

RUBY ON RAILS: A Simple Decision?



Jocelyn Amon

2008

Terms of Reference

As part of the Bachelor of Information Technology course at Nelson Marlborough Institute of Technology, students are required to complete a third year project. This report presents the findings of the research undertaken by Jocelyn Amon into the Ruby on Rails technology and forms part of the overall project. Dr Clare Atkins has been assigned as project supervisor.

As an information technology research project, the focus is on researching an aspect of IT of interest to the researcher and, preferably, of interest to a wider audience. The project will provide an opportunity for the author to integrate many of the skills acquired while studying at tertiary level and will impart an understanding of Ruby on Rails and its future in web development.

The project report is required to be completed by 10 November 2008. Project presentation has been set for 21 November 2008.

Abstract

The Ruby on Rails web application development framework was released to the general public in 2004. Since then it has made a significant impact within the web development community and has been the catalyst for debate amongst developers more conversant with the traditional development frameworks. The exponential growth of websites devoted to Rails development and the recent proliferation of publications on the subject, confirm that this newer technology has captured the interest of a large segment of the web developer community. Many of these developers and their management have an interest in knowing whether or not to adopt this new technology. This study describes the features that differentiates Ruby on Rails from previous web technologies and discusses the issues that require consideration in its adoption.

Traditionally, information systems research has adopted a positivist epistemology and therefore a quantitative approach is favoured in the selection and analysis of information. This paper argues that since information systems are a social construct, a greater emphasis on qualitative evaluation is justified. Therefore, while some secondary quantitative data are presented, the majority of the findings are of an interpretive nature and an analysis of the information contained in web forums, blogs and articles is presented and evaluated.

A background outlining the relevancy of the study is given and the research approach used is discussed. An overview of the Ruby on Rails framework is followed with a discussion of the technology's unique features and philosophy. Using a primarily interpretive approach, the researcher concludes that there is demand for simpler and faster web development frameworks. Ruby on Rails meets the web development community's requirements by simplifying and integrating functionality. The unique features and design philosophy built into Rails has raised the bar for web development tools and developer productivity. Competing technologies must now match these expectations and, in many cases, have already done so by implementing a 'Rails-like' design. Developer enjoyment in their work

emerged as an important theme; an aspect seldom given consideration in the evaluation of development software.

Adoption of new technology is a complex issue and influenced by many factors, a deeper understanding of which is beyond the limitations of this study. Regardless of whether Ruby on Rails is on track to becoming the foremost web development framework of choice, it has already been recognised as significant and has initiated a paradigm shift within the web developer community.

Contents

1	Introduction	1
2	Background.....	3
2.1	The Web Development Paradigm.....	3
2.2	Research Approach	4
2.3	Academic Literature.....	6
3	Ruby on Rails.....	8
3.1	Overview.....	8
3.2	The Ruby Language.....	8
3.3	Model-View-Controller Design Pattern.....	14
3.4	Code generation	16
3.5	ActiveRecord	17
4	Rails Philosophy	18
4.1	Keeping it simple	18
4.2	Convention over Configuration	18
4.3	Don't Repeat Yourself	19
4.4	Agile Development	20
4.5	Open Source Software	21
5	Implementation and Development	23
6	Software Evaluation.....	26
6.1	Programming Environment.....	26
6.2	Suitability of Purpose.....	27
6.3	Productivity.....	30
6.4	Arguments Against	30
6.5	Dynamics of Software Adoption	32
7	Conclusion.....	34
8	Glossary.....	37
9	References	39

Table of Figures

Figure 3-1	Tiobe Programming Community Index.....	10
Figure 3-2	Dynamic versus Static Typing.....	12
Figure 3-3	Model-View-Controller (MVC) Design Pattern.....	14
Figure 3-4	Ruby on Rails MVC Implementation.....	15
Figure 3-5	Rails Directory Structure	16
Figure 5-1	InstantRails Console	24
Figure 6-1	Twitter.com and Yellowpages.com visitor analysis.....	29

1 Introduction

The many favourable reviews published since the 2004 release of Ruby on Rails (aka RoR or simply, 'Rails') has provoked interest from web developers seeking better tools and managers requiring business advantage. Much of this positive response originates from respected luminaries in the computing world (RubyonRails.org, n.d.):

Rails has become a standard to which even well-established tools are comparing themselves to. -- Martin Fowler

Ruby on Rails is a breakthrough in lowering the barriers of entry to programming. Powerful web applications that formerly might have taken weeks or months to develop can be produced in a matter of days. -- Tim O'Reilly

Before Ruby on Rails, web programming required a lot of verbiage, steps and time. Now web designers and software engineers can develop a website much faster and more simply, enabling them to be more productive and effective in their work. -- Bruce Perens

These and similar comments give good reason to investigate Rails further. For many it was the book *Beyond Java* (Tate, 2005) that became the catalyst for the re-evaluation of the web development paradigm. The book's author, Bruce Tate, has gained a reputation as a long-standing Java advocate and web tools developer. Much interest and controversy was generated when he instigated a Rails crusade and questioned Java's dominance in the market. Tate put forward a strong argument that Java fails to deliver effective solutions for the web and declared that applications built on Java-based frameworks were becoming increasingly complex and arcane.

So, is there a need for the web development community to sit up and take notice? Will Rails become a leading web technology? What criterion is valid in assessing *this* new technology? These are some of the questions that developers and management need to consider if they have an interest in optimising competitiveness, productivity and effectiveness in the increasingly fast moving environment of the web.

While time limitations precludes an in-depth discussion on all aspects of the downside of Rails technology adoption, this paper attempts to address some of the more significant issues, in particular, whether Ruby on Rails has a major role to play in the future of web application development.

2 Background

2.1 The Web Development Paradigm

Demands placed on web applications and, consequently, designers and developers (often one and the same) have resulted in a plethora of technologies, tools, frameworks and architectures that frequently serve to further complicate this field. Unlike conventional software, applications designed for the web face additional concerns:

- data persistence issues across various platforms
- increased sensitivity to response times
- increased security vulnerability
- greater emphasis on attractive, sophisticated and accessible user interfaces
- raised customer expectations

A few years back, the ability to create an archetypal ‘brochure’ style web site would merely entail knowledge of HTML and perhaps a scripting language. Today’s web environment has become increasingly complex, placing a heavy demand on the developer to work with many technologies and concepts such as:

- management and updating of web content
- web site user interface design: usability design, style, colour, branding, marketing, navigation, cascading style sheet protocol
- personalisation, internationalisation and localisation
- user management: security, registration, login
- e-commerce: shopping carts, payments processing
- adherence to industry standards: browser issues, W3C compliance, business standards
- application development: design, dynamic content generation, database interaction, data persistence, prototyping, future-proofing, testing, deployment, maintenance, scalability, web server integration
- separation of concern: model, view, controller (MVC)
- legal issues: confidentiality, intellectual property, disclaimers, advertising, privacy
- paradigm shifts and related technologies: Web 2.0, web services, SOAP, REST, AJAX, XML, Agile
- integrated development environments (IDE)
- validation of data and business rules

In the journal article, *Characteristics of Web Development Processes* (Lowe & Henderson-Sellers, 2001), the differences between conventional and web

development are explored. The writers identify that the most obvious difference is in the technologies used and the ways in which they interconnect with web development, requiring the merging of sophisticated business requirements, complex information architectures and highly component based technical environments. The resulting linkages between technical and business architectures are much tighter and the information architecture requires a higher level of sophistication than for conventional systems. It was determined that web systems have unique characteristics which are poorly addressed by conventional development practices.

There are many disparate tools and frameworks available to assist in managing web development tasks. Developers include several of these technologies within their toolbox resulting in a compounding of complexity. Traditionally these have been based on the PHP, Java and Visual Basic programming languages. The environment resulting from newer technologies such as AJAX, SOAP, REST, XML, Web services and other Web 2.0 associated innovations, further complicate web development. As application complexity increases, so does the need for effective development tools. A single technology, encompassing most needs, would assist in simplifying development by providing a homogenous development platform and design philosophy.

The response of many within the industry would indicate that Ruby on Rails may meet this need for a better development technology. This paper presents an exploration of this possibility.

2.2 Research Approach

Research in the field of information systems has traditionally taken a positivist approach through the application of quantitative methodologies using measurable data. Typically, software development has often been evaluated using quantitative methods through measurement of lines of code generated. This approach often gives a misleading view of productivity and ignores issues of usability, maintainability, flexibility and suitability; factors that are not easily measured. Additionally, a positivist, quantitative approach to research can overlook the human aspect of software creation.

A qualitative approach to research allows for a more interpretive and therefore subjective analysis. Myers (2008) notes a shift in Information Systems away from the technical and more toward the human issues of management and organisations, resulting in an increased interest in the application of qualitative research methods.

During initial investigation into Ruby on Rails, a qualitative approach emerged as a valid and effective strategy. Although it is possible to gather empirical data on technology uptake, productivity gains, deployment statistics and technology comparisons, this form of analysis gives only part of the picture. The Rails technology appears to be more about philosophies and how people function than being just another technology. Therefore, rather than drawing up a list of criteria for web development applications then measuring how Rails meets them, it would seem more appropriate to look at the Rails paradigm holistically, noting differences from what has gone before.

