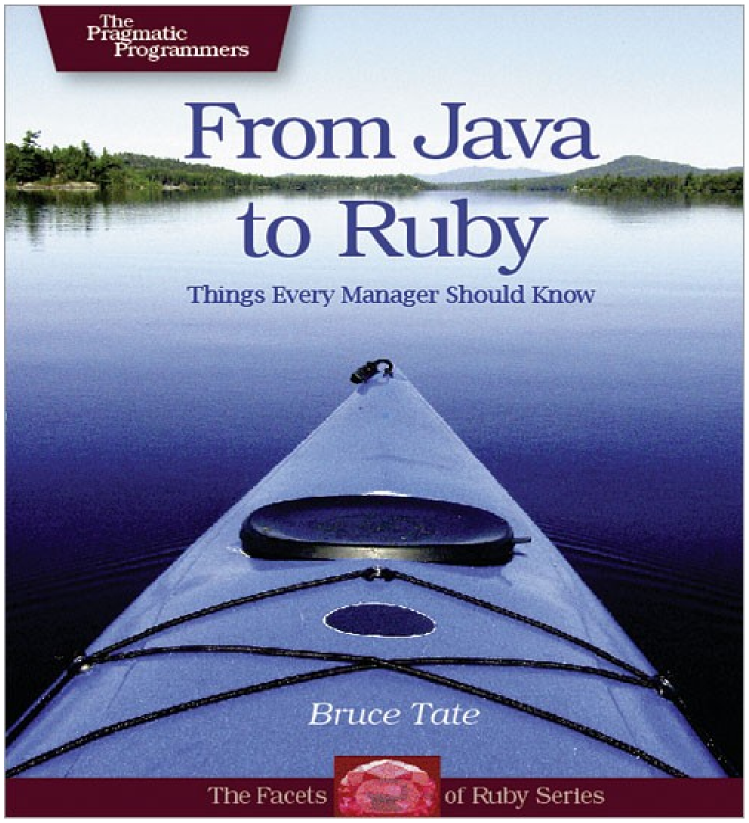

The Pragmatic Programmers
From Java to Ruby
Things Every Manager Should Know
Bruce Tate
The Facets of Ruby Series

From Java to Ruby

Things Every Manager Should Know

by Bruce Tate

Book Summary
by Paul Klipp

Table of Contents

Chapter 1 - Introduction

New major programming languages emerge every decade:

Fortran	1954
Cobol	1961
C	1971
C++	1983
Java	1996

We're due for another.

Java is becomming too complex, with too many frameworks and poor J2EE performance.

It takes an explosion in popularity for a new language to generate the supporting community to convince pragmatists and conservatives to adopt it. Such an explosion is underway with Ruby.

Those who have made the switch to Ruby early will have a huge advantage over those who wait.

How to migrate to Ruby:

1. Identify what pains you about Java and gather information to demonstrate how Ruby will help.
2. Do a small test project in Ruby.
3. Deploy Ruby on a larger scale.

Risks

- the language could stagnate
- Ruby might not scale well
- the small Ruby community means limited 3rd party offerings
- Ruby is not as structured as Java and has less automation

Chapter Two - Pain

How painful is Java to you? If your programmers and clients are happy, then perhaps you shouldn't change.

What's good about Java:

- large community, easy to hire programmers and consultants
- thriving open source community
- mature, safe language
- proven scaleablity
- variety of standards and options

Technical strengths of Java

- Two-phase commit
- object-relational mapping - there is no mature Ruby framework for this.
- distributed objects - Java has more ways to manage distributed objects than Ruby.

Java is big and powerful - far too big and powerful to be the best solution for most database-driven web apps.

What's wrong with Java?

- Lower Productivity
    - Java copied some of C++'s problems as a trade off in order to attract C++ devs in the early days.
        - -syntax - Java programs have 2-4 times as much code as Ruby programs
        - -not purely object oriented - some elements are not objects
        - -static-typing - more productive languages use "dynamic typing"
- longer code is harder to maintain
- Java devs most compile code often to test
- Java's primatives are not object-oriented
- Productivity study: http://apge.mi.fu-berlin.de/~prechelt/Biblio/jccpprt_computer2000.pdf
    - showed dynamic languages are more productive than Java or C or C++

A project has two kinds of complexity:

- essential complexity: an accounting system is at least as complex as accounting rules.
- accidental complexity: introduced complexity. Martin Fowler says that with Java, especially since the introduction of EJB, accidental complexity is most of the total effort.

