the source code or developing new ones can be time-consuming and resource-wasting due to the cost of recompilation.

Interpreter-based programming languages offer a different approach. HTML, CSS, and JavaScript are examples of interpreter-based programming languages, widely used for web application development (W3C, n.d.). These languages enable developers to create source code files that can run across various environments as long as an interpreter (i.e., web browser) is available. The process involves interpreting the source code into machine code at runtime (refer to Figure A2, Appendix A). Applications based on interpreter-based programming languages offer convenience and flexibility but are generally considered slower than those developed using compiler-based programming languages due to the translation process that occurs when the applications are running.

Hybrid programming languages combine the benefits of both approaches (refer to Figure A3, Appendix A). For instance, Java (Oracle, n.d.) enables developers to create source code files once, compile them into intermediate bytecode files, and then distribute these bytecode files for users to process using an interpreter, such as the Java Virtual Machine (JVM) or Java Runtime Environment (JRE). This hybrid approach offers a balance between the convenience of interpreters and the efficiency of compilers, making it well-suited for business applications. The popularity of Java has led to its adoption in various industries, including database management systems (ABC News, 2009) and Android application development (Google, n.d.). However, hybrid programming languages do require the installation of an interpreter on user computers, which can be a burden to users.

To address the issue of requiring interpreters at the user side, adaptive compilation has emerged as a new type of programming environment. With adaptive compilation, developers create and test source code files in a just-in-time environment using just-in-time compilation. When it's time for distribution, the source code file is compiled ahead-of-time for different operating systems, and the appropriate binary code files are distributed (refer to Figure A3, Appendix A). This adaptive compilation approach combines the benefits of interpreter-based programming languages (convenience and flexibility) with the efficiency of compilers. Moreover, users do not need any additional components (e.g., interpreters) to run the programs.

All types of programming languages have relevant applications, but adaptive compilation is one of the most advanced and comprehensive technologies in programming to date. This development will have a significant impact on the selection of programming languages for IS

students, enabling them to directly benefit from the latest technologies. In essence, learners will derive advantages from modern technology that promotes flexibility, efficiency, and user convenience.

Practical Usability

Practical usability in programming languages refers to the convenience of developing, distributing, and running source code files. As reviewed earlier, interpreter-based programming offers this convenience by allowing developers to create source code files once and distribute them across different environments (e.g., Windows, macOS, Linux, etc.). Hybrid programming compiles source code files into intermediary files, such as bytecode, which are then distributed and interpreted by users on various computing environments. In contrast, adaptive compilation quickly compiles source code files for just-in-time testing, and then compiles these files again ahead of time for distribution across different environments. This approach is particularly convenient because it eliminates the need for users to install additional components or interpreters.

Practical usability also involves multi-platform support, enabling developers to create applications that can be used on multiple platforms without significant modifications or additional software requirements. For instance, Dart/Flutter allows developers to create mobile apps, desktop applications, and web applications with minimal adjustments needed (Flutter, n.d.).

While other programming languages can accomplish similar tasks, they often come with limitations. For example, C++ can be used for multi-platform development, but this requires additional components or adjustments. Similarly, JavaScript can be used for cross-platform application development using external frameworks like React Native; C# with .NET (Microsoft, n.d.) enables application development for various platforms, including Android, the web, and Windows, although these approaches are not integrated into the core language.

What sets Dart/Flutter apart from other programming languages is its comprehensive development environment, which integrates all tools around the Dart programming language. This unique combination makes it a more practical choice for developers seeking to create cross-platform applications that can be easily deployed across different environments, leveraging Flutter's professional-grade GUI development framework (Flutter, n.d.).

Pedagogical Value

Programming languages have undergone significant evolution, transitioning from non-structured to structured and ultimately, to object-oriented. Today, most modern programming languages are object-oriented (e.g., Java, C#, C++, Python, Dart/Flutter, etc.). Some programming languages offer basic features of object-oriented programming, while others provide more comprehensive support.

For instance, JavaScript enables the convenient creation of objects; however, this process differs from that found in more formal object-oriented programming languages, such as Java. This divergence is due to JavaScript's focus on efficient coding (e.g., functional programming). While JavaScript's object-oriented features can be beneficial for students to learn, it may not provide a comprehensive understanding of object-oriented programming principles.

