What do you think of developing for the command line first?

Asked 16 years, 4 months ago Modified 7 years, 11 months ago

Viewed 2k times

22

What are your opinions on developing for the command line first, then adding a GUI on after the fact by simply calling the command line methods?

eg.

> W:\ todo AddTask "meeting with John, re: login peer review" "John's office" "2008-08-22" "14:00"

loads `todo.exe` and calls a function called `AddTask` that does some validation and throws the meeting in a database.

Eventually you add in a screen for this:

```
==================================================

Event:    [meeting with John, re: login peer
review]

Location: [John's office]

Date:     [Fri. Aug. 22, 2008]

Time:     [ 2:00 PM]
```

```
[Clear]  [Submit]

=====================================================
```

When you click submit, it calls the same AddTask function.

Is this considered:

- a good way to code
- just for the newbies
- horrendous!.

Addendum:

I'm noticing a trend here for "shared library called by both the GUI and CLI executables." Is there some compelling reason why they would have to be separated, other than maybe the size of the binaries themselves?

Why not just call the same executable in different ways:

- `"todo /G"` when you want the full-on graphical interface
- `"todo /I"` for an interactive prompt *within* `todo.exe` (scripting, etc)
- plain old `"todo <function>"` when you just want to do one thing and be done with it.

Addendum 2:

It was mentioned that "the way [I've] described things, you [would] need to spawn an executable every time the GUI needs to do something."

Again, this wasn't my intent. When I mentioned that the example GUI called "the same `AddTask` function," I didn't mean the GUI called the command line program each time. I agree that would be totally nasty. I had intended (see first addendum) that this all be held in a single executable, since it was a tiny example, but I don't think my phrasing necessarily precluded a shared library.

Also, I'd like to thank all of you for your input. This is something that keeps popping back in my mind and I appreciate the wisdom of your experience.

language-agnostic command-line

Share

Improve this question

Follow

edited Jan 31, 2015 at 13:19

vines
5,215 ● 1 ● 28 ● 52

asked Aug 20, 2008 at 22:27

anon

In Windows you mark an executable as either a console program, or as a gui. You can't have both enabled at the same time. – Brad Gilbert Dec 20, 2008 at 16:38

Mostly true. But a CLI app can act as a wrapper for another GUI app. If the CLI app does not pass the /G flag, the GUI app simply does not create a window. See MsDev.com/.exe for a similar example – MSalters Jan 12, 2009 at 15:43

21 Answers

Sorted by: Highest score (default)

16

I would go with building a library with a command line application that links to it. Afterwards, you can create a GUI that links to the same library. Calling a command line from a GUI spawns external processes for each command and is more disruptive to the OS.

Also, with a library you can easily do unit tests for the functionality.

But even as long as your functional code is separate from your command line interpreter, then you can just re-use the source for a GUI without having the two kinds at once to perform an operation.

Share Improve this answer Follow

answered Aug 21, 2008 at 1:08

Mark Cidade
99.8k ● 33 ● 229 ● 237

7

Put the shared functionality in a library, then write a command-line and a GUI front-end for it. That way your layer transition isn't tied to the command-line.

(Also, this way adds another security concern: shouldn't the GUI first have to make sure it's the RIGHT todo.exe that is being called?)

Share Improve this answer Follow

answered Aug 20, 2008 at 22:31

Stu
15.8k ● 4 ● 45 ● 74

5

Joel wrote an article contrasting this ("unix-style") development to the GUI first ("Windows-style") method a few years back. He called it Biculturalism.

I think on Windows it will become normal (if it hasn't already) to wrap your logic into .NET assemblies, which you can then access from both a GUI and a PowerShell provider. That way you get the best of both worlds.

Share Improve this answer Follow

answered Aug 20, 2008 at 22:32

Matt Hamilton
204k ● 61 ● 392 ● 321

5

My technique for programming backend functionality first without having the need for an explicit UI (especially when the UI isn't my job yet, e.g., I'm desigining a web application that is still in the design phase) is to write unit tests.

That way I don't even need to write a console application to mock the output of my backend code -- it's all in the tests, and unlike your console app I don't have to throw the code for the tests away because they still are useful later.

Share Improve this answer Follow

answered Aug 20, 2008 at 23:02

Jon Limjap
95.3k ● 15 ● 103 ● 153

3

I think it depends on what type of application you are developing. Designing for the command line puts you on the fast track to what Alan Cooper refers to as "Implementation Model" in The Inmates are Running the Asylum. The result is a user interface that is unintuitive and difficult to use.

