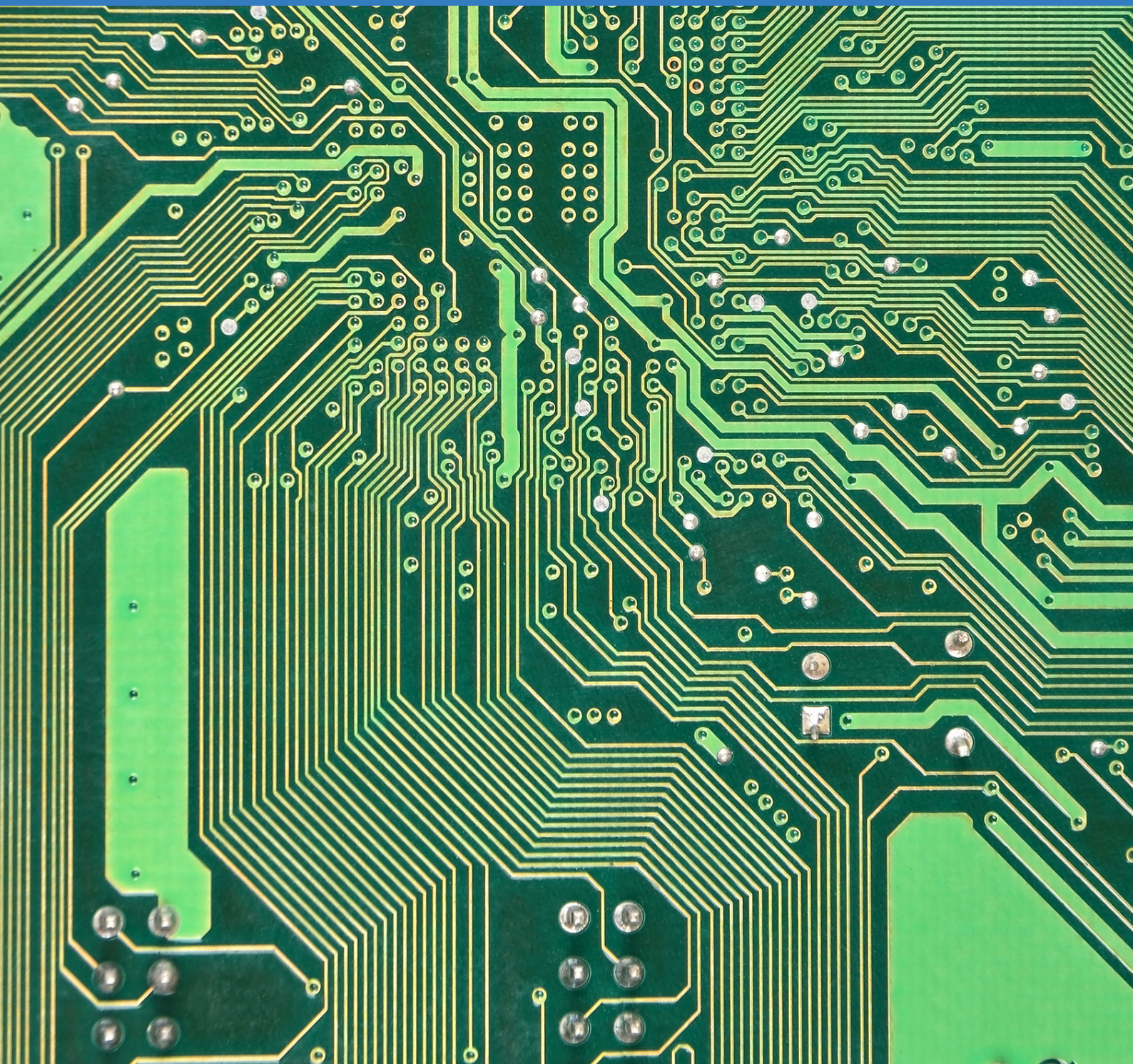
Choosing a Software Stack

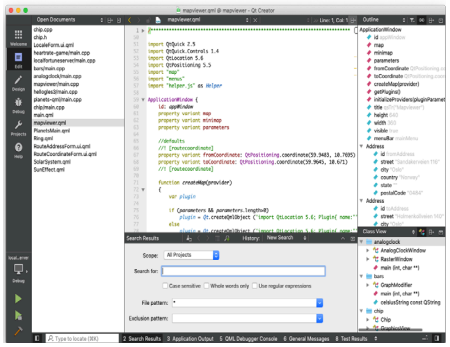
Is Qt right for your project?

