Multilanguage Programming

Steve Vinoski
Member of Technical Staff
Verivue
Westford, MA USA
http://steve.vinoski.net/
vinoski@ieee.org

Characteristics of Exceptional People

- Name a trait that exceptional people from different walks of life all have in common

Exceptional nurses

Exceptional dancers

Exceptional athletes

Exceptional musicians

Exceptional chefs

Exceptional software developers

Exceptional mechanics

Exceptional authors

Extensive *Vocabularies*

Extensive Vocabularies

techniques

similarities

moves

algorithms

effects

recipes

approaches

patterns

variations

tools

combinations

abstractions

situations

How many of you believe

you've already learned

the last programming language

you'll ever need?

The Blub Paradox

- In his essay "Beating the Averages" Paul Graham introduced this idea to discuss power of programming languages
- He uses a hypothetical language called "Blub" that lies in the middle of the language power continuum

The Blub Programmer

- Sees languages less powerful than Blub as useless due to missing Blub features
- Sees languages more powerful than Blub as "just weird"
  - this is because the Blub programmer can think only in Blub
  - can't relate to features that do not correspond directly to features in Blub

"...an 'X programmer,' for any value of X, is a weak player. You have to cross-train to be a decent athlete these days. Programmers need to be fluent in multiple languages with fundamentally different 'character' before they can make truly informed design decisions."

Steve Yegge

http://steve-yegge.blogspot.com/2007/12/codes-worst-enemy.html

Why Multilanguage

- Enhanced developer productivity
- Increased practicality of solutions
- Avoiding accidental complexity
- Better integration with other systems
- Take advantage of advances in hardware, compilers, VMs, and languages themselves
- Lower development costs
- Improved career possibilities

Productivity

- In The Mythical Man Month Fred Brooks provides this software development equation:

$$\text{effort} = \text{constant} * (\text{number of instructions})^{1.5}$$

- He also cites a number of independent studies that find similar equations
- 10x larger means 32x more effort
- Brevity really *really* matters
- Smaller programs are easier to develop and maintain

"Many of the classic problems of developing software products derive from this essential complexity and its nonlinear increases with size. From the complexity comes the difficulty of communication among team members, which leads to product flaws, cost overruns, schedule delays. From the complexity comes the difficulty of enumerating, much less understanding, all the possible states of the program, and from that comes the unreliability."

Fred Brooks

"No Silver Bullet: Essence and Accidents of Software Engineering"

Examples

Tim Bray's "Wide Finder"

- Simple and not unusual problem: analyze textual website logfiles looking for hits on certain resources, and print out a count of the top 10
- Try to do it in a way that takes advantage of multicore processors (Tim wanted to exercise a T5120 multicore Sun box)

Wide Finder Results

- Fastest solutions were in Perl
  - Python solutions were also fast
  - darn those “slow” dynamic languages ;-)
- JoCaml, a functional language, was at the top for awhile
  - it combines OCaml with the join calculus for concurrency and distribution
- Erlang, which according to its creator Joe Armstrong wasn’t made for problems like this, was 3rd among languages
- Popular languages like Java, C++, C#, C weren’t anywhere near the top

Native XML

- There's an impedance mismatch between XML and mainstream programming languages
  - navigate XML via tree structures, callbacks, or pull parsing
    - easily lose the relationship to the original XML
- What if XML were native to your programming language instead, not as strings but as a normal program text?
- Result: better productivity, and smaller code that's easier to relate to the XML it manipulates

ECMAscript for XML (E4X)

Write literal XML in E4X:

```
var foo = <person name='Munster'/>
foo.address = '1313 Mockingbird Lane'
foo.address.@type = 'home'
```

To represent this XML:

```
<person name="Munster">
  <address type="home">
    1313 Mockingbird Lane
  </address>
</person>
```

E4X is an extension of JavaScript

Note how E4X treats XML natively, not as strings!

E4X in the Wild

- Apache CXF supports server-side E4X (and JavaScript) service impls for JAX-WS 2.0
  - E4X impls are 5x smaller than Java code
  - http://steve.vinoski.net/pdf/IEEE-Scripting_JAX-WS.pdf
  - http://cwiki.apache.org/CXF20DOC/javascript.html
- Project Phobos, part of Glassfish
  - https://phobos.dev.java.net/overview.html
- Scala, a JVM-based functional language, also supports literal XML

Concurrency

- Multicore systems are commonplace
- Number of cores per chip keeps rising, and software needs to take advantage of them
- Mainstream languages are poor at helping with concurrency, due to shared state
- How many developers are truly expert at concurrent programming?

