Jasm Programming Language Overview

admin (at) Jasm Tech (dot) com

Jasm *(noun)*: Take thunder & lightning, a steamship & a buzz-saw; mix them together, and put them in a woman. “She has jasm”. The term originated in 1842. Synonyms: Spirit, energy, spunk, zest.

Introduction

- Jasm makes the software engineer’s job easier, and therefore faster.
- We write computer programs by typing text into an editor and trying to imagine what it will do. It is hard to program this way, and it is easy to make errors. This next-generation language describes a better way.
- I have made a career-long project of finding the best tools, structures, and practices for preventing bugs and making code easier to understand. Making a language designed for this simplifies things greatly.
- The details of the language are about implementing best practices automatically and making the programmer’s job easier. The language is a vehicle for doing that in a way that is easy to write, read, and understand.
- Jasm is easy to learn because it takes concepts that programmers already know to their ultimate conclusion.
- Jasm is revolutionary because of the sheer number of evolutionary advances. This is the next-generation programming language.

Why Jasm is better

- Cuts development time in half by reducing bug rates and creating boiler-plate code automatically.
- Eliminates 80% of the worst types of bugs and prevents project-killing bug quagmires.
- Simplicity: Jasm is easy to use and makes best practices natural & automatic. Debugging & maintenance are faster & easier.
- It is easier to understand complex software if the language is simple and visualization tools are integrated into it.
- Designed for high portability.
- Multi-tasking is hard. There are 4 kinds of multitasking, and each is done differently in existing languages. Jasm code can use any of them interchangeably, simply, and reliably.
- Native support for security, distributed applications, and massive data sets.

The numbers

- In most programming languages, 20-50% of the code is boiler-plate: Typing things in one place to match another, checking procedure arguments, hand-coding common objects and procedures, consistency checking, and so forth. Jasm does this automatically, and sometimes invisibly, to avoid cluttering files. So coding goes 20-50% faster. *(20-50% boiler-plate common consensus)*.

- Most bugs are from syntax errors not caught, boiler-plate errors, errors handed incorrectly, test cases not keeping up with code changes, bad math, flow control, and arguments not checked *(various internet sources)*. Jasm prevents 3 of these, helps with 3 more, and can't fix your math. So more than 50% of common bugs will not happen with Jasm.

- The most difficult types of bugs to solve are from order-of-initialization, pointers, memory deallocation, race conditions, buffer overruns, and access to memory that is no longer valid *(various internet sources)*. They can cause project-killing quagmires. Jasm prevents 4 of these and helps with the other 2. So more than 80% of the serious bugs will not happen with Jasm.

- High-integrity languages prevent bugs, both common and serious. They reduce the cost of development by 50% because there are 70% fewer fixes and 90% fewer bugs seen by the customer *(various vendor sources)*. Low-integrity languages have dangerous ways to introduce bugs and increase development costs by 50% *(various vendor sources)*.

- Programmers spend 35-75% of their time debugging *(various internet sources)*. No language has been designed for debugging before, so there are no numbers for the time saved. Most languages allow debugging to some extent, however.

- There are no numbers for the simplification of a visual & graphical IDE (Integrated Development Environment); only one other language has it. People pay $10,000/seat to use that language.

- Integrating visualization tools into a programming language hasn't been done before. Visualization tools do things like show you where some component is defined, how it is defined, and what it is connected to. There are no numbers for how much it will speed up the creation of complex software or reduce errors. Independent visualization tools claim a 30% speedup in comprehension and finding things in large complex software *(various vendor sources)*. Integrating the tools will improve navigation speed and reduce errors as well.

- Language complexity slows down projects. Programmers take longer to come up to speed. They spend time figuring out how to do something, what this line of code does, or why it is doing that. A highly complex language can slow down a project by 50% when compared to a simple one *(various internet sources)*. Complex languages often have other advantages to help compensate for this.

- There are no numbers for natively supporting true enumerations, symbol lists, state machines, binary search trees, string localization, and relational file managers. They simplify programs and make them easier to understand. Even though they are fundamental programming components, no other language has them. Half of existing languages partially implement enumerations.
- Porting a compiler takes a team about 4 months *(various internet sources)*. The Jasm portability scheme will allow it to run just about anywhere with two easy ports instead of one port per target system.

Languages with similarities

- **Eiffel** is the most similar language. It is a high-integrity multi-tasking language with a visual/graphical IDE. Eiffel is not used much because it is expensive ($10,000/seat), complex, and the programmer must learn to think in the Eiffel way. The free version of Eiffel is crippleware.
- **Ada** is a high-integrity multi-tasking language. It is not used much because the core language, though well-designed, is old and archaic. It has been kept up to date, but updates were design-by-committee. It is also complex. Ada is a niche language used in defense work.
- **HAL/S** is an archaic high-integrity language. It was quite readable compared to other languages of its day. HAL/S was used in aerospace.
- **C#** (C-sharp) has the safety and multitasking built-in, but at the cost of great complexity. When looking for a better programming language I was attracted to what C# can do until I saw the 527-page spec and said “I don’t want to learn this”. It takes a long time to learn.
- **Go** has simplicity and multitasking, but does not significantly reduce bug rates.
- **Java** is highly portable and prevents the worst bugs, but does not prevent most bugs. It also needs a lot of boilerplate.
- **Rust** prevents the worst bugs but does not prevent most bugs.
- **Kotlin** prevents the worst bugs but does not prevent most bugs.
- **V** is a C-language upgrade with some safety and simplicity improvements.

