MARCH 31, 2021

CONNECTING COMPUTER SCIENCE AND POLITICAL SCIENCE

A COMPUTER SCIENCE STUDENT'S PERSPECTIVE

PAUL HEIDEKRÜGER

Seminar: Forschungsdesign – Analyse und Vergleich polit. Systeme

Dr. Markus B. Siewert

Connecting Computer Science and Political Science: a Computer Science Student's Perspective

In his second moral letter to Lucilius, Seneca ponders discursiveness in reading. The philosopher argues that reading many authors, that is books, at the same time will make one's mind unsteady and discursive. He draws comparisons to frequent travellers, who will have lots of acquaintances but no friends, food that leaves the body immediately after it is eaten, offering no nutritional benefits, treatment of patients with different medicines, that will not lead to a quick recovery, wounds that will not heal when they are treated with different salves and finally, plants which will never grow to their full potential when they are repotted again and again.

His conclusion: "There is nothing so efficacious that it can be helpful while it is being shifted about. And in reading of many books is distraction." [1]

After reading this letter several times, my interpretation of Seneca's second moral letter is twofold. Firstly, and this to me is the obvious one, our mind's abilities, and especially those concerning multitasking, are limited. Therefore, by reading too many books, although it might yield a sense of productivity and learning, really, one is overwhelming their mind. Instead, one should be focussing on a limited number of books only, read unclear passages several times before moving on and revisit them frequently over time. Reading books is no different than studying for university for example. If we really want the learning to last, we cannot study six subjects at once. Hence, if we really want to learn what we read in books, we cannot read six books at once.

My second interpretation, which follows from the first to some extent, is that Seneca is alluding to the act of context switching and the fact that it is expensive. I am using the word expensive in the computer scientific sense, meaning that something requires a lot of a given resource. In this case that would be brainpower.

---

[1] Seneca, 'On Discursiveness in Reading', in *Moral Letters to Lucilius*, trans. Richard M. Gummere, vol. 1, 3 vols (London; New York: William Heinemann ; G. P. Putnam's Sons, 1917), Letter 2, https://en.wikisource.org/wiki/Moral_letters_to_Lucilius/Letter_2.

One might be able to relate this to their working life and especially those days where they were busy all day, but, in the evenings, the question of what one had accomplished that day would feel rather hard to answer. Those days might consist of a lot of context switching. Putting out figurative fires in lots of different places all day but no being productive. The productivity for that day consisted of context switching.

It appears that the expense of context switching is relative. It only becomes expensive when time is a constraint. Coming back to the university example, surely, you could study six subjects at once if a semester lasted a year instead of a couple of months. That means there exists an ideal ratio of context switches per time. If one is making too many context switches in a given amount of time, one must balance out the ratio by extending the time frame. Reading six books at once becomes feasible once we have enough time to spend per book. We might still make the same number of switches between books, but the time between those switches lasts longer when the overall time frame is increased.

As I was reading the Seneca letter and thinking about the above analogy, the term 'context switching' immediately reminded me of our operating systems lecture at university.

In a computer, the part that is responsible for doing all the computational work is called the central processing unit (CPU). It contrasts the graphical processing unit (GPU), which is, in oversimplified terms, responsible for making things look good on your computer monitor. The CPU on the other hand is responsible for crunching the numbers. For example, imagine a calculator application. If you type in '40 + 2', necessary computations will be performed by the CPU, but rendering the application with the result on your computer's screen will be done by the GPU. Mind you, rendering the application will still involve lots of number crunching, in that sense CPU and GPU are not that different, but that is beside the point.

Now, the addition above will be performed by a CPU in what is called a cycle. Think of a cycle as a unit of operation. For the sake of simplicity, we will assume that this addition will take exactly one cycle. However, when the expression is typed into the calculator application, there is still a lot going on in the background. The calculator might not be the only programme that is running, plus there is the whole operating system hosting the programme, essentially meaning that lots of tasks go on in parallel when using a computer. This might lead to the assumption that computers are able to do several things at once – true multitasking!

However, this is only partially the case. Whilst a computer may have a CPU consisting of several cores, e.g. four, allowing for operations to be processed in true parallelism, the cores themselves are only able to process tasks sequentially. Notice that the number of tasks that can be performed in parallel by the CPU cores, e.g. four, bears no proportion at all to the total number of tasks that must be processed. The reasons that computers are still able to do so many things at once is the fact that the CPU cores work at rates in the orders of several gigahertz, that is billions of cycles – remember additions – per second.

Every task that a CPU works on has a given context. In our calculator example that would be everything required to execute and run the programme, plus everything that we have done so far, which could be previous calculations for example. If the CPU decides that it wants to work on something else than the calculator programme – there might not be anything to do because it is waiting for user input – it will save the current context of the calculator programme and load the context for its next task. As not all computer memory is equally fast, switching contexts takes time. That means, a scheduler – that is the mechanism that decides what task the CPU core should work on next – must take into account that switching from context A to context B might be more time consuming (expensive) than switching from A to C, depending on which memory it must access. In other words, context switching is expensive. Having to wait even a millisecond for example would mean that we are sacrificing millions of cycles just for getting something from memory.

Notice again that we are facing a time constraint. From the perspective of the user, the computer should be doing its work in real-time, making the overall time frame very short and the ratio of context switches to time frame especially important.

