Technical Findings and Risk Update: Rails 2.3 to Rails 3.2 Migration

This document complements the previously proposed migration plan, summarizing key technical challenges and risks identified during the recent proof-of-concept (PoC) migration from Ruby 1.9.3 Rails 2.3 to Ruby 2.7.8 Rails 3.2.

Key Findings from the PoC Migration

1. Gem Compatibility Issues

Several gems required careful updates and replacements to ensure Ruby 2.7.8 compatibility:

- **`nokogiri`**: Updated to version `1.14.5` due to compatibility issues.
- **`json`**: Version `1.8.5` required syntax adjustments.
- **`yajl-ruby`**: Introduced explicitly at version `~> 1.4.1`.
- **`ffi`**: Migrated to a specific GitHub branch (`1.13.1`) to handle Ruby 2.7 native compilation issues on Apple Silicon.
- **`pg` gem**: Adjusted to `1.3.5` after addressing compatibility issues with Rails 3.2.
- **`rmagick`**: Major version upgrade (`4.3.0`) due to ImageMagick compatibility; significant environment configuration required.
- **`rack`**: Upgraded to version `1.4.7`, but still constrained by Rails 3 limitations. Known issues such as request length handling, session management, and security vulnerabilities persist.

Special Notes for Legacy Gems Compatibility:

- **activemerchant (1.75.0)**: Ensure transaction and braintree gateway integrations are thoroughly tested due to significant API changes since version 1.43.3.
- **omniauth-facebook (4.0.0) and omniauth (1.3.2)**: Verify authentication flows comprehensively; token handling and callback behaviors may differ from legacy versions.
- **dropbox_api (~> 0.1.19)**: Confirm Dropbox integrations for backward compatibility with existing file management logic.
- **will_paginate (3.0.12)**: Pagination logic must be validated in views and controllers due to method signature changes.
- **paperclip (2.3.16)**: Extensive regression testing required for file uploads and image processing workflows.
- **aws-sdk (1.6.9)**: Check thoroughly AWS interactions, especially authentication and bucket management; legacy SDKs have known incompatibilities with newer AWS policies.

Risk: These dependencies represent a high risk due to their deep integration in critical application areas (image processing, database connectivity). Continuous testing is required to ensure stability.

2. Core Rails Components and Syntax Changes

Several breaking changes in Rails 3 required extensive compatibility patches:

- **Routing DSL**: Legacy routes (`map.connect`) must be rewritten entirely. Backward compatibility was not feasible; routes needed explicit modernization.
- **Named Scopes (`named_scope` to `scope`)**: Compatibility patch required to temporarily support legacy syntax; recommended long-term refactoring.
- **Controller Filters (`before_filter` to `before_action`)**: Required compatibility patches to ensure a smooth incremental upgrade path.
- **Removal of `verify` method**: Introduced compatibility patches for controllers using this deprecated method.
- **ActiveRecord Serialization and Encoding Changes**: Custom UTF-8 encoding logic needed to be fully removed or replaced.
- **Incompatibility with Ruby 2.7**: The latest version of Rails 3 does not support Ruby 2.7. Making it compatible would be temporary and complex. It is recommended to move directly to Rails 4, which already supports Ruby 2.7.

Risk: Compatibility patches introduced short-term stability but increased technical debt, making the Rails 4.x upgrade more reasonable to reach a stable temporary version. A clear refactoring plan is required.

3. Soft-deletion Logic (acts_as_paranoid)

The legacy soft-deletion mechanism (`acts_as_paranoid`) posed significant challenges due to:

- Deprecated methods (`find_with_deleted`): Required custom patches to maintain existing functionality.
- New default scopes: Rails 3 changed default scope handling, impacting data retrieval logic.

Risk: These workarounds are fragile. A dedicated phase for re-implementing soft-deletion logic using modern libraries or approaches (e.g., updated `acts_as_paranoid` or explicit `unscoped`) is the recommended way to face it.