The passion within the Rails community is a noticeable characteristic and cannot be disregarded if any real evaluation into this framework is to be achieved. From its inception, it has been the aim of Rails' creator to produce a better technology through the implementation of his strongly held views of how a web development tool should work. Programmer response to Hansson's (2004) call to evangelise the technology, has brought Rails to the attention of management and the rest of the developer community, producing a great deal of interest.

Unlike most successful technologies, Rails did not have large corporate backing; the media and industry buzz resulting from user passion has been essential for Rails' promotion. Terms such as 'obsession' and 'passion' are not usually a consideration in research into computer technology and it is rarely (if ever) a factor in quantitative research. Themes frequently encountered when researching Rails such as 'beautiful code', 'fun programming', 'the joy of code' and 'beauty in design' are indicative of a subjective response from the programming community and the feasibility of an interpretive approach to this research.

For those working in the field of creating artefacts, the tools of the trade are integral to the work undertaken. This applies whether it is a tradesperson crafting utilitarian objects or an artist expressing creativity. It is possible to contend that a software developer is both and is just as passionate regarding their choice of programming language and development framework. Research methods that allow for an

interpretive analysis allow the human dimension to become a factor in the research and make the eliciting of alternate areas of insight a possibility.

A primarily qualitative research method was also considered suitable given the newness of the technology and the lack of Rails sites and Rails users available for study locally. Conversely, there is a vast amount of documentation on the technology available on the Web. Also, the publication of Rails texts has seen exponential growth over the past year. From less than a dozen Rails related books available in late 2007, there are now well over forty.

It is the intention of this research to describe the facets of Ruby on Rails which make it unique and those features which have inspired the enthusiasm of the developer community. Issues and areas of debate are also a consideration. An interpretive approach is applied in viewing the available literature to discern common themes. The literature for evaluation includes books, magazines, scholarly papers, blogs and forums. It is recognised that a high-level of subjectivity is inherent in the choice of which texts to include and what information within those texts to highlight. However, it is felt that this approach will provide a more rounded picture than a quantitative approach which can only show what has, what is and be indicative of what may happen; these methods cannot determine technology peaks. Triangulation using both quantitative and qualitative methods would have been the ideal but time limitations preclude this.

The prime focus of research effort has been in the reviewing available literature, determining major themes, deciding relevancy and describing some of the more unique Rails features.

2.3 Academic Literature

An internet search revealed minimal academic exploration into the significance of Ruby on Rails in the field of web development. This is possibly due to the relative newness of the technology but given a wide-spread interest and Rails' enthusiastic adoption by high-profile web development experts, a lack of published research was not expected.

An early evaluation of the technology was incorporated into a study of web technologies by Casal (2005). This research focuses on web development frameworks and how they relate to the Semantic Web and Higher Education. In his

comparison of leading edge web frameworks, Casal evaluates and compares several of the more popular technologies; Apache Struts, Spring, Tapestry, Java Server Faces (JSF), Cocoon and Ruby on Rails. In particular, Cocoon and Ruby on Rails were chosen for an in-depth analysis. The study rates Cocoon highly as being extremely powerful and “in the forefront of current thinking” in web tool design. However, a question is raised regarding the high level of programming knowledge that is required to use it. While acknowledging Rails’ ability to get a solution up and running quickly, Casal raises concerns of resistance from the Java community, scalability and enterprise readiness.

Guzewich, Kent, Pfeifer and Shank (2006) identify six key quality attributes of Rails which they rank as follows: simplicity, reusability, extensibility, testability, productivity and modifiability. The authors’ case study describes tactics that Rails’ employs in successfully achieving these goals. In summarising, they mention that limitations in PHP and J2EE (Java) became the motivating force behind Rails and that the goals set in its development have “certainly been met”. They conclude that Rails is ideal for small to medium web projects using small teams and Agile processes. The theme of technology simplicity is stressed and it is noted that, using Rails, complex applications can be difficult to implement but that this would be no different with an alternate framework.

Deckert (2006), a computer science student, evaluates the Rails technology from a ‘hands on’ point of view. The technology was found to be easy to use and robust but he suggests more work could be done in removing code redundancy. Again, Rails’ use for small to medium applications is recommended with scalability issues being a possible area of concern for larger developments.

Further student case studies and evaluations were also retrieved but these mostly covered the same ground. There appeared to be a consensus that the technology is easy to grasp and productive. In several cases it was noted that Rails is a joy to work with.

Due to the lack of scholarly papers, the majority of the information sources drawn upon for this study has been through the more informal sources available on the web.

3 Ruby on Rails

3.1 Overview

Ruby on Rails is an open source framework written in the Ruby programming language and is one of the newest technologies available for building dynamic web applications. Danish computer scientist David Heinemeier Hansson's requirement for a rapid application development (RAD) tool targeted specifically at web development, led to his development of the Ruby on Rails framework. The framework evolved from Hansson's very successful web-based project management software, Basecamp.

Following the 2004 release of this technology by Hansson's company, 37signals, Rails has been the subject of wide interest and debate among developers. Much of the controversy was prompted by the 'opinionated' nature of the software as well as the doubt that often surrounds innovations that deviate from traditional thought. The frameworks inherent simplicity encouraged some to view Rails applications as 'toy' software, incapable of scalability into real world enterprise environments.

Rails will run on a wide-range of operating systems (including Linux, Macintosh OS X and Microsoft Windows) and is a 'full-stack' solution for web development, meaning that it provides all the necessary elements required to produce a fully-fledged web application. There is no need to bolt-on third party software, therefore eliminating the complexity involved in introducing disparate conventions, programming languages, interfaces and philosophies.

In creating the technology, Hansson's goal was to provide software that would satisfy 80% of the potential market. By omitting seldom used functionality, a simplified product, suited to a large majority of developers, was produced.

The Rails framework has been built using the Ruby programming language and incorporates Hansson's strong views on how web applications should work. Some of the most significant and unique features of the Ruby on Rails framework are described in this section.

3.2 The Ruby Language

In reality, Rails technology is best termed 'Rails on Ruby', rather than 'Ruby on Rails'. It is the Ruby language that makes possible many of Rails' unique strengths.

The open source interpretive Ruby programming language was released to the public in 1995 by its creator, Yukihiro Matsumoto. This relatively new language is a blend of earlier object-oriented languages and is interpretive (i.e. not compiled), fully object-oriented (more so than Java), dynamic (can grow code ‘on the fly’) and reflective (can change during execution). Not only has it been used for the development of the Rails framework, it is also the language used when developing applications in Rails. The rise in popularity of Rails has been mirrored by that of Ruby. The language has experienced a steady rise in programmer popularity since Apple’s announcement in 2006 that Rails would be shipped with the Mac OS X Leopard (Macintosh News Network, 2006) and in 2007 with leading programming tools developer CodeGear’s release of an integrated development environment (IDE) for the Rails platform (dBusiness News, 2007).

Design Philosophy

In describing his design approach, Matsumoto (2000) explains the importance that joy holds in his life. One of his aims in creating Ruby was to facilitate the joy programmers feel when they are concentrating on the creative side of programming. This aim is achieved through implementing the principles of conciseness, consistency and flexibility in the Ruby language. He views a programming language as a user interface and so follows user interface best practice in his design strategy. With Ruby, the programmer can spend less time fighting the language, with more of the effort being put toward application design.

A feature of Ruby that makes it easy to work with is its compliance with *The Rule of Least Surprise* as described by Raymond (2003). In a treatise aimed at Unix developers, Raymond observes that software having a connection to user’s pre-existing knowledge is easier to work with. By paying attention to the traditions of the target users, avoiding “gratuitous novelty” and “excessive cleverness”, usability is improved.

Industry Acceptance

Independent of its links to Rails, Ruby has gained recognition and many developers view it as superior to the more traditional coding languages.

