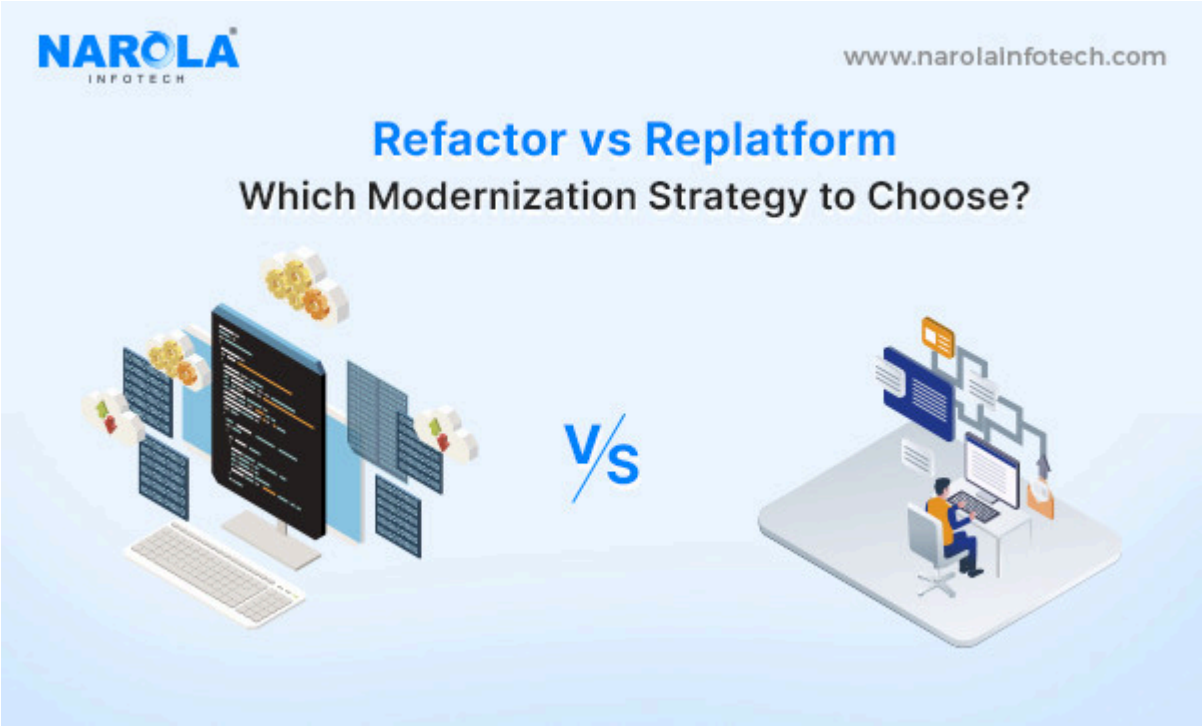
Refactor vs Replatform - What's Right for You?



Refactoring and re-platforming are two typical methods that are routinely explored while modernizing software systems. Both have their advantages and disadvantages, so picking one over the other isn't easy.
If you're trying to decide between refactoring and replatforming your program, this blog post will go over some important considerations.

Refactor

The goal of refactoring is to improve the code's structure and/or writing while keeping the code's exterior behavior unchanged. A more efficient, scalable, and maintainable code structure is the primary goal of this effort. Implementing this approach typically entails reducing large projects to more manageable parts, utilizing contemporary programming languages or frameworks, and getting rid of technical debt.

To avoid introducing new problems or regressions, refactoring necessitates an in-depth familiarity with the current system and meticulous preparation.

Replatform

However, replatforming occurs when an application is moved from one platform to another without substantially altering its codebase. When the current platform is unsupported, too old, or doesn't work with new tech, this is the method most typically used.

By preserving the application's functionality and taking advantage of the new platform, replatforming attempts to keep business disruptions to a minimum. Transferring information, settings, and integrations to a new setting while modifying them as needed to make them compatible is a common part of this process.

We'll now explore some of the key factors to consider when choosing an approach.

Comparing Refactoring vs Replatforming

1. Cost

The time and resources needed to rewrite or rearrange code might make refactoring more expensive upfront. But if it boosts the app's performance and cuts down on maintenance, it might end up saving money in the long run.

