Comp 601 Personal Notebook

Jason Bell 3078931

April 2, 2017

Professor: Dr. Huntrods

Introduction

Hi all!

My name's Jason Bell, and I'm having a late start here. Crazy family issues for the last couple of weeks. Fortunately the first couple of units are pretty straight-forward, so I should be up to speed pretty fast.

I'm 38, and finished my BSC in computer science here at AU, in september. I'm located in Chilliwack, B.C., about an hour outside of Vancouver. I'm a former restaurant manager, and was driven to a career change by the 2008 recession. Trying to run a business, while costs of goods go up and business volume drops, does not make for the happy fun times.

For the last couple of years I've been a research assistant to Dr. Vive Kumar. Much of the work was collaboration with private companies and covered by NDAs. Broad strokes: most of what I do involves creating server back-ends, web services, and machine learning; particularly of classifiers.

For my undergraduate major project I implemented a custom programming language and compiler, that I dubbed "Pie" both as a nod to Python as an inspiration, and in a fit of whimsy. Because Pie is tastey. Like python, it is dynamic type and white spaced. It has a deliberately simple and concise syntax, and unit testing and test coverage analysis as a langage feature. It's syntax is more-or-less a hybrid of Python and C#. It runs on the .NET runtime because it has excellent support for runtime compilation, and because it is my preferred platform in general.

My graduate project will be a continuation of this project. Some of the important questions to be investigated are:

1) How to make a programming language novel? The graveyards are full of dead programming languages. I like Pie a lot, but it needs more to really stand out.

2) In the spirit of it already supporting unit testing, can the language do even more to support good programming practices? Can design patterns be a language feature? Is this even worth doing? I do know one thing: such features should be optional. I'm a fan of giving people tools, but not forcing them to use them.

3) What can the language do to support learners? Dr. Kumar and I have discussed the possibility of and IDE that tracks the key strokes of the user, and recognizes red flags, or large divergences from a set of solutions.

4) Both for fun and to demonstrate mastery, I'd like to implement a custom runtime. I've already written a rudimentary proof of concept of just-in-time compilation, and can use Boehm (https://www.hboehm.info/gc/) for garbage collection. This is the garbage collector that Mono used prior to the open sourcing of .NET. However, while this would all be really fun, I need to investigate whether it's worthwhile doing. Is doing this within or outside of the scope of a master's project? I've already had people say that what I did for my undergraduate project is

almost master's level, so I'm genuinely unsure. Further, the practical programmer in me sees that I'd be re-inventing the wheel. Why do that when I can just run on the .NET runtime? Can the compiler be designed in such a way that it supports multiple platforms? (in other words, I can create my custom version for fun and mastery, alongside a more robust one that runs on the .net runtime).
Regardless, I've read a bunch of programming language papers over the last few days, and will be commenting on and summarizing them. If anyone is interested, I encourage checking out:

Snyder, A. (1986, June). Encapsulation and inheritance in object-oriented programming languages. In *ACM Sigplan Notices* (Vol. 21, No. 11, pp. 38-45). ACM.
The paper is a bit old, but does a great job of covering object-oriented programming concepts. I'm not going to bother summarizing it as it is a review paper that doesn't introduce new concepts, but I encourage reading it. It does a solid job of clarifying terminology. For example, I often find that programmers get "encapsulation" and "data abstraction" confused. So in the spirit of COMP 601 being a survey course, it's a great survey of OOP principles.

So, How are Things?

Right, so I'm sorry I haven't been around to be part of the conversation. My brother went to prison, so that's been fun. And my mother is not doing well health-wise (not helped by my brother going to prison!). I've been dealing with the fallout from that and taking my mother to appointments for the last couple months. It has not been the fun times.

Thankfully I managed to get the first couple of assignments in, and got a pretty great grade on the first one. Hopefully I can manage to contribute as much as I can for the remaining weeks. I've been sitting on notes for these for several weeks and haven't had the mind-set to write them out. Though they were integral to my assignment submissions. My commentary for the units till now will be reflection upon the research I did for them, in the context of my assignment submissions.

If this were a more difficult course, I think I would have withdrawn. But, fortunately I have done research before, both at Athabasca University and University of British Columbia. So, the course content has not challenged me thus far, aside from my family issues. However, I feel bad for not contributing more to the group discussion, and will do as much as I can.

Anyways, I'm moving on Wednesday, so will be away from the roar. I'll be spamming the heck out of you all over the next week. Enjoy.

Unit 1 – Exploring Digital Resources

My search topic, using Google Scholar, was "future programming languages". I chose this topic because for my undergraduate major project I implemented a custom programming language and compiler: Pie. The biggest challenge that I am faced with it is ensuring that the language has features that are of interest, and result in it succeeding. Otherwise it will just be an intellectual exercise. While an intellectual exercise is fine, I'd like it to actually take off.

Article the first

Sammet, J. E. (1972). Programming languages: History and future. *Communications of the ACM*, *15*(7), 601-610.
This article was highly inspirational, in terms of pointing a new avenue to answering the above question. I had been hung up on this question for some time. In fact, my supervisor, Dr. Kumar,

has been encouraging me to write a journal paper about Pie for some time. However, I have not felt ready to do so. This course, and courses to follow, have done a great job of helping me in that regard.

It's important to note that this article was published in 1972: the "future" it speaks of is today's past. Even it's view of the past is different from now. For example, it discusses Fortran, COBOL, and ALGOL extensively, but does not mention C at all: C was released in 1972 and presumably hadn't gained wide attention, assuming it had even been released yet when this was published.

Regardless, the important lesson I gained from this paper is that when trying to discover what makes a language popular, one should look to the past as well as the future. It sounds like common-sense in hind-sight, but I guess that's why hind-sight is 20-20.

Thus, my first assignment was a review of the history of successful programming languages, and what made them popular. I considered more recent languages as well as the "first generation" ones, because it's no longer 1972. One conclusion I reached, that the paper did not, was that languages can be successful by taking a step backwards. C was invented as a backlash against the commonness of verbose, English based, high level languages. Systems programmers wanted a language that was more concise, allowed direct access to hardware, and mapped more closely to machine code.

