Using Rails to Create an Enterprise App: A Real-Life Case Study

By David Keener

http://www.keenertech.com

Overview

Ruby on Rails is clearly an exciting and ground-breaking technology. But how good is it at solving the types of problems that corporate developers need to solve? My team put Rails to the test in a real-world internal application for AOL that's used by hundreds of employees. This case study, the Exception Request Tool, is a non-trivial application that features:

- Multiple databases; of multiple types
- A Legacy database (Remedy)
- Significant AJAX functionality
- Search capability
- Email (or corporate spam)

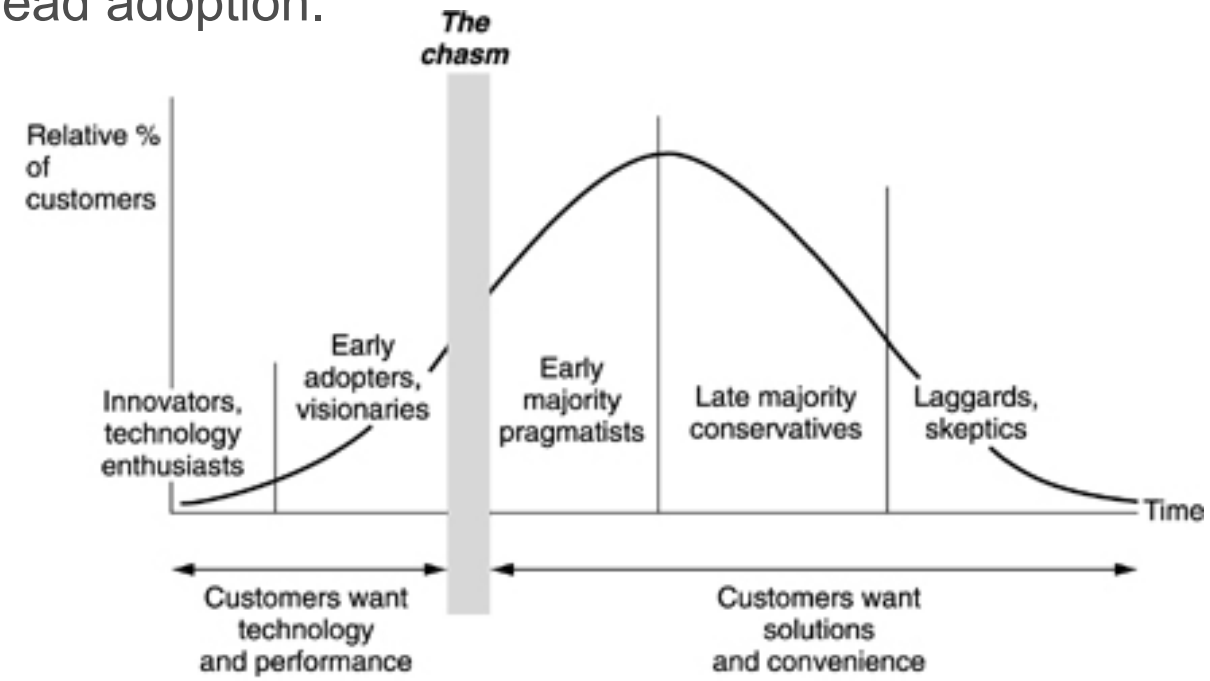
What We Wanted to Know

The underlying question affecting the adoption of Rails for corporate development is: “How good is it at producing web sites that aren’t blogs?”

- Does Rails really speed up development?
- Can it be effectively integrated with legacy databases?
- How hard is it to set up a Rails production environment?
- How hard is it to incorporate AJAX and build **Web 2.0** applications?
- How effective is Rails at generating and sending automated emails?
- What pitfalls should developers watch out for?

Rails Adoption

Rails is in a unique place, just to the left of the “Chasm”. Its capacity for successful delivery of real business applications will probably be a key factor in crossing the chasm between early adopters and widespread adoption.