KDAB

Till Adam | Chief Commercial Officer



The choice of a software stack is so important, it's worth treating the initial selection as a strategic decision.



Qt provides a powerful cross-platform development environment, but it may not be right for every situation.

Choosing a software stack

One of the most difficult choices when starting any new software project is selecting the programming language and framework your team will use to create it. Should you stick with Qt because it's the best tool for the job? Should you switch to something using web-based technology or designed explicitly for mobile? Is Python a better choice to integrate in machine-learning capabilities?

The initial software selection process is so crucial because it shapes how your software is created, its natural limitations and capabilities, the third-party resources you're able to use, who is qualified to work on it, and how long the software will be easily maintained.

However, determining the right framework can be very difficult. Web resources often provide conflicting guidance or are subjectively based on a single developer's perspective. Rarely does anyone create a substantial program in multiple frameworks that would allow a true comparison – creating a completely duplicate program of any complexity is very difficult and time consuming. As a result, software stack comparisons are really too shallow to allow readers to understand the true consequences of a choice, especially when applied to the specifics of your project. It's not surprising then that developers often follow the course of least resistance. Without a clear reason to switch, they default to the same language and framework as their previous project, reusing software that is already familiar.

Since the choice of a software stack is so important to guiding the project's future course, it's worth treating initial software selection as a strategic decision rather than the unconscious assumption that it can sometimes be. At KDAB, we often help companies in their software assessment process: frequently the question is if Qt is the right choice. We have certainly done a good deal of Qt development and believe it's a great tool. However, it's not the only tool in the toolbox and there are occasions where it's not a great fit. Here is a list of considerations that we use to help customers select a software stack that can stand the test of time, whether that's Qt or something else.

Existing codebases carry a lot of knowledge and experience with them but don't let that hinder you from a newer, better approach.

The talent pool for expert programmers can be a significant factor when deciding on your toolchain.

1. Existing codebase

How much code do you already have? Is it cleanly factored, well documented, and can it be easily reused? Existing development is a consideration that can often overwhelm other decisions. Just remember that if you have a lot of spaghetti code that can't easily be reused, it shouldn't carry the same weight as solid, reliable, and modular code. Qt and C++ is not exempt from this consideration: you can inherit terrible code written in any language. Don't let an existing legacy hinder you from making a clean break to a newer, better approach.

On the other hand, existing codebases carry a lot of knowledge and experience with them. This is often not encoded in documentation or comments but is the result of maturation over a long period of time and the accumulation of many decisions, workarounds, and fixes.

Starting from scratch with a new framework or programming language carries the risk of repeating mistakes from the past. As ever, weighing these aspects is not a black or white decision and requires nuanced and careful consideration.

2. Skills and availability

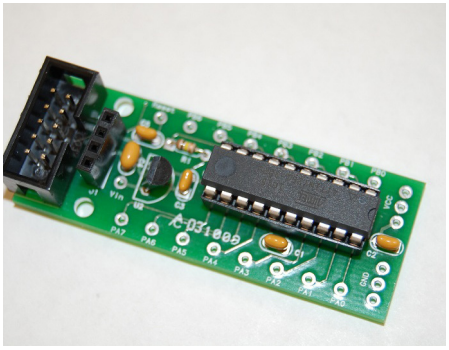
Is your development team already trained up on the software framework in question? Is it easy to find new hires who are fluent in your chosen framework? Is it easy to retrain developers to learn the new framework?