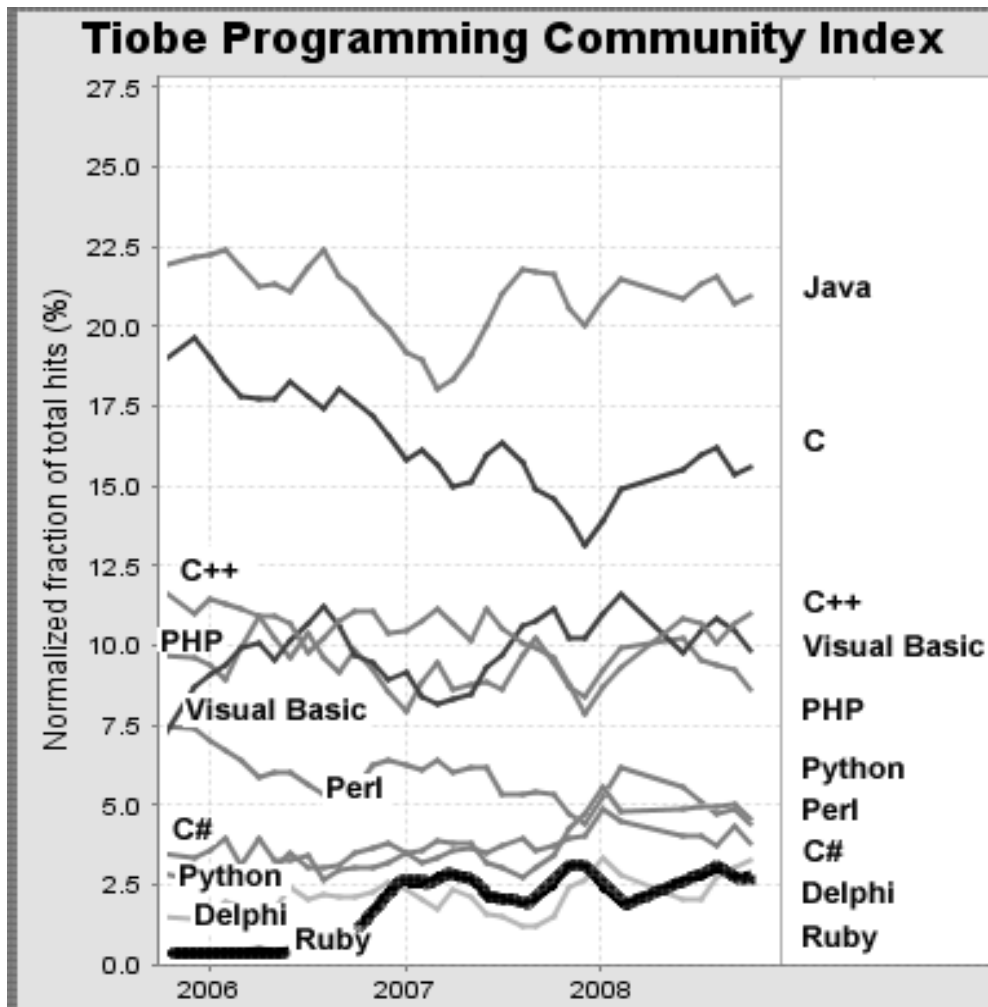


Figure 3-1 Tiobe Programming Community Index (Tiobe, 2008)

The graph in Figure 3-1 shows Ruby has increased in popularity since the release of Rails in 2001, based on the Tiobe Programming Community Index^{*}. Ruby recently edged Javascript from the top ten, putting Ruby on a par with Delphi (aka Object Pascal) and closing in on Python and Perl. It also competes well with the Microsoft-backed C# language.

^{*} The Tiobe ratings are based on the number of skilled engineers world-wide, courses and third party vendors. The popular search engines Google, MSN, Yahoo!, and YouTube are used to calculate the ratings.

There are several factors that make Ruby a good fit for the Rails philosophy. One of these is that, unlike the majority of programming languages, it is closely aligned to the English language. To illustrate this, the following line shows the ubiquitous ‘Hello World’ example implemented in Ruby:

```
puts "Hello World!"
```

The same code in the leading programming language, Java, is shown below:

```
public class HelloWorld {  
    public static void main (String [] args)  
    { System.out.print("Hello World!"); }  
}
```

As can be seen, the Java code is more verbose and is not as easily interpreted as the straight-forward style used with Ruby. It can also be noted that Java requires braces to define each block of code and semi-colons to end each statement. This requirement can be the cause of annoyance for coders, requiring them to remember every instance and dealing with the consequences when they are inadvertently left out. Ruby does not force the developer to adhere to unnecessary rules. The language includes many other features that accelerate and simplify coding.

Dynamic versus Static Typing

Another factor differentiating Ruby from Java is the object typing method implemented. The norm for object-oriented programming languages has been to statically type data items. That is, to assign a data type to each data variable. This provides for better validation, ability to optimise storage and provides a form of documentation.

Ruby’s dynamic typing approach does not impose this requirement, making it possible to define variables ‘on the fly’ and to easily change the data type as required. This can increase the chance of data conflicts causing run-time errors (e.g. using a string value in a mathematical function) but it provides gains in decreased programming time and code flexibility.

Those advocating dynamic typing argue that a competent programmer implementing good unit testing practice should not have to pay the price of slower coding and loss of flexibility by being straitjacketed with static typing. They argue for ‘duck typing’, a popular term which derives from the view that ‘if it walks like a

duck, quacks like a duck, then it is a duck’ - therefore, presumably, one does not need to be told that it is, in fact, a duck. While there are claims that static typing assists with documentation, the use of dynamic typing improves readability through conciseness.

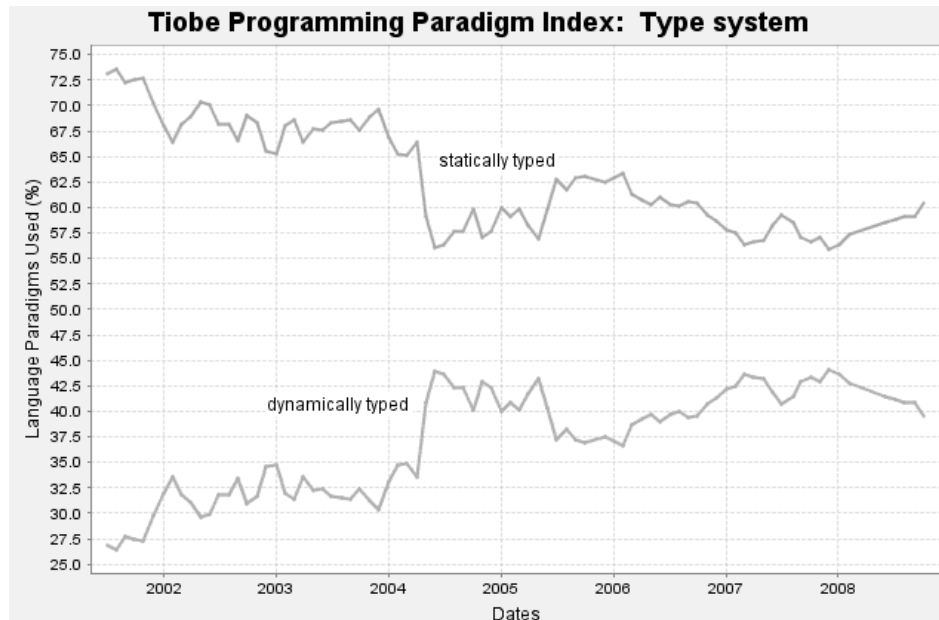


Figure 3-2 Dynamic versus Static Typing (Tiobe, 2008)

The Tiobe graph in Figure 3-2 shows the closing of the gap between programming languages that use static typing and those using dynamic typing. This rise in the popularity of dynamic typing languages is attributable to the popularity of Ruby and similar languages such as Python and Perl. The shift is noted in this excerpt from the CIO magazine website article:

The former second-class citizens of the programming world have leaped to the fore, changing the face of enterprise software development. With the rise of Web 2.0, scripting languages (also called dynamic languages) are now often considered important tools in a developer's arsenal. (CIO Magazine, 2008)

Ruby's dynamic capability has enhanced its reputation for flexibility. It is this characteristic that has enabled the development of the application specific language that forms the Rails development environment. By allowing 'syntactic sugar' to be added to the language, Ruby has been evolved to include concepts defined within the Rails framework to improve code readability. This feature allows Ruby to be extended to provide a Domain Specific Languages (DSL). It is this moulding of the language environment that contributes much to Rails' simplicity and clarity by

providing code that uses meaningful language components. For instance, when modelling a data table, the class can contain elements such as ‘has one’, ‘has many’, ‘validates_uniqueness_of’ etc.

Flexibility

Dynamic typing coupled with the interpretive benefits of Ruby provides the flexibility required for strategies that involve reflective and meta-programming techniques. The ability for programs to change dynamically at runtime provides for the insertion of new methods, creation of new classes as well as running new code submitted as a string. This capability vastly extends the power of the Ruby language and makes it the ideal choice for Rails development.

However, it is the interpretive nature of Ruby that has been the reason for some criticism of the language. Unlike Java, it is not precompiled and therefore must be interpreted at runtime; a feature necessary to allow for dynamic typing capability. There is much debate for and against the use of interpretive languages for real world programming tasks, the main argument against being slowness of execution. With improved hardware, speed has now become less of an issue. An added benefit of interpretive languages is the increased productivity achieved through the gaining of instant feedback as code is written.

Beauty in Simplicity

‘Beauty’ is a term often applied to Ruby because of its inherent simplicity and intuitiveness. Hansson (as cited in Black, 2006, Foreword section) describes the language as being “incredibly rich and expressive” and refers to it as having “beauty”.

Beauty as an attribute of software and its relationship to complexity is a theme recognised by author David Gelernter, professor of Computer Science at Yale University. In his book, *Machine Beauty: Elegance and the Heart of Technology*, he argues the importance of beauty as an engineering concept and how it is applied as “the ultimate defence against complexity” (Gelernter, 1998).