In contrast, standard object-oriented languages like C++, Java, and C# offer more formal and systematic approaches to object-oriented programming. These languages integrate concepts such as encapsulation, inheritance, and polymorphism, which can be complex for learners to grasp. However, these complexities can also provide a deeper understanding of object-oriented programming principles.

Dart/Flutter provides the basics of standard object-oriented programming, simplifying these concepts into more practical and simpler approaches that are suitable for introductory general programming language courses. For beginners, Dart/Flutter offers a solid foundation and helps build a strong understanding of object-oriented programming principles.

For advanced learners, Java, C++, or C# can be used to explore the more in-depth coverage of object-oriented programming concepts, possibly after they have grasped the basics of object-oriented programming.

Designed Purposes of Programming Languages

A programming language is usually designed for a task. For instance, C, C++, Rust, and Go, are designed to develop systems software programs, making them particularly useful for computer science or computer engineering students.

Python and R, are popular choices for quantitative data analysis (R, n.d.; Python, n.d.). These interpreter-based languages offer the convenience of immediate interaction, making

them ideal tools for data analysis. Their applications include data science, data analytics, and business analytics.

Structured Query Language (SQL) is the international and US standard for relational database management systems (Kelechava, 2018). Unlike other programming languages, SQL is goal-oriented or non-procedural, focusing on achieving a specific outcome rather than following a procedural sequence.

For developing web pages, HTML, CSS, and JavaScript are the standard tools (W3C, n.d.). These languages are widely used and provide a straightforward way to create interactive web pages. While other programming languages can be employed to develop web pages, they require conversion into the standard programming code, specifically HTML, CSS, and/or JavaScript. For instance, C# can be used to build web pages, but the resulting C# code will ultimately need to be converted into HTML, CSS, and/or JavaScript for use in a web browser. This conversion process can be time-consuming and may require additional programming effort. In contrast, HTML, CSS, and JavaScript are designed specifically for web development and provide an efficient way to create dynamic web pages. As a result, they remain the preferred choice for most web development projects.

Web server-side scripting languages, such as PHP, ASP.NET, and JSP (among others), are well-established and widely used. However, their primary focus lies in managing web servers (resource management) rather than general-purpose programming.

Mobile application development has become a trend, with two main streams: Android and iOS. For Android applications, Java was previously the standard programming language; however, Google has since emphasized Kotlin for Android application development. For iOS applications, Apple recommends Swift, which is a compiler-based programming language and thus can be used only on Apple systems. As a result, developers of mobile applications had to choose between either Android or iOS. To overcome this limitation, multi-platform frameworks (Jetbrain, 2025) have been introduced such as Ionic (2013), NativeScript (2014), React Native (2015), Dart/Flutter (2017), .NET MAUI (2022), and Kotlin Multiplatform (2023). These frameworks allow developers to write source code once that can be easily run on multiple platforms. Notably, Dart/Flutter employs the most recent technologies, including just-in-time and ahead-of-time compilation.

Industry Demand

Programming languages' popularity can fluctuate over time. For example, COBOL was once highly sought-after for business data and process management during the millennium bug era (Gaskin, 2000). Although its popularity has decreased somewhat, it still maintains a presence in certain sectors, such as finance institutions, where legacy systems continue to rely on its functionality.

Visual Basic enjoyed widespread popularity when Windows desktop applications were prominent in the 1990s and early 2000s. However, its appeal has declined in recent years (Anderson, 2023) as users have shifted their focus towards new computing environments, such as mobile apps.

Despite these fluctuations, there is ongoing debate about programming language rankings. Different sources may have varying rankings, but according to a recent assessment as of 2024, the top programming languages are Python (#1), Java (#2), JavaScript (#3), C# (#7), HTML/CSS (#10), Dart (#19), among others (Cass, 2024).

When selecting programming languages for IS major students, industry demand is a critical factor to consider. However, other factors must also be taken into account, as trends can change in the future. In fact, it may be more important for students to understand programming concepts, which will enable them to learn other urgent programming languages later after completing a general programming course.