From: Diffusion of Innovations, by E. M. Rogers, 1995

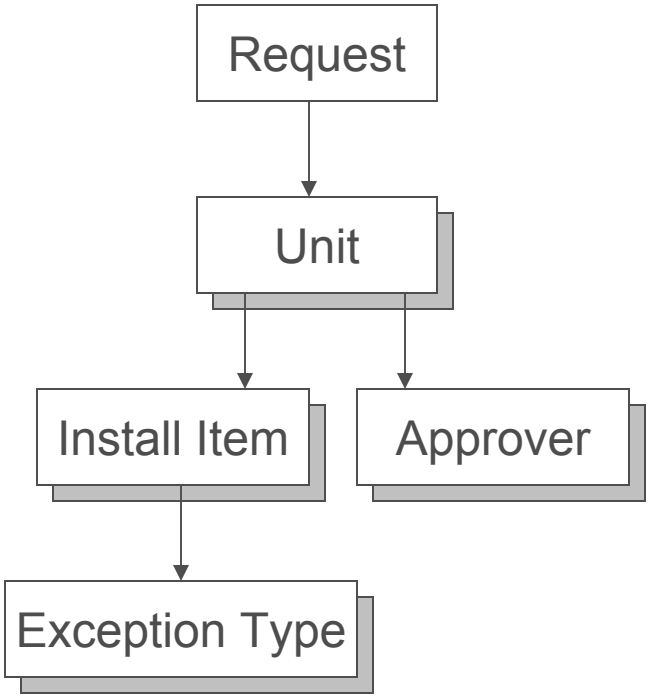
The Case Study Application

The Exception Request Tool (ERT) allows AOL employees to request production installs for time periods when such installs are not normally allowed. Key features included:

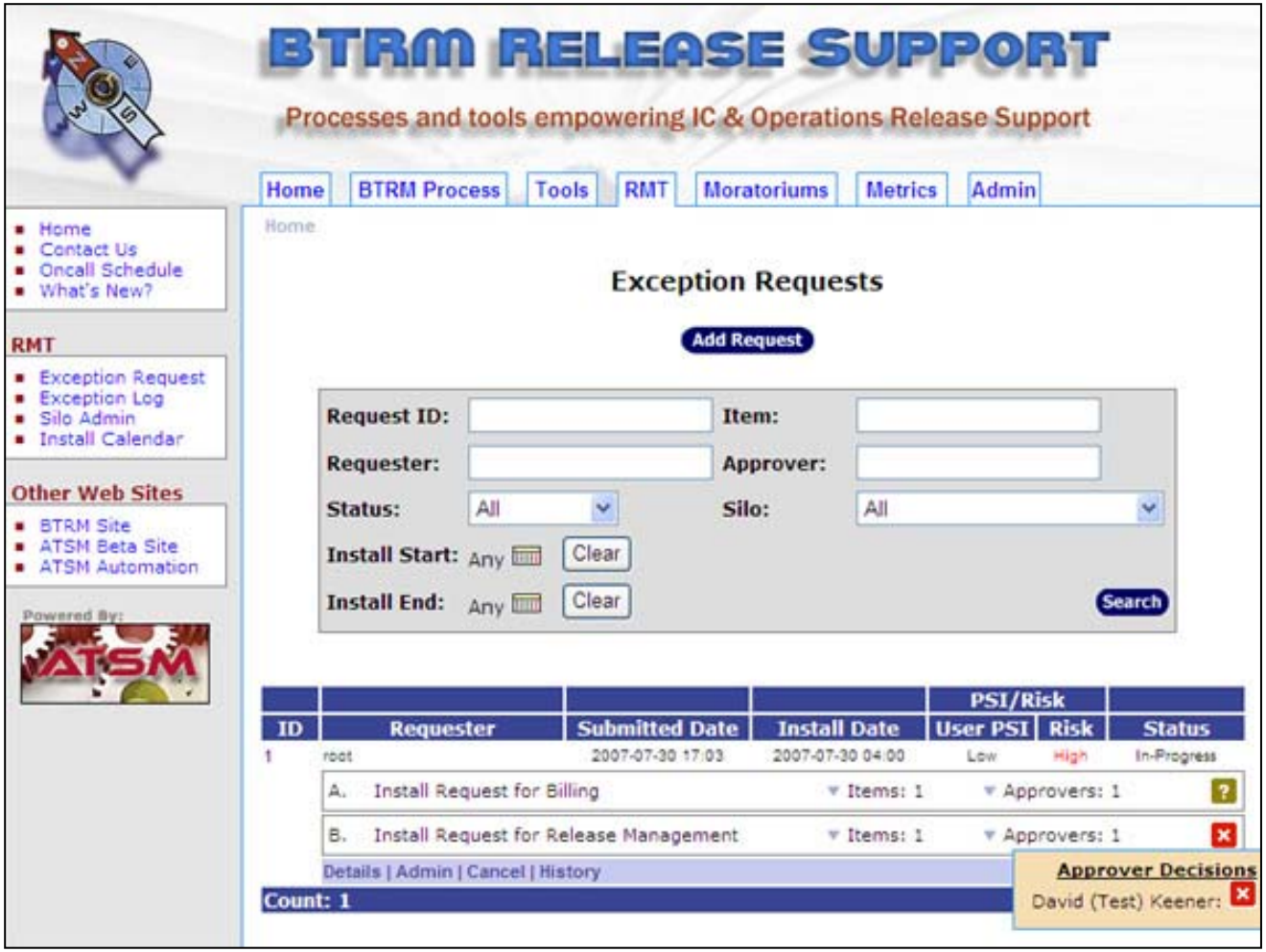
- Relatively standard CRUD features, plus….
- Confirm and edit capabilities for requests
- A search feature
- A history feature (for Sarbanes-Oxley support)
- Administrative capability to modify <u>all</u> aspects of a request
- Automated emails of various types
- Approval mechanism for upper management
- Access to legacy database (Remedy)