37signals also advocates designing your user interface first in Getting Real. Remember, for all intents and purposes, in the majority of applications, the user interface *is* the program. The back end code is just there to support it.

Share Improve this answer Follow

answered Aug 20, 2008 at 22:37

bmavity
2,508 ● 2 ● 24 ● 23

2

It's probably better to start with a command line first to make sure you have the functionality correct. If your main users can't (or won't) use the command line then you can add a GUI on top of your work.

This will make your app better suited for scripting as well as limiting the amount of upfront Bikeshedding so you can get to the actual solution faster.

Share Improve this answer Follow

answered Aug 20, 2008 at 22:34

Cristian
43.9k ● 26 ● 89 ● 99

2

If you plan to keep your command-line version of your app then I don't see a problem with doing it this way - it's not time wasted. You'll still end up coding the main functionality of your app for the command-line and so you'll have a large chunk of the work done.

I don't see working this way as being a barrier to a nice UI - you've still got the time to add one and make is usable etc.

I guess this way of working would only really work if you intend for your finished app to have both command-line and GUI variants. It's easy enough to mock a UI and build your functionality into that and then beautify the UI later.

Agree with Stu: your base functionality should be in a library that is called from the command-line and GUI code. Calling the executable from the UI is unnecessary overhead at runtime.

Share Improve this answer Follow

answered Aug 20, 2008 at 22:37
Steve
1,879 ● 3 ● 15 ● 9

2

@jcarrascal

I don't see why this has to make the GUI "bad?"
My thought would be that it would force you to think about what the "business" logic actually needs to accomplish, without worrying too much about things being pretty. Once you know what it should/can do, you can build your interface around that in whatever way makes the most sense.

Side note: Not to start a separate topic, but what is the preferred way to address answers to/comments on your questions? I considered both this, and editing the question itself.

Share Improve this answer Follow

answered Aug 20, 2008 at 22:37
anon

2

I did exactly this on one tool I wrote, and it worked great. The end result is a scriptable tool that can also be used via a GUI.

I do agree with the sentiment that you should ensure the GUI is easy and intuitive to use, so it might be wise to even develop both at the same time... a little command line feature followed by a GUI wrapper to ensure you are doing things intuitively.

If you are true to implementing both equally, the result is an app that can be used in an automated manner, which I think is very powerful for power users.

Share Improve this answer Follow

answered Aug 20, 2008 at 22:48

Mike Stone
44.6k ● 30 ● 114 ● 140

1

I usually start with a class library and a separate, really crappy and basic GUI. As the Command Line involves parsing the Command Line, I feel like i'm adding a lot of unneccessary overhead.

As a Bonus, this gives an MVC-like approach, as all the "real" code is in a Class Library. Of course, at a later stage, Refactoring the library together with a real GUI into one EXE is also an option.

Share Improve this answer Follow

answered Aug 20, 2008 at 23:08

Michael Stum
181k ● 119 ● 407 ● 540

If you do your development right, then it should be relatively easy to switch to a GUI later on in the project.

1

The problem is that it's kinda difficult to get it right.

Share Improve this answer

Follow

answered Aug 20, 2008 at 23:47

Jason

1

Kinda depends on your goal for the program, but yeah i do this from time to time - it's quicker to code, easier to debug, and easier to write quick and dirty test cases for. And so long as i structure my code properly, i can go back and tack on a GUI later without too much work.

To those suggesting that this technique will result in horrible, unusable UIs: You're right. Writing a command-line utility is a terrible way to design a GUI. Take note, everyone out there thinking of writing a UI that *isn't* a CLUI - don't prototype it as a CLUI.

But, *if you're writing new code that does not itself depend on a UI*, then go for it.

Share Improve this answer

Follow

edited Aug 21, 2008 at 0:15

answered Aug 20, 2008 at 22:32

Shog9

159k ● 36 ● 235 ● 240

1

A better approach might be to develop the logic as a lib with a well defined API and, at the dev stage, no interface (or a hard coded interface) then you can wright the CLI or GUI later

Share Improve this answer Follow

answered Aug 21, 2008 at 0:30

BCS
78.3k ● 69 ● 194 ● 298

1

I would not do this for a couple of reasons.

Design:

A GUI and a CLI are two different interfaces used to access an underlying implementation. They are generally used for different purposes (GUI is for a live user, CLI is usually accessed by scripting) and can often have different requirements. Coupling the two together is not a wise choice and is bound to cause you trouble down the road.

Performance:

The way you've described things, you need to spawn an executable every time the GUI needs to d o something. This is just plain ugly.

The right way to do this is to put the implementation in a library that's called by both the CLI and the GUI.