I should note that the while this paper inspired the direction for my first assignment, I consider some of it's conclusions to be wrong. For example, it suggests that future languages (future in 1972) would be made of natural languages: writing code in English. They couldn't have been more wrong, with C, C++, and their C-like descendants coming around the corner. In the same vein, they also suggest that languages would stop being based upon procedures. Very wrong. Finally, they suggested the advent of user-defined languages. While we see some of that today, in the form of the Groovy language, that was very optimistic for the time.

One point where the paper was spot on: they suggested that "improvements in the computing environment" will contribute to how languages are created. This is very true. I suspect that programmers in 1972 couldn't have fathomed the kind of tools and IDEs that we're spoiled by.

Article the Second

Ray, B., Posnett, D., Filkov, V., & Devanbu, P. (2014, November). A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 155-165). ACM.

I selected this article because I had seen graphs around the interwebs, showing programming language usage levels. For example, this one shows language usage graphed by number of mentions on Stack Overflow, compared to number in GitHub repositories: http://redmonk.com/sogrady/2016/02/19/language-rankings-1-16/
Although I am skeptical about the value of Stack Overflow data, it's still an interesting graph.

Therefore, it was interesting to see a scientific article written with a similar goal, though it also tries to find correlations between code quality and programming language traits (static versus dynamic, functional versus procedural, etc). This paper does not use Stack Overflow data, only GitHub. Fortunately, GitHub still provides a huge source of data, both in terms of amount code in repositories, but also commit history and team data. Naturally, in the context of Pie, I'm not just interested in popularity, but what contributes to good coding habits as well.

The conclusions are interesting. One is common sense: that using a memory manager results in higher quality code. Clearly, through avoidance of memory leaks and fragmentation.

It found that functional code produces higher quality code than procedural code. I hypothesize that this is because truly functional languages allow the output of a function to only be determined by the inputs: whereas in procedural and object-oriented languages, state elsewhere in your code can affect your results. I could be wrong, but I found this extremely interesting because one of the conclusions in my second assignment was that the current rise of functional languages (Scala, F#) will continue. I have been thinking a lot about how to incorporate it into Pie… though I've been hindered by the fact that I'm so used to object-oriented code, that I have a hard time wrapping my head around some of the more arcane ideas in functional programming.

The last conclusion of the paper, that was real food for thought for me, was that static type languages result in higher quality code than do dynamic type languages: interesting to me because the most recent version of Pie is dynamic type. I may need to seriously research this more, and figure out why this is an issue for dynamic type languages. Further, is it bad enough to matter? Does the more concise syntax and flexible capabilities of dynamic type languages outweigh its downsides? My preferred language, C#, is static type, but supports dynamic types if required for interoperating with other languages like IronPython. What about doing the opposite with Pie? A dynamic type language that allows you to declare the type of something if necessary?

Food for Though

I wouldn't say that I go out of my way to visit any particular site for programming information. I usually Google my way to a solution, no matter where it is. Sometimes it's a blog post, sometimes it's in a forum for the technology I'm using, sometimes it's the MSDN, but often it's Stack Overflow. While I don't go out of my way to visit Stack Overflow, it does seem to be a good forum to ask questions. It presents an opportunity for people to ask questions and provide answers, but more importantly: it has a voting system that (hopefully) results in the best answer usually rising to the top. It is also used enough that there is a large repository of previously asked question. I have never had to actually post one: usually if I have a question, it's already been asked and answered.

Clearly the best place to find answers to my questions is the source: if I'm working with a .NET technology, I'll probably go to the MSDN as an authoritative source. Failing that, I approve of Stack Overflow's voting system as a means of filtering poor answers. I've had to fall back on blog posts, but often find they have errors, are outdated, or don't discuss something as completely as I'd like. For example, while I was researching the basics of creating a just-in-time compiler, I found a great post on how to get started… but that was it. It showed how to created an executable block of memory, write machine code to it, and execute it. Sounds good, but when one tries to push onto the stack it fails, as whatever requirements are necessary have not been met: it results in an access violation. I assume that there are extra steps require to setup a stack, but the blog did not show how, and Google was not helpful in answering this question. The lesson: ultimately there's no replacement for digging deeper and finding an expert to answer your questions.

Unit 2 – Internet Searching and Tools

First, let me say that WayBack Machine was a real blast from the past. Speaking of someone who was born in 1978, looking at yahoo from the year I graduated from high school was fun.

In researching "other technologies", I opted to do some sleuthing on TOR: The Onion Router. TOR anonymizes internet traffic by routing it through a large number of nodes, or layers: the layers of an onion, if you will. The result is that the traffic is very difficult, though not impossible, to trace. TOR also provides access to "hidden services" through "onions": websites that are accessible through TOR, but not from the world-wide web.

What are the benefits of this service? It allows for legitimate activities that may be illegal. For example, whistle-blowers can use it to leak information with less danger of being identified as the source. People who live in countries where it is dangerous to criticize governments can use it as a forum to do so. It can be used to bypass nation-wide firewalls, such as the one employed by China. For individuals who have concerns, whether legitimate or not, about data collection by corporations and governments, it provides a way for them to use the internet without this fear.

The negatives of the service are that it naturally provides a place for criminal activities such as drug dealing, weapon sales, and distribution of child pornography. As demonstrated by the Silk Road: a black market that has been shut down by the FBI twice, and is currently back up again.

TOR appears to work well at anonymizing world-wide web traffic. While it's slower due the traffic being routed through many nodes, "what is my address" sites are unable to see your real address. On the other hand, my experience visiting hidden services consisted of following broken links, for the most part. This even though there are search engines for said services. I suspect that those services change address frequently, and search engines and hidden wikis can't keep up. I also suspect that keeping up with those hidden services requires more effort than I'm interested in investing, especially given the danger of stumbling across something illicit. Even the legitimate hidden services seem to be of poor quality.

This leads me to the question of how secure it is. The impression I get is that it's "secure", but not "perfectly secure". Because it's impossible to be 100% secure. The traffic could still be followed through the many nodes, but I suspect that this isn't worth the effort of doing for 99% of traffic. Government security experts probably put their effort into the child pornography distributors and drug dealers, rather than following the traffic of a conspiracy theorist, or someone criticizing the president of Turkey.

I have a friend who works for an internet security and data mining company in Atlanta, Georgia. His opinion is basically just that: it's "secure enough" for general usage, but he wouldn't assume safety for illegal activities, even ones like whistle-blowing. This friend of mine basically said: "I don't know for a fact that the NSA can trace TOR traffic, but yeah… I would assume that they can. But unless you're doing something really naughty, they don't care about you."

