# 15 Angular Upgrade Mistakes and How to Avoid Them

HOUSE OF ANGULAR

Upgrading Angular often reveals hidden issues that slow down development and frustrate teams. Drawing on years of hands-on experience at House of Angular, we've put together this guide to **15 real-world mistakes** that can derail your upgrade **and how to prevent them**.

## Planning Mistakes

### 📌 1. Underestimating the size and complexity of the project

Without clearly estimating the size and complexity of the codebase, it's easy to assume the upgrade will be simple. In reality, both factors heavily impact the required time, effort, and team involvement.

✅ **Start with a technical audit:** review modules, dependencies, internal libraries, and potential technical debt. Don't guess, diagnose.

### 📌 2. Overlooking the number and diversity of dependencies

Modern Angular projects often rely on dozens of external packages, from UI libraries to state management and internal tooling. Skipping a full review can lead to upgrade blockers, conflicts, and extra refactoring mid-migration.

✅ **Inventory all dependencies** early in the planning stage. Include third-party libraries, in-house packages, schematics, dev tooling, and peer dependencies. Understand what's connected before upgrading anything.

## 📌 3. Not checking long-term support for dependencies

Some libraries may appear stable but are no longer maintained or compatible with newer Angular versions. Failing to check this beforehand may result in a core dependency breaking the upgrade, causing unexpected delays.

✅ **Verify active support and compatibility for each critical package.** If support is dropped, plan alternatives, such as replacement, forking, or rewriting. Don't let outdated dependencies disrupt your upgrade process.

## 📌 4. Overlooking breaking changes in key dependencies

Some libraries introduce significant changes between versions, such as Angular Material's shift to MDC components in version 15. If you miss these, you might discover mid-upgrade that you can't move forward without a costly rewrite.

✅ **Identify critical libraries that introduce breaking changes.** Read changelogs, migration guides, and deprecation notes early. If migration is needed, plan it as a separate task with a clear timeline.

## 📌 5. Ignoring team activity and development dynamics

An Angular upgrade often results in widespread changes across the codebase. When many developers actively work on the project, especially on tasks that conflict with the update, coordinating and merging the changes becomes significantly more challenging. In the worst-case scenario, the upgrade will block other teams or features from progressing, unless the process is well-structured.

✅ **Take your team's workflow into account.** Plan how the upgrade will impact other active branches and how they will, in turn, affect the upgrade. If the scale is large, consider splitting the migration into stages and working in coordination with affected developers. A well-managed rollout minimizes disruptions and maintains steady velocity.

## Upgrade Process Mistakes

### 📌 6. Not having a defined upgrade process

In projects with a higher technical debt, the scale of an Angular upgrade can quickly exceed estimations and become unmanageable. The app may become unstable, with potential problems that are difficult to detect.

✅ **Plan the upgrade like a project.** Analyze the current state, break work into stages, and assign responsibilities. A structured upgrade process results in predictability, even in complex codebases.

### 📌 7. Skipping multiple versions in a single upgrade

Jumping several major versions at once may seem efficient, but it increases the risk of unexpected issues, broken dependencies, and bugs. It also makes it harder for the team to understand what changed between versions and why it did.

✅ **Upgrade in order.** Go one version at a time. This makes testing and debugging easier, helps to avoid version mismatches, and supports maintaining team clarity.

## 📌 8. Bundling too many changes into the upgrade

It's tempting to "clean things up" while upgrading – refactor old code, fix unrelated bugs, or remove deprecated APIs. But merging all of this at once creates PRs that are hard to test, review, and deploy safely.

✅ **Keep the upgrade focused.** Avoid mixing unrelated changes. Handle Angular upgrade first, then tackle additional cleanup in follow-up steps. This improves code quality and traceability.

## 📌 9. Not going step by step

Merging all upgrade-related changes at once can result in oversized pull requests, which can affect hundreds or even thousands of files. Such PRs are harder to test, review, and merge, and they often block other developers working on the same codebase.

✅ **Break the upgrade into smaller, logical stages.** Use separate PRs per module, feature, or dependency group. This reduces friction in code review, minimizes conflicts, and helps maintain team velocity.

## 📌 10. Skipping updates of non-essential packages

When upgrading Angular, it may seem fine to leave out packages that aren't strictly required for the upgrade itself. But these unpatched dependencies can become a lasting risk. Older versions may contain unresolved bugs, compatibility issues, or unpatched security vulnerabilities.

✅ **Take the opportunity to review and update all packages.** An upgrade is the perfect time to fill these gaps. New versions often include not just features and bug fixes, but also critical security updates that reduce future maintenance overhead.

## 📌 11. Not using available upgrade tools

Tools like Angular's checklist or Nx provide vast support for upgrades: migration schematics, version compatibility checks, and even commit breakdowns. Ignoring these resources means doing more work manually and increases the risk of inconsistencies, missed steps, or delays.

✅ **Use the tools tailored for the job.** Refer to the Angular Update Guide for step-by-step instructions. In Nx, leverage the `nx migrate` command for automated migrations and smarter dependency handling. Automation helps to avoid human error and saves time.

## 📌 12. Skipping verification of project tooling

A successful upgrade isn't just about building and running the app. It also requires checking all supporting tools: unit tests, e2e, Storybook, CI/CD pipelines. These often break silently if not reviewed after the upgrade.

✅ **Test the full project ecosystem.** After the upgrade, verify that test suites run, builds succeed in CI, visual tools like Storybook still work, and deployment pipelines behave as expected. Partial success isn't good enough in production environments.

# Bonus: Mistakes After Upgrade's Completion

## 📌 13. Assuming the upgrade is done after review

Even with green tests and approved PRs, unexpected issues can still surface, especially in the UI. Minor regressions often slip through automated tests, particularly in projects with heavily customized components or legacy styling.

✅ **Manually verify the deployed app.** Plan for a hands-on QA pass. Check for subtle UI issues and unexpected regressions in production-like environments. Staying alert in the first days after release helps catch what testing didn't.

## 📌 14. Not adopting new features enabled by the upgrade

Upgrading Angular usually unlocks access to major improvements. While most of these features are backward-compatible and not required immediately, delaying their adoption often leads to stagnation in architecture and developer experience. Older APIs and patterns will eventually be deprecated or removed. Relying on them too long can lead to rushed rewrites in the future.

✅ **Plan your modernisation.** You don't have to switch everything overnight, but it's worth identifying which new features can bring the most benefit and planning follow-up work to slowly adopt them. The earlier the shift, the more time your team gets to benefit from better tools, patterns, and productivity.

## 📌 15. Not maintaining a regular upgrade cadence

After a major upgrade, it's tempting to "leave it be" for a bit. But without a lasting update strategy, the project will slowly drift several versions behind again, reviving all the pain of future migrations. Minor, regular upgrades are easier to plan, test, and

deploy. They reduce the risk of regressions and keep the team familiar with the latest Angular standards.

✅ **Establish an upgrade routine.** Check for updates quarterly or on a per-release cycle. Staying current makes future upgrades easier and prevents technical debt from accumulating again.

No two upgrades are identical, but the issues we've outlined, based on years of hands-on experience of our company, are surprisingly consistent across different projects. We hope this guide has brought to your attention common mistakes made in the process of updating Angular and helped you understand them better.

If you need **a trusted partner to help you plan, audit, or execute your upgrade**, book a Free 30-minute Angular Consultation. Contact us if you're ready for a smoother, smarter upgrade!