

How to get Rails web applications accepted in industry

H. J. Mackenzie¹

¹Managing Director, HARD software, hjm@hardsoftware.com

Abstract. Ruby and Ruby on Rails have been proven as an excellent high productivity framework for the development of web applications, but has had relatively minor impact on the software development practices of major in-house commercial software development, where a Microsoft only culture usually dominates. HARD software has recently developed successful risk management and trading software using Rails as the web application platform for many clients in the energy and finance sectors whose previous IT policies dictated Windows languages on Windows platforms. This talk seeks to demonstrate the real economic advantages of Rails as a commercial web application platform in comparison to common Microsoft only IT policies and practices, how to interact with legacy infra-structures, how to deal with open source hostile IT departments and share the experiences of making Rails work for industry. Finally the presentation will explore the potential for Rails in industry and how there are significant opportunities to use this platform to considerable competitive advantage.

1. Introduction

Ruby on Rails (RoR) is a high productivity, well designed and free web application framework that is very popular for open source friendly environments, such as Web 2.0 start ups and on GNU/Linux and OSX servers. However, the typical existing commercial IT department is Microsoft only land and RoR has had relatively little impact in most established industries and IT departments.

How can we change that?

2. Why Ruby on Rails?

2.1 Why Ruby?

“You can recognise truth by its beauty and simplicity. When you get it right, it is obvious that it is right.” – Richard Feynman, Scientist

A lot of the elegance of design and simplicity of the RoR framework can be directly ascribed to the Ruby language that implements a syntax and structure that allows the implementation of advanced features with a natural and simple language. The fact that it is a dynamic interpreted script language has lead to questions of suitability for web applications from the perspectives of performance and scalability, but this is its real strength in improving productivity.

Modern script languages such as Perl, Python and Ruby have changed the game for rapid development of complex software without the time sapping memory management, type compatibility and long build cycles. Ruby's design is a huge improvement on the back-ended object oriented design of Perl and the syntactical weirdness of Python, and both have had web application environments developed for them with similar objectives as RoR but without the success of RoR to date.

“We can teach an artist to animate; we cannot take the time or expend the effort to teach him to be an artist” – Chuck Jones (1989)

Although experienced Ruby programmers have been difficult to find, this is becoming less of a problem. Skilled programmers' in an object oriented language such as C++, Java, C# or Perl/Python have no difficulty becoming proficient and productive in a short period of time.

2.2 Why Rails?

“Rails is the most well thought-out web development framework I’ve ever used. And that’s in a decade of doing web applications for a living. I’ve built my own frameworks, helped develop the Servlet API, and have created more than a few web servers from scratch. Nobody has done it like this before.” – James Duncan Davidson, Creator of Tomcat and Ant

Although a little redundant at an OSDC, it is worth noting that ruby, rails, most libraries, gems, and plugins are all open source. Whilst attendees at this conference will all see this as a good thing, others do not (refer section 3.1).

It is a good thing. No project hold ups waiting for accounts to approve the \$3000 license for a programming environment for the new guy, no cash flow issues for start ups, no delays trying out the framework for a prototype, no restrictions or delays when scaling the application to 1000 servers, no expensive third party libraries that are both costly and have run-time license requirements and fees. If you work only on open source projects you probably take all of this for granted. I don't.

“Tim Bray director of web technologies at Sun Microsystems, say that Ruby on Rails is twice as fast to develop as Java and twice as maintainable as PHP. Gartner has predicted a fourfold increase of Ruby developers to four million by 2013.” – Computerworld NZ, September 2009

RoR is well designed and imposes a disciplined and proven Model-View-Controller (MVC) design on every web application and to do anything else with the framework is wasted effort. If you don't want to use the RoR design, don't use RoR! Some of the design and practices in RoR do take some time to adjust to, but you always know that each technique works in real applications and, even though it may not be the way you usually implement that facet of your application, you do learn effective web application techniques when you use RoR.

The two significant themes in RoR that pervade every aspect of the design are 'Convention over Configuration' (CoC) and Don't Repeat Yourself (DRY). CoC effectively means that you have to stick to RoR ways of doing things, such as naming fields and source files, designing database tables and structuring applications, but the significant payoff is that you write **much** less code and it just works. RoR generates significant parts of your application such as form scaffolding, javascript and AJAX support, database models and migrations and all in reliable and robust code. Obsessive RoR unit, functional and integration testing makes sure that it stays that way.

For new projects where no existing RoR expertise exists, I would **strongly** recommend buying and reading the O'Reilly (oreilly.com) and Pragmatic Programmer (pragprog.com) Ruby and RoR texts and watch the free Railscasts (railcasts.com) **before** starting the real application. However, developing a throw away prototype application whilst you read the references, is even better and will result in significant time and resource savings when delivering a production quality final product.