Gelernter contends that any attempt to use mathematics to reduce complexity is doomed to failure. It is only through the holistic view of a problem and its solutions that the multifaceted nature of the real world can be managed. This appears to be much of the power behind Ruby and, subsequently, Rails.

3.3 Model-View-Controller Design Pattern

The Model-View-Controller (MVC) development architecture describes the method of structuring code to achieve a 'separation of interest' between data (the model), behaviour (the controller) and user interaction (the view). It gained broad acceptance following the publication of the MVC as a design pattern by Fowler (2002, p. 330) and is widely regarded as the ideal approach when developing GUI applications and, in particular, database-driven web applications.

The separation of layers theoretically allows any single layer to be changed without affecting the others. This enables the underlying database architecture to be substituted or the user view to be transformed, with very little impact on other application components. Applications that do not follow the MVC pattern tend, for example, to include functionality through the use of scripting code within the presentation layer (HTML). A tightly bound relationship between layers complicates issues of enhancement, debugging, scalability, flexibility and deployment.

The MVC pattern has been incorporated into the majority of contemporary web development frameworks but interpretations tend to differ. Variations occur mostly in relation to the degree of coupling between layers and in the enforcement of layer separation. The diagram in Figure 3-3 shows the MVC design pattern as it is most commonly applied within GUI developments.

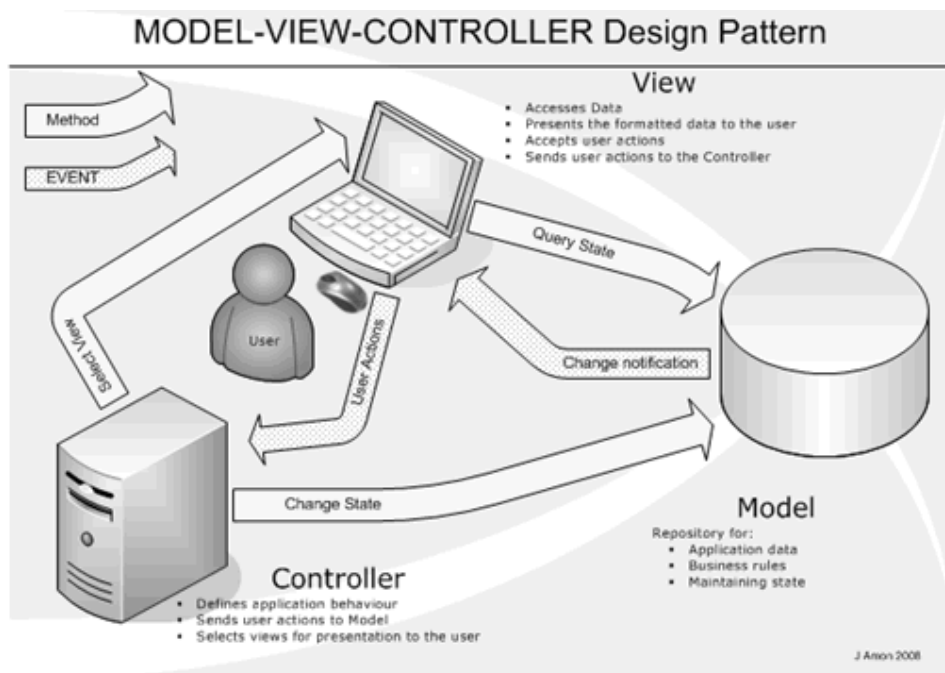


Figure 3-3 Model-View-Controller (MVC) Design Pattern

3.4 Code generation

Rails will facilitate the acceleration of development by generating application code from the database schema. This ‘scaffolding’ capability substantially reduces time taken to obtain a working application. Even though this standard version will display the typical Rails look-and-feel and will not be fully featured, the customer is able to view a working prototype and give their input into the development at a much earlier stage than is usually possible. The iterative development environment that this enables is a step up from ‘the big bang theory’ where the customer only sees their application once all design decisions have been made and built into the software.

The scaffold code files are held in a directory structure that follows the MVC design. These folders, named ‘models’, ‘views’ and ‘controllers’ are automatically generated and named at the time the application is first created. Individual code files are created within each of these folders for each model (class or table) or distinct functional requirement. These files are also automatically generated and named by Rails, ensuring that the Rails conventions for file naming and location are adhered to. The enforcement of these conventions allows Rails to locate needed components automatically (reducing the coding overhead) and also assists the developer in finding wanted code. Figure 3-5 shows the directory structure and code files for a Rails application.

```
    Rails Directory Structure (Partial View)
/app
  /controllers
    application.rb
    account_controller.rb
    blog_controller.rb
    comments_controller.rb
  /models
    user.rb
    blog.rb
    post.rb
    comment.rb
                                /views/
  /account
    signup.rhtml
    login.rhtml
    edit.rhtml
  /blog
```

Figure 3-5 Rails Directory Structure

As each new table, model and/or class is specified, relevant files are generated that contain 'stubs' for the required code. This code will run 'as is' and the developer need only modify the code if the required processing differs from standard requirements.

3.5 ActiveRecord

ActiveRecord is the Rails implementation of the Object Relational Mapping (ORM) pattern. This pattern is defined as "an object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data" (Fowler, as cited in Orsini, 2007, p. 49).

With ActiveRecord, Rails' has added to the pattern by solving two significant limitations. The first is the issue of inheritance which entails the basing of a new class upon an existing one. This is achieved through the integrating of another design pattern describing single table inheritance. The second problem that has been solved is one of associations; handling the relationship between classes. To address this, Rails introduces macros in the form of domain specific language.

Through interrogation of the database tables and business objects, the ActiveRecord component is able to present logic and data in a unified package as a persistent domain mode. Persistence relates to the outliving of data beyond the life of the application that created it. For web applications this can be a complex issue and there are many different strategies that can be applied; the ActiveRecord implementation simplifies these issues for the programmer.

4 Rails Philosophy

4.1 Keeping it simple

In developing Rails, Hansson's philosophy of keeping things simple has improved productivity and enjoyment in the programming task. The importance of simplicity and its role in the computer programming discipline is reflected in the observations of leading practitioners in the field of computer science (as cited in Ringgaard, n.d.):

The key to performance is elegance, not battalions of special cases. The terrible temptation to tweak should be resisted unless the payoff is really noticeable. -- Jon Bentley and Doug McIlroy

Simplicity and elegance are unpopular because they require hard work and discipline to achieve and education to be appreciated. -- Edsger W. Dijkstra

Fools ignore complexity; pragmatists suffer it; experts avoid it; geniuses remove it. -- Alan Perlis

Hansson's philosophy of pragmatism and simplicity is a fundamental characteristic of the Rails framework which he has built using the 37signals 'Getting Real' philosophy encompassed in the following tenets (37signals, 2006):

- Skipping all the stuff that represents real (charts, graphs, boxes, arrows, schematics, wireframes, etc.) and actually building the real thing
- Using less: less software, less features, less paperwork
- Staying small and being agile
- Starting with the interface: get the interface right before getting the software wrong
- Using iteration and lowering the cost of change
- Delivering only what is needed

This philosophy is ideally suited to web based applications, allowing them to evolve on a day-to-day basis unlike traditional software which ships in a box and is restricted to yearly (or longer) upgrades.

4.2 Convention over Configuration

In a Rails application, the convention over configuration precept is integrated into all three layers of the MVC architecture. This pragmatic philosophy is most apparent in the linkage between the model and the underlying database, simplifying

the connection between the two and thereby facilitating any switch of database engine.

A typical Rails configuration specification is shown in the following code block:

```
adapter: mysql
database: myblog_development
username: admin
password: pswd123
host: localhost
```

The majority of the code shown above is generated by Rails when the programmer first begins coding an application. By following Rails configuration conventions, as in the specification above, information is able to be automatically derived and the database table and associated class or model is able to be automatically connected to relevant views and controllers. Understanding the rules and working within the constraints requires less work. Specialised code need only be written for less conventional requirements.

A convention unique to Rails is the use of pluralisation in linking a model to a table. For example, naming a model 'person' will automatically link it to a table called 'people' in the database.

4.3 Don't Repeat Yourself

The principle of 'Don't Repeat Yourself' (DRY) aims at reducing code duplication thereby increasing productivity, reducing errors and facilitating maintenance and refactoring. As an example, Rails will assume that the name for the data model class is the same as that used for the database table and therefore the need for mapping between the two is eliminated. As is often the case with Rails, the programmer can still choose to 'break the rules' and do things differently but the Rails convention is most often the more intuitive (therefore conventional) and breaking the rules incurs extra work and is therefore not encouraged.

In conjunction with the convention over configuration philosophy, the DRY principle enables the Rails framework to derive information directly from the underlying data store. Simply by adding a field to the database, it will then be available for display or manipulation on the browser. If an alternate browser display format is desired from that which has automatically been generated, then this can be

added to the application using CSS specifications. If the field is not required on any or all views, then the relevant generated code can be deleted - removing code requires less effort than coding it in the first place.

The simplicity of the Rails solution makes for ease in refactoring, as generally the code need only be adjusted in one place. To overcome the complexity of such changes in ASP .NET and Java, specialised functions are built into the IDE to propagate changes that require renaming at many different points in the code. While these functions may be useful, they often involve the developer ‘fighting’ with automated refactoring whenever the function makes unintended adjustments.