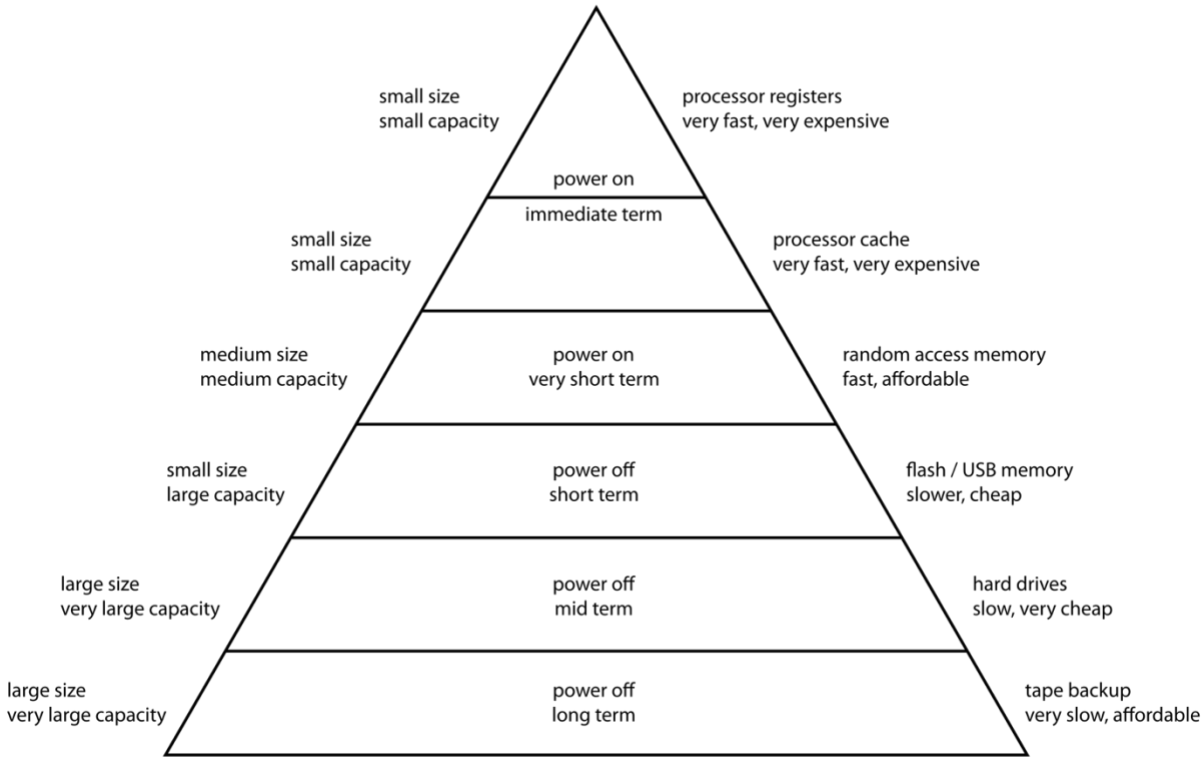


Figure 1: The memory hierarchy of a modern computing architecutre (public domain image).

Personally, I find this parallel to be nothing but beautiful. Roughly two thousand years ago, Seneca was thinking about context switching in terms of reading many books, and today, we are thinking about the same trade-off in terms of CPU scheduling. What we see here is the beauty of abstraction. Underlying CPU scheduling and plants that do not grow when being repotted, is a concept, describing a central property of the two. Seneca's plants and CPUs are connected via a generalisation. We shall call this abstraction by property.

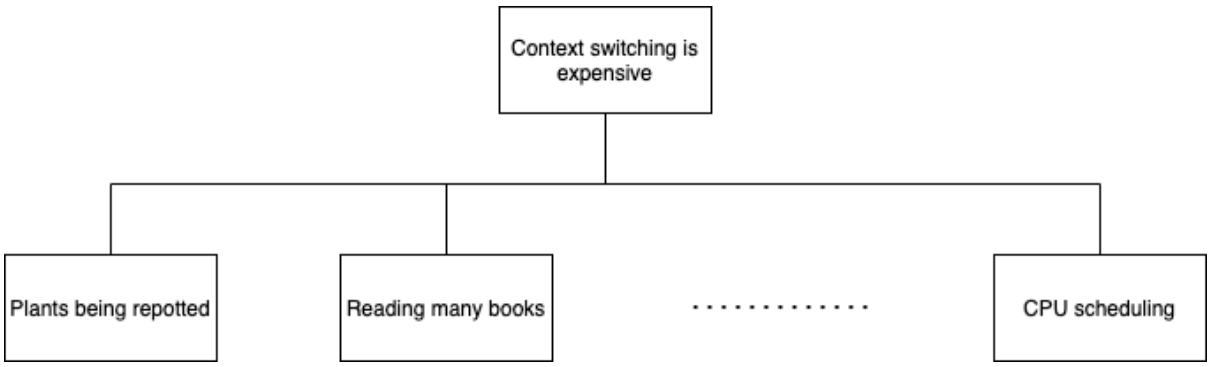


Figure 2: Abstraction in action

Now, you might notice that there is even a far more obvious parallel as Seneca himself uses abstraction in his letter to bring across his point of being overwhelmed by too many books; the moment you say something is like something else, reading many books is like repotting a plant several times, you are using abstraction. You are taking away details on both sides such that the equivalence holds. If we reduce reading many books to its property x and repotting plants to its property y, then x is equivalent to y, i.e. $x \equiv y$.

