GitHub's (Fatal?) Design Flaw

(Spoiler/quick answer: It's convoluted, cryptic, and non-intuitive - even counter-intuitive in some respects)

A principle, maybe even the cardinal principle of good design, is "Simple things should be simple." Certainly this principle is hugely important, if not all-important, in the marketplace. People will pay a 100% premium for Apple products at least partly because "it just works" - no setup required. Very popular technologies such as apps, smart phones, and tablets, are judged to a great extent on their ease (and pleasure) of use.

That's the general consumer market, which is different from the area of interest here - software development. But software developers are "humans in our spare time" - we're consumers and technology users too, and our attitudes are formed in part by our experiences when not on the job.

Once upon a time - that is, before it was so prevalent in regular life - to use technology, you typically had to accept and put up with hassles of installation, configuration, and setup. Sometimes you needed to make 2 or more machines communicate (which involved understanding their differences and mastering at least some features of each), or master syntax. You might have needed to carry out a sequence of steps, but first, modify those steps as appropriate; this might require you to obtain the right "cookbook" procedure, then to change and/or augment it in just the right way. Sometimes you needed to write your own script from scratch, aided only by a generic resource such as a language manual.

All of this meant that back in those days, technology was generally mastered and used only by "geeks." But not any more. These days, people have come to expect a better "user experience" in their daily, non-work lives, and this expectation migrates to the workplace. You've done certain tasks in other areas of life; those tasks are similar to what you want to do now. Carrying out those tasks (not at work) was pleasant, easy, and error-free; so you want/expect this work task to be similarly pleasant, easy, and error-free.

I figure that this is a reasonaable expectation. But perhaps you are a manager who feels that "that's why they call it work" and/or that people at work simply have to put up with hassles. In most workplaces you can force people to use a system, even it it's a hassle for them, simply by ordering them to do so. The follow-on effect can then be:

- you've created an antagonistic situation
- you "win the day" - you get initial and apparent compliance
- but you still lose - they find other ways to get the job done.

You may consider it perfectly reasonable, to require employees to have or acquire specialized knowledge, expend effort, put up with arbitrary procedures, syntax, terminology, menu flows, etc. You may have very good grounds for your opinion, such as security, disaster recovery, team/group collaboration, change management, process control, and automated testing. But generally, people do have choices, even at work, and if you mandate a system, service, or procedure, they may ignore your mandate and get their job done in some other way. Not becaue they love to defy - rather, because of a focus on results, on getting the job done on time, within budget, per spec.

But whatever the reasons, motivations, or workplace conditions, it's the case that sometimes, people who find a system hard to use, won't. Consequently you lose any benefits that system might have offered.

One good example is creating and keeping a source code repository. This is important, as anyone who has ever failed to "back up their work" and then lost it, can attest.

Over the past couple of decades there have been many solutions to accomplish this. "Current favorites" become popular and then fade away, generally for the same reason - they are too complicated, and a hassle to work with. Often these favorite systems have proponents, advocates, boosters. They think that their favorite systems are great; perhaps because they're smarter, or have gone through the work of mastering the system's intricacies and esoteric aspects. Or

maybe, in their boosterism, they are simply motivated to not see defects in "their baby." It's neither necessary nor possible to figure out just why and how this happens - the fact is, it does. Some people claim that a particular source code management system is wonderful, while others respectfully (and sometimes silently) except.

Here are some of the source code managment tools that I've used:

- SourceSafe, then the rebranded Microsoft Visual SourceSafe or VSS
- StarTeam
- Rational ClearCase
- PVCS
- cvs or CVS
- Subversion or SVN (its charter stated that it was to be a "compelling replacement for cvs")
- SourceForge
- GitHub

There are lots of others - Wikipedia lists a couple of dozen. These are all designed to hold source code so as to provide certain advantages. Typically 2 are paramount:

- If you lose your work, for example in a hard drive failur or by deleting or overwriting a file of source code, you can go to the repository and get back whatever version you last backed up, "checked in," contributed to the repository.
- Cutting down the risks that inevitably come up when people collaborate - meaning, more than one person can write and change code. How to make sure people don't delete, overwrite, or otherwise mess up each other's code?

The above-listed packages also offer many other purported advantages, some of which may be vital to certain groups.

When software development projects get really big and complicated, for example involving hundreds of developers working with/on tens of millions of lines of source code, one of the above-listed systems or something like it, would be critically important. They offer a lot of features I don't have time or space to list here, and these features can be vital when working with very large code bases. Similarly, if and when working in a "continuous test, continuous release/deployment" scenario, developers may simply have to master the systems' complexity (e.g. the arcane terminology and syntax), because otherwise the work simply can't be done. There are other situations (e.g. multiple overlapping versions) which require a high-powered source code manageent system.