EJB, XML, and Web Services

EJB is far too complex for most applications.

Java devs embraced XML as an essential tool for managing data. Too much, in fact. Now XML is inseparable form Java, often used for configuration, business rules, to describe documents and messages, and to write programs. By solving one problem, XML introduces unnecessary complexity into Java development.

A proliferation of web services and frameworks introduces confusion. Each advance is significant. New APIs are much more efficient than SOAP. Frameworks like Spring and Hibernate are also a considerable improvement over EJB, but add to the complexity of working in Java.

This proliferation of APIs and frameworks is a symptom of an overly complex language.

One symptom is the overwhelming push to keep inexeperienced developers from making messing in Java projects.

Frameworks are too complex for most projects. You can create a lightweight Java application if you know Java, Servelets, Hibernate, Tapestry, and the tools: IDEA, Junit, CVS, Ant, and CruiseControl. Each of these takes weeks to learn and months to master and most have 800+ page books written about them.

It takes training, mentoring, and lots of mistakes to begin to code efficiently using this combination of tools.

Each too presents a steep learning curve even to experienced programmers. The move from JDBC to Hibernate or JDO can be very painful.

A new hire can take months to become productive and each new technology imposed on a project can set the whole team back weeks.

Competition

Java visionaries know Java is fading. Most consultants believe that Ruby is several times more productive than Java.

Ruby allows metaprogramming: programmers can easily extend Ruby, leading to compounding gains.

If you're a Java shop, you ought to have a backup plan.

Chapter Three - Establishing Your Reward

The visionaries are a good measure of future trends:

Martin Fowler, Dave Thomas, Andy Hunt, James Duncan Davidson, Stewart Halloway, Justin Gehtland, David Geary, and Richard Monson Haefel are all talking about and publishing their enthusiasm for Ruby.

Downloads are also a good measure of future trends:

Spring	280,787
Hibernate	520,061
Rails	500,205

-2006 download stats from Sourceforge and Rubyforge

Ruby and Rails downloads will soon overtake Java framework downloads.

There is a sudden growth in Ruby books being published and a decline in the sales of Java books according to Tim O'Reilly.

http://rewrite.rickbradley.com/ - this blog chronicals the process of rewriting an enterprise Java application that originally took four months into Ruby. The rewrite took four days and demonstrated considerable performance improvements.

Several studies link lines of code to number of bugs and development time, as well as maintanence costs. More code = more cost and more bugs.

Here is an example from the book of code to print the fibonacci sequence written in Ruby and in Java:

Ruby

```
x, y = 0, 1
10.times do
  puts y
  x, y = y, x+y
end
```

Java

```
class Fib {
  public static void main (string args[]) {
    int x = 0;
    int y = 1;
    int total = 1;
    for (in i=0; i<10; i++) {
      system.out.println (total);
      total = x+y;
      x=y;
      y=total;
    }
  }
}
```

Not only does the Ruby application do the same thing with far fewer lines of code and more readable code, but left to run more than ten loops, the Java program would eventually start computing the wrong answers without throwing an exception because of integer overflow, while the Ruby program would automatically grow the number to a Bignum.

The application migrated from Java to Ruby mentioned earlier yeilded these statistics:

Metric	**Java**	**Ruby**
Time (at .5 FTEs)	4 months	4 days
Lines of code	3293	1164
Lines of configuration	1161	113

Reduced repetition and more concise notation lead to higher productivity and lower maintenance costs.