Especially interesting is the fact that abstraction is completely independent of what is being compared, which is exactly why we can draw a parallel between Seneca's moral letters and CPU scheduling.

Per definition, abstraction is a "general idea", i.e. generalisation, but it is not necessarily hierarchical, meaning that an abstraction can be an abstraction of an abstraction. [2] In the computer scientific sense, the relationship between the different layers is then often understood as "is implemented by". So, if layer A is stacked on top of layer B, then B implements A, and A is a generalisation of B. Furthermore, a computer scientific abstraction often involves several hierarchically arranged layers. We shall call this abstraction by implementation.

As a result, per my understanding, abstraction is recursive. Once we derive a generalisation from a property, concept or object, i.e. we have moved one step upwards in our hierarchy, we can apply the same procedure again and find another generalisation of the generalisation and so on. With recursion usually being an algorithmic concept in computer science, that is algorithms that call themselves again and again until a certain condition is met, this begs the question what stands at the top of any abstraction hierarchy? What condition must be met for the abstraction to stop? For now, the fact that abstractions can be hierarchical and recursive will suffice. The rest will be left as an exercise for the reader.

[2] 'Abstraction Noun - Definition, Pictures, Pronunciation and Usage Notes | Oxford Advanced Learner's Dictionary at OxfordLearnersDictionaries.Com', accessed 18 March 2021, https://www.oxfordlearnersdictionaries.com/definition/english/abstraction?q=abstraction.

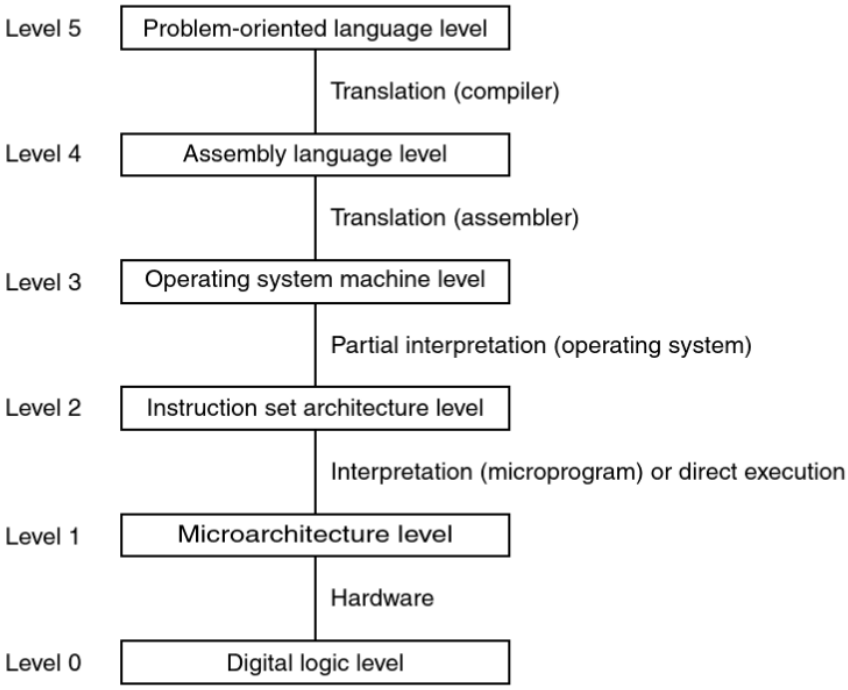


Figure 3: The layers of abstraction used in today's computing architectures.[3]

This is a prime example for abstraction by implementation.

Figure 4: Recursion in action in M. C. Escher's Hand with Reflecting Sphere[4]

[3] Andrew S. Tanenbaum, 'Contemporary Multilevel Machines', in *Structured Computer Organization*, 6th ed., Always Learning (Boston, Mass. ; Munich: Pearson, 2013), 5–9.

[4] Michele Emmer, 'M.C. Escher, Hand with Reflecting Sphere', in *M.C. Escher's Legacy* (Berlin, Heidelberg: Springer Berlin Heidelberg, 2003), 182.

For me, this must be one of the biggest misconceptions about computer science in general; from personal experience, it seems to me that people outside of the academic world (or even inside) never quite understand that computer science is not solely about programming. In fact, for me, programming is just one of several problem-solving tools computer scientists (and students) use. Yes, there are fields in computer science which are specialised on crafting those tools, thinking about their efficient use, making sure they can be used in the most varied conditions, yet they are only a subset of computer science.

It is why I prefer the term computer science over the term informatics for example. It underlines the fact that it is a scientific discipline. This is especially important when considering the German translation of computer science. German does not differentiate between computer science and informatics. It sees both as "Informatik", i.e. informatics, contributing to this misconception.

The same argument could be made for political science as well. Political science is not about being a politician. Whilst many might associate the term political science with the study of today's politics, this completely neglects the behind-the-scenes work if you will that enables such research in the first place.

Consider the fact that in contrast to computer science, political science is part of the social sciences. This means that it is concerned with observing reality and answering question such as how political actors, objects or institutions are, work, act or ideally should be. Especially, it does not deal with facts per my understanding. In the classic mathematical sense for example, $2 + 2 = 4$ holds. Not $2 + 2 = 3$, not $2 + 2 = 5$, not $2 + 2 = 3.9$. To convince one another of this fact, mathematicians can provide formal proofs, showing logically that $2 + 2 = 4$ holds given certain circumstance, i.e. the common understanding of maths. This could be done by showing that $2 + 2 \leq 4$ and $4 \leq 2 + 2$ hold, thereby proving the equation – the fact.