6. A CASE EXAMPLE

At our institution, we utilized the presented framework to screen candidate programming languages and select the most suitable general programming languages for IS major students. Previously, programming languages were chosen arbitrarily or based on popularity.

Initially, COBOL and C++ were taught in our IS major program for some time. Later, Java and Visual Basic were added to the curriculum. Our computer science department also introduced Python as part of an introduction to computer science course.

As we offered C++, COBOL, Java, and Visual Basic, our information systems department began to offer a web design course with HTML, CSS, and JavaScript. A mobile application development

course was later introduced, initially using Java but later switching to Kotlin. In database courses, SQL was used, while business analytics courses employed R.

Recently, our Information Systems department reviewed its programming language courses, driven by factors such as the institution's size. As a consequence, COBOL was removed from the curriculum, and C++ is no longer taught by information systems faculty members. SQL and R continue to be used in relevant courses, while HTML, CSS, and JavaScript remain essential tools of the web design course.

To select a suitable programming language for our general programming course tailored to IS major students, we applied the proposed model. Initially, we considered a range of programming languages, including Java, JavaScript, Python, C++, C#, Visual Basic, Kotlin, Dart/Flutter, and Swift. After analyzing these options, Java and Dart/Flutter emerged as strong finalists. In accordance with the guidelines (refer to Table B1, Appendix B), we ultimately chose Dart/Flutter because it is technologically up-to-date, offers practical usability by enabling developers to create applications for various environments and platforms simultaneously, has pedagogical value in facilitating the learning of object-oriented concepts and more, and meets a specific niche demand in mobile application development – an area with growing importance. Notably, prominent companies such as Google, Alibaba, BMW, Toyota, and others are already leveraging Dart/Flutter.

This process required instructors or course designers to conduct in-depth research directly or indirectly. We then quantified the categories, subdividing the five factors into subsections if necessary. Finally, we reviewed the scores to make the final decision.

Dart/Flutter have been used for a while in a mobile application development course which is recommended for IS major students as an introductory and general programming language course. The outcome appears to be appropriate, as our course offerings have continued successfully.

7. DISCUSSIONS

This paper presents a systematic framework that can be used for selecting general programming languages for IS students. The framework can also be applied to other major students. For instance, the model suggests that C/C++ or any compiler-based language would be suitable for certain purposes, while Python and/or R may be more appropriate for data science, data analytics, or business analytics students.

The items for each category are also flexible. In the future, if new technologies are introduced, the items may change. For instance, adaptive compiler is a recent technology. If new technologies are introduced that are better than adaptive compiler, the new ones may receive higher scores and potentially alter the outcome.

Aside from the strengths of the proposed framework, there are some limitations. Firstly, there may be other omitted factors such as social influence. According to Venkatesh et al. (2003), in addition to perceived usefulness (performance expectancy) and perceived ease of use (effort expectancy), two additional factors affect the use of technology: social influence and facilitating conditions. Social influence is similar to the industry demand construct in our model, but they are not the same. While social influence is driven by other users' opinions, industry demand is more focused on a user's benefit. In other words, industry demand is about a user's benefit while social influence may not be directly related to that benefit. The facilitating conditions construct in the unified theory of acceptance and use of technology (UTAUT) is less relevant to our model because it is primarily descriptive in nature (focusing on explaining what people used), rather than being prescriptive (guiding users as to what to do or what to select).

Secondly, the validity of the proposed model has not been established. With only one case example, it is premature to determine whether the model is valid and reliable. The ongoing assessment of even this single case will take a significant amount of time. Additionally, plans are in place to disseminate this model to other institutions for further evaluation. Until then, this model remains incomplete, but it can serve as a valuable starting point.

Thirdly, measuring the success of this model is unclear. Some possible assessment items include:

- Continuity of a selected general programming course: as more institutions participate in this research, the number of successful courses can be used as an index.
- Students' success in completing the selected course: the percentage of students who complete the course can be

used as an index.
- Students' success in subsequent courses: the percentage of students who take a general programming course and are successful in subsequent courses can serve as an index.
- Students' understanding of how computers work: one of the primary reasons for teaching programming languages to IS students is to help them understand how computers function, thereby enabling them to use computers more effectively. This assessment can also serve as an index.

