A Young Player's Guide to Assignment 3

November 7, 2008

We realise that this project is quite possibly the most ambitious piece of software development you have yet encountered. The purpose of this document is principally to set down some "traps for young players" in the form of a checklist. The intent is that you use the checklist to increase the polish and professionalism of your project (and perhaps more importantly, the marks you receive). There are also subsequent sections that expand on these points and provide advice on how to go about implementing them.

Project Checklist

Initial Proposal

- ☐ Should I quickly run my idea past Peter or Katie before writing up?
- ☐ Is my initial proposal a PDF document? (See Section 3.2)
- ☐ Does my document filename contain my name and student number? (See Section 3.2)

Design Document

- ☐ Have I converted my proposal into the final design document sufficiently? Have I..? (See Section 4.5.1)
  - ☐ included a real screenshot?
  - ☐ updated the functionality to represent the current scope?
  - ☐ updated my class design?

Reflection

- ☐ Did I list all known bugs? (See Section 4.5.4)
- ☐ Did I give consideration to alternatives? (See Section 4.5.4)

Installation Notes

(See Section 4.5.3)

- ☐ Do my installation notes have ...?
  - ☐ the operating system I developed under?
  - ☐ the packages necessary to run the program? Do they require any special installation?
- ☐ Have I specified whether my program must be run through IDLE? OR
- ☐ Does my program run on the command line? If so,
  - ☐ Does my program need to be run from a particular directory (folder)?
  - ☐ Does my program require particular arguments or flags?
- ☐ Have I attempted to make my code cross platform?
  - ☐ Does my program only have relative paths?
  - ☐ Are **all** my filenames lowercase without spaces or special characters?
  - ☐ Do my filenames use the same capitalisation for files as well as within the code?
  - ☐ Have I used the os library to handle path slashes and sloshes?
  - ☐ Have I enabled universal newline support for reading text files?

Demonstration

- ☐ Do I have a plan of how to sell my program and logically demonstrate its features? How will I conclude? (See Section 4.6.1)
- ☐ Have I tested the program on the computer I will be giving the demo on? (See Section 4.6.2)

Code

- ☐ Are any of the lines in my program longer than eighty characters (See Section 4.3.2)
- ☐ Do I have a top-level comment in each and every file? (See Section 4.3.1)
  - ☐ with my name?
  - ☐ with my program name?
  - ☐ with my program's current version?
  - ☐ with what my program does?

- ☐ with license and copyright information?

- ☐ Do I have magic numbers in my code? (See Section 4.3.3)
- ☐ Do I have repeated code in my program? (See Section 4.3.4)
- ☐ Do I have any spaghetti code? (See Section 4.3.5)

Miscellaneous - User Interfaces

(See Section 4.2)

- ☐ Does my program window have a title?
- ☐ Are my menus consistent with other programs? (eg. File, Edit, View, Help, etc)
- ☐ Is my help and about content well spaced and indented? Is the window too long?
- ☐ Are my buttons consistent with other programs? (eg. OK and Cancel - capitalisation, size, alignment and spacing are all standardised)
- ☐ Do my buttons do what they say? (eg. Will Cancel in a Save Dialog actually cancel?)
- ☐ Are the colours I've used what the user expects? (eg. a bright pink Cancel box could be jarring for a user)
- ☐ How well does my interface align with user expectations?
- ☐ Does my interface mainly require the user to use the keyboard or mouse? Or do they have to keep switching between them?
- ☐ Could there be aspects of my program that are counter-intuitive to people other than me?
- ☐ Could there be aspects of my program that may annoy the user? (eg. think popup ads)
- ☐ Could there be aspects of my program where the user feels lost? (eg. long delays with no feedback while something loads)

Contents

1 Staff Assistance

We urge you to talk to us (Peter and Katie) about your project - your ideas, problems you are having or if you just want to get some informal feedback over the course of your project.

