# **SWG 2022-02-07 Agenda**

• Update on the GHC.X.hackage proposal

• Update on advertising the group

• Is GHC stable? (see below)

• What can we learn from Aeson 2.0 (see below)

## 

## **Is GHC Stable?**

Here is a private exchange between me and somebody else (person **X**) responsible for maintaining community infrastructure just recently.

[**ME**] Could you say a bit more about ‘*regarding future versions of GHC: I usually want to err on the side of stability, but i don’t think that perspective is terribly compatible with GHC development anymore*’? What do you think has changed, and when did it start?

[**X**] idk really; GHC 8.10 had quite a few releases (up to 8.10.7) and didn’t really feel “stable” or “reliable” until it got fairly far along; 9.0 looks like it’s gonna stop at 9.0.2, and it’s unclear how quickly development will proceed on 9.2; what i mean by that is GHC 9.2.2 still isn’t out even though 9.2.1 was released in October > > IIRC 9.2.1 has some known issues (the only ones i know of are with codegen or FFI stuff, particularly on macOS ARM), but even so ~3 months without a point release feels longer than expected

I think it is relevant and useful for us to think about whether this is just a moment we are going through and whether there is anything we can do to manage such disruptive phases.

Please note, I am a huge fan of GHC central and the amazing job they do. I am more concerned about the pressures they are being put under and ways we could manage that so that (1) people in the community (high information people also responsible for maintaining critical infrastructure) do not feel the core tools are unstable and (2) the GHC core tools team do not feel perpetually under pressure.

In particular can we identify particular releases that everyone should try and focus on supporting and regard the others as ‘work in progress’ towards the next stable release. In the current situation I would regard the 8.10 series as our focus for stability. Before that it would have been the 8.6 series. I think this is the way the community actually has been working for quite some time, but because we don’t coordinate around the reality people are being put under unnecessary pressure due to unrealistic expectations.

## **What can we learn from Aeson 2.0**

It is clear that the recent changes to Aeson are proving hard work to propagate through the ecosystem. If we had a do-over (assuming we were tasked with maintaining Aeson) would we do anything differently?

Again, I am not trying to be mean to the Aeson maintainers! Actually I have picked two of my very favourite Haskell engineering teams to look at these issues. The aim here is to try and get a better understanding of what is going on and help make the situation easier.

Notes:

* Publish GHC.X.hackage proposal to discourse?
* Possible wording clarification: “ecosystem stability”?

Action Items:

* Chris: Advertise on Reddit and Haskell Cafe

# Possible proposals

Chris seems to be suggesting

* Make an annual GHC release, marked "long term support";
  + we will backport here for XX future years
* And at the six-month mark, make a "Development release";
  + we will not backport here, or at least only backports to fix total breakage.
* No patch release (either LTS or Development) makes a breaking change, notably APIs (of course)
* The next LTS release is based on the last Development release.
* No promise that the LTS release makes no breaking change to the preceding Development release; there might indeed be breaking changes.
* Observation: refactoring of the compiler's codebase (which is very desirable, and repays technical debt) is in tension with backporting.

Question (SPJ): what reason would we have to believe that the LTS release was going to be more reliable than the development release? Is it just that we promise to invest more in backporting?

Trevis: What can we learn from other large open-source projects? SPJ: is anyone willing to go and find out? Adam might look into Rust?

Trevis: Postgres SQL maintains FIVE branches. How do they do that?

Mikolaj: my experience is that it's hard to predict the future or create a schedule that survives the contact with reality --- but then just adjust and communicate; be agile. I think the 8.10+9.0+9.2 confusion was in part a problem with trying to stubbornly stick to schedule that failed in this case

Syd’s idea:

* Every release is one-version backwards compatible and one-version forward compatible. SPJ question: what does that mean? You mean GHC 9.2 can compile every module that GHC 9.0 can? I don't see how to achieve that if there is ANY library API change.
  + Syd: SPJ is making me realise this probably isn’t realistic, but the usual way to do this is to have a forward-compatible release first.  
    So first you make sure that GHC 9.0 can compile anything that GHC 9.2 will be able to. Then you maintain that in 9.2, and in 9.4 you “drop” support for 9.0.  
    For an example of this; smos does this using cross-version E2E tests; <https://docs.smos.online/development/end-to-end-tests>
* This way no single release breaks anything, but users do still have to do some deprecation-warning fixing.
* Sadly: lots of work to maintain from the ghc perspective

Examples of how breakage could be handled that SPJ added Syd to add:

Good: When text 2.0 was coming, bodigrim made my library forward-compatible: <https://github.com/NorfairKing/validity/pull/98>  
Less helpful: When aeson 2.0.0.0 was coming, the breakage was not minimised and the whole community is still behind on upgrading. The security problem was also not even mentioned in the changelog so many people didn’t understand why the breakage was so big.