Materialized Typescript Risks

Risks when converting a javascript codebase that's typed with JSDoc to Typescript.

This is a concise look back on a (still) partial typescript conversion of a Javascript codebase with around 200k LOC. *Update mid 2024: 61.8% complete at 350k LOC, end 2024: 83% at 417k LOC, end 2025: 96% at 470k LOC.*

Before we started the conversion, I wrote down risks I saw. A year later I looked back and reported to the team which of these actually materialized.[1]

TL;DR: most did.

I was critical from the start, so take this with a grain of salt.

The overall assessment of the outcome of the conversion differed between people (many like it) but the general agreement is that there is no way back.

Upsides you would miss by only looking at the risks: Many of our formerly pure **Java-devs are happy with Typescript** and we now have **API description** through autogenerated types. Also JSDoc-tooling is worse than Typescript tooling in most IDEs. But not in Emacs … (insert snark about other tools). *Update 2024: JSDoc tooling has improved substantially.*

Contents

[1] I asked for — and got — the OK to write about these here.

1 Expected and came to pass

- Loosing time with bike-shedding all the things again
  - **Many Discusssions about TS instead of our own codebase.**
  - **Lost at least 3 months of structure discussions, so the structure became more inconsistent, making it harder to find stuff by intuition.**
- Use more and more TS features that make programming more complex
  - **Adding many options quickly, problems to understand the many ways to define types. Need more time to think about stuff.**
- Rush the conversion, so the path back will be too expensive: compiling TS away will be worse than with the old JSDoc
  - **Had to remove some JSDoc typings because of inconsistencies in the IDEs.**
  - **After three months 30% were already converted ⇒ old JSDoc typings lost.**
- eslint-changes due to problems with TS => **infrastructure worsens; worse linting**
- Changing more flexible approaches that did not work well with TS
- Typescript nudges us towards more complexity.
  - **Java-Style in Web => More code in pure TS instead of using the browser as a platform**
- Developed dependencies on tooling that doesn't work as well in Emacs.
  - **Working with Typescript is much slower than with JS, because the native JS2-mode does not support Typescript ⇒ Dependency on the much slower language-server interaction.**[2]
- IDE performance
  - **All deactivated Sonar Lint for Typescript in IntelliJ, because it is too slow.**

[2]The language servers used via other IDEs are also much slower than JS2-mode, but devs using these are used to the slowness. And yes, our codebase is kind of big.

2 Expected, unsure

- Increased build times?
  - they tripled within a year of the introduction of TS, not sure about the cause.
  - avoided most direct build time effects by transpiling via Babel instead of via tsc.
  - **Update 2024**: The size of the codebase almost doubled in one year.

3 Expected, can only be seen over time

- Being locked in a Microsoft-controlled language.
- Disconnect from Javascript standard
  - **Update 2024**: disconnect between TS and JS begins to show:
    Some files were "not converted, because they were never programmed with typescript in mind"

4 Unexpected

- Emotional Lock-In of Java devs.
- Code reviews take 50% longer due to weak or strange typings.
- Cannot simply cherry-pick a fix from a converted Typescript file to its Javascript original in an earlier release (we *could* have expected this).

5 General drawbacks

- With typescript we leave the path of using only **transpiling and polyfills that will go away automatically** as support for modern Javascript improves in the browsers of our customers (IE11 died — we were so close — now the culprit is Safari).
- Browser-based debugging of typescript-tools gets us into source maps, but we cannot **test code live in the web console** and then **copy it verbatim into a file** (except if Browsers gain native TS support — would that be available in Firefox, too, or would it create more lock-in?).

6 Conclusion: conversion from JSDoc to Typescript

- ✓ Many Java devs lost their fear of the web as platform.
- ✓ More people write type information.
- !! Most of the expected risks materialized.
- !! Our infrastructure suffered due to incompatibilities.
- !! We will never get rid of transpiling. What we ship to the browser will never be what we write.
- !! We spent three months mostly talking about Typescript during which the codebase grew with too little coordination. **Whatever you choose, avoid this!**
- !! We moved faster than planned, so we have no viable way back, even if we decided to go back.

This is a comparison to a pure Javascript codebase **with JSDoc**. From that experience, I consider such a conversion as a net-negative. Others like it a lot, though, so — as said earlier — take this with a grain of salt.