Peter
Office: 78-316
Phone: (07) 3365-3461
Email: pjr@itee.uq.edu.au

Katie
Office: 78-320
Email: duczmal@itee.uq.edu.au

2 Assignment Expectations

As mentioned previously, we realise that this project is quite possibly the most ambitious piece of software development you have yet encountered. Although this is a first year course, the type of marking scheme we use is quite different from that in other first year courses. We will assess you hard for you to find out where your limits are so you can grow as a Software Engineer - this doesn't mean we're trying to push you to past your breaking point or "that we're just out to fail you". See the proceeding section for some frequently asked questions and common misconceptions.

2.1 Questions and Misconceptions

I need to do this assignment in order to do better than pass!

It is possible to get a 6 (a pass with distinction) without having completed Assignment 3 by achieving near full marks on all the other assessment items. Certainly this assignment adds some "buffer room" to the marks needed for the other assessment for a 6 but it is only if you are aiming for a 7 that Assignment 3 is necessary.

What's all this project ranking stuff about?

As stated in the assignment sheet, because each student chooses their own project, it is difficult to be prescriptive about the marking scheme. After running this assignment for a few semesters, we believe that a project grading is the best way to give you a feel for how the project might go. We want you to

feel free to talk to us anytime about your project to keep it on track with your goals.

This assignment is too open-ended. There's no clear criteria for what I have to do.

Assignment 3 can be particularly stressful because of its complexity and its openness. We choose to leave Assignment 3 open to give students an opportunity to do something they are really interested in or something they really want to do. Some students can find the freedom that Assignment 3 provides a burden because the assignments vary from person to person and hence it is more difficult for you to get a sense of your progress than in the previous assignments which had generic criteria.

I've seen some of the examples on the course website. I could never do something that complex.

These are examples of the "best of the best" - the assignments received marks in the high 30s out of 40 marks. The students that received high marks were wholly passionate about their project - it was more than just an assignment but was something that they continued doing in their leisure time. It isn't necessary to go to these lengths for Assignment 3 in order to do well in the course.

Having said that, these projects may not have necessarily been "Five" projects when we approved their initial proposals however they showed us something with the "wow" factor on the day by adding more polish or more features. Don't be afraid to try, you may surprise yourself.

Isn't this course about programming? Why do I have to do all the other stuff?

Certainly one of the key outcomes of the course is learning the principles of object-oriented programming but remember that the course is not called Introduction to Programming but rather Introduction to Software Engineering which covers a greater area than programming . In the course of the assignment we expect you will spend many hours looking at documentation, planning your approach, diagnosing bugs and errors, strategising where on your project to focus your attention, considering how to design your user interface and others. All of these stand you in good stead for future software development. Unfortunately this can be obscured when you are in the thick of a difficult assignment - this is an appreciation that develops over time.

I've been putting all my time and energy into this assignment. My other courses are suffering.

As Assignment 3 students from previous semesters will tell you, time management is an extremely important aspect of this assignment. You need to make tradeoffs - it isn't necessary to get a perfect Assignment 3 in order to do well in this course. We aren't looking at you to pour every spare minute into the assignment to the detriment of all your other courses.

I only received 19 out of 40 marks. I've never failed an assignment before.

19 out of 40 is not failure for Assignment 3. Understand that it is not necessary to produce a flawless Assignment 3 in order get a 6 or a 7 overall - many students in previous semesters received 7s while still receiving around 20/40. We can appreciate that for a student looking for top marks, receiving 50% for a very difficult assignment can be very confronting but this does constitute failure.

3 The Proposal

3.1 Approach

Many students will spend a few hours trying out their idea to see how workable it is for them. This is a reasonable approach. We do warn you however not to overcommit to your initial idea. Writing the entire assignment beforehand and reverse engineering the proposal from it is unwise as the project may require significant revision to be appropriate. Please feel free to sound us out on project ideas informally before submitting your initial proposal - it could save a lot of headaches later on.

