KDAB's Software Development Best Practices

Desktop Development

David Faure | Managing Director KDAB France

The software engineering community has learned hard lessons over time about the best way to build software. Following tried-and-true practices provides a huge number of benefits: more resilient software, faster release schedules, higher quality products, more effective teamwork, and happier developers.

While desktop development shares a lot of best practices with other types of software, a few things make it stand apart. Development of an application that sits on a user's desktop, no matter what that consists of, requires a flexible design. Desktop applications are bigger than programs for embedded and IoT and they demand an architecture more complex than server or cloud apps with their tightly controlled environments. Here are some of our best practices for building these complex desktop applications.

1. Cross-platform

1.1. Multiplatform Builds

Targeting multiple operating systems is much more common in the desktop realm than in the embedded space. And even though it adds to your server build times, you should **always build all platforms in your CI workflow**. It's easy to introduce changes that work fine on your development OS but break on a different OS, forcing other platform developers to fix your bugs. Ideally, your CI system would build each platform as part of the pre-commit checks so developers can commit code that works consistently.

Another reason to build on all platforms is that you benefit from behavioral differences of each platform's compilers. **Compiling your code with all platform-specific compilers** forces you to write standards-compliant code and ensures that you get a diversity of warnings. Different compilers are better at detecting different types of errors, so ensuring that you have no warnings on any compiler helps improve code quality.

1.2. Adding Linux

Desktop applications generally need to be built for Windows and Mac OSX. But even if you don't intend to support Linux users, it's a good idea to **build your project under Linux** too. Not only because of additional compiler code validation and warnings as mentioned above, but because a large number of developers live in the Linux world. Including this OS in your build means any Linux developers on your team may be more proficient and productive, especially if they are able to use many of the open-source debugging and profiling tools that Linux has available.

In fact, regardless of developer skillsets and tool availability, the best idea is to always build on all three main platforms if possible. That gives your team the widest available suite of static analyzers, runtime memory checkers, optimizers, debugging tools, and compiler options to bring to bear on making a great product, not to mention making it available to the widest number of users.

2. Cross-device

2.1. Screen Resolution

Why do you need to worry about screen resolution? The OS mostly insulates your app from worrying about a diversity of display or GPU issues, and GUI frameworks and toolkits (Qt and most others) make it easy to adapt to changing screen resolutions. But because screen resolution can vary widely and produce vastly different results, you still need to **actively manage and test against the range of resolutions your app will support**. Decide on what your application's minimum resolution is and test all your app's features at that resolution to ensure dialogs are all on screen, windows have scrollbars when needed and can be scrolled to reach all content, and buttons or controls aren't placed in unreachable off-screen areas. Make sure you have the other extreme covered too by keeping a 4K, 8K, or higher resolution monitor in the design and testing mix. Test your application on high DPI screens to ensure text is readable, controls are visible, and regions are large enough to click. Also check that all dialogs can be user resizable and that your layout manager changes field layout in an intelligent way – in other words, useful for the user and in a way that still looks designed.

2.2 . Monitors

Desktops today will often have more than one monitor – even for non-power-users. Make sure that your application can use a multi-monitor configuration to its advantage by **giving it the ability to create separate windows** for many of your application features. That lets work be distributed as users see fit across those monitors, whether your app has dockable toolbars, preview windows, specialized editors, project notes, inspectors, configuration panels, or other features. By letting your users create windows that can stay open while separated from the main application window, they can maximize their monitor space and arrange content as needed for their workflow.

3. Size

3.1. Speeding up big builds

Desktop apps tend to be loaded with features. That means more lines of code, which means more time waiting on compilers. Considering multi-platform, unit tests, and CI build systems that build all variants in a single build, you've got yourself a lot of compile time on your hands. Thus, it makes sense to **do everything reasonable to reduce compilation time**. Extra seconds in the build increases the length of the code/build/test process across the entire team. Make sure you have pre-compiled headers turned on and your build tree is set up to use them. If you're not using it already, investigate ccache. This can save a huge amount of time by reusing cached compiler results if files haven't changed. If you've got a setup that supports it, you can also consider using a distributed compiler to split the build task onto multiple computers.

3.2. Multithreading

Building multithreaded apps always requires a bit of care to prevent corrupted data or data races. That's why it's important to **document your multithreading approach** for every variable, object, or class. In other words, comment which variables are being used in which threads and which mutex or synchronization object is protecting each variable. This accounting takes a bit of time but can be tremendously worth it. By knowing the author's intent and how their synchronization is intended to work, you can avoid introducing inadvertent multithreading bugs that can be extremely difficult to reproduce and fix. Another way to fight multithreading issues is to **use a thread sanitizer**. This compiler tool adds runtime and memory cost to perform its testing, but it can help detect and avoid data races that are the bane of multithreaded programming.sts and might otherwise not be found until pre-release.

3.3. Plug-ins

Desktop apps often need to be extended with third-party or other optional features, requiring a plug-in based architecture. But to make plug-ins effective, you have to ensure they have access to all the bits that make them work properly. That often requires continually expanding the scope of the plug-in interface. In turn, this drives a desire to make nearly everything in the program a plug-in since that approach ensures the plug-in APIs are all being properly exercised and are complete. However, if you've ever built an application with plug-ins, you know that they introduce a number of problems into development. Plug-ins can prevent cross-application compiler optimizations, and because plug-ins run as modules loaded by the application, they can be very difficult to test and debug.

If you need to incorporate plug-ins, make sure that you **design plug-ins to add features that are truly optional**, and bring all other functionality into the main development branch. This helps you keep the plug-in scope from taking over the entire application and turning development into an awkward and slow process.

4. Environment

4.1. Configuration

Desktop applications might need remote configuration troubleshooting, and that's why it's important to **have a text editable configuration** instead of in a database, binary file, or Windows registry. That lets support staff coach a fix over the phone or send a replacement "safe mode" configuration that resets potentially bad settings. Whether it's XML, INI, plist, or a custom text format, a text configuration helps ensure that your configuration has the same simple editing behavior on all platforms. It also allows you to offer customization of esoteric setting options, saving you from building a potentially complex GUI dialog that would get used by a tiny fraction of your user base.

4.2 Shared Libraries

When you're decomposing your application, if you **architect chunks of the application as shared libraries**, it makes it easier to unit test those subsystems. Link your main application and test executables to shared libraries, not statically linked libraries, to help save space and build time for developers. It will also make it easier for you to create a distributed application where there may be several primary components, and to distribute patches to your user.

What is KDAB's Software Development Best Practice series?

This series of whitepapers captures some of the hard-won experience that our senior engineering staff has developed over many years and projects. Offered up as a grab bag of techniques and approaches, we believe that these tips have helped us improve the overall development experience and quality of the resulting software. We hope they can offer the same benefits to you.

View all three parts of this whitepaper series online at: ***www.kdab.com/publications/bestpractices/***

About the KDAB Group

The KDAB Group is the world's leading software consultancy for architecture, development and design of Qt, C++ and OpenGL applications across desktop, embedded and mobile platforms. KDAB is the biggest independent contributor to Qt and is the world's first ISO 9001 certified Qt consulting and development company. Our experts build run-times, mix native and web technologies, solve hardware stack performance issues and porting problems for hundreds of customers, many among the Fortune 500. KDAB's tools and extensive experience in creating, debugging, profiling and porting complex applications help developers worldwide to deliver successful projects. KDAB's trainers, all full-time developers, provide market leading, hands-on, training for Qt, OpenGL and modern C++ in multiple languages.

www.kdab.com