Features of Ruby that increase productivity:

- Closures - Reduces repetition with groups of items
- Pure object-orientation - less to learn, less to code, easy to read
- Continuations - make better web servers. The next generation of web servers may well be written in Ruby.
- Optional Parameters - a hash map (single table of parameters) lets devs specify only those options needed.
- Open Classes - make it easier than Java to test and extend code.
- ObjectSpace - allows devs to enumerate all objects defined in an app which makes debugging easier and allows for easier impementation of certain algorythms.
- Freezing - locking an object makes it easier to catch bugs.
- Message passing and missing method - quickly add dynamic methods at run time.

Ruby has something to offer all levels of developers:

- simple scripting for beginners and administrators
- Object-oriented programming for experienced developers
- advanced techniques like functional programming, metaprogramming, and domain-specific languages for advanced developers and architects.

Adding Rails provides significant advantages for web-enables database applications:

- Convention over Configuration: drastically simplifies configuration
- Database strategy - rails discovers and adapts to the database structure
- provides excellent default values automatically
- rapid feedback - make a change and reload the browser. Java devs must compile, build, and deploy to see thier changes.
- Built-in testing - rails automatically creates default test cases and fixtures
- AJAX - the ease with which AJAX can be used in Ruby on Rails app greatly improves the user experience.

Other languages provide some productivity improvement up front, such as PHP, Python, and Visual Basic, but none so much as Ruby and none that are so easy to maintain. Any productivity measure can't ignore maintainability.

The larger implications of increased productivity

Higher productivity means:

- smaller teams
- improved time to market
- more simultaneous projects
- less need for internal documentation
- These benefits combined can lead to exponential savings well in excess of lower development costs alone.

Ease of use lowers training time and even novices can safely start adding value very quickly.

The author estimates that he can train a novice Java dev Ruby on Rails roughly four times faster than teaching the same dev to build a web-enabled database application in Java.

Chapter Four - Pilot

Choosing the right pilot project is important.

H Technical Risk L	Difficult Learn more	Learn more Bad if you fail
	Easy Low payback	Success likely but less learned
	L Political Risk H	

Build the team

The best pilot team has:

- some experience with dynamic languages
- small size
- freedom to experiment and make their own choices

Types of pilots:

- Classic - difficult technology but low political risk. Ruby is choosen because Java is already failing.
- Trojan Horse - low risk (both technical and political). Dev builds a small stealth app to show abilities.
- Race - easy app with high political risk. Show how fast/easy/cheap it is to use Ruby for an important but simple project. Good for winning upper management support.
- Bet your Business - the 37signals approach. Just do it and hope it works. High political and technical risk.
- Rescue - save a floundering, high-risk, Java project using Ruby.

Chapter Five - On an Island

After a successful pilot, a next step might be the creation of "island"applications that require limited interaction with the outside world.

Ruby is more than a scripting language. Even without Rails, it's an excellent language for:

- driving testing
- integrating enterprise applications
- working with text files
- using images
- building and accessing web services
- using middleware like LDAP
- working with databases
- better than Java for web development

Ruby's Major Features (Core Libraries)

- Language Support - includes the most basic objects and the basic building blocks of the language: syntax, expressions, threads, decisions, loops
- Input/Output - Ruby's I/O borrows from Pearl and C to create an impressive libraryof features to access devices and other programs
- Data Munging - impressive assortment of text and data management includes high-level string support with regular expressions and templating and extensive math support.
- Communications - several libraries support TCP/IP, HTTP, SOAP, and other APIs. Also supported are serveral web development features such as CGI.
- Development Support - interactive interpreter is good for trying out ideas. Rake makes it easy to build projects. Unit testing, documentation, and benchmarking tools built in. Gems and setup are used to package and deploy code.
- User Interfaces - Not as extensive as those for Java or .Net, but you can use several cross-platform toolkits.
- security - encrypt files, test user input for dangerous code, lock down parts of the application.
- Integration - allows for integration to Microsoft tools, DLLs, and other languages such as C.

As a scripting language, Ruby has all the features of Perl but with a cleaner design and more expressive syntax.

Ruby is frequently associated with LAMP development strategies.

Typical LAMP share-nothing architecture:

Client	Application Server	Database Server
browser	Python, Perl, PHP or Ruby Fast CGI Apache or Lighttpd Linux or FreeBSD	MySQL, PostgreSQL, or SQLite

Shared nothings make scaling easy by adding hardware. Each application can fulfill requests independantly. only a few layers, such as the database, are shared across a cluster.