In social sciences however, this is not possible as the following (variation of a) famous example demonstrates. Say, you want to investigate the colour of swans. You go ahead, exploring different swan populations in Germany. Finally, you conclude "all swans are white".

In logical terms, you are saying, for all x of the set X the predicate P holds, where X is the set of all swans to ever live, and $P(x)$ is "The arbitrary swan x from the set of all swans is white".

If $P(x)$ holds for all swans, then the following logical formula will evaluate to $true$. The formula constitutes a fact if we evaluate it on X and P as per above.

$$\forall x \in X : P(x)$$

Figure 5: Logic in action

Notice two things. Firstly, I used abstraction here as you would have to give more context for this to be mathematically correct. Secondly, formulas cannot be $true$ or $false$ without an interpretation they are evaluated on. An interpretation assigns a meaning to variables in our examples that would the set of swans and the predicate of a swan being white.

If you were not looking at swans but numbers instead, you would be able to provide mathematical proof for your statement without having to look at all numbers. The proof is constructed in such a way that it abstracts from concrete numbers and looks at their properties instead. That means you can show that something holds for every number (in a given set) without showing it explicitly for every possible number.

But, as you are dealing with reality, you cannot come up with such a proof. The set X – the swans – is unknown to you. Yes, you looked at all swans in Germany, but you did not consider swans in other climates. If you would have done so, you would have discovered that in Australia, a black kind of swan exists.[5] Also, even if you were able to look at all swans in existence today, that still excludes all swans that have lived and will ever live.

What's the point of this? When dealing with reality, we never have the whole picture. We always consider a fragment. In order for this fragment to be of scientific use, social sciences have come up with a vast set of tools that help researchers look at reality in such a way that their results fulfill certain criteria, which include objectivity, reproducibility and transparency.

It is what I previously described as behind-the-scenes works and what is know as the scientific method, forming a structured way to make sense of reality – and a key example for abstraction. In fact it is a kind of abstraction that we have not discussed before. We shall call it abstraction by flow.

[5] Carles Carboneras and Guy M. Kirwan, 'Black Swan (Cygnus Atratus)', Birds of the World, 4 March 2020, https://birdsoftheworld.org/bow/species/blkswa/cur/introduction.