Database Objects

- Request: A request for a production install to support one or more business areas
- Unit: A collection of items to be installed to support a business area
- Install Item: Something to be installed, usually software
- Approver: A person who approves a unit, or a set of install items
- Exception Types: For metrics, categorizes aspects of each install item



Exception Request Tool



Screenshot used with AOL permission.

Process Change

The first, and perhaps most striking, change due to Rails, was its impact on the development process.

- Rails facilitated delivery of an early prototype <u>while</u> requirements were still being drafted
- The prototype helped generate lots of new requirements via discussion of real screens & functionality
- New requirements led to major scope creep, which…
- Totally blew the original deadline
- But the customer was happier…the product was really what <u>they</u> wanted
- So the customer agreed to adopt a more “agile” approach to development going forward

Environments

Applications need a place to run, where they will be properly supported. Environment set-up was not a trivial exercise.

- Dev Environments: Easily set up on Mac & Windows
- Production Environment: Apache 2.2, Mongrel, Mongrel Cluster, etc.
- System administrators have no experience setting up Rails environments – falls to the dev team
- No standardization of Rails environments & recommended plug-ins – falls to the dev team
- At AOL: No access from prod box to outside Internet
- Need to allocate a non-trivial amount of time to deal with environment setup issues

Database Support

- Rails is somewhat immature in the realm of database support, though improving
- Sterling support for MySQL, but other databases can be more problematic
- Oracle support is good
- Support for Sybase is **pathetic** – Got it working for Linux, but not for Windows and Mac
- If not using MySQL, need to allocate time to address database connectivity issues

Migrations

- Migrations simplify database builds
- Database neutral
- A truly wonderful feature
- Don’t know how I lived without them
- Can also set up test data / lookup data

```
class CreateRiskLookup < ActiveRecord::Migration

  def self.up
    create_table :btr_risk_lookup do |t|
      # t.column :name, :string
      t.column :risk_name, :string, :null=>false, :limit => 50
      t.column :risk_desc, :string, :null=>true, :limit => 100
    end
  end

  def self.down
    drop_table :btr_risk_lookup
  end
end
```