Most Ruby web applications use this approach.

Ruby is good for LAMP because:

- it's open source
- it handles text very well
- it supports all of the necessary protocals
- Ruby is supported by Apache and Lighttpd and it has its own web server (Webrick)
- Ruby has very efficient database integration.

LAMP is gaining recognition as a Java alternative.

Stateless applications scale better, but they are hard to write. The app must be designed to manage state data itself. Web servers use stateless architectures. New Ruby frameworks are experimenting with a "continuation" feature to address this issue. Three such frameworks are:

- Wee - doesn't use Ruby continuations
- Borges - a port of the smalltalk coninuation server "seaside"
- Iowa - a framework started by the author of "seaside"

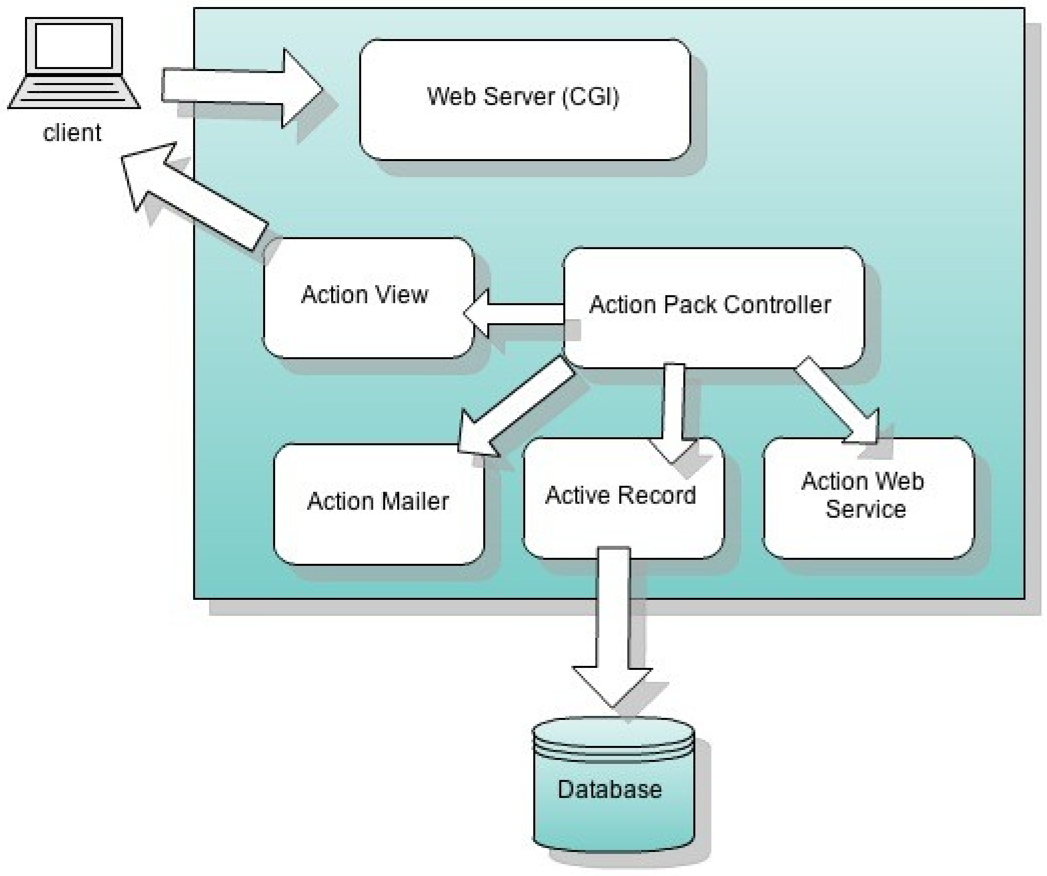
These are still immature, but important to the future of apps in which state management is a problem.

Rails is the only broadly-used Ruby framework. This focus gives Rails a huge productivity advantage over the many competing Java frameworks.