In contrast, replatforming typically requires fewer changes to the codebase, which means it is initially less expensive. Nevertheless, there can be unforeseen expenses linked to fixing technical debt or keeping everything compatible with the new platform.

2. Time

For complicated apps with large codebases, refactoring could be more time-consuming than re-platforming. To make sure the changes are applied correctly, it is necessary to plan, test, and iterate thoroughly.

Replatforming, on the other hand, is a fast process since it just entails lifting and moving the application to a different platform. The actual time frame, however, can change if the migration is very complicated or if any problems crop up throughout the process.

3. Risk

If not executed with caution, refactoring can introduce new flaws or regressions. But it's also a chance to fix things that are already broken and make the code better overall.

Conversely, replatforming aims to keep the application's current functionality, so it might be safer. On the other hand, problems with compatibility or other surprises could arise during the migration.

4. Long-term Goals

If you are seeking to update your program, make it more scalable and easier to maintain, or include new technologies, refactoring could be the way to go. You can better respond to changing business requirements and future-proof your codebase with this.

If moving to a new platform rapidly with little impact on business operations is your top priority, replatforming can be the way to go. It simplifies reaching immediate goals, but fixing technological debt or scaling concerns may necessitate reevaluating the migration plan down the road.

5. Compatibility and Interoperability

Reduced interference with subsequent systems is achieved by refactoring's granular control over preserving compatibility with preexisting interfaces and dependencies.

However, some more work may be needed to make sure that replatforming integrates well with current workflows and external services, which could cause some migration-related disruptions or compatibility problems.

6. Complexity of the Application

Applications with modular architectures or well-defined components are frequently better suited for refactoring since it enables targeted enhancements without redesigning the entire system.

Yet, re-platforming may provide an easier way to update monolithic applications with tightly linked components by dividing the application into smaller, more manageable pieces.

Refactoring Use Cases

1. Improved Code Readability

- Your codebase is cluttered, uses complex logic, or lacks proper naming conventions, making it difficult for you or other developers to understand.
- Frequent code reviews and maintenance become time-consuming due to poor readability.

2. Maintainability and Reusability

- Your code contains duplicated functionalities scattered across different parts of the application.
- Modifying a specific feature requires changes in multiple locations, increasing the risk of errors.

3. Performance Optimization

- Your application experiences slow loading times or inefficient resource usage due to poorly optimized code.
- Certain sections of code contain redundant calculations or operations.

4. Bug Fixing and Debugging

- Your code contains hidden bugs or errors that are difficult to identify due to its complexity.
- Debugging specific issues becomes time-consuming due to poorly structured code.

5. Preparing for Future Growth

- Your application is reaching its capacity, and future expansion or adding new features might be challenging with the current code structure.
- The codebase lacks proper modularity and separation of concerns, making it difficult to scale.

Replatforming Use Cases

1. Outdated Technology

- Your current platform is no longer supported or lacks security updates, making your application vulnerable.
- The technology used is outdated and hinders adding new features or functionalities.
- Scaling your application on the existing platform becomes difficult and expensive..

2. Performance Issues

- Your current platform struggles to handle increasing traffic or data volume, leading to slow loading times and crashes.
- The platform architecture is inefficient and needs a complete overhaul for better performance.

3. Limited Functionality

- Your current platform lacks the essential features needed for your application's functionality.
- Integrating additional features becomes complex or impossible on the existing platform.

4. Security Concerns

- The platform you're using has known security vulnerabilities or is prone to data breaches.
- Implementing industry-standard security measures becomes difficult on the existing platform.

5. Mergers and Acquisitions

- Two companies merge, and their applications need to be consolidated onto a single platform.
- Integrating disparate systems from both companies becomes cumbersome.

Which is Best for Your Business: Refactor or Replatform?

Finally, while deciding whether to refactor or replatform, there is no universally correct solution. There are pros and cons to both approaches; picking the right one for your business will rely on its unique requirements.

Prior to making a decision, it is crucial to thoroughly assess the expenses, timelines, risks, and desired outcomes. The success of any migration depends on careful preparation, clear

communication, and precise execution, regardless of whether you opt for replatforming or restructuring.

Source: [Refactor vs Replatform]