Cross-Database Migration Issues

- Migrations generally “do the right thing” across databases…but not always
- Oracle sequences – names too long

  In model: set_sequence_name shortname_idx
- Indices – names too long

  In migration: add_index :btr_install_item_exceptions, [:install_item_id, :exception_type_id], :name => 'btr_installitemex'
- Indices – delete in Oracle, not MySQL

  In migration: Needed conditional logic based on database used.

Conditional Logic in Migrations

- Perform different actions based on database
- Only remove legacy table in MySQL (dev)
- Use class connection, not default
- Conditional logic also needed for Oracle indices

```
def self.down
   adapter = User.connection.instance_variable_get("@config")[:adapter]
   puts("Database Adapter Detected: " + adapter)

    if adapter == "mysql“
        drop_table :btr_users
    end
 end
```

Killer Legacy Database

Our legacy database put Rails to the test. Our legacy database possessed the following features:

- Database: Sybase – Could only get access working in proposed Linux prod environment – No access from development environments
- Non-numeric primary keys
- Hundreds of database columns
- Read-only database access
- Query Performance Issues
- Case-sensitive column and table names!

Legacy Database Cheat Sheet

- Explicitly define the names of legacy tables

  In model: set_table_name ‘T135’ # case-sensitive in Sybase
- Explicitly define the name of the primary key

  In model: set_primary_key ‘xyz’ # case-sensitive in Sybase
- For tables with non-numeric keys, use SQL
- For tables with 100’s of columns, use SQL
- With query performance issues, break SQL queries into two parts:
  1. Get the ID’s using whatever criteria are needed,
  2. Get the subset of desired columns for just those ID’s
- If doing SQL, create generic methods in model; only the model should have to know about the bare-metal SQL

Dual Query Tactic

```
# Step 1: Get a list of matching ticket ID's

res = Ticket.find_by_sql("SELECT Change_Request__ AS Change_Request" +
                         " FROM CM_Change_Mgmt " +
                         " WHERE Component___NUM_ LIKE '" + str + "%'")

# Step 2: Convert result set to comma-delimited list of single-quoted ID's

lst = bld_list(res)

# Step 3: Get a subset of the 300+ available columns. By the way, it's
# possible that multiple tickets may match

tickets = Ticket.find_by_sql(
           "SELECT CM_Change_Mgmt.Change_Request__ AS Change_Request, " +
                  "CM_Change_Mgmt.Component___NUM_ AS Component, " +
                  "CM_Change_Mgmt.Short_Description, " +
                  "CM_Change_Mgmt.Outage_Tkt_Yes, " +
                  "CM_Change_Mgmt.Outage__ AS Outage_ID, " +
                  "CM_Change_Mgmt.Affects_Members_, " +
                  "CM_Change_Mgmt.NUM__ AS NUMID " +
             "FROM CM_Change_Mgmt " +
            "WHERE CM_Change_Mgmt.Change_Request__ IN (" + lst + ") " +
               "AT isolation 0")
```