Share Improve this answer Follow

answered Aug 21, 2008 at 13:19
17 of 26
27.4k ● 13 ● 68 ● 85

1

John Gruber had a good post about the concept of adding a GUI to a program not designed for one: Ronco Spray-On Usability

Summary: It doesn't work. If usability isn't designed into an application from the beginning, adding it later is more work than anyone is willing to do.

Share Improve this answer Follow

edited Dec 20, 2008 at 2:14

community wiki
2 revs
Kristopher Johnson

0

@Maudite

The command-line app will check params up front and the GUI won't - but they'll still be checking the *same* params and inputting them into some generic worker functions.

Still the same goal. I don't see the command-line version affecting the quality of the GUI one.

Share Improve this answer

answered Aug 20, 2008 at 23:02

Follow

Steve
1,879 ● 3 ● 15 ● 9

0

Do a program that you expose as a web-service. then do the gui and command line to call the same web service. This approach also allows you to make a web-gui, and also to provide the functionality as SaaS to extranet partners, and/or to better secure the business logic.

This also allows your program to more easily participate in a SOA environement.

For the web-service, don't go overboard. do yaml or xml-rpc. Keep it simple.

Share Improve this answer

Follow

answered Aug 20, 2008 at 23:59

Christopher Mahan
7,619 ● 9 ● 54 ● 66

0

In addition to what [Stu](#) said, having a shared library will allow you to use it from web applications as well. Or even from an IDE plugin.

Share Improve this answer

Follow

edited May 23, 2017 at 11:45

Community Bot
1 ● 1

answered Aug 21, 2008 at 2:43

grom
16.1k ● 19 ● 65 ● 67

0

There are several reasons why doing it this way is not a good idea. A lot of them have been mentioned, so I'll just stick with one specific point.

Command-line tools are usually not interactive at all, while GUI's are. This is a fundamental difference. This is for example painful for long-running tasks.

Your command-line tool will at best print out some kind of progress information - newlines, a textual progress bar, a bunch of output, ... Any kind of error it can only output to the console.

Now you want to slap a GUI on top of that, what do you do ? Parse the output of your long-running command line tool ? Scan for WARNING and ERROR in that output to throw up a dialog box ?

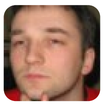
At best, most UI's built this way throw up a pulsating busy bar for as long as the command runs, then show you a success or failure dialog when the command exits. Sadly, this is how a lot of UNIX GUI programs are thrown together, making it a terrible user experience.

Most repliers here are correct in saying that you should probably abstract the actual functionality of your program into a library, then write a command-line interface and the GUI at the same time for it. All your business logic should be in your library, and either UI (yes, a command line is a UI) should only do whatever is necessary to interface between your business logic and your UI.

A command line is too poor a UI to make sure you develop your library good enough for GUI use later. You should start with both from the get-go, or start with the GUI programming. It's easy to add a command line interface to a library developed for a GUI, but it's a lot harder the other way around, precisely because of all the interactive features the GUI will need (reporting, progress, error dialogs, i18n, ...)

Share Improve this answer Follow

answered Aug 25, 2008 at 22:08



Thomas Vander Stichele
36.5k ● 14 ● 58 ● 60

0

~~Command line tools generate less events then GUI apps and usually check all the params before starting. This will limit your gui because for a gui, it could make more sense to ask for the params as your program works or afterwards.~~

~~If you don't care about the GUI then don't worry about it. If the end result will be a gui, make the gui first, then do the command line version. Or you could work on both at the same time.~~

--Massive edit--

After spending some time on my current project, I feel as though I have come full circle from my previous answer. I think it is better to do the command line first and then wrap a gui on it. If you need to, I think you can make a great gui afterwards. By doing the command line first, you get all of the arguments down first so there is no surprises (until the requirements change) when you are doing the UI/UX.

Share Improve this answer Follow

edited Jan 15, 2009 at 1:53

answered Aug 20, 2008 at 22:51

wusher
12.4k ● 22 ● 73 ● 96

0

That is exactly one of my most important realizations about coding and I wish more people would take such approach.

Just one minor clarification: The GUI should not be a wrapper around the command line. Instead one should be able to drive the core of the program from either a GUI or a command line. At least at the beginning and just basic operations.

When is this a great idea?

When you want to make sure that your domain implementation is independent of the GUI framework. You want to code *around* the framework not *into* the framework

When is this a bad idea?

When you are sure your framework will never die

Share Improve this answer

Follow

answered Jan 14, 2017 at 8:14

SystematicFrank

17.2k ● 7 ● 61 ● 106