The DRY approach is also applied to system documentation, with the code acting as the primary reference source rather than producing separate system documentation. This is made possible through the readability of Ruby (using DSL) and the Rails high level of abstraction. Eliminating the need for maintaining separate documentation ensures that synchronisation problems and currency issues are minimised.

4.4 Agile Development

Agile software development methods encompass various methodologies that promote rapid work methods while delivering a product that meets customer requirements. The Agile philosophy evolved over a number of decades but was defined in 2001 through the documenting of twelve principles, known as the Agile Manifesto. Seventeen authors from a variety of computing backgrounds contributed to the defining of these principles (The Agile Manifesto, 2001) which is summarised as encompassing the following four values:

- individuals and interactions over processes and tools
- working software over comprehensive documentation
- customer collaboration over contract negotiation
- responding to change over following a plan

To succeed in today’s turbulent business environment there is a need for relentless innovation (Highsmith, 2001). This aspect of web development was also noted by Lowe & Henderson-Sellers (2001) in their finding that “a good initial architecture should allow growth to occur in a controlled and consistent manner.” The majority of earlier software development practices involved detailed specifications and a

‘water-fall’ approach. These practices discourage re-work, are a hindrance to innovation and delay software delivery. Too often the users were unable to see the end result until the software was fully complete, by which time the market may have moved, requirements may have changed or misunderstandings may have been hard-wired into code.

An emergence of Agile technologies was the development community’s response to the inadequacies inherent in the more traditional development tools. These newer methods include Scrum, Crystal Clear, Extreme Programming, Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method.

Programmer Dave Thomas, co-author of the Agile Manifesto and author of several well-received programming texts, establishes in his book, *Agile Web Development with Rails*, that “Agility is part of the fabric of Rails” (Thomas, 2007). He describes how Rails conforms to Agile objectives by meeting the needs of individuals and interactions. Rails’ simplicity leads to transparency and ease of interaction between development team members and between developers and customers. Rather than eliminating documentation, Thomas explains that Rails has been designed so that the source code will form the basis for generated documentation.

The adherence to DRY principles, a convention over configuration approach, the Rails scaffolding ability and Ruby’s immediate feedback provide the tools and an ideal platform for the rapid development of high quality software. The provision of an operable version of the software early in the development cycle and an allowance for incremental development, affords the prospect of a project management approach that encourages early customer inspection and involvement. It also enhances the ability to respond to customer requests for change and adaptation. The principle of incremental development in small steps is a characteristic of both the Agile philosophy and Rails development.

Rails’ instant feedback ability facilitates collaborative development. Simple adjustments can be made ‘on the fly’ with the user seeing the immediate result.

4.5 Open Source Software

The open source movement has its origins in the rise of the internet and the availability of free software distributed via bulletin boards. In many cases, the

underlying code was also provided, allowing developers to bug-fix and enhance the software. Often the improved code was then provided to the software author, enabling these enhancements to be used in improving the original product.

This philosophy of freely providing software and the underlying code gained further traction with the release of the Linux operating system followed by programming languages such as Perl, PHP and Java. The defining feature of this type of software is the collaborative development environment that it fosters and the dedicated following that it inspires within the user community.

Both Ruby and Rails have been released into the public domain based on the open source philosophy of openness and sharing. The initial development of the Rails framework was controlled by Hansson enabling him to “secure one vision and establish one culture” (Hansson, as quoted in Builderau, 2006). Since then a more collaborative approach has been adopted in keeping with open source ideals. Although there remains a small team who carefully control changes to the actual Rails software, Hansson states that there has been an explosion in contributions from dedicated programmers, many of them stretching Rails’ current capabilities into new directions. The refinement of Rails can be attributed to the open-source nature of the development process which allows programmers to examine and have input to the process.

The sense of community derived from contributing to the common good and the ability to influence the development of the Rails’ product has added to the enjoyment factor for many programmers. The programming paradigm need no longer be one of working in solitude; for many, it has become one of sharing, support and human interaction.

5 Implementation and Development

To gain an understanding of the concepts and structure behind the Rails framework, hands on experience was considered useful. For this exercise, the Rails software and an IDE were installed onto a Windows XP professional operating system. Several tutorials were then followed to gain an overview of the Ruby language, Rails framework and the level of effort required to implement functionality. A description of the software used and the process followed is set out below.

Rails Installation

The Rails software is available in several forms and in various versions. If required, Ruby can be loaded and implemented independent of Rails. Rails can then be loaded separately. For the purposes of this exercise, it was decided to use InstantRails which includes all the components necessary to develop and run a Rails application. Additionally, it also includes example applications which speeds up the learning process as these can be examined and ‘tweaked’ to gain an understanding of the workings of Rails.

Initially, the latest release of InstantRails (version 2.0) was installed. Because this release had compatibility issues between the Rails version and the example applications included in the release, it was decided that the previous release of InstantRails (version 1.7) would be used instead. This version incorporates Rails version 1.2.3 running on Ruby version 1.8.6.

InstantRails bundles and configures open source components into the one download: an Apache web server, the Rails framework, the Ruby programming language and the MySQL database software. The web server allows web applications to be run and tested in a standalone environment.

The single 60 megabyte file download was obtained online from RubyForge (http://rubyforge.org/frs/?group_id=904). This file unzipped into the required directory structures and no further configuration was required to run the example applications. It was found useful that none of the InstantRails components required any Windows registry configuration as this makes the process of installation and software removal straight-forward.

A shortcut was created on the Windows Desktop to run the InstantRails.exe program. This program activates a GUI console (refer Figure 5-1) from which the developer can run the following tasks:

- Activate/deactivate the Ruby console environment
- Activate/deactivate the Rails console environment
- Stop/start the MySQL web server
- Stop/start the Apache web server
- Activate/deactivate the Rails application session (in Mongrel)
- Display log files for MySQL, Apache and InstantRails
- Adjust configurations for any of the installed components
- Create/add Rails applications

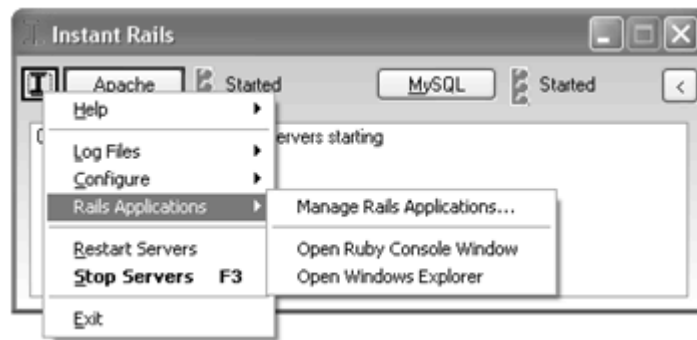


Figure 5-1 InstantRails Console

Using InstantRails

The InstantRails installation incorporated two sample Rails applications:

- *Cookbook* – a simple recipe management application
- *Typo* - a comprehensive blog website management application

The *Cookbook* application was configured (via the InstantRails console) to use a specific port, in this case 3001. A Mongrel session was then initiated from the InstantRails console which loaded the application. The application was then able to be accessed via a Firefox browser installed on the PC. This was achieved through the use of either URL addresses <http://127.0.0.1:300> or <http://localhost:3001>.

Another alternate method of access was made possible by configuring a website, www.cookbook.com, which enabled the URL 'www.cookbook.com:3001' to also be used. The *Typo* application was similarly able to be run.

The example applications were explored and code enhanced through the addition of table fields and the implementation of CSS to alter the user view.

Application Development

There are three main methods of editing Rails application code: from the Rails console, through a text editor or by using an Integrated Development Environment (IDE). For the purposes of this exercise the Aptana RadRails IDE (<http://www.aptana.com/studio/download>) was installed. This facilitates the location and editing of code as well as the running of Rails commands.

Several tutorials were followed to gain an understanding of Rails. In particular, Sonjaya Tandon's website (<http://sonjayatandon.com/05-2006/how-to-build-a-secured-web-application-with-ruby-on-rails/>) was helpful in providing a tutorial covering user login, registration and verification. The *Ruby for Rails* publication (Black, 2006) provided a tutorial that also proved useful. This involved building a simple music cataloguing application consisting of three tables: work, composer and edition.

Through installing Rails, making minor configuration changes, examining sample applications and completing tutorials, a good understanding of the Rails technology was able to be achieved.

6 Software Evaluation

6.1 Programming Environment

A key requirement of any software development tool is that it is conducive to productivity and creativity. Developers have noted that they are at their most productive when they are ‘in the zone’, a term used for the trance-like state that transpires when creativity and clarity of thought is at its peak and it is a concept familiar to creative people in all fields of endeavour.

Chopra & Dexter (2007) note that programmers link beauty and simplicity when they describe this mental state. The authors find that this uncluttered frame of mind is required to produce good code and that messy code is a symptom of a messy brain and is likely to be incorrect.