ActiveRecord is one of the significant reasons that developing RoR applications works as well as it does. To illustrate the effectiveness of the design of ActiveRecord, here is an excerpt of code from

'Pursuit of Beauty' [Hansson 2006] that needs little explanation and would require a least an order of magnitude more lines of code to write in PHP.

```
class Post < ActiveRecord::Base
  has_many :comments
end
class Comment < ActiveRecord::Base
  def self.search(query)
    find(:all, :conditions => ["body = ?", query])
  end
end
Post.find(1).comments.search "hi"
```

2.3 When do you use Ruby on Rails?

RoR is not a one size fits all framework that should be used in all circumstances. However for non-trivial web applications, RoR provides a rapid application framework that does not constrain the design and scales well to create significant web services. HARD software has developed many PHP applications, including the market leading electricity bidding application, but has now adopted RoR for all new web applications and servers.

The Ruby and Dynamic Languages page on the Thoughtworks (2009) website succinctly lists the sorts of applications that they believe are most suited to RoR.

- Web 2.0 applications that make use of REST and/or AJAX designs
- Small to medium web-based applications with aggressive time-to-market goals
- Low-cost internal prototypes and pilot applications
- Highly-targeted internal applications and utility programs
- So-called "soft layer" APIs on hardened transactional systems
- Build systems for complex enterprise systems.

You wouldn't use RoR to develop your next iPhone application (although Titanium developer lets you use Ruby), but you would certainly have your iPhone application communicate with your RoR server.

3. Why not Ruby on Rails?

3.1 You never lose your job for picking IBM/Microsoft/SAP/Oracle

“Somehow, as an industry we fool ourselves into thinking market leader is the same thing as standard.” – Martin Fowler (2009)

The one defining characteristic of management over the last 20 years with regard to Information Technology (IT) is **risk aversion**. There are many case histories of career ending ambitious IT projects that have failed in very public and embarrassing ways and the board and senior management are damn sure that they are not going to be responsible for one of those projects.

Unfortunately, the net result of this conservatism is that it is difficult to get to use anything that doesn't have the 'industry standard' seal of approval and that includes open source software, non-Microsoft development languages and standard products that are sometimes very poor implementations

in comparison to OS software (Sharepoint and IIS are standout examples), and anything the board hasn't heard of, which is often a lot.

I was once involved in a meeting where I was proposing a RoR solution that was an excellent fit for the proposed application, to be told that no solution would be considered unless it was .NET, and that .NET MVC was an alternative to RoR that was acceptable to use. .NET MVC is the Microsoft 'answer' to RoR and other web frameworks and should really be called .NET VC as it has no standard database model component, although there are separate libraries available, they are not integrated into the framework (Tokumine 2009). It is an immature product with marginal community support to date and really only the .NET libraries to implement the application, and is an obvious response from Microsoft so that they can say 'we have one of those'. This was considered the **less** risky option.

The board and senior managers can not be expected to be IT experts and even IBM's recent pragmatic adoption of open source would not have been noticed at this level of management. The only solution here is to educate and persevere, and hopefully get the support of the local IT department (see section 3.2).

3.2 Dealing with the in-house IT department

The difference between the board and senior management and the IT department is that the IT department have heard of open source software, Ruby and RoR and they don't want it. Whereas management are risk averse, IT departments are typically driven by **insecurity**. IT is an ever moving field, and is difficult to be able to be conversant in all of the wide range of possible operating systems, languages and application frameworks, and you know stuff they don't.

There are some IT departments that you will never win over. You can pretty much pick them straight away when they give you the company's voluminous software development standards and none of them list anything that doesn't have the Microsoft name all over it, with the exception of Oracle, but only because the accounting system has always used it, but they will be changing to MS SQL Server as soon as the vendor supports it. They usually are run by someone with little software development experience so they are not going to let anything in the door that is non-standard as they are still struggling to cope with the standard stuff. These are prospects that are never going to change until the next coup, so move on.

Fortunately most IT departments aren't so inflexible and can be won over and can even be your most valuable allies, especially if they hate the out-sourced IT provider (if they have one), which they usually do. Senior management won't listen to you as an outsider with a glaring self interest but they will (usually) listen to their own IT manager.

The approach to the IT department should be based on technical reasons to use RoR (productivity, ease of maintenance, rigorous testing, reliability) whilst mitigating their concerns (no existing internal skills, ongoing support if you disappear, explaining open source licenses and their implications and considerable benefits). Often IT departments approach RoR with concerns that are not really relevant to open source software, such as how do I get support for the environment if there is a serious bug because there is no commercial entity providing the support for a fee? Whereas in reality you have a much better chance of a timely fix with popular OS software like RoR because the known bugs and problems are all disclosed and solutions posted by the community or can be diagnosed and fixed because you have the source. Try reporting a bug to Microsoft and see how quickly it is fixed, although active communities do exist for commercial products and can help to solve problems with a black box investigative approach.