Rails is a Ruby application framework for web applications connected to relational databases. It includes a suite of core frameworks:

- Active Record - performs all databsae access. Discovers fields and structure and adds some things automatically to your classes.
- Action Pack - handles presentation
- Action Mailer - handles email integration
- Prototype - uses AJAX for things like drag and drop
- Action Web Service - allows you to integrate Rails apps with other applications using other frameworks
- Ruby - Rails uses Ruby metaprogramming to make it esier to define database classes and for simpler configuration with Active Record. Rails also improves some of Ruby's core frameworks for working with HTML.

A model of RAILS – illustrating that it consists of a suite of smaller frameworks



More about the main components of Rails

Active Record

Traditionally when creating a database-driven application, you have to define the database and object model independantly and then write some code to connect them. Active Record automatically discovers database fields and adds them to object. It just requires that the programmer stick to simple object naming conventions.

An Active Record class might look like this:

```
class Book<ActiveRecord::Base
        has_one :author
        has_many :chapters
end
```

A Java class with the same functionality would need dozens of lines of code and a configuration file.

Action Pack

Action Pack handles both view and controller parts of the MVC design pattern. It invokes Active Records and moves data between controllers and views.

Action View

Action View does for Rails what JavaServer Pages do for Java.

Middleware

Ruby lacks a standard database API, but there are three strategies for getting around this limitation:

- Driect API - there are drivers for most common databases including MySQL, Oracle, PostgreSQL, SQLite, DB2, MS SQL Server and others
- DBI - Database Interface uses a library similar to the Perl DBI. Direct API is still more efficient, but DBI is moving in the right direction.
- Mappers and Wrappers - database mapping in Ruby is not as popular are in Java (EJB, Hibernate) but there is a database mapping framework for Ruby called OG. Many people are probably taking advantage of the fact that it's easy to write your own in Ruby. Wrappers, such as Java's Spring and Velocity, have their Ruby parallel in Rails' Active Record. Active Record also handles relationships, which is rare or poorly implemented in Java frameworks. Interestingly, you can use Active Record without Rails.

Security

Most Ruby devs use either the Ruby/LDAP extension or one of the many mature plug-ins for their preferred web framewoks.

Communications

Ruby has excellent support for ReST, a simpler model of web services (and prefered by Amazon.com) but also for SOAP. Ruby also has direct support for low level protocals like TCP/IP and HTTP. The EML RCP (remote proceedure call) is very useful for Java bridging strategies There are also many other communication APIs avaliable.

XML

Ruby support for XML is fantastic. Since Ruby is so well-suited to managing text strings, it works with XML better than Java does. The most popular XML framework for Ruby is based on ElectricXML and is called REXML.

Chapter 6 - Bridges

Integrating Java and Ruby Applications

The next step is to plan your move to Ruby from within an environment built on Java. Approaches can be mapped on a simple matrix representing strategic versus tactical planning and coarse versus fine data transport methods.

Strategic vs. Tactical

Service-oriented architecture is a strategic investment because you have to adapt and extend your existing applications to interface with others using services.

A Java bridge is a tactical solution because it involves minimal changes but is less flexible in the long run.

Coarse vs. Fine Trasport

Coarse transport methods move volumes of data efficently using minimal interface, such as in an SOA strategy.

Fine transport, such as scripting business rules in Ruby and putting them in a Java engine, trades efficiency for convenience.

Scenarios

- Migration - You've decided to move all Java apps to Ruby
- Interapplication Integration - Use the best technology for the job, but make all apps cooperate.
- Simplification - You can use Ruby inside a java project to simplify certain functions. For example, Ruby can manage data in an application often better than XML. Ruby can be used within a Java app for generating web pages, business rules, testing code, configuration, and running scripts.

Because Java is not productive for web development and because users are demanding more AJAX interfaces, developers will increasingly build more Ruby frontends onto Java backends.

JRuby

JRUBY is a Ruby virtual machine written in Java and will bring many of Java's advantages to Ruby, such as JDBC, transaction libraries with two-phase commit, and enterprise integration libraries. JRuby will make it easier to run Java and Ruby apps side by side. JRuby allows Ruby devs to reference and use Java classes from within Ruby. This allows Ruby devs to use Java libraries. It also allows programmers to embed a Ruby interpreter into a Java application. Using JRuby could significantly impace how Java developers think about writing software. Ruby can be a powerful tool for the Java developer.