3.2 Format

We want your document in PDF. More specifically we will not accept .doc or docx formats. ITEE has support for printing PDFs (see `http://studenthelp.itee.uq.edu.au/printing/pdfprint.html`) There are also many free PDF virtual printers available for Windows, just search for PDF printer on `download.com`. Linux allows fairly straightforward printing of PDFs out of the box.

We would also appreciate if you could put your name and student number in the filename - dozens of files entitled `assign3proposal.pdf` can become confusing.

3.3 Project Ideas

You may have a firm idea of what you want to do for Assignment 3 in your mind. However if you're having difficulty coming up with a project idea you can take inspiration from programs that you use everyday (like writing a simple version of Word, Firefox, Photoshop or iTunes) or if you have a casual job you may like to write a tool to help you or your boss or you could ask yourself the question "What program could I write that would make my life easier?"

You may also be able to refine your focus by deciding what type of program you want to write. For example,

- a game (arcade, side-scrolling adventure, puzzle - have a look on `zone.com` or old DOS adventure games for ideas)
- a network application (like a small web server, instant message or IRC client, or networked game)
- a web application (using Django, Google App Engine, or raw CGI)
- a computer graphics application (image processing, image rendering)
- a multimedia application (playing sound or music or video)
- a information security application (encryption, steganography)

Also browsing through available Python libraries may inspire you. A few interesting ones are:

- Pywiimote (connects your Wii remote to your computer using Python)
- Django (a Python based web framework)
- PyTesser (adds OCR capabilities to Python)
- Pedro (a Python library for subscription based network communication - suitable for a network game)
- msnp.py (a MSN Messenging API for Python)

There is a list of previous proposal topics we've received in Appendix A. It should be noted that the designs varied greatly across the zero to five spectrum but all had the potential to be five projects with the right features and additions.

3.4 Be detailed

The more detailed you are about your project, what you are trying to achieve and how you intend to go about it, the more advice we can give you about its scope and methodology for implementing it.

3.5 Decompose the project

Even with the most enthusiasm and best of intentions, life can often get in the way of a serious project. It's a really good idea to break the project down into smaller components. Categorise these components into those that you will implement versus those that you may implement given enough time. You may even want to go as far as having a wishlist category for those really amazing ideas that you think may be unrealistic to implement in the timeframe. This is useful because it helps you clarify your direction and give you priorities to focus on.

4 The Project

It should be noted that this project is more difficult than Assignments 1 and 2. In fact, Assignments 1 and 2 would be considered as "One" projects on the project complexity scale. This is true not just in the scope of the project but a higher standard of software development is also expected.

4.1 Time Management

Time management has been the prime problem with Assignment 3 projects in previous semesters. As with projects in industry, people often underestimate the time required to implement functionality. Too often students ask for help too late. Two tips: first, after getting project approval spend some time experimenting with the key aspects of the project (libraries, algorithms, etc) to get a feel for how difficult these are. Second, set some deadlines for particular functionality to be established (think the point of no return). If these are exceeded consider either simplifying the functionality to a fallback position or asking either Peter or Katie for advice.

4.2 User Interfaces

We don't expect you to be a user interface expert in this course however keeping the few tips from the checklist in the back of your mind while implementing can help provide a more stylish product.

A lot of user interface design revolves around two things: user expectations and usability. Excluding complete computer beginners, users have interacted with programs under a particular operating system and have implicit expectations of how they behave. Colours, positioning, alignment, type of widgets, keyboard and mouse shortcuts can all have meanings the user associates with particular aspects of programs. When programs have aspects that differ from these expectations, users can find them jarring and either are put off or quit using

the program entirely. Typically games have less rigid user expectations than productivity programs, however there may be styles that can be used to align with user expectations. For example, emulating an older style game could lend itself to 8-bit graphics and internal PC speaker beeps. Example projects with excellent game interfaces are located on the course webpage.

Usability is a very large area of which a very tiny portion will be discussed here. Consider Microsoft Word (or any modern word processor) which has been experimenting with usability for many years. There are often several ways to do common actions - Paste can occur from the menu, the toolbar or using the Ctrl-V shortcut. Typically, less experienced users will use the mouse more than power users. Thinking about ways to avoid forcing the user to constantly move from keyboard to mouse again will improve usability.