Food for Thought

I can't say that I rely on the internet for information more than I did 10 years ago. I'm 39, so I was already pushing 30 10 years ago, and access to the internet was already ubiquitous. Instead I'll go back further. When I graduated high school in 1996, the internet was still a new and novel thing, at least for the general public. Every company didn't have a website, and search engines were in their infancy: they were more like collections of links than search engines. The internet was just that small. So, not the amazing source of information that it is today.

In university dorms, internet wasn't available yet. When I attended UBC, you still had to go to the library and use a computer there to access it. When doing research, you had to actually go to libraries and pore through physical books (gasp). No speedy googling of papers, because Google didn't exist yet. I have mixed feelings about this last point. On the one hand, I love that I can find papers on any subject, right at the tips of my fingers. On the other hand, it takes a way some of the work that I think was useful experience. The experience shouldn't be just finding the

paper, it's also going to a place, experiencing lots of interesting literature (not all of which is online), and meeting people.

I use Google extensively: it meets my needs just fine. I'm sure that Google does all kinds of highly advanced tracking of my behaviors, but I don't really care. I don't do anything illegal, and if Google wants to track me looking at cat gifs on Reddit, whatever. So that would be it's downside: usage tracking, but it doesn't affect me personally. Finally, Google Scholar is amazing.

Bing seems like an alright search engine in it's own right: it seems to give search results similar to those of Google. I still use Google though, even though I'm an avid user of Microsoft products in general. It's just habit, and Google works fine. It does annoy me some that programs like Skype assume that I want to search with Bing rather than Google. I must say though, that I took a look at Microsoft Academic: Bing's equivalent to Google Scholar, and the first impression was good. I like that it lists fields of study to the side, and presents you with pertinent articles. But, that's just a convenience. Next time I need to research something I may try it out.

I'm surprised that Yahoo and Altavista are even still around.

I'd like to have more to say about how the engines impact me personally, but they don't, really.

What would affect me personally, and would be a deal breaker, would be if it was proven that an engine is biasing the search results. Or even censoring legitimate sites. Censoring illegal things such as child pornography is perfectly fine, but if it's ever shown to cross the line where political viewpoints and the like are censored or made harder to find? That'd be an issue for me.

In the sense of being 100% secure, I don't think there's such a thing as a secure engine: simply because 100% security is impossible. There are engines that don't track and keep user information, such as DuckDuckGo. But that still on the open world-wide web, so I have little doubt that there are many ways for third parties to track user activity. DuckDuckGo does allow a TOR routing mode, which likely provides as much security and anonymity as the general public is likely to get.

So aside from anonymizing search engines, or using an engine through TOR, I don't see any engine as secure. Even then they're just mostly secure.

As for efficacy, engines like Google and Bing work exactly the same through TOR as they do without. Just slower. I don't have much experience with DuckDuckGo, but I note that it speaks of returning links that are "best" rather than most visited. What is "best", and who decides? I feel like that's something that could lead to the kind of bias and censoring I mentioned before.

Unit 3 – Information Source Assessment Techniques

When searching with the terms "future programming languages" I used Google Scholar. For this exercise, I repeated it with regular Google. I would deem most of the results to be poor for academic research. In fact, most of the results for this search are BuzzFeed-style clickbait. "12 up-and-coming programming languages developers should get to grips with" reads no differently to me than "10 incredibly cute kittens: #3 will leave you in tears!".

These sorts of sites also contribute to religious wars about programming languages: which is "best" (pro-tip, there is no "best" language). The sites are relatively content free while presumably convincing the naïve that their list of languages is the best list. I would call these low quality.

As I talk about in food for thought, the Wikipedia links that come up in this search are at least reviewed by other users. I would call these medium quality. They provide mostly accurate information, and a stepping stone to more.

The high quality sources would be from peer reviewed journals, which don't even come up in the google search. So, off to google scholar with you.

Food for Thought

I feel that the simple description of the positives and negatives of Wikipedia, outlined in the unit text, is apt. I don't really have anything to add to it: it's great for having largely correct information due to peer review, while occasionally falls on it's face because the reviewer's credentials aren't known. As I've seen it put: Wikipedia is generally reliable, but occasionally someone will edit it to have false information. However, it's usually a short time before another user corrects it.

The lack of credentials would be the primary reason it's not to be used as a primary source: using a reference from it isn't any better than citing "some guy" you met on the street. Contrast with journal articles, that go through a peer review process by people who are experts in the field.

I think Wikipedia could correct this by designating users as "verified". A user has shown proof that he holds a degree in an area, and edits by him or her are flagged accordingly. Perhaps those edits can only be changed by another verified user. I see some challenges to this, however. First, as evidenced by the regular fund raising messages found on Wikipedia, they have limited resources. I imagine that the cost and time of verifying users properly is outside of their means. Second, experts are not immune to bias. I picture a topic that has the interest of many lay people, but only one verified user, who holds a monopoly over the page's content. So, frustrating politics will ensue.

This being said, Wikipedia tends to be my first port of call when learning something new. I wouldn't completely rely upon it, but it at least gives a solid introduction to and overview of the topic. Especially when it comes to programming topics, it tends to be quite accurate. For example, I've yet to get an incorrect algorithm from it. Wikipedia also provides an avenue to other sources, since it's references are often to valid primary sources like journal articles.

I think Wikipedia is fine, and doesn't need to change. It's a mostly reliable source of information for general use, and a solid introduction for more advanced subjects. The ability for anyone to add incorrect information is a problem, but one that appears to self correct quite quickly. This should continue as long as Wikipedia has a large and diverse set of users. For an example of the biased mess that it can become with a small number of users, see Conservapedia.

I would say that in the context of academia, there is nothing like Wikipedia that is acceptable as a primary source, unless you count peer reviewed journals. But nothing as convenient as Wikipedia. You still have to put some effort into finding your sources.

Unit 4 – Managing and Presenting Research

The first challenge I'll suggest for the implementation of Edmonton's transportation plan is that, frankly, it seems huge and ambitious in scope. Therefore, also slow and difficult to implement. Simply adjusting different forms of transportation to coordinate with each other would be a huge

task. How do you do so without inconveniencing current users? How do you do so, while moving as few train and LRT stops as possible, to minimize cost? Are there politics involved? Are people used to things being a "certain way" for their form of mass transit going to go along with a plan easily? What about the time and cost of training people for new routes? Or keeping the public informed of route changes? That's a lot of plates to keep spinning.

