When deciding what to work on, we usually focus on issues that have a lot of thumbs-up reactions on the first comment, what we call the "popular issues".

Some popular issues are, for whatever reason, topics on which we cannot find a good way to make progress. Since those issues where we *do* make progress get closed, the result is that the list of most-popular issues is now full of only those issues where we have conspicuously not made progress!

In the interests of transparency, this wiki page discusses the status of the ten most popular issues. It is only updated occasionally and so may not be entirely up to date; for the most up to date information, please see the latest comments on the relevant issue. (Please avoid asking for an update on issues, otherwise they become full of people asking for updates and nobody can find the actual updates.)

Code Push / Hot Update / out of band updates (#14330)

There are three main areas that people are referring to here:

- **Modular application delivery**: the ability to package a single app into multiple separate archives when compiling it, and download them independently as needed. This is supported on Android via deferred components. We suspect it is not possible to achieve this on iOS with Apple's current guidelines and tools. We have not yet attempted to provide this on desktop or web, primarily because we have more important issues to resolve on those platforms first.
- **Dynamic extension loading**: the ability to download some Dart code that wasn't written when the app was first published, which adds a new feature to the app. This could be done on the fly. It may require the core app to be larger since we can't know ahead of time what is needed by each future extension. There are various solutions in this space, such as combining the rfw package and an FFI-based or Wasm-based solution (e.g. package:wasm). There is an example that provides a proof-of-concept for this combination of packages: a Flutter desktop application that knows nothing about being a calculator downloads an interface description specifying all the buttons and their layout to show on the screen, and downloads a C program compiled to Wasm to perform the calculations. The Flutter program is merely a bridge between these two downloaded files. We are looking for feedback from people using this feature; please add your experiences to issue 90218.
- **Dynamic patching**: the ability to update the Dart code of an app in the field by downloading a patch (of sorts) and providing it to the Dart VM. This would require a reload of the app to take effect. Dynamic patching was previously on our roadmap for 2019. After investigating this in greater detail, we decided not to proceed with that work. There were several factors that led us to this decision:

- To comply with our understanding of store policies on Android and iOS, any solution would be limited to JIT code on Android and interpreted code on iOS. We are not confident that the performance characteristics of such a solution on iOS would reach the quality that we demand of our product. (In other words, "it would be too slow".)
- There are some serious security concerns. Since these patches would essentially allow arbitrary code execution, they would be extremely attractive malware vectors. We could mitigate this by requiring that patches be signed using the same key as the original package, but this is error prone and any mistake would have serious consequences. This is, fundamentally, the same problem that has plagued platforms that allow execution of code from third-party sources. This problem could be mitigated by integrating with a platform update mechanism, but this defeats the purpose of an out-of-band patching mechanism.
- There is currently no out-of-the-box open source hosting solution for patching applications, so we would either have to rely on people configuring their Web servers accordingly, or we would have to create integrations for proprietary third-party services, or we would have to create our own bespoke solution. Hosting patches is a space we are not eager to enter. Having people configure their own server leaves them open to making mistakes with potentially serious implications as explained in the previous point about security. Depending on third-party services puts Flutter in an awkward position of having to pick winners and exposes us to the risk of those projects themselves making policy changes that would affect this feature.

Let flutter be installable via homebrew (#14050)

Currently, we see this as a lower priority than our other release-related work (such as working towards SLSA compliance). There are a number of other mechanisms for obtaining Flutter today, so this does not immediately unblock anyone, it is "merely" a convenience. That said, we recognize that homebrew is a pretty idiomatic way of getting software for developers on macOS, and so the request is quite valid.

If anyone would be interested in implementing an official homebrew installation path, the best thing to do would be to reach out on the #hackers-releases channel of our Discord (see [[Chat]]). Implementing it would require integrating into our release pipeline, so familiarity with that would be extremely helpful. It would also require carefully negotiating how Flutter's primary distribution mechanism (shipping the `git` repo directly) should interact with Homebrew's mechanisms, so familiarity with both of those would also be needed.

flutter run should support wireless debugging of iOS devices (#15072)

The current status is described in some detail in this comment. In brief, this is something we periodically investigate, but currently it seems that the underlying Xcode features are insufficiently reliable to really bring this to production.