4.3 Code Style

4.3.1 Top-level comment

At the top of every source code file we're looking for a top level comment with your name, the name of the program, current version, what it does and also license/copyright information. In the case of multiple files, there should also be a description of what purpose of the particular file is. Looking at open source code from the internet may give you an idea of what these should look like. Developers that may want to extend your program can look at these comments to work out the state of the project.

4.3.2 Line wrapping

Excluding some unbreakable formulas, each line of code should be no longer than eighty (80) characters. While not apparent on modern widescreen monitors, eighty characters is the width of a standard Linux terminal shell and more importantly the width of a standard printed page. You should manually break lines in order to make your printed code readable.

4.3.3 Magic Numbers

As a general rule there should be no numbers passed directly to functions. Eg

```
a = 32 * 32
```

In the above example there is no context to indicate the purpose of the number `32`. Numbers such as these are referred to as "magic numbers". These are considered poor style as it makes it more difficult for another programmer to read or indeed for you to read months down the track when you have forgotten

the significance of the number. This problem can also occur for any type of variable (there are also magic strings).

4.3.4 Repeated Code

As the amount of code you write grows you will notice that common code fragments will emerge. Periodically look through your code for these fragments and ask yourself "Can I rewrite this so I don't have multiple bits of code doing the same thing?" Alternately, if you find yourself copying and pasting entire swathes of code stop and think whether you could create a class, procedure or a loop instead. Your ability to generalise code will develop over time as you gain more experience prgramming.

4.3.5 Spaghetti Code

Somewhat related to the previous section, spaghetti code occurs when the flow of your program is convoluted from an obfuscated class structure or procedures. Again, the more you program, the better your ability to write clear and concise code and avoid the spaghetti.

4.4 Backup and Version Control

As you gradually scale up the development projects you take on - backup and version control of your code becomes more important. At its simplest you may elect to keep progressive copies of your assignment on a USB drive or similar - whenever you reach a coding milestone. There are also more sophisticated backup (such as unison) and version control systems (such as CVS, SVN and git). These typically require a bit more setup. Feel free to talk to us if you are interested.

4.5 Documentation

4.5.1 Good Proposals help make good final documents

Even though the proposal is not assessed as such, creating a polished proposal can save time later and serve as a starting point for the final assesable documentation.

Details about what packages you use, how the program works will hopefully require few alterations for your final document. Bedding this down early will allow you to focus on the reflection and installation documentation.

Note that if you included a sketch or mock-up in your initial proposal, we expect you to include a real screen shot for your final design documentation.

4.5.2 Start Early

Often students leave documentation until the last minute in favour of coding last minute features. Don't underestimate the time required for documentation. This assignment requires a detailed project breakdown and the reflective component needs to display a grasp of critical analysis.

4.5.3 Installation

The installation notes you submit are used for us to try out your program. The notes should be about half a page. You should specify:

- the operating system you developed under
- which packages are necessary and whether they require any special installation
- whether your program must be run through IDLE or from the command line and if from the command line whether:
  - your program must be run from a particular directory
  - your program requires particular arguments or flags

We (Peter and Katie) are both Linux users and while we don't expect you to develop on our platform of choice we do ask that you take a few simple steps to keep your code relatively platform independent. Please be aware that we run your programs ourselves before the demonstrations in order to give maximum feedback possible. Taking these steps will show off your project in the best possible light.