Another challenge that makes some aspects of the plan ambitious: Edmonton is in Alberta. I don't say that facetiously, as I've lived in Alberta. How well is encouraging people to walk and ride bikes in a province where the weather wants to murder you for half the year? How are you going to keep the roads well maintained when it literally gets cold enough to crack the streets? I suspect that the answer to the first question is "you won't": people are going to get out, or not. The answer to the second is "as well as we can": it's up to the project managers to decide what is "good enough" road maintenance, and how to achieve it.

So, that's a lot of challenges. And the project managers get the fun task of figuring out how to juggle them all to a solution, in a cost effective and timely manner. I'm skeptical that this is a project that would be possible to be on time and in budget: the goal would be to get it as close as possible.

Food for Thought

I'm not a fan of learning objects as the principle tool of education. They have to be maintained, or wind up ridden with broken links. When used as the only teaching tool, they are frankly lazy. If I'm paying $900 for a course, I want to receive a great textbook and accompanying well planned study guide… and preferably a physical textbook, not an etext. When I took the Intro to Java course, I was presented with a set of links to Wikipedia articles and Oracle documentation. All reasonable ways to learn, but I paid for more than that. I'm perfectly capable of finding that information myself, without paying $900.

Learning objects are a great way to supplement education, especially if they're done in a modular and reusable way. But you still need the traditional teaching methods: a book, a great study guide, and someone you can ask questions of.

Unit 5 - Modeling and Simulation

Unit 5 – Modeling and Simulations

In my own experience with machine learning, simulations can simplify a problem. Often the data that you have is so convoluted, large, and seemingly monolithic, that it's impractical for a human to find patterns in it through observation. Furthermore, humans have preconceptions and biases that can affect how they interpret data. If one can create a model that reasonably approximates the data, things get easier. The model may not 100% accurately represent the data, but it's still useful as long as it's "good enough": what metric is used to determine "good enough" being subjective. Simply, a reasonable model and simulation can make an unsolvable problem solvable.

The reference material for simulations, provided by the course, is somewhat dated (1996). I'm not sure that I agree with everything that it says. For example: "manual simulations allow you to observe carefully how a simulation is constructed". I'm not sure how this distinguishes from modern automated simulation tools, such as machine learning libraries. I can, however, see this being the case in 1996. Speaking of it's "certain tactile quality that can't be equaled" also seems

jedi hand-wavey and vague, but again, I can see it being useful for education in 1996, when people hadn't grown up surrounded by computers, tablets, smart phones, etc.

I also disagree with it's assertion that low cost is an advantage of manual simulations. This may be the case… as long as you're performing a trivial simulation that only requires a dice, pencil, and paper. Again, I can see this being plausible in 1996 when computers would have been incapable of more advanced simulations, but it's no longer the case today.

On place where I would expect a manual simulation to perform better is when dealing with physical materials. Given a new airliner made of some new material, I would much rather be a passenger if those materials had been tested under real conditions, rather than just in a computer. When there are lives at stake, there's not substitute for manual testing.

Another advantage of manual simulation is that it eliminates artifacts created by a computer's inability to generate truly random numbers. A computer can at best generate "pseudo-random" numbers: numbers that a random based on some seed. If you know the seed, and the algorithm, you can predict the numbers. For example, two simulations that both use the Unix epoch (January 1, 1970) as the seed may have the same result.

Clearly, the primary advantage of computer simulations is performance. While the course material (1996) refers to hundreds or thousands computer dice rolls in the time of a manual one, today that's more like millions. A computer simulation can process volumes of data that are simply impossible to process manually.

A second advantage of computer simulations is automation. Once the simulation is setup, it can be run automatically on any volume of data, and run again with another set of data. Start the simulation, knock off for a cup of coffee, see the results when you get back.

Another advantage of computer simulations is that you can also automate its validation. Once the simulation has finished doing it's thing, it can automatically compare it's results to real data. This step could be just as intensive as running the simulation was, so it's useful to be automated.

I would say that the final advantage to computer simulations is low cost. Although, again, the course material talks about manual simulations being "free", that's only the case for trivial dice roll simulations. I think, for example, of claims by 9/11 conspiracy theorists that the buildings could not have fallen, as they were designed to withstand the impact of airliners. Except, that's not something that would have been testable through manual simulation (or even computer simulation, considering that they were designed in the 1960s). This is a manual simulation that did not happen simply because of impractical cost.

The connection that Fishwich makes between simulation and children is that children learn through modeling. They role play, and build models of things (speaking from experience: Lego is awesome). While I am not an educator, nor do I wish to be one, this seems to be a reasonable statement.

According to Fishwich, the steps of computer simulation of a physical thing are: 1) build a mathematical model and 2) execute the model on a computer.

Closed-form simulations require the assumption that the thing being simulated can be represented with linear equations: this may not be the case. A system of linear equations may also become too complex if it tries to take into account all of the potential variables in the thing being simulated. Because of these limitations, the simulation may not accurately reflect the thing being simulated: the thing has been "contorted" to conform to the limitations of closed-form systems.

A petri net describes a distributed system as a network of places, transitions, and arcs. Arcs can only run from a place to a transition or from a transition to a place: not from a place to place. The transition describes the conditions necessary for the movement between two places to occur. Unless there is a policy in place, transitions are nondeterministic: if multiple transitions are possible, any one of them may fire.

The advantage of petri nets in the context of computer simulations is that that they can be precisely described in mathematical terms, and therefore the model can be implemented in code.

Unit 6 - Statistics and Probability

Chapter 3, section 1 questions from "Introduction to Probability":

1) Because we have a set, and order matters, we want the number of possible permutations. Because we have four people, there is no repetition:

P(n,k) = n!/(n-k)!

Where n is the number of people, and k is the number of seats.

P(n,k) = 4!/(4-4)! = 24

2) We have four ways to select exterior color, and three ways to select interior color. We want one of each, so by the rule of product, the number of combinations is 4 x 3 = 12.

3) A word being 32-bits by default shows that this book is a bit dated. Regardless, we're selecting from a set of 2 items, 32 times, with repetition allowed. 2^32 = 4,294,967,296. Not coincidentally, an unsigned 32 bit integer can old a value between 0 and 4,294,967,295.
10) In the absence of more information, the probability of the 13th card being an ace is the probability that any card will be an ace: 4/52.

