Considering a Rewrite?

A 7-point executive checklist to determine readiness and alternatives

When your team's delivery pace slows and technical debt piles up, a rewrite can seem like the fastest way to get back on track. But rewrites are high-risk moves—and without the right technical foundations, they often reproduce the very problems they were meant to solve.

This quick diagnostic is designed to help engineering leaders assess whether their teams are truly ready to execute a successful rewrite—or whether foundational gaps in design, refactoring, testing, and requirement clarity might sabotage the effort. It also points to safer, more sustainable alternatives you may not have considered.

Before you commit to a rewrite, use this tool to identify hidden weaknesses—and discover whether your team is better served by strengthening its capabilities first.

This diagnostic contains 7 checkpoints, each on its own page, and a section on interpreting the results. Leaders of large organizations will likely need to delegate to get the answers for each checkpoint and advice is given as to what questions should be asked when delegating.

The factors evaluated are:

- Design Discipline in Daily Work
- Refactoring Capability
- Test Creation
- Test Maintenance
- Understandability of Tests
- Pressure and Shortcutting
- Workflow Quality

1. Design Discipline in Daily Work

In your current system, are your engineers consistently improving design—even within small changes—by breaking down large algorithms, refining abstractions, and applying Design Patterns as part of their normal workflow on a daily basis?

☐ Yes ☐ No

Why it Matters

One of the most powerful forces impeding software development teams is the design quality of the code they work with. Code doesn't decay on its own—people cause decay through inattention to design.

A team that isn't actively improving the design of your current system is unlikely to do better with a new system, and will likely recreate the same problems you're facing today.

How to Answer

Ask your senior technical leads whether recent code changes (pull requests, commits) show evidence of efforts to improve design quality—or if they primarily focus on adding functionality within the existing design.

2. Refactoring Capability

Do your engineers routinely refactor code—making safe, transformative improvements to design that provably preserve behavior—as part of their regular development workflow?

☐ Yes ☐ No

Why it Matters

One of the major reasons for engineers not updating design is fear that they might damage behavior of the system in the process.

This causes the system's design to decay in ways they can't immediately see and creates a self-reinforcing cycle: as design gets worse, engineers attempt fewer design changes—causing further decay. Refactoring offers a safe way to change design without the risk of damaging the system's function, allowing engineers to start improving the maintainability of the system over time, rather than consigning the codebase to rot.

Engineers that do not understand and regularly apply true transformative refactoring will likely allow the new codebase's design to harden, reproducing the same issues.

How to Answer

Ask your senior technical leads if there are any portions of the design that engineers are afraid to change for fear of causing bugs or major slowdowns. If any such areas exist, your team likely does not fully understand how to refactor safely (even if they believe they do). Conversely, if there are no parts of a mature system with known technical debt that your engineers hesitate to change, your team likely has strong refactoring skills.

3. Test Creation

Do your engineers create tests that fully protect all new code they write?

☐ Yes ☐ No

Why it Matters

Full test-coverage provides a safety net that helps developers distinguish between intended and unintended consequences.

A fully tested module, component, or subsystem can be modified with far greater confidence and speed than one with incomplete or no test coverage because the tests will serve as an early warning system helping developers ensure they create only the intended effects—without introducing defects or other negative side effects.

Engineers that do not write tests for all new or updated code in the current codebase will likely not do so in a new codebase as well, leading to the same problems you have today.

How to Answer

Ask your senior technical leads if recent changes (pull requests, commits) consistently come with full, partial, or no test coverage. They might also tell you that it's inconsistent, depending on the developer and/or the code being updated or created. If the answer is anything other than consistently full coverage, your engineers are not creating the kind of test coverage you need.

4. Test Maintenance

When a test fails, is it treated as a strong signal that something is wrong with the production code?

☐ Yes ☐ No

Why it Matters

Having a test suite isn't good enough. You need a good test suite for your system. Tests that lay dormant until they fail and then get deleted weren't really helping teams work with confidence, they were providing a false sense of security. Tests that fail erroneously on a regular basis require constant evaluation, turning their feedback into noise and generating extra busywork for developers.

A test suite is code. It must be designed and maintained as such. Test-Driven Development (TDD) gives developers the mental framework they need to build and sustain a meaningful, reliable, resilient test suite that will stand the test of time.

Engineers who aren't engaged in full TDD in the current codebase are unlikely to start with a new codebase, dooming a rewrite to the same problems you are facing today.

How to Answer

Ask your senior technical leads the following questions:

- Are there test suites where engineers regularly disable or delete tests that fail?
- Are there test suites for which failure triggers a diagnosis of the test before diagnosis of the production code?
- Are there test suites that technically exist, but aren't run on a regular basis because the feedback they provide is not valued by engineers?
- If the answer to any of those questions is "yes", then you should answer "No" to this checkpoint.