Shared State

- Sharing state among threads and managing it is the root of concurrency problems
  - mainstream languages force us to do this
- Asking the developer to guard and synchronize and lock and protect?
  - miss any shared state, your app is wrong
  - lock too coarsely and your app is too slow
  - lock too finely and you increase chances for deadlock

"What if the OOP parts of other languages (Java, C++, Ruby, etc.) has the same behavior as their concurrency support? What if you were limited to only creating 500 objects total for an application because any more would make the app unstable and almost certainly crash it in hard-to-debug ways? What if these objects behaved differently on different platforms?"

Joe Armstrong, creator of Erlang

as quoted in
http://weblog.hypotheticalabs.com/?p=217

Erlang

- Erlang began life in 1986 for developing highly reliable distributed concurrent systems
- Developed at Ericsson for telecom equipment
- Open sourced in 1998, it's been used to develop systems with guaranteed nine nines reliability (31.5ms downtime per year)
- Under active development, version R12B-2 came out in April 2008

Erlang Concurrency

- Erlang processes are *very* lightweight
- Create and destroy 1 million in 0.53s on MacBook Pro
- Contrast with other languages on the same machine:
  - Java: 250,000 threads in 48.6s
  - C++: 7044 in 1.3s, then out of resources

Threads Without Limits

- No artificial limits on threads changes how you approach problems
  - for example, Erlang servers scale better than heavy-thread languages (search for "Apache vs. Yaws" for example)
  - No more thread pooling, leader-follower, or other non-trivial patterns

Message Passing

- Erlang avoids shared state, uses message passing instead
  - the type of message passing originally intended for OO languages
    - very fast, asynchronous, same host or across hosts
- Erlang variables cannot be re-assigned; they're bound once and that's it, to avoid mutable state
- No explicit code for concurrency guards, locks, synchronization etc. required

Reliability

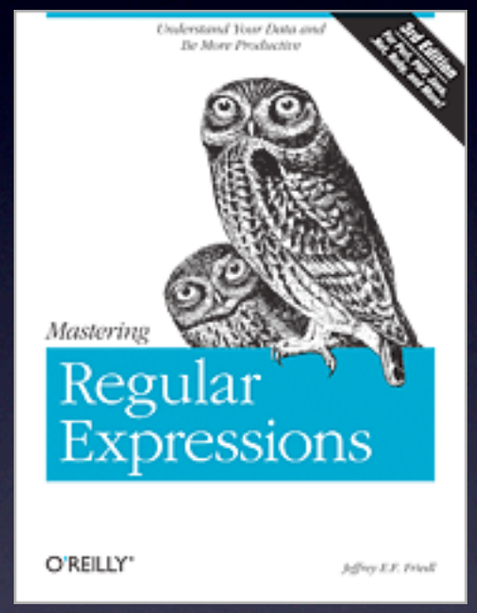
- Enabling highly reliable systems is a primary goal for Erlang
- It encourages designs that accept that failure will occur and must be dealt with
  - Processes can be arranged in distributed supervision trees, where supervisors watch and restart failed processes
- Code can be loaded into running systems
- The Open Telecom Platform (OTP) libraries provide common application behaviors supporting reliability

Erlang, Middleware, and an Observation

- Most of my career has been writing C++ and Java middleware; wish had I known about Erlang years ago
  - I could have produced more reliable systems that scaled better...
  - ...and cost a lot less to develop and maintain
- This often results when you try new languages
  - Generalization of Greenspun's Tenth Rule: "*Any sufficiently complicated platform contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of a functional programming language.*"