Chapter 6, section 2:

2) Expected value is the weighted mean of the values in the distribution: E(x) = (0 x 1/3) + (1 x 1/3) + (2 x 1/6) + (4 x 1/6) = 1.33

The variance is calculated similarly, except with the squares of the values in the distribution. Then subtract the square of the expected value:

Variance: Var(x) = (0 x 1/3) + (1 x 1/3) + (2^2 * 1/6) + (4^2 * 1/6) – 1.33^2 = 1.89

Standard deviation is the square root of the variance: Sigma = 1.37

I don't have much to say about this unit frankly. Nice little review of probabilities.

Unit 7- Ethics

Selection of Data:

1) These students should include their data points, assuming that they go ahead with the report. Given that they decided that an unsteady power source generated incorrect results, they should not do so: they should redo the measurements after taking corrective action. As it is, they have good reason to suspect that their data may (or may not be) incorrect. If, after repeating the measurements with a steady power supply, they still get the anomalous data points, then they have an interesting topic to discuss. Why do their measurements not match the theoretical values, while the other group's measurements do?

2) If they go ahead with a report with the anomalous data, they should be open about them and provide possible explanations. If they do the report with the original data, explain that unsteady power may have affected the results (though again, I think they should redo the measurements). If they repeat the experiment and still get the anomalous results, look for other possible measurement errors. If none are found, propose other explanations, using what is known about the properties of the material. Or, present the paper as a request for another group to repeat their measurements to see if the results are duplicated again.

3) Fix the issue that they suspect is causing the points: unsteady power supply. If they still get those results despite fixing the problem, they have an interesting topic to discuss.

What is plagiarism:

1) This is flagrant plagiarism. They're pasting other people's text into their report and claiming it as their own.

2) This is not plagiarism. Well known data points like these are, well… well known. They were established long ago, and have the same values no matter where you get them from. There is no need to assign credit for them. It would be different if they were working with a material with properties that are still poorly understood, and used measurements made by a single research group. Ie, measurements that aren't well known.

3) This is plagiarism: although the supervisor rewrote large sections of the thesis, the student is still claiming it as his or her own words.

4) Unless the author is explicitly quoting the other authors, with credit given, this is plagiarism. The author should express the ideas in his or her own words, while giving original credit for those ideas.

5) On the surface this seems a bit more grey, but I interpret this as the author writing a discussion of, or critique of, the book. If that's the case, I would think it's acceptable as long as he or she obtained permission from the author to use the figures and text. If not, I would call it plagiarism. It's highly subjective though. How many figures, and how much text? I would consider use of any figures without the permission of the author to be plagiarism. And is the "block of text" a few sentences in quotes? Or is it entire chapters? The former would not be plagiarism, while the latter would be.

A case for plagiarism:

1) No, the way he used the information was not normal or acceptable. He should have made them explicit quotes, or reworded them, while still giving credit.

2) I could possibly see forgiving the student if the supervisor didn't discover this before it was submitted: the mistake should have been discovered right away. If the supervisor either didn't see the error, or didn't review the proposal before it was submitted, I can see room for forgiveness. That being said, I'm not sympathetic towards someone who still doesn't understand plagiarism by the time they're in graduate school.

3) I think the action of the head of the school was correct. Supervision aside, if someone doesn't understand how to avoid plagiarism by graduate school, perhaps they shouldn't be in graduate school.

Credit for the discovery of pulsars:

1) Bell should have shared the Nobel Prize. First, she wasn't a graduate student: she was post-graduate. Second, although she "accidently" discovered the signal, I have to assume that there was still a lot of skill and expertise required to recognize it. Third, she was the second author listed for the paper on the discovery, of five authors. Clearly she must have had a greater role than just being the one who got lucky.

2) Bell may have only been a research assistant, but for the reasons I describe above, she should have shared in the prize.

Industrial sponsorship of academic research:

1) I see no harm in simply asking the company to let the student publish. But if the company says no, a contract is a contract.

Sharing research materials:

1) The other laboratory has good reason to not want to share the chemical yet. Perhaps there are safety concerns that they have until they understand it's properties better. Perhaps they don't want their chemical to be stigmatized because their so-far poorly understood chemical caused problems for someone else's experiment. Wanting to wait until the chemical's properties are better understood seems reasonable to me.

Fabrication of a grant application:

1) No, the student should be expelled. He made a factually false claim in an application for funding. It doesn't get more clear-cut than that.

2) As above. Continuing in the program with funding despite the paper he claimed was submitted for publication has not even been completed? No. Also pretty clear-cut.

3) Other universities that the student applies to have a right to know about this incident, as it may be repeated. I can't think of any ethical reason for this incident to be confidential. The student screwed up, period.

Reviewing a journal paper:

1) If the reviewer is confused because they are not familiar with the subject, they should have said as much or declined to do the review, rather than simply rejecting the paper. If the paper is confusing because it is poorly written, then sure, the reviewer was right to reject it. If the fact in question is one that has support in a distinguished and reputable journal, then the reviewer should not have rejected it for not conforming to the orthodoxy found in textbooks. At the same time, I'm not sure if one journal article is enough to justify using an unorthodox claim.

2) The editor's actions were not correct: he should have sent the paper to multiple reviewers, and ensured that they were qualified to competently do so.

3) The author's arguments seem reasonable, and should result in a proper review of the paper by multiple qualified reviewers.

Canals on Mars:

This clearly warns of the fallibility of human senses and preconceptions: seeing what we want to see. It's interesting to note that this phenomenon occurs to this day with Mars, in the form of pareidolia. People want to see faces, pyramids, and beings, so convince themselves that they can discern them in complex photos of terrain.

Polywater:

This case shows the importance of replicating experiments after publication. After other groups tried to duplicate the results and failed, it became apparent that the experiment was faulty.

Cold Fusion:

As above. Although it seems like scientists were skeptical, they were open minded enough to at least try to replicate their results. They all failed to do so.

Blind Referring

Devroye seems fraught with emotion in this discussion. I'd have taken it more seriously if it was written with less emotion and casual tones.