4. Customizations to Core Gems (`Money` Gem)

A critical risk emerged from legacy customizations of the `Money` gem, specifically:

- Ruby 2.7 introduced breaking changes around type coercion, causing collection and creation of currency records to fail when used with `Money` objects.
- Compatibility patches were required to adjust internal methods and avoid runtime errors (`FrozenError`, `TypeError`).

Risk: This issue is a critical risk to financial computations and demands careful and thorough testing. Consider replacing the custom fork of the Money gem with a modern, maintained version.

5. Ruby 2.7 Language-Level Breaking Changes

- Strictness with frozen string literals caused unexpected runtime errors (`FrozenError`).
- Method signature changes (especially related to keyword arguments and coercion rules) introduced subtle issues that were difficult to identify.

Risk: These subtle breaking changes significantly increased the required testing coverage. Ensuring comprehensive test automation before further upgrades is mandatory.

Key Findings from PoC Migration

1. Significant Performance Improvement

- The PoC demonstrated a **25X improvement in application boot speed**, significantly boosting developer productivity and reducing deployment time.
- Dependency management was smoother than anticipated, allowing most existing gems to function with minimal adjustments. This simplifies the temporary pipeline setup, reducing infrastructure complexity.

2. Gem Compatibility and Dependency Management

Most dependencies required minimal adjustments, with critical gems updated successfully:

- `nokogiri`, `json`, `yajl-ruby`, `ffi`, and `pg` successfully updated.
- `rmagick` required detailed environment adjustments due to ImageMagick compatibility.
- Activemerchant, omniauth-facebook (4.0.0) and omniauth (1.3.2), dropbox_api (~> 0.1.19, will_paginate (3.0.12), Paperclip (2.3.16), aws-sdk (1.6.9).

3. Critical Breaking Changes in Rails

Key areas requiring immediate attention included:

- **Routing syntax**: Old DSL (`map.connect`) required complete rewrites.
- **Scopes and callbacks**: Needed temporary compatibility layers.
- **Soft deletion (`acts_as_paranoid`) and Money logic**: Introduced complex compatibility issues, requiring careful management.

Updated Risk Overview

Area	Risk Level	Mitigation Strategy
Money gem customizations	Critical	Immediate refactoring and extensive automated testing.
Soft deletion logic	Critical	Complete rewrite of soft deletion methods across critical models.
Rails compatibility patches	Medium	Plan to refactor patches early in Rails 4.x phase.

Area	Risk Level	Mitigation Strategy
Ruby language changes	High	Prioritize full testing coverage before next upgrades.
Gem dependencies changes	Medium	Pay attention to guarantee compatibility required

Recommended Adjustments to Timeline and Resources

Short-Term (Weeks 1-6)

