Dennis McDaid

2.1.2015

Gallatin Rationale & Booklist

Primary Advisor: Dennis Shasha

**Background**

Throughout software’s recent history, developers have been juggling the question of how exactly the politics of programming fit into both the technical and social aspects of the engineering process. Specifically, the question of democratizing development comes into play, and whether or not software should be free to all for manipulation and redistribution. This phenomena is known as the Open Source movement. Spearheaded by figures like Richard Stallman (founder of GNU) and Tim Berners-Lee (founder of the World Wide Web), the free culture and open source movements exist to encourage a more fluid creative process, with hopes that more voices lead to greater progress. Once considered a sort “hippie” approach to software, recent years and new tools have helped to push this movement out of the niche and into mainstream software design.

**An Interdisciplinary Approach**

This colloquium aims to discuss why we design programs the way we do, approaches to development, and aspects of social theory which can be used to better navigate the field. By viewing the construction of software as a creative process in which strategy is just as valuable as execution, a higher level of understanding and potential is gained; that’s where the interdisciplinary approach comes into play. Anyone can code, but it takes a fine eye and consistent questioning of *why* to be a strong developer.

Plato and Aristotle’s classics inform the structure of programs in the same way a good algorithm does. *The Republic* introduces the theory of forms, and specifically goes into detail when discussing the “Ideal State.” The ideal is not something we can ever build ourselves, but rather it functions as a template for human invention to replicate. One of the first things taught in a Computer Science 101 class the concept of Object-Oriented Programming, which is closely based on this idea. Objects must all adhere to a certain set of requirements which are templated in the code.

When describing the ideal chair, for example, attributes take precedence over details. A chair is a chair if and only if it has some means to support itself and contains a seat which a given user can rest on. As long as it fits these conventions, the number of legs, color, or material it is made from are all variables. *How to Design Programs* follows this model when building an object-oriented programming foundation. The details are all fields of an object which can be changed, but in order for an object to be meaningful, it must match certain constraints to which other objects adhere.

In *Categories*, Aristotle breaks all things in the universe down to a few simple traits, including qualities, quantities, state of beings, and relationships. When discussing object-oriented programming, these are fundamental values which are necessary in order to give the pieces of a program value and unique functionality from object to object. With an ideal object established, we use Aristotle’s categorization to fill in the blanks of what separates this given version of an object from all others.

This object-oriented model is usually adhered to within the field, but like Plato and Aristotle, it isn’t the *right* way to design software, just *a* way. It’s not just limited to object-oriented vs. functional programming either, technical choices like algorithms or data structures, and aesthetic choices like user interface can be handled in any one of a dozen or more ways. Texts like *Introduction to Algorithms* and *Code Complete* inform readers of the various approaches they can take in software development, but it takes strategizing and teamwork to determine what the “best of all possible” method will be for a given problem. There is never a correct answer, only the one that best fits the current constraints, which is a commonly accepted mindset in both the fields of computer science and philosophy. Is the object-oriented model the best way to handle different entities within a program? While strict conformities help to categorize various objects, what advantages can be gained by making the process more fluid? These are the kind of questions that can be answered through democratizing software.

The opportunities I’ve had through both internships and research have forced me into positions where this kind of decision-making is required. For the last two years, I’ve worked at NYU’s Housing Office as a web developer, and if I strictly stuck to implementations of a hashmap or linked list I learned in a Data Structures course, our projects would be months behind schedule. These are tools, not tenets, and an inability to recognize patterns and social trends throughout the field has the potential to leave developers in the dust. I can speak from experience, having learned probably just as much from Stackoverflow (a website dedicated to programmers helping other programmers) as I have from a textbook. Sites like this don’t pay users for answering questions posed by others, and more often than not there will be three or four different propositions of what the best way is to handle a problem. Even if confined to a cubicle, the modern developer exists in an ecosystem of s Locke-ian community, where everyone pitches in with the only goal of trying to help the community and push new ideas forward.

