The Occidental Computer Science Comprehensive Project: Goals, Format, and Advice

Maryo Botros
justinnhli@oxy.edu
Occidental College

Abstract

Although they have not been adopted on a large scale, self-driving cars already exist and as with any new technology, they pose some ethical dilemmas. One of the major ethical dilemmas posed by self-driving cars is the allocation of responsibility. Ideally, self-driving cars should never be involved in any accidents, as that is one of the things they are meant to prevent, but accidents have occured because of self-driving car technology and sensors can sometimes be faulty when perceiving the environment, so an accident is a possibility, but understanding who is responsible can be an issue. Just about all self-driving cars on the road today require that a driver be in the driver's seat and this person has just as much responsibility as any other driver on the road. However, this is only a temporary solution as self-driving cars are meant to be completely autonomous and there won't be all people in the car who will be considered proper passengers. When we reach the point where cars are completely autonomous and the computer processing artificial intelligence is the entity that is "driving," who is responsible for the safety of the passengers in the car as well as the safety of others traveling and walking on the same roads? Is the vehicle owner, vehicle manufacturer, passengers? There are two major forms of responsibility that can be used to analyze the issues with allocating responsibility when it comes to self-driving cars: task responsibility, commonly referred to as "forwards-looking responsibility," and blame responsibility, often called "backwards-looking responsibility. Having a task responsibility means to be obliged to do something. Having a blame responsibility means that one is to be blamed if something goes wrong (First). Blame responsibility is often associated with punishments or with duties to compensate. What will happen with our responsibility ascriptions when driverless cars are introduced? One thing should be clear: since the users of fully automated vehicles. Users of fully automated vehicles have no control over the vehicle, other than their choice of a destination, it would be difficult to hold them responsible either for safety (task responsibility) or for accidents (blame responsibility) because we cannot hold people responsible for something they have no control over. (First) There are three main alternatives for what can be done instead. First, we can hold other people responsible instead, such as the vehicle manufacturers and the people responsible for the road system (including the communication and coordination systems used to guide the vehicles). The second option is to hold the artificial intelligence built into the vehicles responsible. The third is to treat traffic accidents in the same way as natural accidents such as tsunamis and strokes of lightning, for which no one is held responsible (First). Although the future is always difficult to predict, the first option is by far the most probable one. Previous experience shows that this is how responsibility is assigned when a human is replaced by an automatic system. For instance, if an aviation accident unfolds after the pilot turned on the autopilot, we do not blame the artificial intelligence that took over the flight, and neither do we treat the failure as a natural event. Instead, we will probably put blame on those who directed the construction, testing, installation, service, and updating of artificial intelligence. Such an approach is not unknown in road traffic. In the past few decades, proponents of the Vision Zero approach to traffic safety have had some success in achieving an analogous transfer of responsibility to vehicle and road system providers, although human drivers are still in place.

1 Computer Science Comps

Occidental's "Comprehensive Requirement" is a graduation requirement that serves two goals: "to provide an opportunity for senior students to synthesize the essential[s] [...] of their academic field", and "to provide an opportunity for students to demonstrate competence in their field" [2]. For computer science, the breadth of the field means that comps leans heavily towards the second goal: most CS comps are large coding projects that showcase what a student has learned in their four years at Oxy. In that sense, a CS comps project is more like a Studio Arts comps project: you could draw from multiple subfields or focus on just one, but the essence is to take on a complex project of your choice while situating it within the methods of CS.

Since comps is a significant undertaking, the CS major curriculum has two courses to support comps projects. Most students will take CS Junior Seminar in the spring of their

junior year, then take CS Senior Seminar in the fall of their senior year. One of the main goals of Junior Seminar is to prepare students for their comps project, including selecting a topic, exploring the literature, and writing a detailed proposal of what they will do. The entirety of Senior Seminar is dedicated to students completing their comps; however, it is necessary to separate the Senior Seminar course from the comps graduation requirement. Senior Seminar is a required course for the CS major, and it concludes at the end of the fall semester with a grade. Comps, on the other hand, is an Oxy graduation requirement, and concludes at the end of the senior year with either "fail", "pass", or "pass with distinction" [2]. While the sole purpose of Senior Seminar is to support students working on comps, it is possible for a student to complete the *course* without completing the *graduation requirement*. This could occur, for example, if the student ran out of time to finish the project, but submitted all the assignments for Senior Seminar. In that case, the student will continue working in the spring senior semester to complete their comps, until they have a "pass" or "pass with distinction" result. In short, the timeline for comps is:

- **Junior Spring Semester**: Junior Seminar, finishing with a comps proposal
- **Junior-Senior Summer**: Additional research/tutorials
- **Senior Fall Semester**: Senior Seminar, ideally finishing with a comps paper
- **Senior Spring Semester**: Buffer if comps was not completed

For many students, selecting a topics for comps is a major source of stress. While this is an important decision, it is also one for which you will receive a lot of support. Choosing a topic, together with ensuring that you have the relevant background and experience for the project, is a major component of Junior Seminar. There have also been students who change their project during Senior Seminar, and still finished their comps within the semester; although this is not recommended, this delayed timeline shows how much leeway there is for exploration and backpedaling.