5. Understandability of Tests

Are the teams writing tests that also serve as documentation of requirements?

☐ Yes ☐ No

Why it Matters

Tests that verify the system's design or technical functions may provide developers some confidence for a limited period of time, but ultimately tend to lock a product down and impede forward progress. By contrast, tests that verify requirements provide both developers and stakeholders with confidence and continue to provide value for the life of the requirement in question.

This is the essence of Behavior-Driven Development (BDD), which elevates tests to the level of requirements and unifies the lowest (most detailed) level of requirements with testing. Teams that engage in BDD analysis with their partners in Product produce very-high quality tests that can be read by any individual contributor involved and that provide enduring value.

If a team is not engaged in BDD with your current codebase, they are unlikely to do so with a rewrite, leading to the same kinds of problems you are facing today.

How to Answer

If you're feeling adventurous and hands-on, ask your senior technical leads if there are any tests they'd be comfortable reviewing with you. A properly-written test generated using the process of BDD is something that you should be able to understand from a business perspective, even if it's at a level of detail you don't usually work with.

If you prefer to delegate, ask your senior technical leads what level of involvement Product has in test development. If Product is actively engaged in writing tests (probably using BDD), monitoring test status, and occasionally reading old tests, then you can answer "Yes" to this checkpoint.

6. Pressure and Shortcutting

Do your teams prefer failing to meet artificial deadlines over making technical compromises?

☐ Yes ☐ No

Why it Matters

When teams take shortcuts, they are knowingly choosing a long-term cost for what they believe to be a short-term gain. Doing this as part of a regular habit destroys a codebase's maintainability over time.

In the case of a real deadline (e.g., a government regulation is about to go into effect), it might make sense to "do it wrong, now and clean up later". However, artificial deadlines such as a Sprint boundary are often treated the same way, causing the product's maintainability to decay over time.

A team that is regularly under pressure to deliver on those kinds of deadlines is likely to fold under pressure and start making the product unmaintainable, which would be just as true for a rewrite as it is, today.

How to Answer

Ask your senior technical leads if they believe the teams are sacrificing code quality to meet deadlines. If you use iterations, make sure you remind them that the iteration boundaries are a form of deadline. If you determine that there is a pattern of teams taking "shortcuts", then you should answer 'No' to this checkpoint.

7. Workflow Quality

Do your teams work to well-defined, fully understood requirements with very few mid-implementation surprises or delays?

☐ Yes ☐ No

Why it Matters

Working against shifting requirements or unstable dependencies is not only demoralizing for engineers, it is a source of risk and waste for development organizations.

The essence of Agility is that requirements change and the direction of a product should change with it. However, that doesn't require the rug to be pulled out from under a team every Sprint or Story. Frenetic requirements changes and unstable dependencies create an environment of chaos and constantly "scrambling" to meet last-minute changes. This, in turn generates risk and rework in addition to burning out your best team members.

If an organization is not working in a way that provides development teams with the stability they need to be successful on incremental changes, now, there's no reason to believe that will change during a rewrite and the same problems are likely to resurface.

How to Answer

Ask your senior technical leads if the teams feel like any of the following are true:

- Requirements change in the middle of implementation
- They deliver on requirements only to find out it wasn't what their stakeholder wanted
- They are constantly scrambling to deal with last minute changes
- They are blindsided by dependencies in the middle of implementation

If your senior technical leads confirm that any of these effects are occurring, then you should answer "No" for this checkpoint.

Interpreting Results

There are two outcomes from this checklist: Either you answered “Yes” to every checkpoint or you answered “No” to at least one of them.

If You Answered “Yes” to Every Checkpoint

If you answered “Yes” to all of these checkpoints, then your organization likely has what it needs to make a rewrite successful.

You should carefully consider the tradeoffs involved. Your teams already have everything they need to refurbish your current product. The costs and benefits of refurbishing your current product should be weighed carefully against those of a full rewrite.

Here are some example follow-up questions to ask yourself when making this decision:

- How long will it take for refurbishment to make a demonstrable impact on capacity?
- How long will it take to get to a minimum viable replacement on a rewrite?
- Does your Product team have the skills and processes needed to define a minimum viable replacement?
- How long will it take to define the high-level requirements for a minimal viable rewrite?

If You Answered “No” to Any Checkpoint

If you answered “No” to any of these checkpoints, it’s very unlikely that a rewrite will help solve your delivery problem. You should consider investing in the teams' skills and processes before commissioning a rewrite project.

At Producore we have experts across all the necessary skills to help guide your teams toward rewrite-readiness and executive-level consultants that can partner with you to ensure your goals are being met.

If you want to know more, let’s open up a dialogue.

LinkedIn www.linkedin.com/in/maxguernseyiii/

Email max@producore.com