This is another area where concrete contributions would be most welcome. If you would like to help, experiment with the work described in the issue so far and reach out in the #hackers-ios channel of our Discord (see [[Chat]]) with your results.

Design a new vector file format (#1831)

A design document containing both a detailed study of the problem and a strawman proposal have been published (comments welcome). The primary goal of the strawman proposal is to see if it is possible to create a format that is implemented entirely on the GPU (the thought being that creating yet another CPU-bound format doesn't really bring the industry forward). The next step is to experiment with implementing the proposal. Unfortunately all our shader experts are currently busy on higher-priority problems (like improving rendering performance and reducing jank), so this work has stalled.

As usual, contributions are welcome. Reach out to Hixie directly (either by e-mail, ian@hixie.ch, or on our [[Chat]] channels) if you are interested in helping out.

Improve the indexability (SEO) of Flutter apps on the web (#46789)

This feature is one that is recognized as important. There are some prerequisites, like improving Flutter's deep linking and accessibility features, which we have to deal with first. There are also other issues, like those around performance, plugins, and embedding, that are currently higher on the list for people who are currently contributing to Flutter's web support.

Fixing this issue is non-trivial, as Flutter's architecture is one that is fundamentally different than what the web usually expects. If you are interested in contributing, the best place to begin would be to discuss potential approaches on our #hackers-web [[Chat]] channel, followed by writing up a design doc (the process for which is also on the [[Chat]] page).

Enable "hot reload" (not just "hot restart") for Flutter Web (#53041)

Hot reload on web is tentatively on our 2022 [[roadmap]] but it's a very difficult problem so no promises.

Apple CarPlay / Android Auto support? (#26801)

For Apple CarPlay, there is a package called flutter_carplay by Oğuzhan Atalay that gives a Flutter API for controlling the CarPlay API. It's not clear that anything beyond that really makes sense for CarPlay, because Apple's API is template based and so Flutter (with its rendering engine, widgets framework, platform neutrality, etc) doesn't really provide any direct value.

For Android Auto, our understanding is that the situation is similar: there are some templates that you can fill in using Android APIs. To our knowledge nobody has yet created a plugin to expose those APIs to Dart code, but we are not aware of any reason why that would not be possible. (For extra bonus points, one could imagine a package that tries to intelligently fill both CarPlay and Android Auto templates from the same source data, but that may be hard if the templates are too different from each other.)

Currently, it does not seem that we can provide substantially more value here than anyone else could in writing plugins like Oğuzhan's, so we do not intend to work on this. We would encourage people to work with Oğuzhan and/or create new plugins for these features. Should the situation change (e.g. if CarPlay or Android Auto supported a way for us to directly send pixels to the car dashboard, allowing the power of Flutter's widgets to be useful here), we would reconsider this feature.

Add support for UWP (#14967)

UWP is one of the ways of building Windows applications ("Universal Windows Platform"). Unfortunately there are multiple ways to build Windows apps these days and it's not entirely clear which ones Microsoft want to recommend going into the future. It seems, however, that UWP support has been abandoned by Microsoft.

We plan to delete our incomplete UWP support and do not plan to support UWP in the foreseeable future. If you are interested in owning the UWP support in the flutter org repos, please ping @Hixie and @cbracken in #hackers-desktop in [[Chat]]. Owning the UWP support would require a couple of people with contributor access so that code reviews can happen, as well as cycles to maintain the code and corresponding test infrastructure.

Server-side rendering for Flutter web (#47600)

Fundamentally, rendering Flutter web apps to HTML is incompatible with Flutter's current architecture, and therefore this is not something we are likely to ever attempt. It's also not something we think is particularly useful. We see Flutter as the first of a new breed of frameworks that target WebGL and Wasm and leave HTML behind. For more detailed thoughts, see the status update on the issue.

We believe indexability (SEO) can be addressed without server-side rendering; see the issue above for a discussion of that topic.

Automatic/scalable shader warm-up (#32170)

Shader warm-up is available on every platform. To automate the generation of shader warm-up files currently requires first manually writing automated tests, and then running these in CI.

Our medium-term efforts are around removing the need for shader warm-up entirely, and therefore we are not currently planning on working on further automating the creation of shader warm-up files (no point working on something that we want to make irrelevant).

If we manage to remove the need for shader warm-up entirely, we will close issue #32170. If not, we will reconsider whether additional efforts to automate shader warm-up file generation are warranted.