With these, our proposed model can be flexibly applied by IS educators to benefit their students majoring in IS.

Fourthly, the use of the model can be subjective. For instance, one evaluator may have significantly different outcomes than others. To overcome this problem, additional procedural checks can be incorporated to make the process more objective. One possible approach would be for multiple course designers to develop their own comparisons, and then compare these comparisons to reach a final decision. This process can help reduce subjectivity in decision-making. Additionally, visual grids (refer to Figure B1, Appendix B) can be used to facilitate visual comparisons among candidate programming languages, making the decision-making process more transparent and easily understandable.

8. CONCLUSIONS

The conceptual framework presented provides a systematic approach to selecting a general programming language for IS major students. This framework considers key factors such as technical features, practical usability, pedagogical value, designed purposes, and industry demand that influence the selection process.

The framework's applicability can be extended to include other technologies, providing a valuable tool for educators and course designers alike. In this context, different or various items for each construct can be used for different technologies.

With the limitations considered, we hope that the proposed model will contribute to IS education.

9. ACKNOWLEDGEMENTS

We appreciate the anonymous reviewers for their valuable suggestions.

10. FOOTNOTES

[1] Recent versions of JavaScript have introduced more formal object-oriented features, specifically classes (MDN web docs, n.d.).

11. REFERENCES

ABC News. (2009, April 21). *Oracle to buy Sun for $7.4B after IBM drops bid.* Retrieved January 3, 2025, from https://abcnews.go.com/Technology/story?id=7395780&page=1.

Anderson, T. (2023, March 20). *Microsoft's Visual Basic: Why it won, and why it had to die.* Dev Class. Retrieved January 3, 2025, from https://devclass.com/2023/03/20/microsofts-visual-basic-why-it-won-and-why-it-had-to-die/.

Cass, S. (2024, August 22). The Top Programming Languages 2024. IEEE Spectrum. Retrieved January 3, 2025, from https://spectrum.ieee.org/top-programming-languages-2024.

Dart. (n.d.). *Dart documentation*. Retrieved January 3, 2025, from https://dart.dev/guides.

Davis, F. D. (1989). Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly*, 13(3), 319-340.

Flutter. (n.d.). *Flutter: Multi-platform*. Retrieved January 3, 2025, from https://flutter.dev/multi-platform.

Gaskin, J. E. (2000). COBOL Experts: Life After Y2K. Inter@tive Week, 7(2), p. 2/3.

Goodhue, D. L. & Thompson, R. L. (1995). Task-Technology Fit and Individual Performance. MIS Quarterly, 19 (2), 213-236.

Google (n.d.). *Get started with Android*. Retrieved January 3, 2025, from https://developer.android.com/get-started/overview.

Janicki, T. & Cummings, J. (2022). IS model curriculum: Adoption rate of IS 2010 model curriculum in AACSB and impacts of the proposed 2020 model curriculum. *Information Systems Education Journal (ISEDJ)*, 47-56.

Jetbrains (2024, June 10). The six most popular cross-platform app development frameworks. Retrived July 15, 2025 from https://www.jetbrains.com/help/kotlin-

ISSN: 2473-4901
v11 n6451

multiplatform-dev/cross-platform-frameworks.html.

Kelechava, B. (2018, October 5). The SQL Standard – ISO/IEC 9075:2023 (ANSI X3.135). *ANSI*. Retrieved January 3, 2025, from https://blog.ansi.org/sql-standard-iso-iec-9075-2023-ansi-x3-135/.

Kernighan, B. W. & Ritchie, D. M. (1988). *The C Programming Language, 2nd ed.* Prentice Hall PTR, Upper Saddle River, New Jersey.

Laudon, K. & Laudon J. (2020). *Management Information Systems: Managing the Digital Firm, 16th ed.* Pearson, New York, NY.

Leidig, P. & Salmela, H. (2020). IS2020: A competency model for undergraduate programs in information systems. *The Joint ACM/AIS IS2020 Task Force*. Retrieved January 3, 2025, from https://www.acm.org/binaries/content/assets/education/curricula-recommendations/is2020.pdf.