However, these are not the only situations or scenarios. More and more these days, the power of certain systems, tools, and languages is so great, that very good results may be produced by just one developer, or a couple or a few; and in the process, they just don't create all that much source code! So all they really need, is some simple functionality. Allow others to read and use your work, protect against loss. Additional functionality, such as versioning, integration with continuous testing or deployment systems, or enabling multiple developers to work on the same small piece of source code at the same time, they just don't need.

I don't mean to imply that the work itself is simple nowadays. If John and Jane are working together to develop a new system, dividing up the work and figuring out who does what, can be tricky. They're unlikely to perfectly define the intricacies of their collaboration at the outset; nor do they expect to. The question "Who does what?" gets revisited and the answers revised as they work together. They make it up as they go along. In the process there can be false steps, failures, and errors. Both the management and the work itself are typically difficult and complex.

But, though source code management systems might claim to aid in managing this complexity, they don't really, not to any significantly helpful extent. That's because the chief difficulties in this sort of collaborative work are not simple matters of defining who has control over which lines or blocks of source code; rather, it's a matter of just what those blocks of code actually do. John needs to ensure that his changes and additions don't somehow screw up Jane's, and that he does provide her what she needs, and vice versa. Meaning, the code that John writes, has to be correct. Source code management systems can only assure that John's code is John's, is protected, etc. They can't even begin to

approach the topic of code correctness.

Fortunately they do not need to. In these cases, our needs are simple. Here are two typical needs, similar but not identical:

- I just want to check some code into a place where others can access it. By "access" I mean that I want to let them read it, copy it, and if they want, make changes to their copies and save them - as long as they don't/can't delete, nor overwrite, my original work.
- I just want to put some code out there where others can read it and if they want to, make a copy for their own use. They don't need to write to "my" repository and I don't want them to be able to.

All of the above-listed systems will can be used to fulfill the above needs, but, they are also all "overkill." In these cases, they violate the design principle "simple things should be simple." Specifically, they typically are too arcane and difficult to use, for no added, incremental, or additional benefit.

Additional benefit over or compared to what? You may ask. Answer: over tools and techniques people use day-in, day-out, such as Google Drive or Microsoft OneDrive, or just emailing files to one another. People by the millions use these techniques to share things; satisfying needs pretty much just like those stated above.

GitHub specifically

You might think (or hope) that if you want to satisfy the above simple needs, it'd be easy to do with GitHub. After all, GitHub it is at least the 3rd generation of similar tools (that I know of - meaning CVS, Subversion, GitHub) and it is very widely and enthusiastically hyped. Seems like it should be a good tool, given all that.

But it isn't - not for this purpose. When it comes to fulfilling simple needs, GitHub fails the classical design principle. To do simple things, is not simple.

Let's compare the procedure for sharing things on Google Drive and GitHub. In making this comparison, I will presume that the initial account setups have already been done; also I will skip the "authentication" (logging in) steps, for all of these are rather simple.

Google Drive:

You actually have two options here:

1. Using a browser, go to the Google Drive web interface/your URL. Then, using whatever file manager (like Windows Explorer) you favor, just drag the document to your Google Drive. You may need or want to first pick, or create, a destination folder.

   - or -

2. Use the mapped/virtual drive and/or folder that appears in your favorite/preferred file manage (like Windows Explorer). Your Google Drive shows up there looking just like a hard drive - you just copy things there and they're automatically sent across the internet to your Google Drive.

After either of the above simple steps, just tell your collaborator where to find your stuff by sending them a URL. And that's all! You're collaborating, sharing, ready to go.

GitHub:

First it should be noted that you don't have an option like option 1 above (though it may seem like you do, should, or must). Yes, once you've created your remote GitHub repository, there is a URL for it. You and/or others can go there,

look at your repository, see anything you've already loaded/put up there. You can even log in, which gives you certain powers. But, you will hunt in vain for any button or GUI element that says "upload." You just can't do it. You will intuitively want to do this 'cause "it just stands to reason" that you should be able to add things to your own repository. You can't - not with this web interface. Sorry - tough luck.

In lieu of the above, you can download specialized software (called GitHub Desktop), and install it on your system. It's a long process which among other things involves downloading a 110MB installation file - the intallation routine tells you "This may take several minutes." It's not a very polite install either; a slow taskbar progresses from left to right not once but twice, you can't choose where the software is installed, often there's no visible progress and when a progress screen does appear, it's often hidden. Once installed, you can try to use this program to accomplish what you want. If you do this you will find that to accomplish anything, you need to both figure out the GitHub Desktop's non-intutive UI, plus master GitHub's specialized vocabulary.