The talent pool for expert programmers in a particular framework, language, or domain can be a significant factor when deciding on tools. This is especially true for a language as powerful and nuanced as C++. We've found throughout our years of training developers from diverse backgrounds that Qt tends to tame the complexity of C++ and provides an excellent framework for programmers of many levels to be productive. The ability to make developers at different skills effective contributors comes from Qt's cleanly structured design and complete library along with QML, which due to its similarities to Javascript, makes user interface development dramatically simpler

Since almost all tools do not shine in all environments, the platform that your app targets is a major consideration.

and closer to web-based development. Even with this acknowledgement, it still remains hard to recruit high quality talent that is able to tackle challenging areas of Qt and C++: optimizing, modifying plumbing, or extending and customizing.

Outside of Qt, it is easier to fill jobs for web-based technologies. If your project primarily sticks to the basics, this could be of benefit. However, if you need to dig under the hood of your web-based framework, you may be in the same situation – excellent people are rare regardless of the language. The importance of talent pool and language choice will vary, depending on your industry, locality and ability to compete on salaries and other benefits.



Most hardware can support all development tools, but you should choose wisely at the low end.

3. Platform and silicon

Since most tools shine in some environments but not in others, the platform that your app targets is a major consideration. One of Qt's strengths is building cross-platform applications. However, while it provides an acceptable UI for all platforms, it won't be fine-tuned for each platform. If you are only building mobile apps, for example, you may want to consider a framework like React Native or Flutter that specializes in cross-platform mobile development. Similarly, if you're only targeting one of the two mobile environments, Apple IOS or Google Android, you may want to investigate the best tool for each environment – Swift or Kotlin, for example. Moreover, if your app has a desktop and web-based interface, it may make sense to use an HTML5-based tool like Electron that can more easily cross the gap between the two environments.

Finally, the size of the processor can introduce serious constraints. While an ARM microprocessor can run nearly any modern toolchain, microcontrollers have a number of constraints and can be more limiting in your language and framework choices. While Qt does offer a trimmed down version to target microcontrollers, it will require a big focus on performance to shoehorn your application into a small micro. You're probably better off picking a smaller lighter-weight framework that's designed to go into small applications like Crank

The sophistication of your graphical needs can be a big driver in selecting an appropriate programming environment.

Storyboard, if your main focus is on the low end of the hardware performance spectrum.

4. User interface requirements

What does your application need to display: simple text, responsive mobile pages, 3D graphics, complex visualizations? The sophistication of your graphical needs can be a big driver in selecting an appropriate programming environment. If you can live with console input and text output, nearly anything will do. If, however, you're building a lot of sophisticated graphs, you may want to consider a language with extensive data visualization libraries like R, Julia, or Python. Then again, if you're looking to do 3D manipulation or integrating 3D graphics into 2D UIs, that's an area where Qt really shines; web-based technologies will likely struggle in these types of applications. Because of its ability to create software closer to the metal, Qt also does very well in areas where data management is paramount, such as scientific applications, high-throughput measurement systems, or real-time data collection.

Consider your requirements for remote access, such as frequency of use, UI needs, and flexibility.

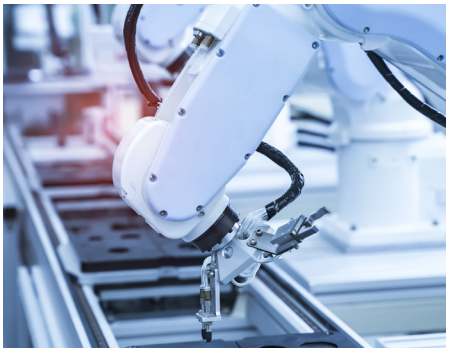
5. Remote access

If remote access is required, web-based solutions can often be a natural choice. The upside is that development will naturally use a browser interface, making remote browser access identical to local access. However, this benefit is also a downside since web technologies are constrained to the browser making it difficult to create an optimal desktop experience. It's possible to create a tailored desktop and remote UI with Qt, as it has become very versatile in being able to support remote and browser-based UIs through WebAssembly or WebGL. The requirements of remote access should be considered: will it be used for control more frequently than local access, is it for maintenance support, is it for training, oversight, or administration? Based on the frequency of use, UI needs, and flexibility, you can make a more detailed assessment on whether Qt or web-based tools are better for your remote access requirements.

Custom hardware is often found in vertical markets, making it challenging to use programming environments isolated from the hardware.