Development tools that are helpful in easing the developer into this state are therefore desirable. Feedback from the programming community indicates that the Rails philosophy is closely matched to what is required for this to occur. Rails ensures that much of the complexity (and therefore distraction) is hidden from the developer. Additionally the Ruby language is rated highly for its intuitiveness and as being a joy to work with.

Casal (2005) observes that much of the design principles behind Ruby on Rails are more of a psychological benefit rather than a technical one with, for example, the implementation of such features as MVC and object-orientation being more to do with providing an efficient working environment for programmers rather than a choice of superior technology.

Larry Wall (1999, p. 4) discusses the programming paradigm in his talk on Perl, a programming language he developed in 1987. He argues that Perl is a post-modern language unlike most others which suffer from ‘modernism’: his definition being “taking a cool idea and driving it into the ground”. He compares previous languages to hammers and everything being made to look like something to be hammered, resulting in dysfunction. In contrast, he describes Perl’s post-modern qualities in terms of duct-tape as in “if all you have is duct-tape, then everything starts to look like a duct” and that duct tape is more notable for its use on anything *but* ducts. With Perl the focus is primarily on the person doing the solving, not the problem to be solved. And so it is with Ruby, which shares many of Perl’s attributes. It is

flexible enough to be moulded to various application environments; it does not attempt to strait-jacket the programmer with static typing and unnecessary grammatical rules. Perhaps now is the time for programmers to stop hammering and be allowed to adopt a more flexible approach to development?

Ruby architect, Matsumoto (interview by Venners, 2003) also stresses the value of the human-side of programming when he comments that it is the people who are the masters and machines, the slaves. Too often computer engineers will focus on the machine when they should be focussing on the programmers, a concern he has borne in mind when working on Ruby.

In commenting on Ruby on Rails (in particular, blogs, newsgroups, book reviews, forums and other informal communication channels) it is noted the high level of programmer enthusiasm for Rails development amongst programmers. Their frequent use of subjective terms such as 'fun', 'beautiful', 'joy', 'happy', 'aesthetically pleasing' and similar phrases were manifest. A typical comment from a blog extolling Rails' virtues is as follows:

Beautiful code is one of those things that are hard to explain, it's about feeling good about a single statement, expressing what you want to say in a way that's aesthetically pleasing, that's understandable, that's right.
(Charman-Anderson, 2006)

Too often management's attitude to keeping their valuable programmer resource happy is to pay them more, provide more perks and give them impressive job titles. What is likely to be of more importance in keeping developers loyal to their work and, by extension, the company they work for, is the enjoyment factor. Having effective, productive and fun tools to work with is likely to have a very positive influence on worker retention as well as productivity.

6.2 Suitability of Purpose

Concerns have been expressed as to whether Rails is a suitable choice in many circumstances. Hansson states that in implementing his principle of convention over configuration he was targeting his product at what most web developers do most of the time and, in this respect, he hoped to meet 80% of the markets needs. Therefore there may be instances where other technologies will be a better fit. In particular, if there is a need for strong integration with legacy systems, then Java's ability to interface with a wide-range of technologies may make it a better choice.

In the majority of cases, Rails has proven itself in the marketplace. For sophisticated requirements, such as those that relate social networking and other forms of Web 2.0 applications, Rails' has had a proven track record. In a paper outlining the design and construction of a Semantic Web 2.0 Application Framework (SWAF) it was determined that the dynamic typing abilities of Ruby on Rails was well suited to the Resource Description Framework (RDF) model (Oren, Haller, Hausworth, Heitmann, Decker & Mesnage, 2007). The decision was made to build the SWAF as an extension of the Rails framework. They were able to create SWAF generators to "automatically generate scaffold functional code for classes of RDF(S) data".

Recently Scholastic UK, part of the largest children's book publisher Scholastic Inc, developed an on-line children's book club using Agile processes and Rails technology. It was noted that "the site was developed 20% faster, on-budget and delivered with more functionality than originally scoped" (RealWire, 2008). The site received more than 5000 visitors in less than a month.

Social networking site, Twitter, attracts in excess of 25 million visitors per month (refer Figure 6-1), a figure which continues to grow exponentially. There was much debate (Atwood, 2007) as to whether achieving this level of scalability is straightforward in Rails but the recent success in converting the USA Yellow Pages site to a Rails platform is positive evidence that traffic on this scale is manageable. Figure 6-1 shows the volume of visits to both these sites over the past year. Whatever the technology applied to the problem of such high volumes, the resulting architecture would necessarily entail a complex solution.

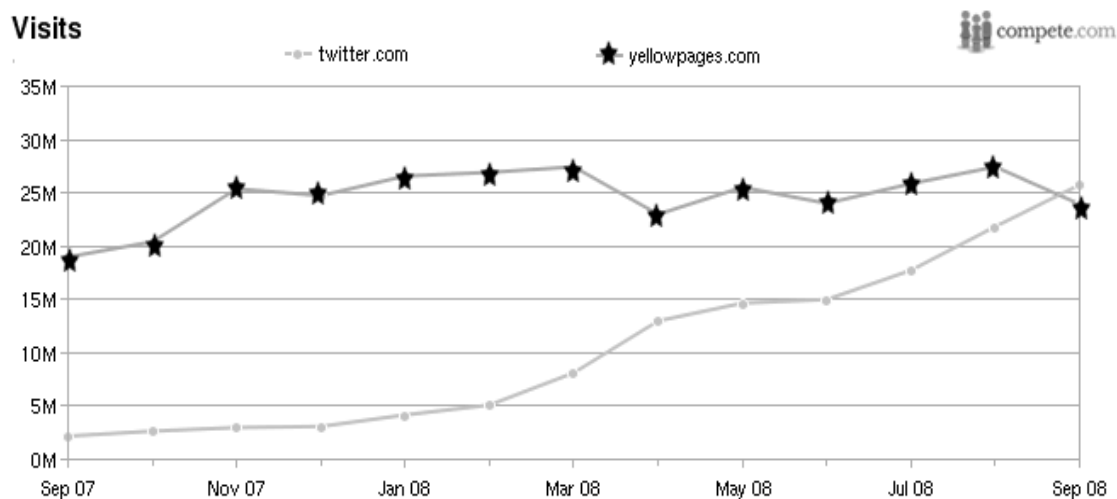


Figure 6-1 Twitter.com and Yellowpages.com visitor analysis (Compete, 2008)[†]

The instances of Rails implementations, given above, demonstrate that Rails has the ability to provide a solution for a wide-range of scenarios. Ideally, however, the platform is best suited to the following situations:

- Web applications such as ecommerce or social networking
- No requirement for cross-database access
- An ongoing requirement for incremental change, both before and after development completion
- Small team environment
- Flexibility in choice of database schema (e.g. no legacy system interfacing)
- Flexibility in the deployment environment

If imitation is the sincerest form of flattery then it is evident that Rails has a large number of devotees amongst those working with competing web technology. The terms ‘Rails-like’ is widely applied to the newer releases of existing developer platforms and a raft of new rivals. Just a few of the many newer frameworks inspired by Rails are Merb (Ruby), Waves (Ruby), CakePHP (PHP), AsProject (ActionScript), RIFE (Java), Catalyst (Perl), Trails (Java), Grails (Groovy) and IBM’s WebSphere sMash (PHP/Groovy). Even Ruby has its imitators, for instance Microsoft’s IronRuby which has been developed for the Microsoft .NET environment.

[†] Compete triangulates multiple data sources, including ISP, Panel & Toolbar to estimate U.S. traffic.

6.3 Productivity

Primarily it is claims of enhanced productivity that has brought Rails to the attention of the developer community. There is little or no debate as to whether Rails applications are quicker to develop; the only area of disagreement appears to be in how this is to be measured and the significance of the increase. In the past, it has been common for productivity to be measured in lines of code produced. With frameworks having a higher level of abstraction, it is of more relevance to value *fewer* lines of code used in achieving required functionality. Less code equates to faster coding, less debugging, less complexity and improved readability. There is wide agreement that Rails applications require considerably less code than similar solutions written in Java and that productivity, therefore, is significantly higher. Tate (2005, p. 82) estimates that productivity, using Ruby over Java, is boosted three to five times higher. This productivity is reflected in the gains made using Rails (as opposed to J2EE) with some putting the increase as high as ten fold (Bradley, 2005).

6.4 Arguments Against

The arguments against Rails are many; most of these originating from developers firmly attached to other popular web development tools. A comprehensive list of reasons why Rails is unlikely to become mainstream originates from a developer who at the outset declares that “Rails is a fantastic framework built on a wonderful language” (Cedric’s Weblog, 2006). His main arguments, echoed by others elsewhere, are listed below with relevant counters.

Ruby is too complex

Claims that programmers of the more mainstream languages would find Ruby difficult to learn are not demonstrated in the figures showing the language’s rapid uptake. (refer to Figure 3-1).

Rails is too clever

Rails is criticised for having too many slick features and too much ‘magic’, making the technology incomprehensible to regular corporate developers, with the jump from imperative to object-oriented programming being too difficult. The counter to this is that many programmers *did* survive the object-oriented transition, even though it involved a substantial mind-shift, added to programming complexity and

the payback was not immediately obvious. Rails, in contrast, simplifies and offers the immediate reward of increased productivity.