As a rough guide, comps projects should be equivalent to the work of an upper-level CS elective, and should either dive deeply but narrowly into a subfield of CS, or reach broadly across subfields or across disciplines without being too shallow. The only constraint on the topic is that CS+Math and CS+X students must select a project that fits within their theme, and should check with their academic advisor to make sure this is the case. Although double majors could do both comps on the same topic, they must have aspects that are unique to each major. For example, a cognitive science and computer science double major could do both projects around biologically plausible neural networks, but the cognitive science project might be about comparing its activity to real brains, while the CS project might instead focus on its algorithmic performance. We will spend more time on this in class, but you might consider the following prompts in brainstorming your project:

- What topics were you most interested in from your classes? What area of CS would you like to dive deeper into?
- Alternately what topic were you unable to take a class in (either due to full enrollments, or because Oxy doesn't offer it)?
- What interdisciplinary topics or CS-adjacent topics would you like to explore?
- What projects might be a good showcase of your abilities to potential employers in your desired field?

A list of past comps projects can be found online.[1]

2 The CS Comps Final Paper

The two main deliverables for CS comps is the code from the project, submitted together with its documentation as a GitHub link, and a paper about the project. While the code component is straightforward and uncontroversial, some students may chafe against the idea of a paper.

Given these goals, the final paper should be written for an advanced CS undergraduate: someone who has finished Math Foundations, Data Structures, and Computer Organization, but may not have taken the appropriate upper-level courses. The paper does not need to explain everything, but should provide enough relevant details so the average reader can understand your results; a walkthrough of the sections of the paper can be found in Section 4. The paper will be written in LaTeX, to make all final papers uniform in style and to allow for psuedocode and mathematical equations; a template will be provided and required. To ensure there is sufficient context and detail, the paper should be about seven pages in length (excluding figures, tables, code, and references). More details about additional requirements of the paper will be provided in Junior and Seminar Seminar.

3 Interlude: Using LaTeX

LaTeX (pronounced *lah-teck* or *lay-teck*), often stylized as LaTeX and written as latex, is a document markup language and typesetting system. Building on the TeX language created by Donald Knuth in 1978, LaTeX provides additional commands for common document needs such as sections, figures, and bibliographies. LaTeX is widely used in academia, especially in mathematical fields, due to

[1] `https://www.oxy.edu/academics/areas-study/computer-science/past-senior-comps-projects`

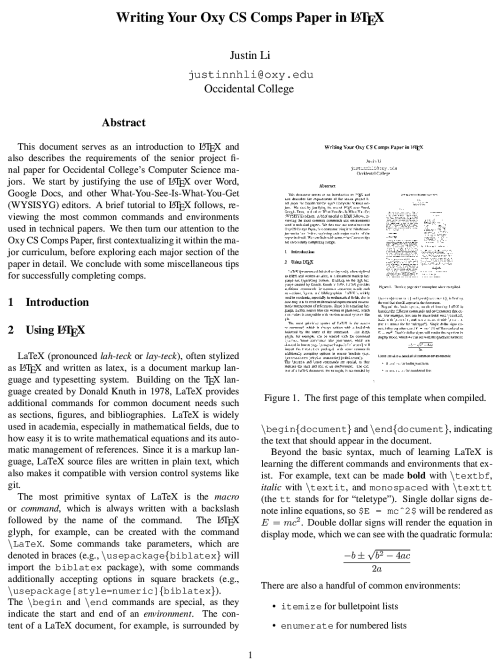

Figure 1. The current page of this paper, as an image.

how easy it is to write mathematical equations and its automatic management of references. Since it is a markup language, LaTeX source files are written in plain text, which also makes it compatible with version control systems like git.

The most primitive syntax of LaTeX is the *macro* or *command*, which is always written with a backslash followed by the name of the command. The LaTeX glyph, for example, can be created with the command `\LaTeX`. Some commands take parameters, which are denoted in braces (e.g., `\usepackage{biblatex}` will import the `biblatex` package), with some commands additionally accepting options in square brackets (e.g., `\usepackage[style=numeric]{biblatex}`).
The `\begin` and `\end` commands are special, as they indicate the start and end of an *environment*. The content of a LaTeX document, for example, is surrounded by `\begin{document}` and `\end{document}`, indicating the text that should appear in the document.