Regular Expressions

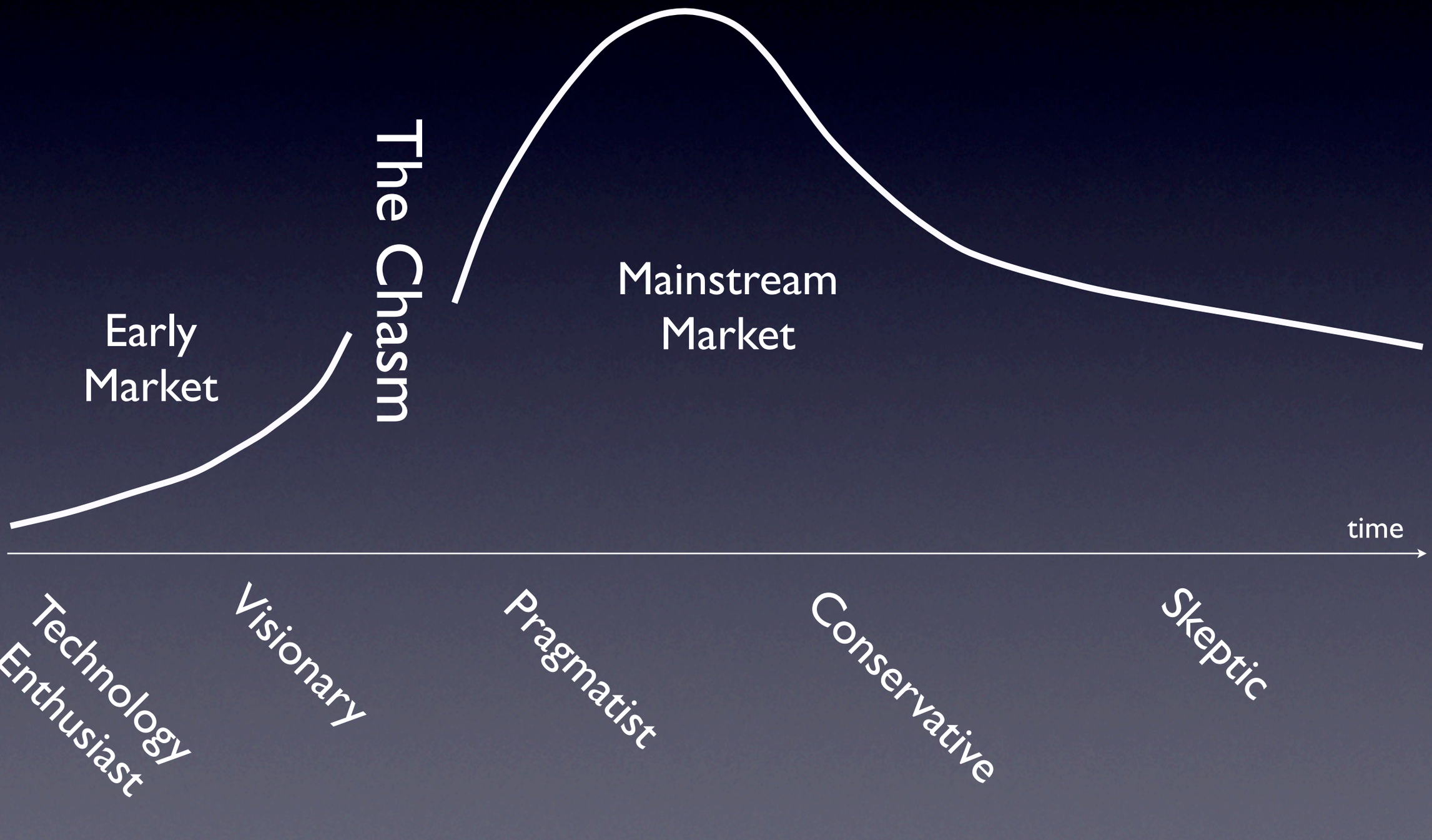
- One of the most important languages a developer can learn
  - they open up vast possibilities for searching, sorting, and data transformation
- Like all good languages, regexps change the way you think about and approach problems



Barriers to Multi-Language Programming

- Ignorance and Fear
- Tools
- Testing
- Cross-language integration
- Learning materials, community and support

Technology Adoption Lifecycle



Technology Adoption Things To Remember

- Disruptive technologies are never "good enough" for Pragmatists, Conservatives, Skeptics
  - 13 years ago, C++ guys said Java was too slow; today, Java guys say Ruby is too slow — hmm...
  - if the technologies are useful, they win anyway due to lower cost
- Different adopter categories never agree on technology
  - real source of many "technical" disagreements
  - Technology Enthusiasts and Skeptics can't even relate