6. Vertical markets

The market for your product may impact the toolkit you want to use. Sometimes a particular tool chain has a strong foothold in a particular area like automotive or point-of-sale applications. Having access to open source libraries, third-party expert consultants, pre-existing software stacks, and third-party offerings can help shortcut the development process. Development using uncommon tools in one of those areas can mean that your team will need to reinvent the wheel for common protocols or hardware devices.



Robotics is an example of a vertical market that needs realtime performance and hardware access.

In some vertical markets the application will need to directly access hardware. If all hardware components are off-the-shelf and have properly encapsulated drivers, then this won't be a problem, regardless of language. However, custom hardware is often found in industrial or robotics designs, making them challenging to use in programming environments that use a virtual machine such as Java or web-based technologies.

The final constraint that vertical markets may impose are software lifecycle, development, and quality standards that must be met for regulation and safety reasons. You'll want to pick a language and toolset that is familiar to certifying agencies and doesn't introduce any problematic concepts like unprompted garbage collection. While the specifics of each standard differ, similar characteristics will apply to medical devices that need IEC 62304 certification, automotive systems that comply to ISO 26262, and industrial systems that require IEC 61508 compliance.

7. Tool interest and longevity

Is the language you want to use "hot" in the development community? A tool's popularity may tell you how easy it will be to have programmers join your team. "New and exciting" needs to be balanced with stability – don't pick something that's so new it is undergoing continual refinement and requires changes throughout the code base on each subsequent tool release.

Finding tools that can easily integrate unit testing into the development process can help maintainability and should be a priority for modern development.

Be sure to consider the long-term strategic impacts of development tool choice outside of engineering.

Another measure of developer interest is the community size. If your tool of choice has a large and active forum on Stack Overflow or Reddit, and ongoing projects on GitHub, you'll know there are plenty of programmers interested and contributing in that space. Newer tools may have a lot of interest but fewer practitioners and smaller online communities. Another measure of desirability is lifespan. Languages and tools that have been around a long time have proven their worth and have lived beyond obsolescence.

Finally, is the software easy to maintain? The more maintainable the code base, the longer the productive lifespan – you'll be able to keep the code fresh and reusable across multiple projects. While maintainability is often as much about programming style and documentation standards, tool characteristics can help or hurt maintainability too. Development environments that contain a hodgepodge of techniques and paradigms make it very difficult for a developer to know where and how to fix a bug or extend functionality without breaking existing code. Tools built on web standards can be difficult to modify with a combination of HTML5, JS, CSS, and custom JavaScript frameworks. However, Qt applications can also suffer from this problem due to the mix of C++, QML, and Qt libraries. Finding tools that can easily integrate unit testing into the development process can help maintainability and should be a priority for modern development.

8. Strategic considerations

Often, development tool choices are not considered by anyone outside of the engineering department. However, with the additional impacts that software tool choice imposes on hiring talent, available consultants, the cost and availability of third-party libraries, codebase longevity, and product stability, it is wise to make these strategic decisions with input from the whole team. Certainly, you want your programmers to feel the choice of development tool is a good technical fit, however, engineers may not be as aware of all the other downstream considerations – just as management, HR, leadership, or other corporate areas may not fully understand the technical aspects

There's a lot to consider when making your initial software selection; while Qt makes sense for a good number of projects, it's not the only kid on the block.

of software selection. Bringing all parties to the table to create an open dialog for selecting critical software tooling will ensure that such decisions not only help the immediate product development but also support the company's longer-term considerations.

We can help you structure your decision-making process around the software options that might be the best fit for you and give you the necessary guidance to head in that direction.

Summary

As you can see, there's a lot to consider when making your initial software selection. While Qt makes a lot of sense for a good number of projects, it's not the only kid on the block. We have lots of experience helping customers evaluate whether Qt is the right choice for a project. We know all about the framework, its license options, its strengths and weaknesses, and much about other possible choices. We have seen people pick Qt and succeed but we've also seen people have problems. We can help you structure your decision-making process around the software options that might be the best fit for you and give you the necessary guidance to head in that direction.

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 runtimes, 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