Using JRuby, developers can access Java APIs, frameworks and services. For example, with JRuby you can write Swing apps in Ruy or use JDBC to access databases.

Java SE6 will include pluggability for Ruby and other languages and will ship with a JavaScript interpreter.

JRuby also allows Java devs to write Ruby unit tests.

JRuby currently passes 85% of the core Ruby test cases, but it will never be able to perform OS-specific or native C calls.

Future plans for JRuby include:

- using Rake to replace Ant
- better implementation of Ruby continuations
- Just-in-time and ahead-of-time compilation
- thread scheduling
- ability to run Ruby from within a Java servelet

Ruby on Rails could form the UI of existing Java legacy systems, keeping the Java as the moded and running the entire application, Java and Ruby, from within a JVM.

Domain-Specific Lanugages

Java does not provide good support for crafting DSL such as MS Office Macros, Spreadsheet functions, and HTML docs in a browser. Ruby is very good for building DSL. JRuby allows devs to use Ruby for scripting withing a Java app.

Supporting Technology

Java to Ruby bridges allow a Ruby app to access a Java API through an interface called a "bridge." They only try to bridge data across languages.

- RJB - Uses the Java Native Interface (JNI), and provides good performance. It is the most difficult bridge to set up, but once in place, it's easy to use.
- YAJB - uses XML for remote proceedure calls. Easy to install, you just drop libraries into the right directory.
- RJNI - this is not actively developed and supports only one-way calls.
- RJAVA - uses TCP/IP. No longer developed and not recommended for production systems.

Advantages - light, simple, convenient
Disadvantages - not good for large-scale development. A more tactical than strategic solution. They are not efficient for bi-directional traffic and they are limited to only Java and Ruby.

Service-Oriented Architectures

An SOA should:

- use common standards for messaging
- use simple interfaces, transfering complexity to message content
- expose coarse-grained interfaces

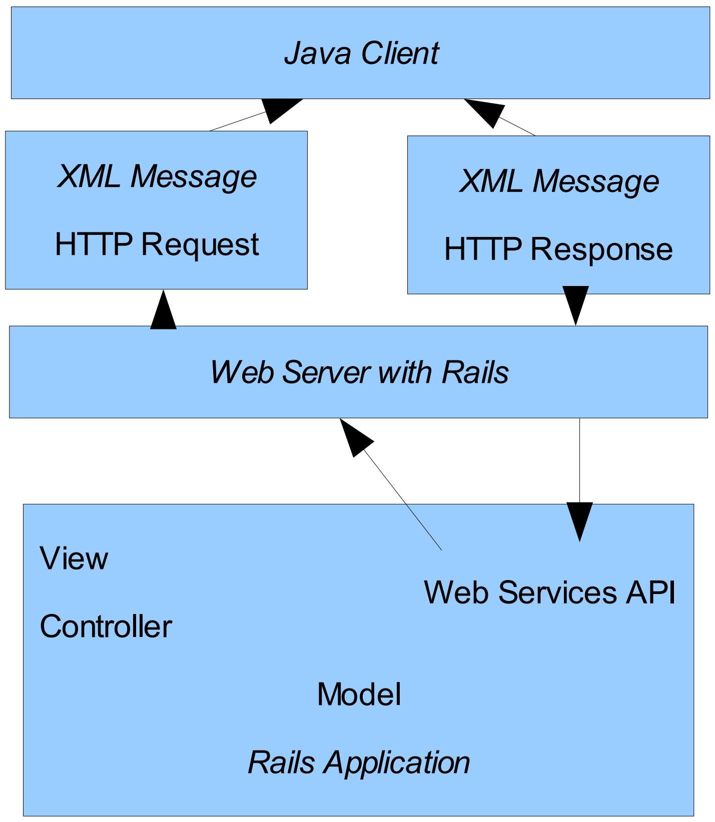
XML

Ruby supports a variety of XML standards that can be used in an SOA scenario. The most popular is REXML, but Ruby also supports natively the much simpler YAML (stands for "YAML Ain't a Markup Language").