No credible IDE

Initially there were concerns that no Integrated Development Environment (IDE) existed for the technology. However, this drawback has now been resolved with over 9 platforms now available (InfoWorld, 2008a). These include:

- SapphireSteel Ruby in Steel Developer Edition 1.2 and Text Edition 1.1.5
- Aptana RadRails 1.0
- ActiveState Komodo IDE 4.3 and Edit 4.3
- CodeGear 3rdRail 1.1
- NetBeans IDE 6.1
- MacroMates TextMate 1.5.7
- JetBrains IntelliJ IDEA 7.0.3 with Ruby plug-in 1.0
- E Text Editor 1.0.20 Beta
- Intype 0.3.1 Alpha

Fanaticism

The Rails community is frequently criticised for what is viewed as their fanatical attitude in support of the technology. It is not uncommon for developers to defend their language of choice and the adopters of Rails are no exception. Perl and Java also have their extremists but it does not make those languages any less useful.

Solitary Ruby Solution

Java offers at least a dozen web development technologies. The fact that Ruby had just the one solution, Rails, was viewed as a negative. However, the Java web development environment has become a miscellany of technologies with developers often required to cobble together solutions using components from various third party vendors. The development environment that this engenders encourages complicated solutions that are difficult to modify and enhance. In addition, upgrades to the third party components incorporated into the solution pose further issues of compatibility, staying up-to-date and deprecation. With the Ruby community's focus on just the one framework, Rails has benefitted from their combined effort in refining and enhancing the platform.

In the past year, two new Ruby web development frameworks have been released; Waves and Merb. While these technologies do resolve some issues that can be an

issue with Rails, the emergence of these competing technologies in the Ruby environment may result in the same issues related to fragmentation as has occurred with Java.

Scalability and Enterprise Readiness

Lack of examples of high-profile enterprise solutions in Rails is commonly cited as reasons for Rails not to be considered a serious contender in real world web development. Recent developments (some of which are described earlier in this paper) show this to be no longer the case with several sites attracting in excess of one million hits per month.

Hansson counters scalability scepticism with the following observation:

We've been using Ruby on Rails for five years. Tons of organizations are scaling it massively to billions of page views. While some Rails sites may have scalability problems that is not necessarily the fault of Rails (Hansson, interview by InfoWorld, 2008b).

Limited Web Host Support

A lack of web hosting companies providing Rails support has been an issue in the past but this is changing rapidly with the increasing popularity of Rails. The Web Host Industry Review (WHIR), a website and magazine supporting the web host industry, notes that:

Already many hosts have introduced support for Rails, and it is safe to assume that more hosts will continue to add their support as Rails becomes one of the key standards for Web development (Whir, 2008).

6.5 Dynamics of Software Adoption

As is often demonstrated in the computer industry, the best technology does not always win in the decision making process. While it is important that the technology is stable and does what is required, other factors are also significant: marketing, timing, market-share and the environment in which the development will occur. Enthusiastic user community support combined with the technology's adoption by high-profile organisations will also influence the final decision.

A sole contractor working on a greenfield project typically has no constraints on the development platform and tools used. In this case it is common for software decisions to be left up to the developer who can then decide on tools based on

productivity, cost advantage, learning overhead and skill-set benefit. The management decisions of larger organisations will demonstrate risk-averse tendencies, the influence of legacy systems and the developer culture and abilities.

The choice of software tools will greatly impact developers as their experience and mind-set will influence the success of any new technology adoption decision. Developers may have many years investment in learning a particular technology and may be recognised as leaders in their area of expertise. It is understandable, therefore, that the introduction of a new technology could engender fear and resentment. There may be a lack in ability or inclination to re-think the software development paradigm. For example, the mind-shift from procedural languages to an object-oriented approach has been a difficult transition for many programmers. An alternate programmer response to new technology is to embrace the new and yet unproven. While this can be seen as positive, it can also result in the introduction of a technology that lacks industry and developer community support and so becomes prematurely arcane and obsolete.

With any new software decision it is necessary to keep in check any urge to possess the most up-to-date technology by weighing risks inherent in being an early adopter against perceived benefits. Risk associated with the choice of Rails has lessened now that it is moving from the early-adopter phase.

It is possible that the lifecycle of many of the current web development tools maybe nearing their end. This concept of programming language lifecycles, and in particular Java, is explored in an article by Sharp (2002) in which he identifies these lifecycle stages as conception, adoption, acceptance, maturation, inefficiency, deprecation and decay. He surmises that Java is now in the inefficiency phase putting it near the end of its lifecycle and predicts that very soon a language shift will occur with “a young pretender will emerge from a research project”. Microsoft’s .NET language, C# was dismissed as a possibility. While no mention was made of Ruby, its rapid rise in popularity, riding on the back of Rails, may very well become a threat to Java.

The real question may not be whether or not Ruby on Rails is better but whether it has the ability to solve enough of the application problems in enough situations to gain traction in the industry.

7 Conclusion

Simplicity is the ideal defence against the increasingly complex task of web development and this has been achieved with Rails by restricting framework flexibility and limiting functionality to the needs of only 80% of the web development market. Developers are able to focus less on the intricacies of interacting with the development environment and concentrate on the business problem, enabling them to invest more effort into innovative and effective solutions.

The fast moving web paradigm calls for tools that allow users to have access to workable solutions early in the process and allow for iterative on-going development. This requirement has been met through Rails' scaffolding capacity and built-in Agile methods.

Significantly, the impact on the programmer environment is greatly enhanced through a lessening of the distraction and annoyance frequently associated with other web technologies that are not fully integrated. A homogenous philosophy throughout Rails makes it possible for the developer to intuit how things work and lessens the need to refer to documentation. The management of complexity through a high-level of abstraction assists the programmer in grasping and solving design issues unique to the business problem rather than the dealing with implementation issues. The overall approach in streamlining and simplifying adds to programmer enjoyment and, as a consequence, productivity and effectiveness.

In an environment where it is frequently left to the programmer to interact with users, gather requirements and evolve systems, there will always be demand for good programmers who are also able to design. These skilled workers require and deserve the best tools available to maximise their productivity and keep them happy in their work. The interest, excitement and feedback from the programmer community, suggests that Rails may offer them an immense improvement in their development environment.

If sensitivity to employee needs is not consideration enough for management to take notice, the economic benefit of the proven increase in productivity should justify serious consideration of Rails in any evaluation of web frameworks. Programmer productivity (in any language) poses difficulties in quantifying and for this reason

empirical evidence comparing Rails to other frameworks is questionable. However, the majority of opinion which puts the increase in productivity much higher than that of previous technologies is difficult to ignore.

It was initially planned that, as part of this research, a fully working Rails application be developed. Although time constraints prevented this from being a possibility, it was felt that a sufficient level of understanding was gained to enable the researcher to grasp the concepts and philosophy encompassed by Rails. There is always an element of bias in any qualitative research and it is acknowledged that this study incorporates the researcher's subjective opinion. The choice of which material to include and what themes to emphasise has been founded on the author's understanding of the topic and personal preference. While the observations given in this study appear to have majority support within the programmer community, there are situations where the Rails approach may not suit, in particular, where the environment is not conducive to the application of Agile methods. One such example of mismatch between organisation mind-set and the chosen development approach cost the company "tens of millions of dollars" (Martin, 2008). Also, where there is a substantial investment in legacy code the introduction of newer technology may pose difficulties with system interfacing and programmer re-training.

It is easily argued that Rails has raised the bar in expectations and has already left its mark on the web development landscape. Most existing technologies have now incorporated enhancements to comply with the Rails way of doing things. The majority of newer technologies openly state that they are Rails clones. While these newer offerings may present advantages for specific niches, none appear to have the general all-purpose functionality that has been built into Rails or the broad acceptance across the industry. They certainly have not generated the same level of excitement among developers.

The future of web development is uncertain but in response to the current financial crisis, the creator of Rails has commented as follows:

...Rails developers are much better positioned to weather the storm as they generally stand for delivering more with less faster. It's the traditional mainstream environments that are going to see much more pressure to deliver. (Hansson, interview by Taft, 2008)

It would seem that Rails is likely to be the answer to many of the perceived shortcomings existing with web development, primarily in improving programmer productivity and making programming tasks more enjoyable through the removal of complexity. If lessening the complexity of web development is a significant factor in deciding on a web development framework, then the choice of Ruby on Rails, whether by management or programmers, is a simple decision.

8 Glossary

The following definitions were retrieved from the Wikipedia on-line dictionary (<http://en.wikipedia.org>).