For example, to check something in, you first need to:

- spot/find/note the "+" symbol (the "plus sign") in the upper left corner of your screen,
- click the little downarrow just to the right of it,
- of the three buttons or options in the upper right of the resulting pop-up, choose "Clone" (no, I haven't explained what this means or why you need to do it, you just have to do it),
- click on the repository you want to clone,
- now go to the bottom of the pop-up and click "Clone {MyRepositoryName}" (note that you have to substitute the real name of your repository here),
- choose a local directory, that is, a directory on your local hard drive or file system,
- wait, perhaps briefly.

After all of that, what you have accomplished is still not your original goal - putting code out there on your remote GitHub repository. Instead, whether you wanted to or not, you've done just the opposite - brought a bunch of code down from your remote GitHub repository, to your local filesystem/hard drive. So, at this point, you can (and must):

- Copy the files you want to put out there into your GitHub repository, into the local directory you just identified in the steps above.
- (nothing appears to happen, there's no indication that what you did just worked or accomplished anything)
- Look toward the top center of your screen - there are two buttons there reading "Changes" and "History." Click "History."
- You should see another pop-up that is titled "1 change" and which offers you the ability to enter a Summary and Description. If you are not already versed in GitHub it may not be clear just what these two fields should contain/what you should type in there.
    - So go ahead and figure out what you think should go into those two boxes and type it in.
- Click "Commit to master".
- You should see a "success message that begins "Created commit..."
- Though you may think you are done, you're not, the above "success message" notwithstanding. If at this point you use your browser to go to the web interface, even after refreshing, you won't find your code out there in your repository.

At this point, your original goal, to use a GUI to drag-and-drop or otherwise put your source code out there into your GitHub repository, has actually met with failure. If there is a way to accomplish the rest of the task using graphical functionality, I have not found it. Instead, you must now:

- Go to the upper right of your screen and right-click on the repository you just tried to add to.
- From the resulting drop-down menu, choose "Open in Git Shell."
- A command-line prompt should appear; it should give you a message something like this:
    ```
    Microsoft Windows [Version 6.1.7601]
    Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

    C:\Leo\CCIS\GitHubRepo\CodeSamples>
    ```
- You can use that last line as a check: the directory shown should match the local repository you chose in a prior

step.