JavaScript & Validations

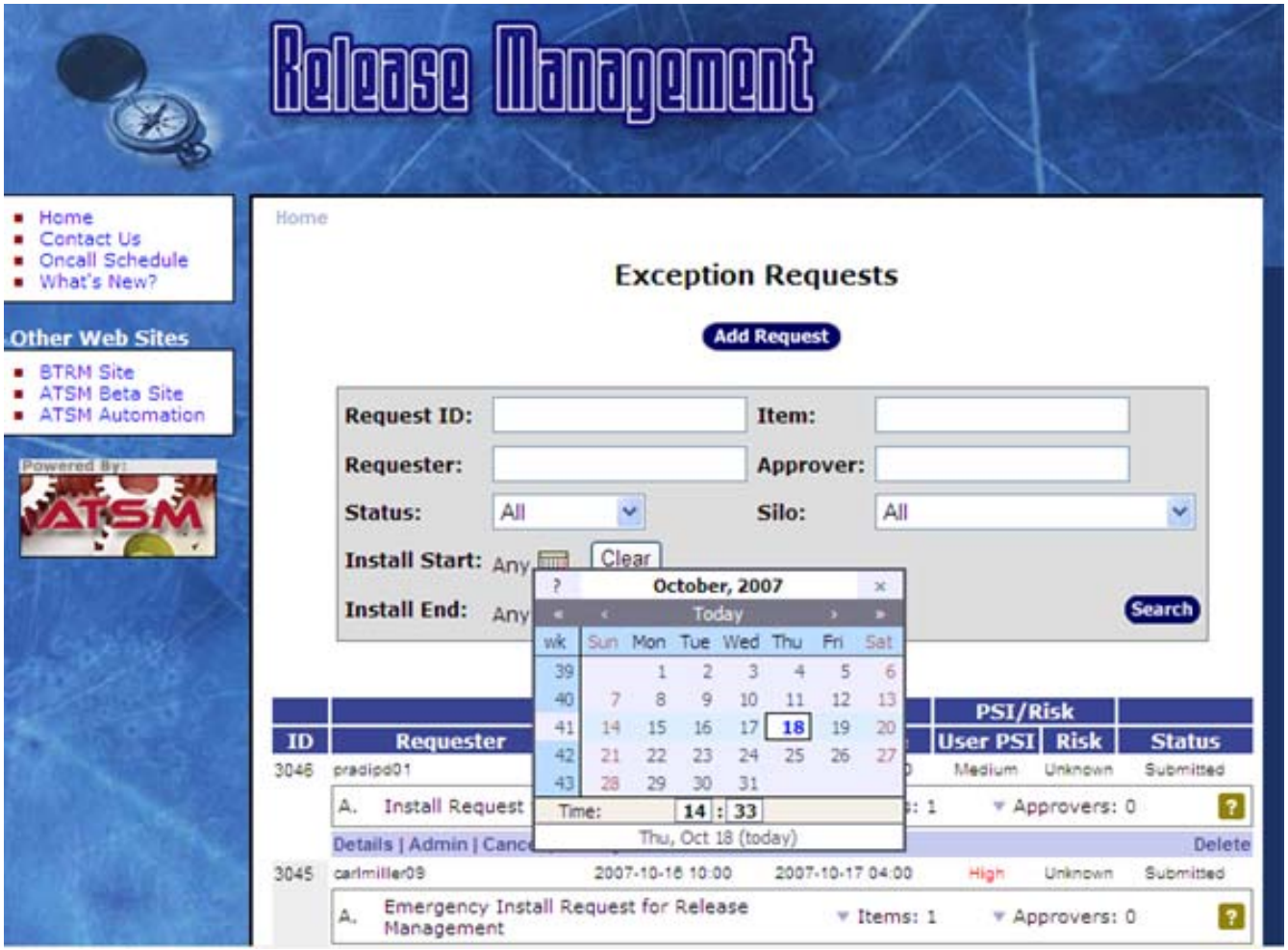
- Rails supports basic JavaScript validations
- JavaScript takes time, requires testing to support multiple browsers, etc.
- JavaScript can seriously slow a project
- Consider custom JavaScript judiciously
- If JavaScript needs are intense, consider integrating other open source libraries (e.g. – Dojo or others)

Date Manipulation

- Rails date manipulation features are awesome
- Rails date entry features are decent
- Really needed a popup calendar component -
- Also needed to be able to validate pairs of dates, i.e. – start and end dates
- Spent a LOT of time on date-related issues, including integrating popup calendar feature

Popup Calendar Component

Popup Calendar
- Date selection
- Time selection



Email in Rails

- Rails includes ActionMailer class
- Easy-to-create email class
- Emails generated based on templates
- Easy generation of dual text/html emails
- Easy configuration
- Can have emails that mimic the web pages almost exactly
- Graphic incorporation more problematic
- Email Features: Incredible!