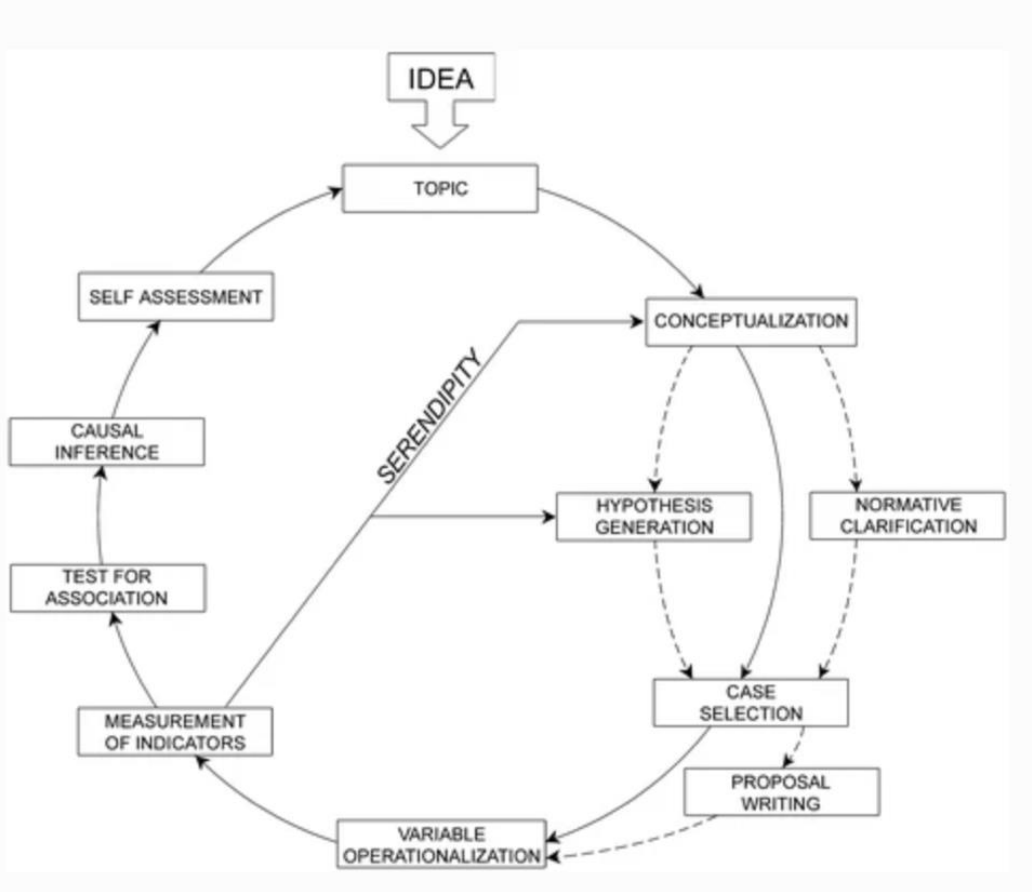


Figure 6: the scientific method[6] – abstraction by flow

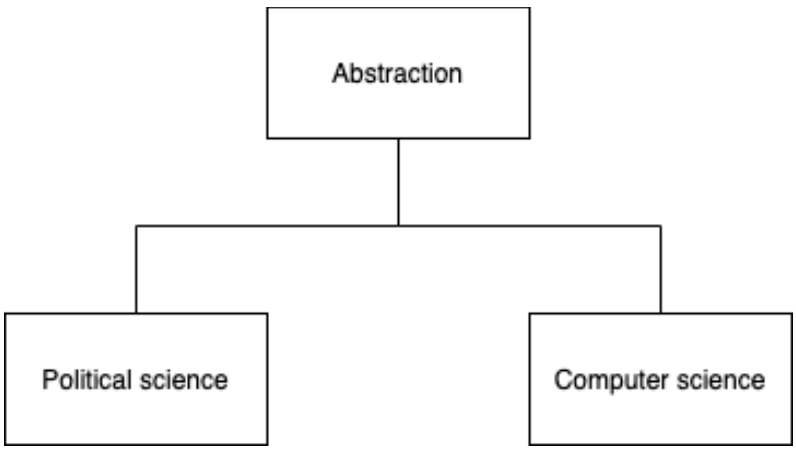


Figure 7: Connecting political science and computer science

[6] Philippe C. Schmitter, 'The Design of Social and Political Research', *Chinese Political Science Review* 1, no. 4 (1 December 2016): 577–609, https://doi.org/10.1007/s41111-016-0044-9.

Therefore, it marks the first parallel between computer science and political science. Let us now use abstraction to make sense of these abstractions.

In a talk on a reverse-engineered approach to human longevity[7], the physician Peter Attia discusses his approach on problem solving, which he mainly credits to Denis Calabrese, one of his mentors.[8]

His approach goes as follows: objective, strategy, tactics; where the strategy implements the objective, and the tactics implement the strategy. Notice that this abstraction fits the computer scientific definition perfectly. He then goes on to illustrate this with the "rumble in the jungle", a major fight in boxing history where underdog Muhammed Ali defeats George Foreman and the objective-strategy-tactics approach becomes very evident. However, this was only the introduction to a talk focused on how to maximise one's healthspan within a given lifespan, where this approach can be applied as well, as Attia argues.

In other words, our goal is getting from A to B, our strategy is what points we visit on the way and the tactics are how we make the journey between those points.

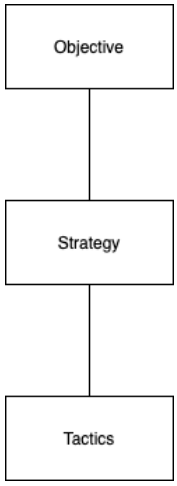


Figure 8: Peter Attia's problem solving approach – abstraction by implementation

[7] *Peter Attia - Reverse Engineered Approach to Human Longevity*, 2017, https://www.youtube.com/watch?v=vDFxdkck354.

[8] 'Mr. Denis Calabrese', American Conference Institute, accessed 18 March 2021, https://www.americanconference.com/speakers/mr-denis-calabrese/.

In computer science, we are not as strict, meaning that there exists no standardised research / problem solving process. In maths, I like to think of it in terms of toolboxes instead of standards. That means one has acquired a certain set of tools for solving math problems. Some tools are only work within a very narrow scope of problems whilst others are incredibly universal.

When confronted with a problem, one can start by looking at which tools can be applied. Then, which tools can be applied to the result of applying the first tool and so on. It becomes especially interesting when you can apply tools from different mathematical disciplines.

Let us first discuss the notion of an isomorphism, inspired by Douglas R. Hofstadter's Gödel, Escher Bach. [9] In abstract terms, an isomorphism can be thought of as a mapping between two different worlds where the essential properties of the elements remain after being mapped. Let us illustrate this with an example by giving an isomorphism between integer addition and a box of matches.

Suppose you are not convinced that $x + y = z$ holds when $x \to 2$, $y \to 2$ and $z \to 4$. Grab a box of matches. Now, for the mapping $Number \to matches$. Take x matches from the box. Take y matches from the box. Now but them side by side. How many matches do you have? We now have z and with it our inverse mapping; by counting all of the matches on the table we get our $matches \to number$ mapping.