AJAX	Ajax (asynchronous JavaScript and XML) is a group of interrelated web development techniques used for creating interactive web applications or rich Internet applications.
CSS	Cascading Style Sheets is a stylesheet language used to describe the presentation of a document written in a markup language.
DSL	A Domain Specific Language is a programming language or specification language dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique.
Dynamic typing	A programming language is said to be dynamically typed, or just 'dynamic', when the majority of its type checking is performed at run-time as opposed to at compile-time.
Greenfield	a greenfield is a project which lacks any constraints imposed by prior work.
HTML	An initialism of HyperText Markup Language, is the predominant markup language for Web pages.
IDE	An integrated development environment also known as integrated design environment or integrated debugging environment is a software application that provides comprehensive facilities to computer programmers for software development.
J2EE	Java Platform, Enterprise Edition or Java EE is a widely used platform for server programming in the Java programming language.
MVC	Both a design pattern and an architectural pattern used in software engineering.
Open source software	Computer software for which the human-readable source code is made available under a copyright license (or arrangement such as the public domain) that meets the Open Source Definition.
REST	Representational state transfer is a style of software architecture for distributed hypermedia systems such as the World Wide Web.
Static typing	A programming language is said to use static typing when type checking is performed during compile-time as opposed to run-time.
Type system	A type system defines how a programming language classifies values and expressions into types, how it can manipulate those types and how they interact. Type categories are dynamic and static.

Web 2.0	A term describing changing trends in the use of World Wide Web technology and web design that aims to enhance creativity, secure information sharing, collaboration and functionality of the web.
Web Hosting Service	An Internet hosting service that allows individuals and organizations to provide their own website accessible via the World Wide Web.
Web service	A software system designed to support interoperable machine-to-machine interaction over a network.

9 References

- 37signals. (2006). *What is Getting Real?* Retrieved October 2, 2008, from http://gettingreal.37signals.com/ch01_What_is_Getting_Real.php
- Atwood, J. (2007). Coding Horror. *Twitter: Service vs. Platform*. Retrieved September 23, 2008, from <http://www.codinghorror.com/blog/archives/000838.html>
- Black, D. (2006). *Ruby for Rails: Ruby Techniques for Rails Developers*, Foreword by David Heinemeier Hansson. Manning Publications.
- Bradley, R. (2005). *Evaluation: moving from Java to Ruby on Rails for the CenterNet rewrite*. Retrieved November 7, 2008, from http://rewrite.rickbradley.com/pages/moving_to_rails/
- Builderau. (2006). *Ruby on Rails: The importance of being 1.0*. Retrieved November 1, 2008, from <http://www.builderau.com.au/program/ruby/soa/Ruby-on-Rails-The-importance-of-being-1-0/0,339028320,339233600-2,00.htm>
- Casal, D. P. (2005). *Advanced Software Development for Web Applications*. Retrieved September 9, 2008, from http://www.jisc.ac.uk/uploaded_documents/jisctsw_05_05pdf.pdf
- Cedric's Weblog. (2006). *Why Ruby on Rails won't become mainstream*. Retrieved September 23, 2008, from http://beust.com/weblog/archives/2006_04.html
- Charman-Anderson, S. (2006). *Happy programming with Ruby on Rails - David Heinemeier Hansson*. Retrieved October 2, 2008, from <http://strange.corante.com/2006/02/08/fowa-happy-programming-with-ruby-on-rails-david-heinemeier-hansson>
- Chopra, S., & Dexter, S. (2007). *Decoding Liberation: The Promise of Free and Open Source Software (1st Edition)*. Routledge.
- CIO Magazine. (2008). *PHP, JavaScript, Ruby, Perl, Python, and Tcl Today: The State of the Scripting Universe*. Retrieved September 21, 2008, from http://www.cio.com/article/446829/PHP_JavaScript_Ruby_Perl_Python_and_Tcl_Today_The_State_of_the_Scripting_Universe
- Compete. (2008). Twitter visitor profile. Retrieved October 27, 2008, from <http://siteanalytics.compete.com/twitter.com/?metric=uv>
- DBusinessNews.com. (2007). *CodeGear™ Releases 3rdRail™ – The Powerful Integrated Development Environment Built Specifically for Ruby on Rails*. Retrieved September 21, 2008, from http://sanjose.dbusinessnews.com/shownews.php?newsid=133662&type_news=latest
- Deckert, F. (2006). *Experience Report on Ruby on Rails*. Retrieved August 1, 2008, from http://www.cs.ubc.ca/~kdvolder/CPSC511/submissions_06_07/florian.pdf
- Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley
- Gelernter, D. (1998). *Machine Beauty: Elegance and the Heart of Technology*, New York: Basic Books.

- Guzewich, T., Kent, M., Pfeifer, A., Shank, K. (2006). *Software Architecture case Study: Ruby on Rails*. Retrieved October 27, 2008, from <http://www.pdffoo.com/ruby-rails/257-9244-pdf.html>
- Hansson, D.H. (2004). Fear-driven technology choices. Retrieved October 26, 2008 <http://www.loudthinking.com/arc/000333.html>
- Highsmith, J. & Cockburn, A. (2001). Agile Software Development: The Business of Innovation, Computer, v.34 n.9, (pp.120-122). Retrieved October 9, 2008, from Academic Research Library database. (Document ID: 79918289)
- InfoWorld. (2008a). *Lab test: Climb aboard Ruby on Rails*. Retrieved October 27, 2008, from http://www.infoworld.com/article/08/07/07/28TC-ruby-ides_3.html
- InfoWorld. (2008b). *Ruby on Rails upgrade eyed*. Retrieved November 4, 2008, from http://www.infoworld.com/article/08/05/27/Ruby-on-Rails-upgrade-eyed_1.html
- Lowe, D., & Henderson-Sellers, B. (2001). *Characteristics of Web Development Processes*. Retrieved October 2, 2008, from <http://services.eng.uts.edu.au/~dbl/archive/2001-Low01c.pdf>
- Macintosh News Network. (2006). *Ruby on Rails to ship with Leopard*. Retrieved September 21, 2008, from <http://www.macnn.com/articles/06/08/08/ror.to.ship.with.leopard/>
- Martin, C. (2008). *Agile Methodologies Turn Dev Team into 'Cargo Cult'*. Retrieved November 7, 2008, from http://www.pcworld.com/businesscenter/article/149586/agile_methodologies_turn_dev_team_into_cargo_cult.html
- Matsumoto, Y., (2000). *The Ruby Programming Language*. Retrieved September 9, 2008, from <http://www.informit.com/articles/article.aspx?p=18225>
- Myers, M. D. (2008). *Qualitative Research in Information Systems*. Retrieved October 21, 2008, from <http://www.qual.auckland.ac.nz>
- Oren, E., Haller, A., Hauswirth, M., Heitmann, B., Decker, S., Mesnage, C., (2007). *A Flexible Integration Framework for Semantic Web 2.0 Applications*. Retrieved September 23, 2008, from <http://www.deri.ie/fileadmin/documents/ieeesw2007.pdf>
- Orsini, R. (2007). *Rails Cookbook*. O'Reilly Media, Inc.
- Raymond, E.S. (2003). *The Art of Unix Programming*. Retrieved September 1, 2008, from <http://catb.org/esr/writings/taoup/html/>
- RealWire. (2008). *New Bamboo And Scholastic UK Create Online Children's Book Club*. Retrieved September 23, 2008, from http://www.webitpr.com/release_detail.asp?ReleaseID=8067
- Ringgaard, M. (n.d.). *Quotations on simplicity in software design*. Retrieved October 9, 2008, from <http://www.jbox.dk/quotations.htm>
- RubyonRails.org.(n.d.) Retrieved September 1, 2008, from <http://www.rubyonrails.org/>
- RubyonRails.com. (2007). *Understanding Rails MVC*. Retrieved October 10, 2008, from <http://wiki.rubyonrails.com/rails/pages/UnderstandingRailsMVC>

Sharp, R. (2002). *Programming Language Lifecycles - Where's Java At?* Retrieved October 27, 2008, from http://www.softwarereality.com/programming/language_lifecycles.jsp

Taft, D.K. (2008). *Can Ruby, Rails Make Developers Shine in a Downturn?* Retrieved October 26, 2008, from <http://www.eweek.com/c/a/Application-Development/Can-Ruby-Rails-Make-Developers-Shine-in-a-Downturn/>

Tate, B. (2005). *Beyond Java*. O'Reilly Media, Inc.

The Agile Manifesto. (2001). *Principles behind the Agile Manifesto*. Retrieved September 9, 2008, from <http://www.agilemanifesto.org/principles.html>

Thomas, D. (2007). *Agile Web Development with Rails*. (2nd Edition.). Raleigh. The Pragmatic Programmers LLC

Tiobe. (2008). *TIOBE Programming Community Index for October 2008*. Retrieved September 21, 2008, from <http://www.tiobe.com/index.php/content/paperinfo/tpci/index>.

Venners, B. (2003). *The Philosophy of Ruby: A Conversation with Yukihiro Matsumoto*. Retrieved October 27, 2008, from <http://www.artima.com/intv/ruby4.html>

Wall, L. (1999). *Perl, the first postmodern computer language*. Retrieved October 27, 2008, from <http://www.perl.com/pub/a/1999/03/pm.html>.

Whir. (2008). *What is Ruby on Rails?* Retrieved November 5, 2008, from http://www.thewhir.com/find/web-hosts/articles/What_is_Ruby_on_Rails.cfm