- However, nothing tells you what to do next. Which is:
  - At the command prompt, type
    - `git push`
- You should get back a message something like this:
  - ```
    C:\Leo\CCIS\GitHubRepo\CodeSamples>git push
    Counting objects: 3, done.
    Delta compression using up to 4 threads.
    Compressing objects: 100% (3/3), done.
    Writing objects: 100% (3/3), 3.52 KiB | 0 bytes/s, done.
    Total 3 (delta 1), reused 0 (delta 0)
    To https://github.com/LeoHeska/CodeSamples.git
       b1d45cb..ab1ed3d  master -> master
    ```

  which basically means "it worked."

Now, and only after all of this, you can go back to your browser, refresh it, and see that the code you wanted to put up in to your GitHub repository is actually there.

Perhaps you may agree with me - this is not a case of "Simple things should be simple." The need truly is simple. The above procedure, is not simple. It's lengthy and esoteric, not to mention system-dependent.

The procedure required you to enter a command at a command prompt. If you are used to working with command prompts, it might have already occurred to you that it might've just been simpler to carry out the whole process using command-line commands, and not bother with installing GitHub Desktop. And you might be right, presuming that you are able to take generic instructions and correctly come up with (syntatically and procedurally perfect) commands. I am not going to give a detailed example of this; one awful example of a thorny esoteric procedure is enough for one document. However, here is how one helpful advisor answered the question "How do I do this?":

> I have recently downloaded `GitHub` and created a repository on it. I am trying to upload an `Objective C` project in it. How do I go about doing this?
>
> asked Mar 6 '12 at 4:37
> Cai Gengyang
>
> Well, there really is a lot to this. I'm assuming you have an account on http://github.com/. If not, go get one.
>
> After that, you really can just follow their guide, its very simple and easy and the explanation is much more clear than mine: http://help.github.com/ >> http://help.github.com/mac-set-up-git/
>
> To answer your specific question: You upload files to github through the `git push` command after you have added your files you needed through `git add 'files'` and commmited them `git commit -m "my commit messsage"`
>
> answered Mar 6 '12 at 4:40
> Justin Herrick
> 1,884616
>
> Hi Justin, I have an account on GitHub and created a repository. Where exactly do I type the "git push" command ? Thanks ... – Cai Gengyang Mar 6 '12 at 7:03
>
> You're supposed to do that from the command line. – 425nesp Sep 5 '13 at 3:32
    ```

> @CaiGengyang: You do that in the Git Bash or the native system console, depending upon which one you chose while installing Git.

Now, just a couple of comments about the above question-and-answers. First of all, when the answer says "its[sic] very simple and easy", that is not in fact the case. To just "follow their guide" requires mastering GitHub concepts which are not straighforward, intuitive, obvious. Also note that that the original asker needed to come back with another question/request for clarification. Finally, note that last answerer points out that there are actually two different answers, "depending on which one you chose while instlling Git." So, there's significant complexity here.

In fact, whether you use GitHub on the command line (or rather, on one of the two command line variants) it's not just a matter of carrying out instructions, or executing the right steps in the right order. You can probably get this job done, though you might well wonder why why you have to do three things to accomplish what other systems do in a single simple step. You will need to understand what the GitHub terms "add", "commit", and "push" actually mean, but fortunately these terms' meanings are similar to the meanings of those same words in common English.

But that is not always the case for/with GitHub. For example, suppose you come up with an addition or change you'd like to offer to the GitHub community, the existence of which is GitHub's chief claim to fame and incremental/additional value. Perhaps the best way of terming what you would probably want to do next, in plain English, is "to offer." Or perhaps you might say you would create or make or publicize an "offer." But GitHub doesn't use such simple, straightforward language. Instead what you create (that is really an offer) is called, in GitHub-speak, a "pull request." There are historical and contextual reasons why this term does make some sense (if you're a GitHead), but be that as it may, the simple English term you might really expect to use, because it's simple, straightforward, and apt, isn't used. Compare the following two sentences:

- My son's a good coder and budding developer. He's written a lot of code and made several GitHub offers that have been accepted.
- My son's a good coder and budding developer. He's written a lot of code and made several GitHub pull requests that have been pulled.

A lay person with no knowledge of coding or GitHub could probably more or less understand what the first sentence meant, but this is not a sentence that you will (likely) run across. The second phrasing is what I've actually read (in an online bulletin/message board) and is, to many people, largely incomprehensible.

I would really hate to tell you how long it personally took me, to figure out just what a GitHub "pull request" really was... (but then I am rather stupid and your experience may differ from mine)

Upshots, results, consequences

One unfortunate but all-too-common result can be a conflict in communication, like this:

- Boss or hiring manager: "Hey, how can I see some of Jane's code?" I want to decide whether to hire her/put her on a project.
- Tech person: "Easily done! Just have her throw it up there on GitHub."
- Boss or hiring manager: "Should she put it in our repository? She doesn't have one of her own."
- Tech person: "Doesn't matter - it's easy either way."

But as you may see, just because in the tech person's view, "it's easy," that doesn't mean it really is. It may actually not be easy at all - or it may be easy (only) for a person who uses GitHub day-in, day-out, has the concepts and the lingo immediately available in their active/working memory, and has all the setup, configuration, vocabulary, and memorization hassles behind them. Unfortunately Jane, though a competent coder, can find herself in a position of being unable to do what the boss or hiring manager now has been told (by someone he trusts and knows is competent) should be easy. Which can look bad on her, and possibly result in a "no-hire" of a person who might actually have been a strong, competent, good contributor.

Other scenarios are basically variants of the above. Somebody mandates or suggests the use of GitHub, folks try it out, find it downright baffling, and just go around it. Jane is likely to say "the document is up there on my Google Drive - just go grab it, OK?" If Jane's in the pre-hiring process, that may work out OK. But in a production environment, if people are bypassing the official system, especially if this is happening unbeknownst to certain critical folks like change or build managers, bad things can result.

In the real world, there are many times when all you need to do is just put something out there where others can get to it, in a secure and reliable manner. And, in the real world, many tools, such as the aforementioned Google Drive, Microsoft OneDrive, Amazon S3, dropbox, WeTransfer, and dozens of others, do support just this. There is a single concept ("put") and it is generally accomplished with a single action or operation. Not so with GitHub:

- Instead of doing one thing to accomplish one task, you have to do 3 things: add, commit, and push.
- None of the 3 are particularly intuitive.
- It's possible to do the 3 in the wrong order, or in the wrong way.
- If you do, it can be difficult to figure out how to recover from/fix the resulting situation, so as to accomplish your original, simple task.

In short, and especially if your needs are simple, GitHub is overly convoluted, cryptic, and non-intuitive - even counter-intuitive in some respects.

Leo Heska
December 4, 2015