Microsoft. (n.d.). *C# language documentation*. Retrieved January 3, 2025, from https://learn.microsoft.com/en-us/dotnet/csharp/.

MDN web docs. (n.d.). *JavaScript Classes*. Retrieved January 3, 2025, from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes.

Oracle. (n.d.). *Learn Java*. Retrieved January 3, 2025, from https://dev.java/learn/.

Parker, K. R., Chao, J. T., Ottaway, T. A. , & Chang, J. (2006). A formal language selection process for introductory programming courses. Journal of Information Technology, 5, 133-151.

R. (n.d.). *The R Project for Statistical Computing*. Retrieved January 5, 2025, from https://www.r-project.org/.

Python. (n.d.). *Python 3.13.1 documentation*. Retrieved January 5, 2025, from https://docs.python.org/3/.

Tharp, A.L. (1982). Selecting the 'right' programming language. *ACM SIGCSE Bulletin, 14* (1), 151-155.

Topi, H., Valacich, J. S., Wright, R. T., Kaiser, K., Nunamaker, Jr., J. F., Sipior, J. C., & de Vreede, G. (2010). IS 2010: Curriculum Guidelines for Undergraduate Degree Programs in Information Systems. *Communications of the Association for Information Systems*, 26, pp-pp.

Smith T. C. & Jones L. (2021). First course programming languages within US business college MIS curricula. *Journal of Information Systems Education*, 32(4), 283-293.

Venkatesh, V, Morris, M. G., Davis, G. B., Davis, &F. D. (2003). User Acceptance of Information Technology: Toward a Unified View. *MIS Quarterly*. 27 (3), 425–478.

W3C. (n.d.). *The web standards model - HTML CSS and JavaScript*. W3C Wiki. Retrieved January 3, 2025, from https://www.w3.org/wiki/The_web_standards_model_-_HTML_CSS_and_JavaScript.