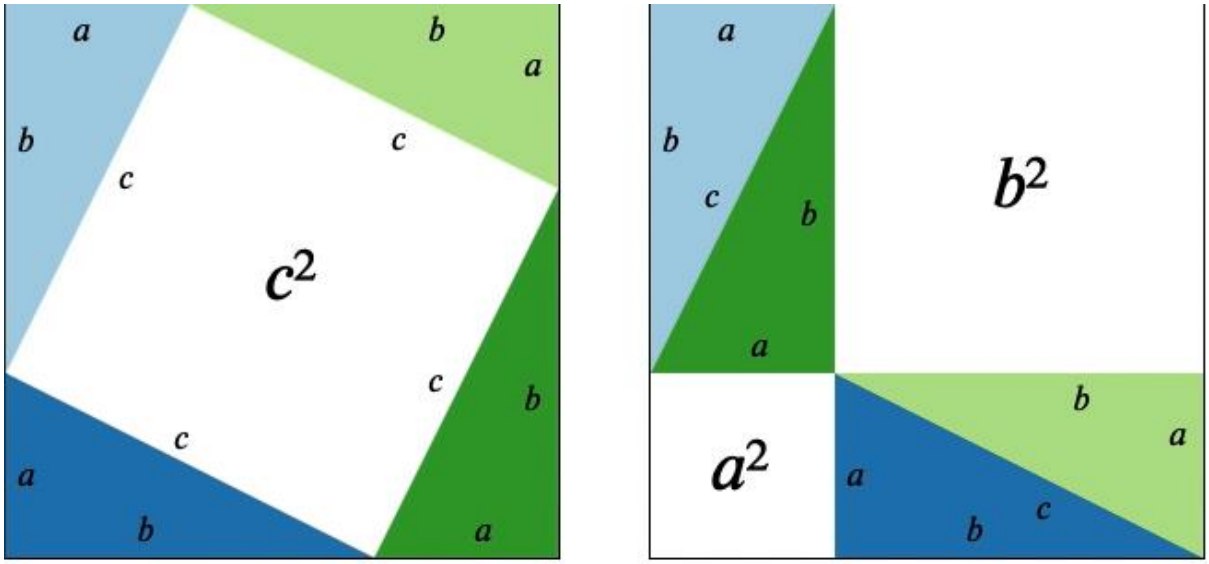
Now, the concept of an isomorphism should become clear; if you are facing a problem in space A that is isomorphic to space B, consider that it might be easier to solve in space B. For example, space A might still be relatively unexplored whilst space B is well known. Therefore, translate the problem from A to B, solve it in B and translate the solution back to A to get your result. A and B are isomorphic.

Building on the intuition of isomorphisms, let us explore how it relates to proofs. For example, consider the famous Pythagorean theorem $a^2 + b^2 = c^2$, stating that in a right-angled triangle, the square of the length of the hypothenuse c, that is the line opposite the right angle, is equal to the sum of the squared lengths of the remaining two sides a and b.

---

[9] Douglas R. Hofstadter, *Gödel, Escher, Bach: An Eternal Golden Braid*, 20th Anniversary Edition (New York: Basic Books, 1999).

Whilst you can prove it formally, as done by Euclid [10] or Einstein [11] for example, we can also prove it visually and therefore a lot more intuitively as $Figure\ 9$ shows.



$Figure\ 9$: $A\ proof\ without\ words.$ [12]

$If\ the\ animation\ does\ not\ work, it\ can\ also\ be\ viewed\ here.$

Ergo, there exist many ways for solving mathematical problems. In terms of the previously introduced problem solving framework, for a given goal, we are free to choose our strategy and tactics within the space of mathematics to achieve it.

[10] Euclid, 'In Right-Angled Triangles the Square on the Side Subtending the Right Angle Is Equal to the Squares on the Sides Containing the Right Angle.', in *Elements*, vol. 1, 13 vols (Oxford: Clay Mathematics Institute Historical Archive, 2008), Proposition 47, https://www.claymath.org/library/historical/euclid/files/elem.1.47.html.

[11] Steven Strogatz, 'Einstein's First Proof', *New Yorker*, 19 November 2015, https://www.newyorker.com/tech/annals-of-technology/einsteins-first-proof-pythagorean-theorem#:~:text=Einstein%20became%20particularly%20enamored%20of,own%20mathematical%20proof%20of%20it.&text=You%20may%20once%20have%20memorized,%2B%20b2%20%3D%20c2.

[12] William B. Faulk, *Pythagoras Proof Animated*, 20 November 2014, Animation, 20 November 2014, https://commons.wikimedia.org/wiki/File:Pythagoras-proof-anim.svg.

Let us now apply this in the contexts of political science. We discussed the empirical research process before and that it adheres to a set of quality criteria. Therefore, we can deduce the following:

$$Objective \rightarrow Answer\ research\ question\ whilst\ adhering\ to\ quality\ criteria$$

$$Strategy \rightarrow Empirical\ research\ designs\ within\ the\ standard\ reserach\ process$$

$$Tactics \rightarrow Surveys, intervies, statistical\ tests\ \ldots$$

Compared to math, we are a lot more constrained in our strategy / tactics here. Yes, we can choose different tactics, but it is more of an off-the-shelf approach as we are constrained by our quality criteria.[13] Also, once we choose a strategy, our choice is a lot more final than with maths since external factors such as money and time play a lot more significant role here.

However, not all of computer science is math. As mentioned before, computer science offers a vast array of different research disciplines. So, to contrast maths, let us choose a very concrete example. Assume, we want to build a new piece of software. Then, we can apply the software engineering lifecycle, which gives way to the following.

$$Objective \rightarrow Our\ new\ piece\ of\ software$$

$$Strategy \rightarrow The\ steps\ in\ the\ software\ engineering\ lifecycle$$

$$Tactics \rightarrow Design\ patterns, frameworks, testing\ suites\ \ldots$$

Now, software engineering marks an especially interesting example as it is – unlike most other disciplines in computer science – about dealing with people too.

The initial phases of the software development lifecycle are all about the customers. They are about understanding what they want and how to model it within the problem domain, i.e. terms that are understood by the customers too and not just the developer.

[13] Claudius Wagemann, Achim Goerres, and Markus B. Siewert, *Handbuch Methoden Der Politikwissenschaft* (Wiesbaden: Springer VS, n.d.).

