10 reasons APIs and development platforms are difficult to use

In 2012 I worked for several customers on a range of projects which were essentially about making it easier for people to use APIs, SDKS, Standards or development platforms.

What all of these have in common is that they're tools people use to build their own custom applications… and they're tools which are expensive to support because users require a significant amount of help to use them.

Why is it so difficult to make these products usable?

What people who work with APIs and development platforms say about it

Just a quick note before we go any further: in this article, I'm using "API and development platforms" as a shortened way of saying "APIs, SDKs, Standards, and development platforms". Sometimes I may even say just "API" to mean all of these. Sorry if that causes any conclusion. My point is that although they're quite different types of thing, they do have enough in comon for us to be able to talk about them as a group.

During the projects, I was lucky enough to be able to talk to around 25 people on the receiving end of these products, as well as the same number again of developers who aren't using the products I was working on, but provided input from their experience of working with other similar products. This gave me a pretty good insight into what's difficult about using these products … as well as some good ideas about how to do it right.

I talked with these people about:

- how they go about familiarizing themselves with this sort of product
- what they find frustrating
- what type of learning or information resources they found useful
- specific examples of information that they've looked up or posted on forums or contacted support about
- specific companies or products that do it well or badly

What are the difficulties?

It turns out there's a lot of common ground across these type of products or tools, both in terms of where organizations go wrong in the way they develop them and package them up as products and in terms of the types of information and other resources that would make them easier for users to explore and work with.

What makes APIs and development platforms difficult to use:

1. **Flexibility.** APIs are – by their nature – flexible: full of potential to become a variety of different implemented solutions. They're just a set of rules and potential. If people didn't need

this flexibility, they'd buy an off-the-shelf implemented product. Traditional information and tutorials focus on how to implement a solution; what's actually needed is additional information that helps users imagine what might be possible.

2. **Complexity.** Complexity is often an accidental side-effect of flexibility. The more features you add to the tool, the more difficult it is for users to identify the features that are relevant to them, and figure out the best way to implement them.
3. **Lack of UI.** Exploration is difficult. With a GUI or website, people can navigate, click and explore functionality. Until people know your API they don't know the terminology of your functionality, so even exploring the documentation is difficult. They can blindly click links within the official documentation … but that doesn't give much sense of context. To really explore, you need to implement; to implement, you need to know the basics of using the API or platform.
4. **Practicalities of delivering supporting information.** The type of product or tool often rely on documentation or specifications generated from code; depending on the tools used, this often restricts what can go in (e.g. graphics and text formatting) and when (i.e. not post-release). This limits your ability to provide information in a usable format.
5. **Technical products.** APIs are highly technical. They require specialised knowledge to communicate about how to use them. Which means expensive people.
6. **Supporting troubleshooting for a custom-built application.** The more flexible a tool is, the more difficult it is to make error-handling relevant and meaningful for an individual implementation.
7. **Development processes.** Finishing the API reference documentation is often a last thought, meaning it doesn't get the attention it needs to make sure it is usable, accurate and complete. Creating additional information beyond the reference documentation doesn't make it beyond "wish-list" status.
8. **Poor requirements management.** The API isn't clearly defined (i.e. in terms of requirements): it's more usually a by-product of other decisions. No user stories specifically about use of the API – it just exposes functionality that exists for a core GUI product or SDK; it may even include functionality that has been developed for a specific customer…or a specific customer demo. Closely related to this is…
9. **…Poor product management.** The API isn't managed as a product – features are added ad hoc; unused features are rarely removed or deprecated. Information about which features are being used and how is either not known, or is not incorporated in decisions about future developments.
10. **Small(ish) closed user base.** Examples of similar products that provide good usability are often generic development platforms and tools (e.g. JQuery) or even programming languages. These products have a large customer base who are happy to collaborate on forums, blogs and other public resources. For many of the products I was working with, the user-base was smaller and often working for competitors to each other – so knowledge-sharing was not available. This

meant that users had to rely only on the official resources provided by the company to get to know the product.

Source: http://communicationcloud.wordpress.com/2013/01/30/926/