Sample Email

Real-Life Email
- Mimics web page
- Dual format

Exception Installation Request

The following exception request was just submitted. To review the request online, click here.

Request Summary

Request ID: 1
Requester: David Keener
Email: davidkeener01@aol.com
Install Date Start: 2007-09-18 04:00
Install Date End: 2007-09-18 06:00

Status

Status: Submitted
Date Submitted: 2007-09-17 06:16
Action, Silo: None
Action, RM: Yes

Request Details

Prod Emergency: No.
User PSI: Low
Business Justification: ffffff
Contractual Obligation: No.
System Outage: No.
Impact, Mem Services : No.
Comments: None

Install Request for Release Management

Install Item	Item Type	Req. Install Date	Risk
fffff	Oth	Start: 2007-09-18 04:00 End: 2007-09-18 06:00	Unknown

Approver	Decision Date	Delegate	Decision
No approvers found			

Spaminator Model

```
class ExceptionRequestSpaminator < ActionMailer::Base

  def submit(ex)
    email_list = @requester.email_address + "," + get_email_rm

    subject    = "Exception Request #{ex.id}: SUBMITTED"
    intro = "The following exception request was just submitted.
             To review the request online, click <a
             href=#{get_local_url}/show/#{ex.id}>here</a>."
    setup_ERT_email(ex, subject, intro, email_list)
  end

  def setup_ERT_email(ex, subject, intro, email)
    @subject    = subject
    @body       = { :exception_request => ex,
                    :request_units => ex.request_units,
                    :requester => @requester,
                    :intro => intro}
    @recipients = email
    @from       = 'support-services-rm@listserv.sup.aol.com
    @sent_on    = Time.now
    @headers    = {}
  end

end
```

Sending Email

- Controller: To send an email…

  ExceptionRequestSpaminator.deliver_submit(@exception_request)
- Email Template – Just another “.rhtml” page

  - Template is “submit.text.html.rhtml”
- Mime Type incorporated into template name
- Can easily support other formats
- Email features – easy to implement
- Not so useful for social networking “green field” apps
- Incredibly useful for corporate enterprise apps

Testing

- Rails supports extensive testing features
- Didn’t use **ANY** of them on the project
- Not a problem with Rails, but a major deficiency in the project
- Need to internalize Rails built-in testing features
- Should incorporate tests as one of the required outputs of each project

Overall Rails “Report Card” – 1

- A+ Prototyping: Extremely fast
- A Agility: Supports agile programming methodologies; not so great for waterfall, etc.
- A Migrations: excellent; cross-database support could be stronger
- A Email: Excellent; great for enterprise sites
- B Dates: Manipulation excellent (A); date entry is just OK (C)

Overall Rails “Report Card” - 2

- B Testing: Features are good; requires some planning; easiest thing to “drop”
- B Legacy DB Support: Pretty good; needs more publicity
- C Database Support: not consistent; still very immature
- C JavaScript: Supports basic validations; complex components and validations are more problematic
- D Environment Setup: Not widely documented; not easy for a production environment *

* Note: See JRuby as an alternative environment for Rails applications.

The “Green Field” Perception

One of the biggest problems facing Rails adoption is the growing “green field” perception, i.e. – that Rails is only good for brand-new applications….

<u>An Excerpt from a Real-Life Conversation</u>:

David Keener: I’d like to promote Rails as a tool to get more done with fewer people.

Executive: Well, it was my understanding that Rails was only good for “green field” applications.

Summary

- Rails can be a good technology for corporate apps, not just “green field” apps
- Yes, legacy databases are harder to deal with than Rails-oriented databases, but then that’s an issue regardless of the technology used
- Rails is mature enough to be a realistic technology choice for many companies
- Plan your Rails project; don’t just “leap”