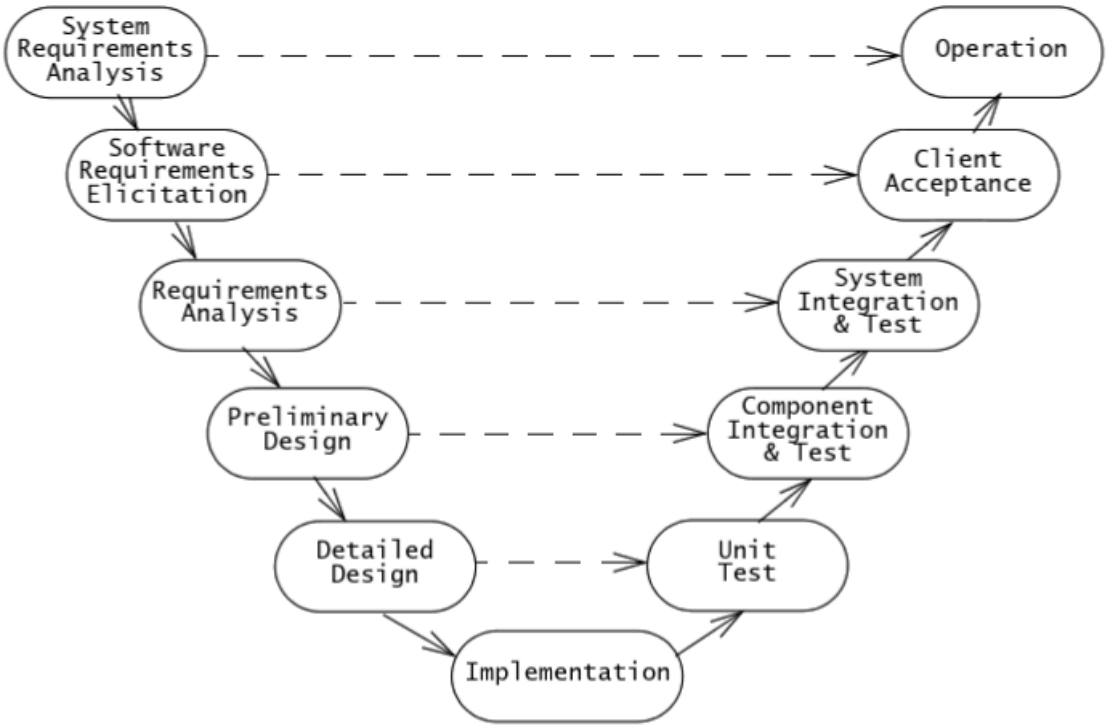


Figure 10: *Software engineering lifecycle*[14] *– abstraction by flow.*

It is here where computer scientists all of a sudden go from thinking about code to thinking about communicating with customers. It is the same when political scientists conduct a survey for example and are suddenly confronted with the problem of how participants will read the questions. Does ordering matter? With what question should we start? Should we cluster the questions?

I personally think that it would be easy to discard such problems as being unscientific as they do not seem to concern the issues at hand. However, it seems to me that this is exactly where the distinction between being good and being brilliant occurs. Excellent scientists will have put extensive thought into the ordering of their questions. Excellent software engineers will have put extensive thought into communication strategies for their customers. It is that which produces excellent science / software, adhering to the desired quality criteria.

The sentiment of discarding such problems also underlines for me what I have experienced as a holier-than-though attitude of natural sciences towards social sciences or even within

[14] Bernd Bruegge and Allen H. Dutoit, 'V-Model of Software Development', in *Object-Oriented Software Engineering Using UML, Patterns, and Java: Pearson New International Edition*, 3rd ed. (Pearson Education Limited, 2013), 624.

natural sciences themselves. I have been thinking more about this ever since listening to a podcast interview recently, which amongst others briefly discussed the statement "Most of the disciplines that have science at the end of them are not actually science." [15] The mentioned exceptions are computer science and neuroscience.

Right now, I am at a point where I think that this statement is too narrow-minded. It seems to focus on the interesting parts of natural sciences, whilst discarding their less interesting parts and focussing on the less interesting parts of social sciences, whilst discarding their interesting parts.

I do think that one must see the bigger picture here and realise that political science and computer science are not that different in some regards. We have dived into abstraction above, but consider their structure as another example.

Both science's research fields cover the spectrum from theoretical – via theoretical computer science and political theory – to practical – via software engineering and political data science for example. Both theoretical disciplines cover the questions of how things ideally should be, what is possible and why certain things cannot work.

There is, in both sciences, a strong contrast between theory and practice. In computer science for example, theoretical scientists are amongst others concerned with the runtime of algorithms. Now, clearly, the same algorithm will have different runtimes on every computer. Circumstances play a huge role here, i.e. which programming language was used, how it was compiled, on which operating system the code is running, what hardware the computer has, what is running in the background, the list goes on.

Theorists are also not concerned with an absolute runtime, i.e. in how many fractions of a second computations can be performed – this is circumstantial as mentioned – but rather they look at the relation between input size and operations used to compute the output. This is formalised in what is called Big O notation.

[15] Tim Ferriss, 'The Tim Ferriss Show Transcripts: Naval Ravikant on Happiness, Reducing Anxiety, Crypto Stablecoins, and Crypto Strategy (#473)', accessed 18 March 2021, https://tim.blog/2020/10/15/naval-transcript/.