Beyond the basic syntax, much of learning LaTeX is learning the different commands and environments that exist. For example, text can be made **bold** with `\textbf`, *italic* with `\textit`, and `monospaced` with `\texttt` (the `tt` stands for for "teletype"). Single dollar signs denote inline equations, so `$E = mc^2$` will be rendered as $E = mc^2$. Double dollar signs will render the equation in display mode, which we can see with the quadratic formula:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

There are also a handful of common environments:

- `itemize` for bulletpoint lists
- `enumerate` for numbered lists
- `figure` for figures (see Figure 1)
- `tabular` for tables
- `lstlisting` for code listings (see Listing 1)

Listing 1. Hello, World! in Python

```
def hello():
    print('hello, world!')
```

This covers the most frequently used commands, but as you might have already inferred, LaTeX is a vast and deep system, and can easily be overwhelming. We recommended reading through the *Learn LaTeX in 30 minutes* tutorial [3] and using the *LaTeX WikiBook* [4] as reference. Beyond that, you should examine and play with the source of this document to gain working proficiency with LaTeX.

One final note on writing in LaTeX. Since LaTeX is only a markup language, the source files must be compiled into a viewable form, most commonly into a PDF file. The source of this document is made up of several files, each with its own purpose:

- `template.tex` - The main LaTeX file containing the contents of the document.
- `oxycomps.sty` - A style file with settings for what the document should look like.
- `references.bib` - A list of bibliography items.
- Other files containing images, build instructions, etc.

Manual compilation of these files is somewhat esoteric, as it requires multiple uses of the `pdflatex` and `biblatex` commands. Instead, it is much easier to use tools such as `pdflatex` or `latexmk` (in the terminal) or Overleaf (online) for compilation. We have also provided a `Makefile` which will automatically update the document as necessary; the use of makefiles is beyond the scope of this document, but see Lambert (2021) [1].

4 Sections of the Oxy CS Comps Paper

4.1 Introduction and Problem Context

This section should motivate why the project is interesting both to you and to the computer science community or the general public. You should also justify the difficult of the project. As a rough guideline, your project should be

either narrow but deep in a subfield of CS, or broadly reaching across subfields without being too shallow. It should be comparable to the amount of work/content in an upper-level elective.

4.2 Technical Background

This section introduces the technical knowledge necessary to understand your project, including any terminology and algorithms. You should assume that the reader is a CS undergraduate like yourself, but not necessarily familiar with AI/ML/HCI/apps/video games/etc.

4.3 Prior Work

This section describes of related and/or existing work. This could be scientific or scholarly, but may also be a survey of existing products/games. The goal of this section is to put your project in the context of what has already been done.

4.4 Methods

This section describes what exactly you will be working on. What are you building? How will it combine/incorporate ideas from the literature? Be specific about what you will be doing: talk about the specific algorithm you will implement/use, the specific dataset/platform/API, and what the outcome of your project will look like. All of these decisions should be justified as well.

4.5 Evaluation

4.5.1 Evaluation Metric

This section describes how you will evaluate your project. What will you be measuring, and how will you measure it? You might think about what would result in an F, a C, or an A for comps. Alternately, think about what are the minimal requirements for passing the class, what you might do if you had more time and resources, and what the best case scenario would be if everything went swimmingly.

4.5.2 Results and Discussion

4.6 Ethical Considerations

Are there any ethical concerns that might arise from your project? You might think about whether your project perpetuates societal inequity (or could be used by others to do so), whether the data/platforms you are using is collected with informed consent and free of bias, and whether you might be subject to technological solutionism instead of working support/better the public infrastructure. Include a discussion of how you plan to mitigate these issues in your project.

4.7 Limitations, Future Work, and Conclusion

4.8 Timeline

A timeline of major milestones, with specific items to be completed by specific dates/months. Note that this timeline must start over the summer; otherwise you will unlikely have enough time to complete a project of the expected scope. As part of the discussion around your timeline, talk about what you already know that would help you with the project, and what you expect to have to learn to be successful. Include programming languages, technical concepts, as well as processes (e.g., user testing).

4.9 Code Documentation

This section will demonstrate that you have thought through the basics of how your code will work. You should include a diagram of the overall data flow of your program, including what the inputs and outputs of each component will be, and how they will be represented.

4.10 Appendices

5 Tips and Advice

References

[1] Lambert, Chase. *Makefile Tutorial*. 2021. URL: https://makefiletutorial.com/.

[2] Occidental College. *Comprehensive Requirement*. 2021. URL: https://oxy.smartcatalogiq.com/2021-2022/Catalog/Academic-Information-and-Policies/Bachelor-of-Arts-Degree/Comprehensive-Requirement.

[3] Overleaf. *Learn LaTeX in 30 minutes*. 2021. URL: https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes.

[4] WikiBook. *LaTeX WikiBook*. 2022. URL: https://en.wikibooks.org/wiki/LaTeX.