Ignorance and Fear

- Some believe all languages must be as large and complicated as Java and C++ (related to Blub Paradox)
  - result: an unwillingness to look at new languages, due to the assumption they'll take years to learn
- More languages you know, learning becomes easier
  - after you know a few, you find many concepts across languages are similar
  - identify language strengths and weaknesses and apply them appropriately

Editors and IDEs

- Eclipse, Netbeans, Visual Studio deal with multiple languages to varying degrees
- Language and editor can become mutual detrimental dependencies
  - can't write the language without IDE, and can't use the IDE with other languages
  - can result in a developer who's unable to move along with technology changes
- Personally, I use emacs (for 23+ years)
  - amazingly rich, and infinitely extensible in emacs-lisp
  - can use it to develop in any language and not have to wait for an IDE provider to give me features

Debugging

- Yes, languages provide debuggers
- Many "dynamic languages" provide interactive prompts
  - aka REPLs (read-eval-print loops)
  - development/debugging merge together
    - interactive development tends to include more exploration and experimentation, leading to more insightful solutions
    - can also help you learn the language faster
- Profiling, tracing, logging typically exist as well
- If nothing else there's always the good old "print" debugger

Testing

- Focus on Test-Driven Development over the past decade means test frameworks are available for many languages
- Unit testing critical to making sure code in each language is correct
- When adding languages to an existing system, be sure to fit the new tests to the old harness
  - make reporting (printed output, return codes) the same
  - important for system test and QA

Cross-Language Integration

- Use Inter-Program Communication (IPC)-based integration (i.e., the network)
- Foreign function interfaces for C
- The JVM is becoming a multi-language VM
  - JVM-based languages can generally talk to each other and to Java
  - Makes Java libraries available to other languages
  - Significant interest in JRuby, Scala, Groovy, JavaScript/E4X (Rhino)
- Microsoft Common Language Runtime (CLR) supports multi-language integration

Multilanguage VMs

- VM-based multi-lingual development is currently crossing the chasm
  - if your organization is to the far right of the technology adoption curve, it'll be a few more years before they'll even consider multi-lingual development
- Ironically, Java programmers have a tendency to be mono-lingual but the JVM lets them try other languages with little churn, little risk, and little cost

Getting Started

- Use another language on the “edge” of your system
  - write a client in a different language
  - write some tests in a different language, works well for distributed systems
    - keep the core in Java or C++ or whatever it is
- Find a recurring problem in your business and consider what other language might be better for solving it
  - coolness doesn’t cut it
  - a good solution can show true business value

Team Issues

- Don’t let a lone programmer plow ahead and leave everyone behind
  - do the work with a team of at least 2, gives you a more balanced assessment of what’s good and what isn’t, and you can help each other learn
- Don’t bite off more than you can chew
  - do your homework
  - don’t make promises based on results read in a blog somewhere, instead do the work and experiments for yourself and prove the value

Summary

- Continuous technology advances mean that learning and applying new languages is inevitable, especially if you want to stay employed
- Don’t fall into the trap thinking that all languages are as big and complicated as Java or C++
- Using the right language can mean significantly increased productivity, vastly fewer lines of code, and greatly enhanced problem-solving capabilities

"If you want a new idea, you have to silence your inner critic. Your sense of right and wrong, of smart and stupid works by comparing new ideas to what you already know. Your sense of what would be a good fit for you works by comparing new things to who you already are. To learn and grow, you must let go of you, you must be young again, you must accept that you don't understand and seek to understand rather than explaining why it doesn't make any sense."

Reginald Braithwaite

as quoted in

http://weblog.raganwald.com/2008/04/why-we-are-biggest-obstacle-to-our-own.html

For More Information

- Reginald Braithwaite's blog: http://weblog.raganwald.com/
- Ola Bini's blog: http://ola-bini.blogspot.com/
- Charles Nutter's blog: http://headius.blogspot.com/
- James Hague's blog: http://prog21.dadgum.com/
- John Lam's blog: http://www.iunknown.com/
- Lambda the Ultimate: http://lambda-the-ultimate.org/
- Ted Neward's blog: http://blogs.tedneward.com/
- Debasish Ghosh's blog: http://debasishg.blogspot.com/
- Oliver Steele's "The IDE Divide": http://osteele.com/archives/2004/11/ides
- My own blog: http://steve.vinoski.net/blog/