If the manager isn't particularly enamoured with the benefits of RoR or open source software in general, then the best approach is to find an ally amongst his/her team who has a GNU/Linux or Mac computer at home to play with, likes new challenges and learning cool stuff or has developed web applications in some other environment and has experienced first hand the design and implementation issues and the real potential benefits of RoR. Help them to prepare a document or a simple prototype in a short time to demonstrate to the IT manager that the benefits are real and the fear of the unknown can be overcome.

3.3 The challenges of out-sourced IT

The out-sourced IT provider has a very different motivation. They don't want any application that is going to make them do any more work unless it is going to be **really** worth their while. I can never understand why private and public enterprises are so willing to abdicate control of their critical IT functions to an external organisation who are openly driven to maximising revenue from them. I have had proposals effectively killed on an economic basis because the IT out-sourced provider has dictated to the client that they must have separate development, test and production servers, and development, test and production database servers, each with fully licensed Microsoft operating system and other licensed products such as their server monitoring software all at an exorbitant annual fee of over \$120,000 and the client just accepts their requirements as a result of their existing out-sourcing agreement.

Again, sometimes you can quickly determine that there is no point proceeding with the client, who is usually sympathetic but unable to change or vary any of the out-source providers requirements or costs.

If there is a possibility of a non-standard approach, the most successful strategy that I have used is the 'alternative' scenario. My current favourite is to provide a simple costing of a cloud server with a well known provider such as Amazon EC2 or Rackspace using GNU/Linux, Mysql and Apache running RoR where the costs are about \$75 US / month for the production server and the test and development servers are only activated when required. That usually gets the client's attention and they start talking to you about ways of working around the present arrangements.

3.4 How to fit into the existing environment

Once you have the client's approval to implement your fantastic RoR application, you still have to deal with their existing IT infrastructure. The main challenges with implementing RoR applications within an existing IT environment are implementing on Microsoft operating systems and legacy database table designs.

Implementing reliable production RoR applications on MS servers is possible but certainly not as simple as a GNU/Linux or OSX implementation. Whereas on GNU/Linux servers and OSX an Apache/Passenger approach or one of the other popular options are simple to implement, MS servers require some messing around. We have found that the approach described in Chapter 8 'Deploying on Windows' of Zygmuntowicz et al. (2008) has been the most reliable production implementation for Windows servers using a pack of mongrels and mongrel services, but note that the mongrel service gem doesn't seem to be being actively maintained and doesn't presently support the latest mongrel versions which causes some complications with the server configuration.

RoR expects database tables to be designed and named using a fairly prescriptive convention although it does have good options to deal with most of the non-rails table designs. The most difficult

problem is that Rails uses a single auto-incrementing ID field as the primary key for Rails tables and there are a lot of tables out there that have composite primary and foreign keys. There is a very good rails gem that enhances ActiveRecord to deal with composite keys, but it does still lead to some complications especially with SQL Server, which is the poorest supported popular database for Rails.

4. Conclusions

It is naïve to suggest that it is always possible to convince a prospective client that RoR will deliver a timely and cost effective web application that will be much cheaper to develop, more reliable and more easily maintainable than presently possible with the in-house development standard. However, with a persistent, patient and educative approach it is often possible to convince the client's IT department and then subsequently their board and senior management that RoR is the IT path to application development enlightenment. HARD software has successfully developed risk management and trading market data systems and deployed them in previously Microsoft only IT environments, and you can too.

References

- Fowler C (2009), "The Passionate Programmer: Creating a Remarkable Career in Software Development", The Pragmatic Bookshelf, 2nd Edition.
- Hansson D H (2006), "Snakes & Rubies: Pursuit of Beauty", www.rubyonrails.org,
<http://media.rubyonrails.org/presentations/pursuitofbeauty.pdf>.
- Jones C (1989), "Chuck Amuck; The Life and Times of an Animated Cartoonist", Chuck Jones Enterprises Inc., 1st Edition.
- ThoughtWorks (2009) "Ruby and Dynamic Languages, How We Do It", ThoughtWorks,
<http://www.thoughtworks.com.au/how-we-do-it/ruby.html>.
- Tokumine S (2009), ".NET MVC vs Ruby on Rails", blog, <http://www.tokumine.com/2009/03/21/net-mvc-vs-ruby-on-rails>.
- Wellington M P (2009), "Opinion: Why Ruby on Rails is a game changer; Ruby adds speed and boost developer productivity", Computerworld NZ, 7 September 2009.
- Zygmuntowicz E, Tate B & Begin C (2008), "Deploying Rails Applications; A Step-byStep Guide", The Pragmatic Bookshelf, 1st Edition.

Biography

Dr. Harley Mackenzie has worked in the risk management, aviation software and the energy industries, with a background in engineering and mathematics, and has been the owner and managing director of HARD software for the last 10 years.

HARD software is an independent consulting and software development company that specializes in the systems analysis, design and implementation of technical software. Harley has developed many trading and risk management applications in the energy and finance industries mainly using open source languages such as Perl, PHP and Ruby and is a member of OSIA.