SOAP has become incrediby complex and is supported by Ruby through a variety of APIs, but most Ruby devs prefer the more simple (and universal) ReST-based web services.

ReST (Representational State Transfer)

ReST-based web services move XML documents using HTTP and TCP/IP. Rails uses a ReST-style web service.



Incomming requests specify a document in XML which is parsed by REXML and sent to the Rails model logic. The REXML layer returns the processed document as a response to the request if necessary.

Using this type of development on both the Java and Ruby side of applications has a variety of implications:

- By using only internet standards, you can communication across languages
- You could pick the best language for each part of an application and use such a model to share data within a multi-language application
- Using glue code, you can interact with legacy software using bridging technology.

This approach scales easily by just letting your network infrastructure perform the load balancing.

Chapter 7 - Ramping Up

This chapter deals with hiring and training Ruby devs. It stresses building skills internally. Hiring developers with prior dynamic language experience, the effectiveness of metoring within small teams. Devs who know: Smalltalk, Lisp, Python, and Perl, learn Ruby easily. Just make sure the devs are also familiar with building object oriented software using these technologies.

The author recommends design reviews - using an outside expert consultant to quickly evaluate an application's design and offer rapid feedback at different stages of development.

Transitioning existing apps to Ruby is easier if you follow these conventions:

- table names should be English plurals (e.g. the people database holds person objects)
- use object identifiers named id.
- Foriegn keys should be named object_id (e.g. person_id)
- Join tables use the names of the tables they join. (e.g. people_courses for a table that joins the people database and the courses database)

Architectural guidlines that will ease migration to Ruby and Rails

- MVC separation
- SA
- ReST-based web services
- Shared-nothing architecture
- HTML and JavaScript on the client
- Use standards such as XML that are supported by Ruby

Deployment

Deployment should be completely automated and recoverable and should manage important dependancies. Capistrano can be used to accomplish all of these goals.

Capistrano allow you to:

- deploy applications to a remote server with a single command
- roll back a deployment to the previous state with one command
- issue parallel commands for deployment tasks across multiple servers
- check code out of SVN.

Rails Scheme Migrations

Ruby on Rails uses a function called scheme migrations to back out changes to a database after an unsucessful upgrade.

The combination of Capistrano, SVN, and scheme migrations can make for a very sophisticated and robust deployment strategy.

Chapter 8 - Risk

Given that many, if not most, software projects fail, it's important to carefully consider potentially risky decisions like changing languages.

You should not look to a new technology to solve process problems.

Political opposition to change may be so strong that any technology change will be doomed to fail.

Ruby is not the technology for every problem. Match it's strengths to your needs.

There remains virgin territory where Ruby apps have not been tested:

- Ultrahigh volume apps
- ultrahigh connections
- Very rich user interfaces
- Ruby has not yet been used in these scenarios. That doesn't mean it's automatically ill-suited to them, but these scenarios do represent unknown risk.

Deal with risk by gathering information, having backup plans, limiting your ambition, and elimiating related risks.

Make sure that you answer these questions to your own satisfaction:

- Does it do what I need?
- Is it still buggy?
- Is it fast enough?
- Can I learn it or find people who know it?
- Can I find support if I have trouble?

Since Ruby, Rails, and related tools are open source projects with a lot of momentum behind them, you are likely to experience three differences from Java technologies:

- Rapidly developing tools
- An active and enthusistic support community
- Tools that are easy to customize

Most people woudn't think to modify Hibernate to suit their needs, but by comparison, extending Rails is entirely doable.

Most of the author's suggestions from minimizing risks have roots in agile develpment, such as building the riskiest elements first, using automated testing, building prototypes (spiking solutions), and a willingness to abandon an idea if it has proven to fail.

In conclusion, dynamic languages offers higher productivity. Ruby offers more maintainable code. Software has for decades been moving to a higher abstration level with object-oriented code. A language like Ruby is the next logical step. Ruby is well positioned to be the next big thing in programming languages.