ISSN: 2473-4901
v11 n6451

APPENDIX A

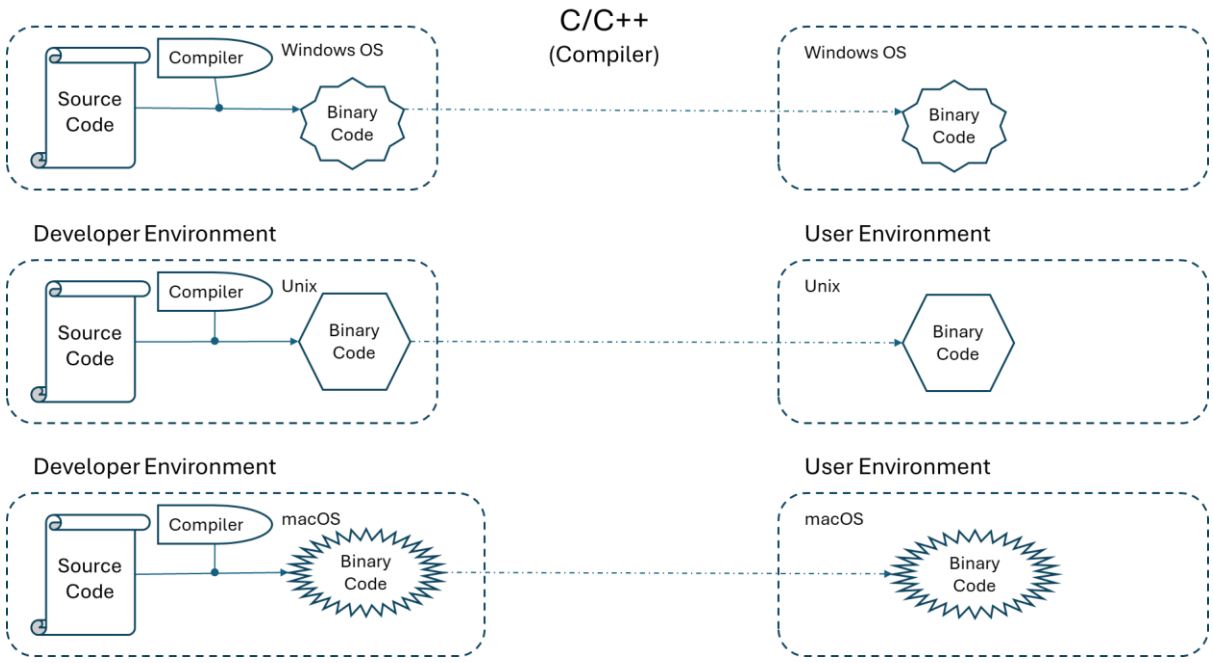

Figure A1: Compilers

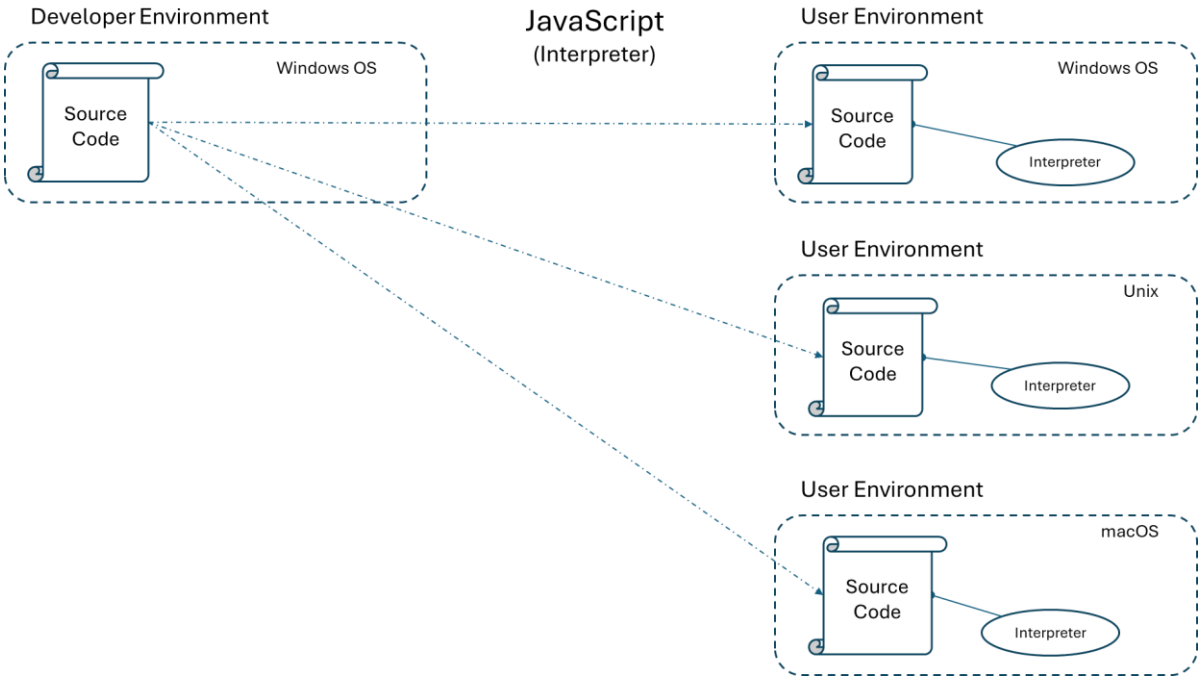

Figure A2: Interpreters

ISSN: 2473-4901
v11 n6451

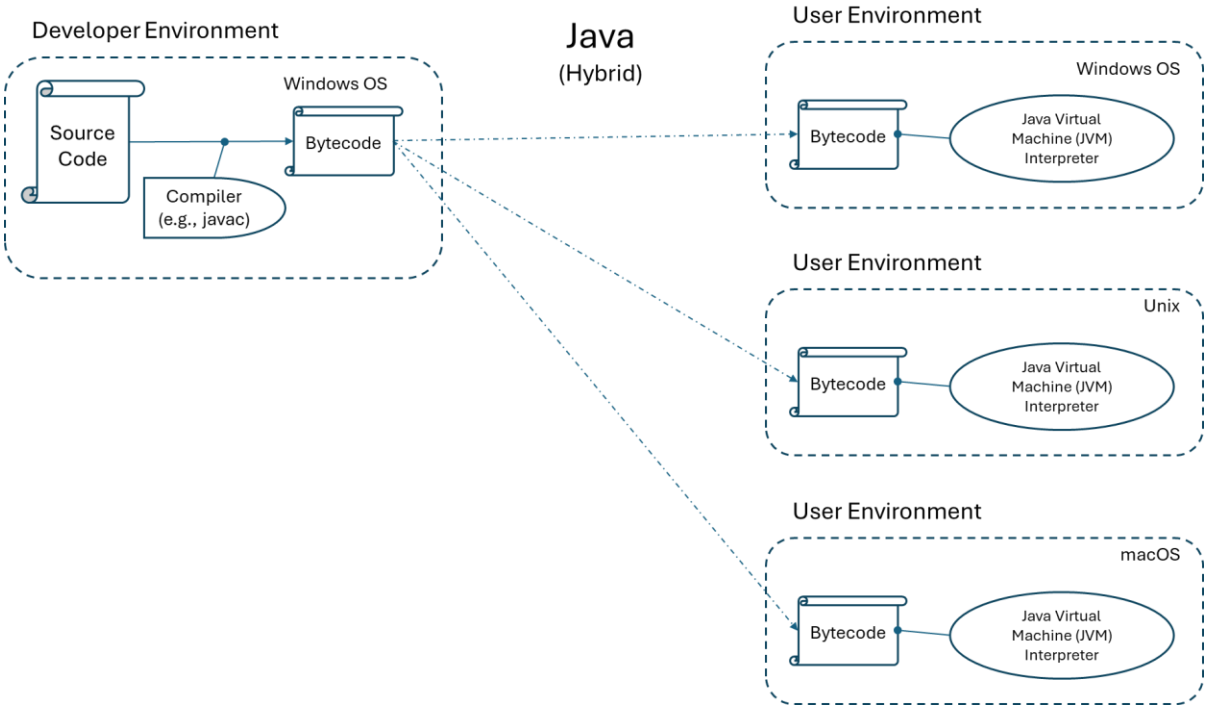

Figure A3: Hybrid

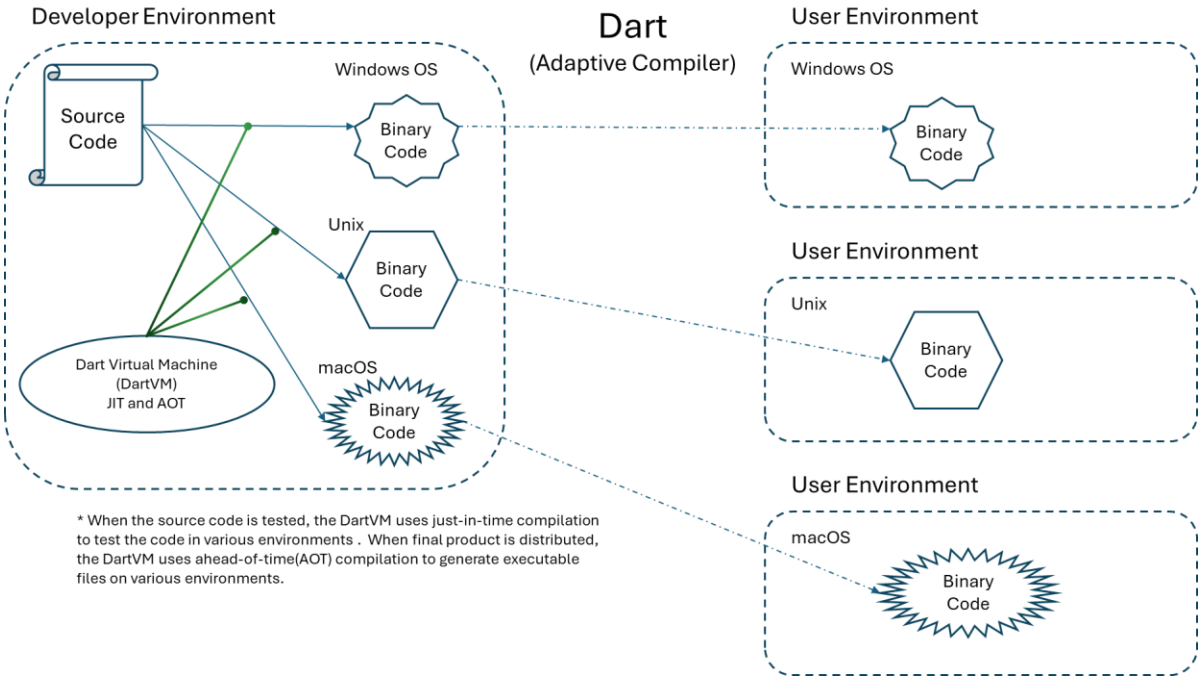

Figure A4: Adaptive Compilers

APPENDIX B

Table B1 presents a comparison of programming languages suitable for IS major students. Although additional programming languages had been compared using tables similar to this one they are not included here due to space limitations. For our institution, the final candidates for general programming for IS major students were Java and Dart/Flutter; ultimately, Dart/Flutter was selected.

Category	Java		Dart/Flutter	
Technical Features	Hybrid programming language - Uses compiler and interpreter	8	Adaptive compilation - Utilizes compilers for just-in-time compiling and ahead-of-time compiling	9
Practical Usability	Support multiple operating systems	8	Support multiple operating systems Support multi-platform (mobile, desktop, web, etc.)	9