Discussions of the programming language software engineering professionals wish they had falls into three equal categories *(various internet sources)*:

- A simpler language that improves productivity
- Support for something specialized (usually multitasking)
- Pie-in-the-sky stuff that may never be possible

Jasm simplifies things to improve productivity and provides superior support for multitasking.

Who is making new languages and why

These are all new general-purpose languages that are catching on; specialized languages are not listed here.

- Google designed **Go** to improve programming productivity for multicore, networked machines, and large codebases. Simplicity is an important feature.
- Mozilla developed **Rust** for high-performance applications. It was to replace C++, but with lower bug rates and to prevent serious bugs.
- JetBrains created **Kotlin.** They made it to solve their own development problems, and then make a product out of it. Kotlin is focused on interoperability, safety, clarity, and tooling support.
- Microsoft created **C#** (C-sharp) as an alternative to Java, but supporting Microsoft products and less bug-prone.
- École Polytechnique Fédérale de Lausanne (Federal Polytechnic School of Lausanne, Switzerland) made **Scala** as a better version of Java.
- Apple created **Swift** to code in a "safer" way, making it easier to catch software bugs.
- Plataformatec created **Elixir** as a higher-productivity language for distributed fault-tolerant computing.

Jasm goals

Simplicity: Simplicity means easy to learn, easy to read, easy to program, easy to debug, and less complex.

Integrity: High integrity languages prevent bugs by performing data checks and disallowing dangerous practices.

Multitasking: Jasm simplifies multitasking.

Productivity: The end goal is more code done better in less time.

Unique / novel features

- **Visual/graphical Integrated Development Environment (IDE)** simplifies viewing, building, and managing software projects. Only one other language integrates this into itself. All other IDE tools are text-based.

- **Visual/graphical software visualization tools** help make sense of large and complex projects. Only one other language integrates them into the language.

- **Super-simple high-performance memory management:** Manual memory management and pointers are a source of bugs and complexity. Most programming languages expose the programmer to memory management and/or pointers. Jasm manages these for the programmer in the simplest possible way. Some languages simplify memory management, but most of them do it at the expense of performance.

- **High-Integrity:** There are few other high-integrity languages.

- **Autogenerated code:** Few programming languages generate code that the programmer can debug through.

- **Design for debug:** No other language is designed for debugging. Most languages allow debugging, though.
- **Enumerations** are a key software engineering design element. Half of the programming languages do not implement enumerations, and the other half only do so partially. Jasm implements enumerations completely and well.
- **Symbol lists** are a key software engineering design element. Some modern languages implement them partially (you must hand-code symbol lists in most languages). Jasm implements them completely and well.
- **State machines** are a key software engineering design element. No other programming language implements them (you must hand-code state machines in all other languages).
- **String localization** is a key software engineering design element. No other programming language implements this (you must hand-localize strings for language translation in all other languages).
- **Relational file managers** are a key software engineering design element. No other programming language implements them (you must hand-code relational file managers in all other languages).
- **Simplicity & high-integrity:** No other language has this combination of important characteristics.
- **Simplicity & multi-tasking:** Few other languages have this combination of important characteristics.
- **Reactive programming** is a technique for safer multiprocessing & distributed programming. Jasm has native support for it, a first.
- **Design-by-contract** is a high-integrity coding system to improve software correctness, particularly in large complex systems.

How companies make money developing a new language

There are many ways companies profit by making a new language; no two seem to monetize their language the same way. Many profit models are viable.

- Sun made **Java** and profited by selling software, services, hardware, and licenses. Oracle now owns Java and makes a lot of money selling Java middleware, and also licenses Java to other companies.
- Eiffel Software's business is developing and licensing the **Eiffel** programming language. They also do Eiffel consulting, development, and training.
- Plataformatec is a consulting & development company. They created the useful **Elixir** programming language. This brings them consulting & development contracts.

- Netscape was in the browser business and created **JavaScript** to make better browsers. They gave it away and profited by improving their core business. Microsoft countered with **JScript** to boost their browser, and more recently **TypeScript** for the same reason.
- Microsoft developed **C#** and the .NET framework as a better version of Java, one they owned, and give it away. They profit by selling C# products. Microsoft also developed **Visual Basic** and makes money by selling VB products and business being steered to Microsoft products.
- JetBrains created **Kotlin** to solve their own internal development problems. They saved money by doing so and give the language away. They profit by selling Kotlin products.
- Google made **Go** to solve their own internal development problems. They saved money by doing so and give the language away now.
- Apple made **Swift** to solve their own internal development problems. They saved money by doing so and give the language away now.

Return on investment for customers

Ounce-per-ounce, software is the most expensive stuff in the universe. Writing it costs $200,000 per man-year *(various internet sources)*. A team of 10 engineers working on a medium-sized project for a year costs $2 million. A programming language that lets them do it in half the time saves a million dollars.