Many of his concerns seem unfounded, even by his own admission. For example, he cites an incident that only happened once in his career. He says referees should be concerned about things like papers being published in multiple journals, or similar papers in the same journal. But is this not the responsibility of the editor, not the referee? The referee just assesses quality of the paper. I would imagine it's the editor who makes the final call, after considering these matters.

His discussion doesn't mention the benefits of blind refereeing: the reduction in bias. I feel that this outweighs other considerations. He worries so much about not being recognized, but will he not be recognized if he passes the refereeing process and is published? What's the value of being recognized by referees if you're not published?

Devroye thinks that referees should be paid. Clearly, they should not be. If paid, especially by a commercial publication, authors are going to be more inclined to give positive reviews of papers than others.

Good referees should be flagged? Who decides what makes a referee good?

Wanting to receive a URL so you don't have to remember a password is just lazy, as is allowing referees to review in any format they wish to.

What's wrong with sending unpaid referees thank you letters? They know what they're getting into, don't expect payment, and I'm sure that they're capable of appreciating the sentiment.

So, I disagree with everything he says: it all comes across as very whiny, petulant, and arrogant.

Ethics Application

As expected, receiving an ethics approval for human research is a rigorous process. I have not had to do so (yet), but work with people who have, so had heard from them how exhaustive it is. All of the concepts from the undergraduate ethics course appear to be covered by it. Informed consent, including being informed of risks, right of refusal to participate at any time, confidentiality, parental consent, and so on. All safe-guards that came into being in the aftermath of the Nuremburg trials.

Unit 8 - Computing Hardware

Unit 8

History of Computing Hardware

For the purposes of this discussion, I'll start with analog computers: there were devices prior to that, which could be considered "computers", but one must draw the line somewhere. At which point is something like a slide-rule a computer, or a simple manual tool?

Unsurprisingly, the first analog computers were special-purpose. This makes sense, given that it was revolutionary technology that would need to fill an important need to receive funding for development. And, of course, to primitive to program. Examples of special purpose analog computers are:

- 1872: Lord Kelvin's tide prediction machine: prediction of tides of course being very important for navigating in and out of harbors.

- 1876: James Thompson's differential equation solver. This was made up of a set of wheels, discs, and drive shafts, that performed numerical calculation.

- World War I: the years in and around world war one saw the advent of analog devices to calculate trajectories for artillery shells and bombs, as the ranges over which these were fired had become impractical to calculate by hand.

Digital computers appeared just before and during world war 2. Again unsurprisingly, their rapid development was driven by the needs of war. They were still used for trajectory calculation: for example, for V1 and V2 rockets. In addition, and perhaps more importantly for determining the outcome of the war, they were used for cryptography. Throughout the war there was a continual arms race, as each side attempted to break the other's codes, leading to increasingly sophisticated computers.

The first digital computers were programmable, but still largely mechanical: programming consisted of adjusting mechanical relays. Because they involved moving mechanical part, they were also inherently slow. This changed with the advent of electric computers, with vacuum tubes. This happened incrementally: at first the calculations were electrical, while memory was still mechanical. Later they became fully electric. The first of these was the American Colossus, in 1943. The superiority of these programmable electronic computers gave the allies a distinct advantage against the electromechanical computers of Germany.

Although computers had by this time become programmable, the process was a laborious one. Mechanical computers required complex reconfiguration of hardware, and even electrical ones required complex redesign of punch-card programs.

Stored program computers narrowed the distinction between data and instructions, by defining a standard instruction set that could be stored in memory. The Manchester Small-Scale

Experimental Machine was the first to introduce this functionality in 1948, and this is the method still used by computers today.

From 1955 transistors began to replace vacuum tubes, which were less efficient, larger, and required constant replacement. This was an important step in miniaturization of computer hardware, making their commercial use more viable, as well as enabling the development of supercomputers. Previously, a vacuum tube based supercomputer would have been impractically large and required a legion of workers constantly replacing the vacuum tubes as they burn out.

This was closely followed by the development of integrated circuits: a concept possible with transistors, but not vacuum tubes. Integrated circuits integrate many transistors into a single chip, allowing for further miniaturization and fewer wires.

Personal Reflection

This trick question isn't much of a trick question for me, because I'm old enough to predate embedded devices, gaming consoles, and the like. I recall using the VIC-20 at a young age, and playing video games that ran off tape cassettes. One of these was the first with spoken audio: Ghostbusters ("He slimed me!"). At the time, this was unbelievably cool: something that I don't think kids today would understand.

I purchased my own computer around 1997: I think it was a super amazing 386. This was a 32-bit computer using the x86 instruction set, still used by 32-bit programs today. I can't recall its specifications, but it's safe to assume that it had far less memory, hard drive space, and processing power than either my phone or tablet.

For much of my life, I did not use computers professionally. I worked in the restaurant industry, and computers were for video games and casual hobby programming. I had never considered becoming a professional programmer until the great recession forced me to consider a career change: running a restaurant in the face of rising costs and declining business did not make for the fun times.

I feel that I should say that even now I see computers as tools. I feel that some people are far too into them: needing to always have the best computer, being elitist about capabilities, and so on. I was speaking to a cab driver once about this (random, I know). He was mentioning a relative who was like that, and who was very excited about the idea of computers having interactive AIs. He was surprised when I said I had no interest in that. I'm no more interested in having a personal relationship with a computer, than I am with a hammer or screwdriver.

Future of Computers

It's widely acknowledged that we've just about reached the limits of what a single CPU can do: we've reached the point where a processor is limited by the laws of physics. I've seen this myself: I'm old enough to remember how in the 90s, computers really did become obsolete within a year. There was a constant exponential increase in power. That's no longer the case. Whereas previously, I would find myself realizing I needed a new computer on a regular basis, my current laptop is a few years old, and is not much slower than brand new ones. New computers are gaining more CPUs, more memory, bigger hard drives, but aren't really getting faster.

The future of computing is parallelization and quantum computing. The former is already in full swing, while the latter is still a way in the future.

Parallelization entails software that is capable of running on multiple CPUs: important for supercomputers, and home computers that have more CPUs. Going forward, it's likely to be more important for software, from business tools to video games, to be capable of parallelization.

Quantum computing is in it's infancy, and to be honest, it's concepts hurt my brain. My understanding is that rather than bits, which must be a single state, it employs qubits, which because of quantum mechanical concepts can be a number of states simultaneously. The end result of a quantum algorithm is probabilistic: the quantum states are condensed into classical binary states, with a probability of correctness.