This raises the question of “what’s the point?” It’s great that developers can feel a sense of identity, but where can we find real gains in redistribution and modification of our own work? When others invest time into changing my code, does that deprive me, as the creator of content, of creative authority? Both companies and individual developers are forced to ask themselves the question of how much they want users to have input and control over final products. Stallman discusses this process extensively in *Free Software, Free Society*, developing an idea of elastic software, which favors building programs which can be manipulated to suit any number of different tasks rather than just doing one extremely well. By opening up the user base and allowing external parties to participate in the development process, programs become more robust and useful to all.

In my development career, I’ve shipped products for NYU staff which were initially intended to simply function as a way for RAs to communicate expectations to residence, and through collaboration and giving users access to the resources we use, that application has vastly outgrown its original purpose and developed into a full-time interface for Hall Directors to communicate and manage RAs as well as residents. This change was substantial, but would have been impossible without democratizing the creative process of software design. Had my co-workers and I not opened up our existing code to stakeholders, the scope of this application would have been much smaller, and its creative potential would have been substantially limited.

But is the open-source model sustainable? Is it even worth it to build software when the end goal is free distribution and the only compensation is some kind of karmic belief that by contributing to the community, a developer truly does make it a better place? These are difficult questions to answer, however the growing trend in the market favors the open-source movement, as companies like Apple, Google, and Facebook have pushed the inner workings of their projects into the public domain. At the moment, it seems like the reward is worth the risk, as it causes developers to want to use these technologies and see how they can expand or warp existing software into new realms.

The only way to determine whether or not this is the best practice to utilize is through testing it. Throughout the research I’ve done in my senior year, open source practices have followed me everywhere. In this final semester, I continue my work on porting the program Sungear from Java to Javascript, a process which has proven quite difficult. While Sungear allows users to simulate gene manipulation and follows the best design standards taught in any good computer science textbook, the framework which it was built in has evolved over the last ten years, and the technology is now outdated. Even with open access to thousands of lines of code, it takes good design techniques and a knowledge of how to collaborate with others in order to get the ball rolling in a meaningful direction. The group I’m working with has chosen to keep the repository we’re using to work on this open and active, so anyone can view and utilize the code we’re writing and manipulate Sungear to work the way they want it to. Tests are already being developed for substituting different genomes with baseball teams and utilizing its functionality for categorizing sports statistics.

**Conclusion**

Gallatin offers an interdisciplinary experience on the path to gaining a degree, and I believe that with the way the computer science field has evolved, this kind of multi-faceted outlook is exactly what the industry needs if it is to continue accelerating at its current pace. The tenets on which good software design is built are fluid, and through acknowledging this we can open a dialogue for a richer development process. I aspire to continue working on projects like Sungear long after graduation, with a large portion of the near future spent dedicated to developing and contributing to various open source projects. Some of the greatest utilities in the modern developer’s arsenal are the ones which are open to all, and I want to be at the forefront of developing for my fellow community members. ~~Discussions of why we design the way we do and the potential of communal coding are the major springboards which will launch my colloquium into being a truly special experience.~~

**Booklist**

* **Ancient, Medieval and Renaissance Classics:**
  + Plato, *The Republic*
  + Aristotle, *Categories*
  + Aristotle, *Metaphysics*
  + Sun Tzu, *The Art of War*
  + da Vinci, *Leonardo’s Notebooks\**
  + Machiavelli, *The Prince*
  + Ibn Khaldun, *Muqaddimah\**
* **Modernity - The Humanities:**
  + Hobbes, *Leviathan*
  + Voltaire, *Candide*
  + Locke, *An Essay Concerning Human Understanding*
  + Locke, *Two Treatises of Government*
* **Modernity - The Social and Natural Sciences:**
  + Quine, *Methods of Logic\**
  + McConnell, *Code Complete*
  + Leiserson, *Cormen, Stein & Rivest, Introduction to Algorithms*
  + Felleisen, Findler, Flatt & Krishnamurthi, *How to Design Programs*
  + Chacon, *Pro Git*
* **Area of Concentration**
  + Williams, *Free as in Freedom*
  + Brooks, *The Mythical Man-Month*
  + Brooks, *The Design of Design*
  + Berners-Lee, *Weaving the Web*
  + Stallman, *Free Software, Free Society*