Twitter Thread by Anson Horton

Anson Horton
@AnsonHorton

Incremental rebuild was a feature of the C# compiler which was meant to increase the throughput after an initial build. It worked on the principle that changes in between builds are localized, and that the information gathered by the compiler from previous builds wouldn't be

entirely invalidated; specifically, some of the information and, indeed, the assembly itself could be updated in an incremental fashion resulting in faster builds.

Both the VS 2002 and VS 2003 compilers exposed this option through the /incr switch on the command line, and the 'Incremental Rebuild' option in the Advanced tab of Project Properties. In 2002 incremental rebuild was enabled by default for all project types.

After we shipped 2002 we noticed that we started to get a large number of bugs (internal compiler errors - ICEs – the worst kind) which were a direct result of the incremental rebuild feature.

These bugs derived from the complexities involved in correctly implementing incremental rebuild and the problems associated with testing it. Consequently, for 2003 we fixed all known issues with incremental rebuild but subsequently turned it off by default for all projects.

Incremental build initially seems like a no-brainer win, it can theoretically improve compilation times by significant amounts, in certain cases we saw single assembly build times improve by as much as 6x.

However, and this may seem counter-intuitive at first, incremental rebuild could also lead to longer build times. Why? The feature had several heuristics to determine whether it should do a full rebuild or not.

One of those heuristics was if > 50% of the files being tracked needed to be recompiled. However, in most cases the files that needed to be recompiled aren't simply the files that have changes, but rather the dependency graph between the changed public interface of the types

within the files and those other source files that depend on those types. So, the compiler would do a lot of work to figure out the dependency graph, and occasionally discover it should actually do a full build anyway.

Due to how incremental rebuild worked, it would then throw out all of the work it had done, and simply perform the normal build at that point, increasing the end-to-end time. Incremental rebuild also had a few other implications that were likely non-obvious to folks.

We wanted to update the existing assembly and PDB, but due to the way they are laid out, incrementally updating them meant they would contain the old data as well and we'd just update the pointers (e.g. in the metadata table) to point to the new locations in the file.

That meant the assembly generated from an incremental build was different than what you'd get from a non-incremental build, and both be larger and slower to load.

The slower to load aspect likely wasn't a big deal; however, the different output spoke to one of the major problems that users had with incremental build. In VS 2002 it was entirely possible to build, have the compiler choke and issue an error, and the simply build again

and have it work. This lowered confidence in the compiler (reasonably), and led to odd conclusions about why the compiler was exhibiting this behavior that were unrelated to the incremental flag, because many users didn't even know it was set.

For example, folks might tweak their code, do another build, and have it work - not because of the change they made, but because a non-incremental build happened. There were at least 13 cases that would cause the compiler to bail on doing an incremental rebuild and perform a

full build instead, including things like 'more than 30 successful incremental builds without a full build', which would prevent the PE from getting too bloated over time. So, whether or not the user actually saw the incremental behavior was difficult for them to predict.

All of this led to the decision to cut incremental build in VS 2005. I should mention that this is incremental compilation of the assembly, incremental rebuild of the solution was most decidedly not cut and was greatly improved in VS 2005 through MSBuild.

The funny thing about this one is that we had initially spent significant design, added complexity to the codebase, and effort in validation (every feature added to the language needed to be separately tested in incremental cases) and when we turned it off by default in VS 2003

essentially no one noticed. We had a few folks who mentioned it in VS 2005, but not because they saw the compiler builds get slower, but simply because they saw the option was removed.

TBH the regular feedback we had was that the compiler was blazingly fast - particularly from folks who had been using C++ for a long while. This is another feature that, looking back, we probably should never have done.

It caused customers tons of headaches for little ultimate benefit. That said, I feel good about our willingness and decision to remove it. Occasionally, at Microsoft we fall prey to the sunk cost fallacy, but it's gotten much better over the years as our telemetry has

dramatically improved and we have lots of additional insight about usage and benefit versus what we had in 2003 when we made this decision. So, if anyone used VS 2002 and wondered what happened to this option, now you know :-)