Elizabeth Schlievert

HCI 584

Summer 2022

Developer Guide

This programmer's guide should go over all the **big picture software stuff** of your project. Think of it as the doc that a new **developer**, who is tasked to take over your project, will read to get up to speed.

**Overview:**

* Overview (in which you can re-use material from your sketch or the user's guide)
* **Condensed version** of the final(!) planning specs, i.e. which parts of the initial specs are actually currently implemented.

**Install/deployment/admin issues:**

* Assume that the developer has already read your user's guide, knows how to run your app and has your project installed. If there are additional things that the dev (but not the user) needs to do or know (e.g. to run as admin, configure something first, install other stuff, deploy the server in a specific way, watch out for that) - this is the place to mention them!

**(End) User interaction and flow through your code (“walkthrough”)**

* Start be a brief(!) recap of what the flow of the user interaction is. If you didn't talk about the UX aspects earlier, mention them here. This should include screenshots where needed.
* Now, think about what happens at each step of the flow **in your code**. Describe, which classes, functions, etc. are involved. You can be as specific as you like here, but assume that the dev is also looking at your code while reading this, so no need to literally "cite" code, unless you think it's a vital snippet.
* However, should certainly mention the names of classes, functions (and in which module they can be found) and any hierarchies that are put  into (class inheritance, multi-level modules (packages), etc.)
* It maybe be useful to use graphics to illustrate some aspects of your explanation. This could be something like class diagrams or flow diagrams or even just simple tables.

**Known Issues:**

* Minor: anything that's a minor bug (non-breaking) that could easily be fixed given some time
* Major: anything that will break, how to possible work around it and maybe how to fix it
* (These could also come up in a user-centric, UX, UI context!)
* **Optional**: Mention any known or suspected computational inefficiencies. This could be something that worked OK for your (small) data set but would be impractically slow or memory-hungry if deployed in the real-world. Or something you used a pure Python solution for but that could be done better with a C-wrapper that's using the GPU instead.

**Future work:**

* Here you can offer some insights into how your project could be expanded. Maybe there were some points in your initial spec that you didn't get to. Or maybe, during development, you found that you could also do something cool ... but didn't have the time or resources to do it.
* Optional: if you talked about inefficiencies, speculate how you could potentially solve them, given more time and money.

**Ongoing deployment/development:**

* If you don't realistically expect your project to be deployed in a serious way, you can skip this!
* If you expect that you project will be ongoing, add anything that is vital to ensure that. If possible explain how add-ons, changes and additions (in some form), should be dealt with so that your project continues to function. I know this sounds very wishy-washy - I'm thinking about unit tests, subclassing, supporting new formats, changing some aspects of the infrastructure - stuff like that. Again, I don't require this or even expect this, but if you have though about these things, mention them here.

**External documentation (i.e. the developer's guide)**

* All modules included are part of python 3 library – standard packages

You should think about (and document) how your app is started. While *you* have

probably always used Run or Debug within VS Code, it would be nice if your user

* would not have to install an IDE, just to run your app!
* This programmer's guide should go over all the **big picture software stuff** of your project. Think of it as the doc that a new **developer**, who is tasked to take over your project, will read to get up to speed.
  + Overview (in which you can re-use material from your sketch or the user's guide)
  + **Condensed version** of the final(!) planning specs, i.e. which parts of the initial specs are actually currently implemented.
  + Install/deployment/admin issues:
    - Assume that the developer has already read your user's guide, knows how to run your app and has your project installed. If there are additional things that the dev (but not the user) needs to do or know (e.g. to run as admin, configure something first, install other stuff, deploy the server in a specific way, watch out for that) - this is the place to mention them!
  + (End) User interaction and flow through your code ("walkthrough")
    - Start be a brief(!) recap of what the flow of the user interaction is. If you didn't talk about the UX aspects earlier, mention them here. This should include screenshots where needed.
    - Now, think about what happens at each step of the flow **in your code**. Describe, which classes, functions, etc. are involved. You can be as specific as you like here, but assume that the dev is also looking at your code while reading this, so no need to literally "cite" code, unless you think it's a vital snippet.
    - However, should certainly mention the names of classes, functions (and in which module they can be found) and any hierarchies that are put  into (class inheritance, multi-level modules (packages), etc.)
    - It maybe be useful to use graphics to illustrate some aspects of your explanation. This could be something like class diagrams or flow diagrams or even just simple tables.
  + Known Issues: You should mention any issues you know about (or suspect)
    - Minor: anything that's a minor bug (non-breaking) that could easily be fixed given some time
    - Major: anything that will break, how to possible work around it and maybe how to fix it
    - (These could also come up in a user-centric, UX, UI context!)
    - **Optional**: Mention any known or suspected computational inefficiencies. This could be something that worked OK for your (small) data set but would be impractically slow or memory-hungry if deployed in the real-world. Or something you used a pure Python solution for but that could be done better with a C-wrapper that's using the GPU instead.
  + Future work:
    - Here you can offer some insights into how your project could be expanded. Maybe there were some points in your initial spec that you didn't get to. Or maybe, during development, you found that you could also do something cool ... but didn't have the time or resources to do it.
    - Optional: if you talked about inefficiencies, speculate how you could potentially solve them, given more time and money.
  + Ongoing deployment/development
    - If you don't realistically expect your project to be deployed in a serious way, you can skip this!
    - If you expect that you project will be ongoing, add anything that is vital to ensure that. If possible explain how add-ons, changes and additions (in some form), should be dealt with so that your project continues to function. I know this sounds very wishy-washy - I'm thinking about unit tests, subclassing, supporting new formats, changing some aspects of the infrastructure - stuff like that. Again, I don't require this or even expect this, but if you have though about these things, mention them here.

**In-code documentation:**

* Do your best to **make it easy for a new developer to understand your code**.
* At the very least, provide good doc strings for modules, classes, methods and functions. Lecture 10 mentions 2 common systems, Google format (which I like in principle but you can create you own flavor of it, as long as you're consistent!) and reST style (which looks a bit weird for humans but can be processed by Sphinx to auto-create external documentation (in html or other formats).  You're also welcome to use other documentation systems (like pydoc) but the use of an auto doc system is not required for this class!
* You should also go over your code again and try to read it with a fresh eye (or have another person do that, if possible). Add # comments in your code to convey additional, pertinent(!) info, i.e. something that's maybe not clear from just reading your code. I like short(!) # comments at the end of some lines, for these notes (like the comment scribbled in the margins of ancient texts ...).
* If your code is long (as mine tends to be), it may help to use empty lines to structure it into visual chunks of code lines that, like paragraphs, should ideally be internally consistent, i.e. do "something" in its multiple lines. Furthermore, it may help to write a full line #  comment at beginning of a paragraph that expresses what is happening in its paragraph.
* If you know (or suspect) that there are potential issues with certain parts of your code, add a # TODO  or # GOTCHA comment about it.
* Again, all of this is so to make it easy for another developer to a) just get a good general understanding of your code by just reading it and b) make it possible to extend your code or maybe even fix a bug.