1. Please avoid hardcoded absolute paths. As much as possible keep the directories you use within your root program directory and keep the paths you specify relative to this root directory. You may also want to have a configuration file separate to your program code where you specify any directories you use so they are easy to change (which you should specify as part of your installation notes).
2. We request that you keep your directory and filenames lowercase and without spaces or special characters. Something like `my_program_name` would be best.
3. Windows considers the files `BaTh.JPG` and `bath.jpg` as the same whereas other operating systems do not. We request that you keep your capitalisation consistent between your files and your code.
4. Instead of writing your paths directly we ask that you use the os library to handle these in a platform independent way. See `http://www.oreillynet.com/onlamp/blog/2008/01/pymotw_ospath.html`

5. Be aware that new lines are either represented as one character or two depending on the operating system. This can be handled seamlessly but using universal newline support: `open('/tmp/hello.txt', 'U').read()`

4.5.4 Reflection

We are interested in what you got out of your project and so we would like you to reflect on what you have achieved. We understand that your project will not be perfect and there will be several things you think could be improved or done in a better way. This document gives you a chance to reflect on all aspects of your project. All known bugs and unwanted features should be clearly listed. You may also discuss issues such as completeness, efficiency, modularity, documentation, usability.

4.6 Demonstration

Later this semester we will post a message to the newsgroup asking you to sign up to a presentation time. The presentation will be 20 minutes long in which you need to give us (Peter and Katie, not the whole class) a demonstration of your program and we will provide feedback about your program and your code. The presentation is also a means for ensuring that you have authored your own work.

In the presentation, we are looking for you to sell your program to us. Show us all the features in the best possible light. In previous semesters it has been possible to secure all possible marks for a particular project band (or occasionally a few more) independent of the quality of other project deliverables.

Typically, you don't need to discuss code with us as part of your demonstration - we will do a code review with you afterwards. You may however choose to talk about code in your demonstration if your project has a very large "behind-the-scenes" component - eg. a physics calculation library, a purely AI program.

During the presentation we will also be providing constructive criticism about your program and your code. Being apologetic or defensive can detract from your presentation, we ask that you accept the feedback as it was intended, to help you improve for future projects.

4.6.1 Plan your demo

It's worthwhile to make a brief plan of how to demonstrate your project. Often students are nervous and can become flustered - having a logical and methodical means for showing all your program's functionality can make the demonstration go much smoother. Also think about how you can conclude your demo - "So yeah that's it" probably isn't the best note to end on.

4.6.2 Test under demo conditions

If you decide to use a different computer to that you developed on, be sure to do a "dry run" to make sure the computer is set up properly and the program runs as expected.

A Previous project ideas

Calendaring/Organisers/Notetakers:

- Calendar and Tasks application
- Note taking wiki-style with integrated Google and Wikipedia search
- Share house management application
- Shopping receipt analyser (reads in scanned dockets to analyse them)
- Study motivation tool (completing uni tasks unlocks RSS feed articles)
- University Course organiser

Games:

- Advanced Sudoku solver
- Asteroids
- Breakout
- Checkers
- Extended Battleships
- Klondike and Spider Solitare
- Map based adventure game
- Mini Golf game
- Ore Mining game
- Safari hunter - side scrolling arcade game
- Scrabble
- Snake game
- Space invaders
- 2 person Chess
- Python implementation of Tank DOS game
- Warhammer battle calculator

Image-related:

- Image processor (brightness, contrast, resize, zoom, saturation, glow, blur, transparency, merge)
- Silhouette drawing game
- Sliding tile picture puzzle
- SVG Viewer/Editor

Network:

- Simple chat server
- IRC client with spell check
- Network shoot-em-up flying game
- Web server

Security:

- Letter-based cipher encryptor and simple steganography application

Specialised subject areas:

- Chess puzzles
- Chip-8 emulator (emulator of a 1970's game console)
- DC circuit analyser and GUI circuit editor
- Geophysics Calculator
- Interactive flash cards for foreign language learning
- Maths demonstrator

Internet/Web Applications:

- Advanced implementation of Yahoo Open Shortcuts
- Online stocks monitor
- YouTube music video playlist application

Work related:

- Employee roster generator
- Network Design GUI

- Python port of a CSV data filter for massive datasets
- Simulator for safety of giant mining trucks
- System for generating tutor timesheets
- Trailer and vehicle hire management for a service station