Pedagogical Value	Most concepts of object-oriented programming are supported	9	Selected and more practical concepts of object-oriented programming are supported. E.g., polymorphism is not directly supported, but uses other feature (named constructors) to implement this practice.	10
Designed Purpose	Initially develped for appliance program development. Now used for enterprise application development. Once was the standard programming for Android application development.	9	Mobile application development for various operating systems (Android, iOS, etc.). Can be used to develop desktop applications and web applications	9
Industry Demand	High	10	Not high, but growing	8
Total		44		45

Table B1: Review of Programming Languages for IS Students

ISSN: 2473-4901
v11 n6451

Figure B1 illustrates a visual comparison of selected programming languages, using a grid that considers three dimensions: technical features (x-axis), practical usability (y-axis), and pedagogical value (circle size).

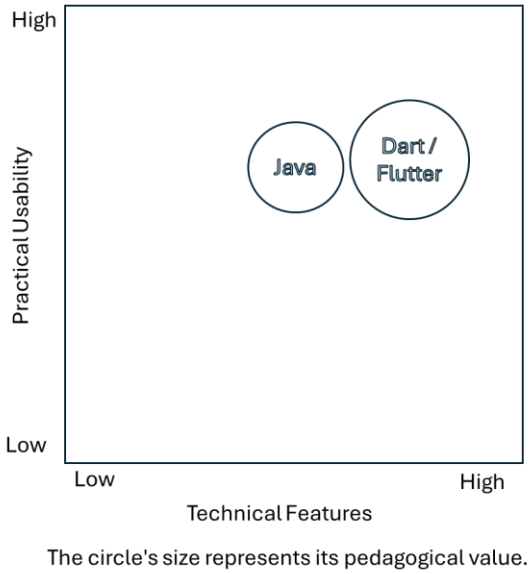

Figure B1: Visual Comparison of Programming Languages (Comparison Grid)

As depicted in the figure, Dart/Flutter has the highest positions for technical features and practical usability. In terms of pedagogical value, it has a similar position to that of Java. For an introductory general programming course, Dart would have a better position (larger circle). However, for an advanced programming course, Java would have an edge over Dart.

This comparison grid does not account for the extrinsic forces: designed purposes and industry demand. If the compared programming languages are tied in the previous comparison grid, they can be compared in another grid with two dimensions: designed purposes and industry demand.

The sequence may be reversed in certain situations. For instance, a two-dimensional comparison grid may be used first to identify more popular programming languages with relevant purposes. If some programming languages remain tied, then a three-dimensional comparison grid can be employed.

If the second comparisons are still tied, voting by course designers can be conducted to finalize the decision.