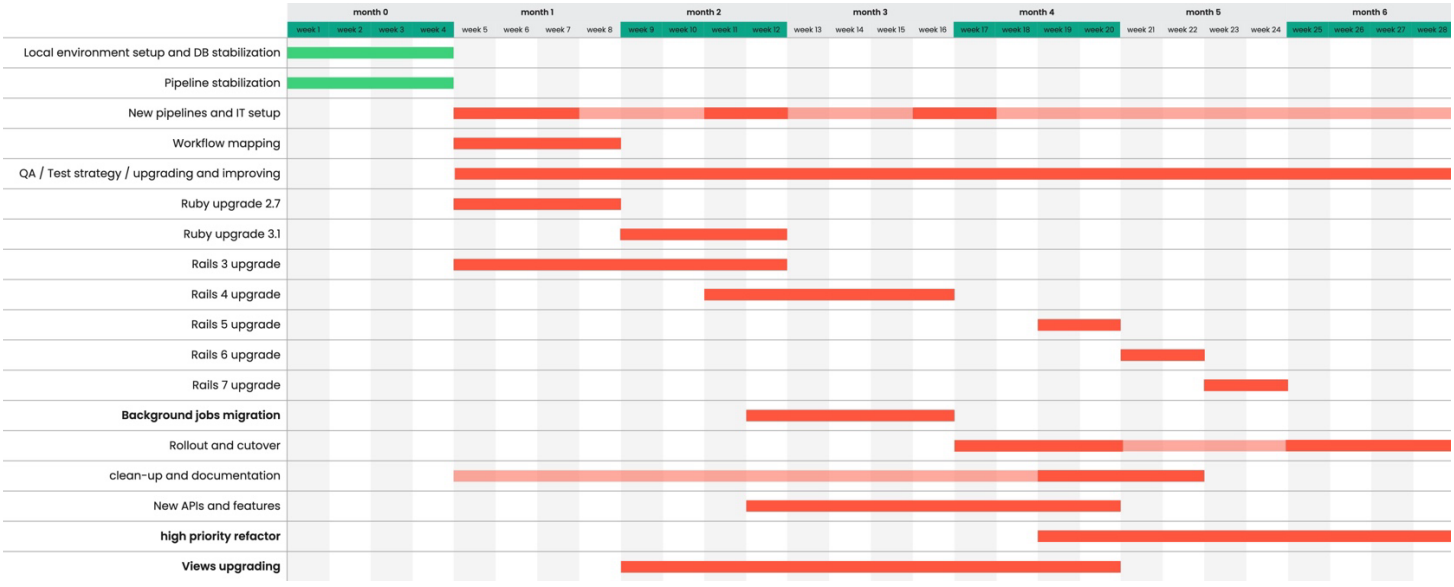
- Achieve **100% testing coverage** for Blurby distribution components involved in Blurb 2.0 coexistence. Current test coverage (~60%) is insufficient for safely managing upcoming changes.
- Clearly identify endpoints and fully document critical business logic to mitigate regression risks.

Mid-Term (Months 1-3)

- Stabilize directly at **Rails 4.x and Ruby 2.7.8**, bypassing prolonged efforts at Rails 3.x. Rails 4.x provides better long-term dependency stability and clearer upgrade paths.
- Refactor the Money logic and soft deletion mechanisms, explicitly isolating impacted models (estimated ~20 core models including Orders, Line Items, Product Options) to reduce complexity.

Long-Term (Months 3-6)

- Gradually phase out all temporary compatibility patches.
- Document clearly the modernized components and restructured business logic for easier future maintenance.
- Ensure proper handling of JS/CSS asset management through Sprockets, carefully considering pipeline changes and distribution impacts.



Resource Adjustments

Role	Allocation	Focus Adjustments
Lead Engineer (SC)	6 months	Oversee refactoring and risk mitigation, architectural design.
Automation Tester (SE/EN)	6 months	Focus immediate efforts on full test coverage.
2 Ruby Developers (SE/EN)	6 months	Prioritize Gems logic refactoring and Rails stabilization.
Frontend Developer	3 months	Coordinate closely on JS/CSS pipeline migrations.

Final Conclusions

- The PoC demonstrated a significant **25X increase in booting speed**. Most dependencies functioned well with minimal intervention, indicating that creating and configuring temporary pipelines will be easier and faster, enabling quicker implementation of changes and new features.
- Initial efforts (weeks 1-6) should focus on **achieving 100% test coverage** specifically for components part of Blurby distribution and Blurb 2.0 coexistence, clearly mapping endpoints and documenting business logic.
- Due to dependency challenges identified in Rails 3.x, it's recommended to **stabilize directly at Rails 4 and Ruby 2.7.8**. Rails 4 offers a more straightforward upgrade path and better-maintained dependencies.
- Extreme caution is advised regarding the custom `Money` logic and the soft-deletion mechanisms (`acts-as-paranoid`). Autosaving and callback complexities have introduced significant instability affecting critical financial and deletion operations. To resolve these issues, **isolated rewriting of approximately 20 critical models** (such as Orders, Line Items, and Product Options) is necessary.
- Pay close attention to the **JavaScript and CSS asset pipeline** via Sprockets. Changes in this pipeline can significantly impact application stability and performance.