Based on my admittedly limited understanding, I suspect that the applications of quantum computing will be limited: for numerical and scientific computing of solutions for NP-complete problems, particularly cryptography. I'm skeptical of its application for general purpose uses, but thanks to my limited understanding, I have no evidence to back up that skepticism.

Hardware Change in the Last Decade

This is a difficult question to answer: hardware hasn't really changed that much in the last decade, in terms of revolutionary changes. Hard drives are pretty much the same as ever: even solid state drives have been commercially available for the last decade. Memory is pretty much the same, just bigger and a bit faster. Input devices have not changed at all. GPUs have not changed that much in how they operate. Graphics rendering benefits greatly from parallelization (pixels can be rendered independently), but their general mode of operation hasn't changed, just the degree of parallelization as technology miniaturizes.

I would suggest then, that the most significant change in computing technology is the shift from 32-bit to 64-bit. 64-bit allows programs to use more memory, thanks to the 64-bit word size for pointers. 32-bit computers had a "3 GB limit", where depending on the architecture, programs could use no more than approximately 3 GB of memory, though technically 32-bit should allow for 4 GB. For comparison, my rather aged tablet has 2 GB of memory, so 3 GB would be a pretty serious limitation today. Contrast with 64-bit, that supports up to 16 exabytes, or 16 billion gigabytes. Certainly, a limit we are unlikely to hit any time soon… though I wouldn't assume that we never will.

Unit 9 - Data Management

Unit 9

Data Management

Apache Cassandra

Apache Cassandra is an open source distributed database system. It is a noSQL database, though this is something of a misnomer. "noSQL" was intended to signify databases that are not relational, but SQL is merely a query language rather than a database. Cassandra does, in fact, support a SQL query language.

I have used Apache Cassandra in my role as a research assistant, and was very impressed with it's capabilities. Particularly, it is extremely configurable: nodes across multiple clusters, degree of replication, and read and write consistency.

Cassandra can be configured to support any number of nodes: each of which may hold a complete copy of the database, or a subset of it. There can be multiple nodes at a single data center, or across many data centers.

Degree of data replication can be configured: how many to replicate at a node, or across different datacenters. Naturally, the more data is replicated, the less it is vulnerable to hardware failure, and it is more likely to be quickly retrieved (more potential sources to download from). Conversely, higher replication potentially makes writes much slower, depending on the write setting.

The read and write settings are of critical importance to the behavior of a Cassandra database. Write settings can be complete, or eventual. If complete, a write operation will only succeed once every instance of the data throughout the database has been updated. This means that read operations will be delayed until the write is complete, but ensures that all reads will reflect the change. If a write is eventual, it will complete once updating a certain configurable number of nodes, resulting in inconsistency throughout the database. The other nodes are updated through regular consistency checks, until all of the data is consistent. Until this process completes, it is possible for different read operations to return different results.

Read settings are similar: a read can be from a single node, multiple nodes, or the entire database. The first being fastest, but less likely to retrieve the most up to date piece of data, and the last being slowest, but certain to retrieve the most up to date data.

What setting to use for reads and writes is entirely dependent on the scenario. For performance critical applications that don't require that data always be up to date, partial writes and reads may be acceptable. And example of this is CERN, which uses Cassandra to rapidly ingest huge amounts of sensor data. Conversely, applications that aren't performance critical, but demand up to date data (for example, banking), a setting with complete writes and partial reads will ensure this.

As a noSQL database, Cassandra is not relational. Further distinguishing it from MySQL and similar, it is also column-based rather than row-based. It is designed to store denormalized data, and have tables that are configurable on the fly. New columns can be added to subsets of the table, or the entire table. The important thing is, it's flexible.

The downside of Cassandra is that it is bulky, and likely not as fast as a local MySQL database. That being said, Cassandra is quite painless to setup. I would suggest that if your data is denormalized, and access needs to be reliable and fast, use Cassandra. Otherwise, stick to MySQL. In fact, of late I do much of my work in Python, and having been using SQLite for rapid prototyping. It's not as powerful as MySQL, but it's trivial to setup and use. So, use the right tool for the job.

Food for Thought

Shamelessly quoting Wikipedia: a database is an organized collection of data.

A more detailed explanation is that it is a set of data tables, views, schemas, indexes, and all the other infrastructure necessary. Data tables define the format of the data: the columns of each record, and data format of each column. Views return user defined subsets of data from those tables: rather than seeing all the columns, perhaps the user only wants a few. Rather than

creating a custom query, they can just use the view. Indexes, which may be B trees or hashmaps, index the data by some key. This (potentially, if done right) can speed up queries that use that key.

Databases can be relational, where tables can be defined in reference to each other: a column in one table may hold keys in another table. A table may simply hold keys from two other tables, linking them together.

Databases can be noSQL, which as I said in the Cassandra section, is a misnomer. Tables are denormalized and nonrelational. For example, a noSQL database would better be suited for ingesting data that is not in a standardized format, such as documents.

Databases can be distributed (Cassandra) or not (MySQL). A distributed database is slower and bulkier, but scales with demand and less vulnerable to hardware failure.

Unit 10 - Operating Systems

Unit 10 – Operating Systems

History

For the sake of this discussion, I will call GM/NAA IO the first "real" operating system, IBM. This ran on the IBM 704, the first widely used computer with floating point operations, rather than being limited to integral ones. This platform and operating system was the one upon which Fortran and LISP were implemented: understandable as both languages would have demanded floating point operations for their respective purposes (scientific computing for Fortran, AI For LISP).

A following operating system by IBM as OS/360, which had a troubled history: it was larger than the space available on hardware at the time, and development was very slow. IBM was forced to implement subtypes of the operating system that could run on available hardware.

IBM released DOS/360 in 1966, and it became the most adopted operating system of the time. Like OS/360, it was released in variants so that it could run on different storage sizes: from 8 kb to 16 kb. It too supported Fortran, and most other high level languages. It could not run concurrent programs yet, thought successors could.

In the 1980s there was little standardization of or popularity of a single operating system for household computers. Floppy disks lacked sufficient space to store an operating system, so computers tended to just use whatever was preloaded on their hardware (Atari, Commodore 64, Vic-20, early Apples). Most supported a command prompt running the BASIC programming language: rather than being a command prompt in the modern sense, they had runtime interpreters of BASIC code as it was entered by the user. An anecdote: when I was a child, my family had a Commodore 64. My mother enjoyed purchasing magazines that came with BASIC source code for simple games and typing them in, then watching them run.

Later, computer hardware became more standardized and more spacious, so there was greater reason to standardize operation systems. This led to the contentious competition between DOS variants, Microsoft Windows, OSX, OS/2, and unix/unix-like. This has since narrowed to Windows, OSX, and Linux, with Windows dominating the home market and Linux the commercial one.

Food for Thought

In the early years of operating systems, competition and different versions made sense: the technology was developing and new ideas were appearing all the time. Further, hardware wasn't standardized yet, so it was difficult to make a single "one size fits all" operating system.

That is no longer the case today. I am entirely convinced that the different brands and versions of operating systems are only a thing for commercial reasons. First, I have some experience with machine code and Assembly. Their details differ on different platforms (Intel vs PowerPC, for example), but their general mode of operation is the same. You have a stack, you have a heap, you have registers, and you have operations. As long as the instructions are being processed in a way that isolated from the rest of the operating system, there is no reason why the instruction set should be something that makes or breaks an operating system. For a well written operating system, it should be simple(ish) to swap out one instruction set processor and replace it with another.

Second, one of the most important messages I got from the undergraduate operating system course is that fundamentally, operating systems pretty much run the same under the hood. They manage memory the same way, they interact with the file system the same way, they load balance threads the same way. They are not that fundamentally different internally. They have pros and cons: for example, my own experience is that setting up a web server or database on Linux is far, far easier than in windows. But these are high level features, and each continues separately because they have momentum and sunk cost in their given niche. Windows dominates the gaming and household computer market, Linux still dominates commercial servers, and Mac OSX dominates special snowflakes who want to create videos and edit photos (that last one is a joke!).

I have one important example of cash cow-ism: older versions of Windows not supporting newer versions of DirectX. For example, Windows XP did not support DirectX 10, allegedly because the kernel could not handle the operations required to use the new graphics card capabilities. So, DirectX 10 being the next "big thing" for video games, one had to upgrade to Windows 8. Never mind that Windows XP supported new versions of OpenGL that were able to use those allegedly inaccessible hardware capabilitites…

Exercises

My solution to this problem would be to port everything to Linux :) . I only half jest. I am no Linux fanboy: I use windows far more than I use Linux. But I loathe using windows for services and would have used Linux in the first place.
More seriously, my solution would be to move the older servers onto virtual machines, then port the software to the more up to date operating systems as convenience and cost demanded. This would be a case by case thing: perhaps some services are so rarely used that it wouldn't even be worth porting it off of the virtual machine.

Unit 11 – Applications

Unit 11

Applications

For this unit I will discuss Apache Ant, as I am not an expert in Java technologies, and this is one tool that I know very little about, yet see often.

Apache Ant is, simply, a Java equivalent to Linux make. Make allows one to setup custom build processes, and performs project management. For example, when run, it will ensure that C++

source files are only recompiled if their last write time is more recent than the last compilation time. This process is written in code in a makefile, which is run by make.

Apache Ant performs the same service for Java. It was implemented due to the lack of free, open source, implementations of make across all unix and unix-like platforms. The lack of a free, standardized tool made it difficult to build projects on all platforms, especially Solaris.

Ant differs from make in that the build file is in XML format, rather than platform specific shell commands. Ant abstracts these behind a consistent, higher level, XML format. Given XML's flexibility, this eases the addition of extensions to ANT. The disadvantage, relative to makefiles, is that XML files are far more verbose and less human readable. Because Ant XML files are not scripts, they do not have a problem flow or maintain states. This means that unlike makefiles, they can not be approached as something that is programmable: in this respect, makefiles are more flexible.

Food for Thought

My most used application, aside from Chrome or a handful of video games, would be Visual Studio. I use it extensively for both C# and Python, and occasionally C++. I have used it since Visual Studio 6 in 1998. It's not without its warts, but each is definitely superior to the previous, especially if you go as far back as Visual Studio 6. That version was notorious, for example, for not adhering to any C++ standard. Given that .NET wasn't a thing yet, this made it an exceedingly poor tool outside of using it for Visual Basic 6. I have unhappy memories of fighting with it to get a piece of C++ code working, because it's compiler did not adhere to the standard.

I have a very high opinion of later versions. I am, as a general rule, not a fan of working with Java technologies. This has nothing to do with the language itself, and everything to do with the tools. Why? Because for years I've been spoiled by Visual Studio. It does a great job of balancing power with ease of use. I never find myself fighting with it like I do with say, maven settings in an eclipse project.

My one criticism of it is that each new version seems to get bulkier and more sluggish. At this point, its pros still outweigh its cons, but I could see myself jumping to Visual Code at some point in the future as that IDE matures.

Final Thoughts

Although this course has it's warts, I enjoyed it. I can't say that I learned much from the specific course materials, but I loved the independent learning. Being free to choose topics and research things of my own choice really helped me find direction in my graduate project.

Not that I was without my challenges: my brother went to prison (don't ask), and it's shocking how much that affects a family and your personal life. My mother, who was already ailing, has been in and out of hospitals constantly. So, it has not been the happy fun times. If I had a higher course load, I would have withdrawn from the course. Fortunately, as I said, I found the material very easy. I just feel bad about not engaging in more discussions with the other students. I look forward to being able to do so more in subsequent courses.

For the independent learning, I give this course solid points. I only have two criticisms:

- I like the landing in principle. The concept is great: editable and commentable wikis? Yes. But it REALLY needs a makeover by a UI expert. So cumbersome, disorganized, and slow.

- I did not enjoy the RLO focus of the course. I'm not an educator, nor do I wish to be. I'd have preferred to do my research without having to think like one. But hey, that's just me. You don't always get to do things the way you want to!

In terms of course outcomes, I'd say I met them all, aside from "multimedia" presentations: all of my submissions were essays.

I did not directly explore the AU library, but that's because I already have Google Scholar setup to see it's contents :) . Otherwise, yes, I made extensive use of ACM and IEEE through the same.

I think my essays did a solid job of building up analytically, to a logical conclusion.

I think my essays were formated well, and my grades for the first two seem to corroborate that.

I did not discuss topics with my peers as much as I should have, because the family life is aweful right now.