An example: consider an algorithm that computes the factorial

$$n! = n * (n-1) * (n-2) * \cdots * 1 = \prod_{i=1}^{n} i$$

Figure 11: Factorial

For every n that we feed into our equation, our algorithm computes n multiplications. Therefore, our runtime will be in $O(n)$. In simple terms, this is a measure for the worst-case runtime an algorithm will have. In this case it is also our best-case runtime as the algorithm will take n steps, no matter the input. As a result, it is in $\Theta(n)$. However, it is common practice to just settle for the $O(n)$ instead.

$O(n)$ describes linear runtime. That means if our input size increases by 2, the number of computations required to compute the output will increase by the same number (in the worst case).

More complex algorithms might result in more complex runtimes such as $O(n^{12})$. From a theoretical point of view, the runtime of this algorithm will still be regarded as acceptable as it is polynomial in the size of the input – worse than linear but still acceptable. However, realistically, if n is large enough, the algorithm will quickly surpass what is computationally feasible.

As there is no ideal scenario for algorithms to run in, theorists abstract away from all the circumstantial details that might affect runtime and construct a theoretically ideal scenario instead. This scenario or in better terms model is called a Turing machine and sort of bridges the gap between theoretical and applied computer science.

It is almost like a switch that happens when going from theory to application. In my mind, it resonates with the isomorphism analogy from before. We go from theory to application and back again, constantly.

Spinning this back to political science, such a shift happens within the social scientific research process as well. Once we have defined our hypothesis, we are concerned with operationalisation. This is where we consciously try to plan out our move from theory into the real world.

Then, when have gathered and processed our data, its back to theory. We are constantly switching from theory to the real world and back again.

It reminds me of the switch from continuous to discrete, which is made very often in computer science. In fact, there is a whole mathematical field concerned with the best ways of translating methods from the mathematical world – often continuous – into the discrete world of computers. Namely, that is numerical mathematics.

Why is this translation necessary? A computer only has a finite amount of memory. That is a finite number of zeros and ones for performing calculations. However, what if one wants to work with numbers that are larger than those that can be represented with the given memory? What if I want to perform exact calculations with numbers that are very small and therefore require a lot of detail (memory) in their representation?

To outline the continuous part of mathematics, think of the real numbers. That is all positive integers, all negative integers, zero, and everything in between, i.e. all rational and irrational numbers. The set of real numbers has a property called density. That means for every two real numbers, I can construct another real number that will be right in between the two. This process can then be repeated infinitely often. So, if we want to store a large enough set of real numbers in a computer, we will eventually run out of memory.

Say you want to develop a flight simulator. That flight simulator should allow players to travel the whole world whilst providing a realistic piloting experience. Now, how does one model the real world in such a way that is visually appealing to players but also computationally feasible? How does one model an aeroplane in such a way that it behaves according to the laws of physics? How does one model realistic weather? How can all of this be done with a finite amount of memory?

Whilst flight simulators are not being actively researched to my knowledge, the methods used to develop such simulators are – numerical mathematics.

Another example. Consider digital photography. We are taking a snapshot of reality and represent it discretely with a finite number of pixels. Then, with this discrete representation, once again, we are trying to find a balance between quality and feasibility. As with the flight simulator, you want your image to look good and be true to what it is depicting, but you also want its representation in memory to be as small as possible. It is a constant trade-off.

Compare this with doing a qualitative comparative analysis (QCA) for example, it is the same step that is being made. We are trying to translate something that occurs in the real world in such a way that it can be represented discretely. That could be either in discrete steps or even binary such that we can use logic – again a field that is present in computer science and political science. In fact, it is analogous to what the scientific research process is about: capturing a precise, well-thought out snapshot of reality, whilst battling constant trade-offs.

I would like to close with the question our operating systems professor asked at the beginning of the semester: “is computer science just an auxiliary science?” That is, does computer science only exist for enabling other sciences to utilise computers in an efficacious way?

I have not been able to answer this question for myself as of yet. However, what has become clear to me is that computer science builds upon several other sciences just as political science does.

Per definition, amongst others, computer science is tied to mathematics, electrical engineering, physics. Computer science uses maths to describe what these products of engineering are supposed to do / doing and uses methods and concepts from these fields for its own research.

And stretching the definition even further, you have psychology involved the software engineering lifecycle. You have disciplines like linguistics involved in natural language processing. There is even a whole discipline of its own – bioinformatics – that ties together biology, medicine and computer science. Plus, you have various smaller fields like legal informatics.

Political science is not too dissimilar in that regard. And people seem to wonder. If you type into Google “is political science” the first suggestion you will get is “is political science a science”. This also ties in with the podcast interview mentioned above. It seems that political science and other sciences too are often regarded as inferior and are trying to disprove the general notion of not being a real science.

My thought process right now is the following. Computer science as well as political science both study computers and politics respectively. Yes, both are not creations of nature. Still, both go about their creation of knowledge in systematic way that is supposed to fulfill quality criteria such as objectivity or reproducibility.

And there is even an argument to be made that political institutions are in fact a natural occurrence, owing their appearance to the "political animals" that humans are. [16]

I am convinced that asking whether certain disciplines are scientific or not runs counter to what science should be about – creating knowledge, collaborating and learning. In fact, we should encourage interdisciplinary research instead of dismissing disciplines. Abstract connections between sciences are way too exciting to be left unexplored.

Note from the Author

This text draws upon my experience as a computer science student at the Technical University of Munich with a minor in political science. Views represented are mine and mine only. Especially, they do not represent those of my university or my employer. The text is aimed at readers who are new to computer science but are familiar with political science.

[16] Aristotle, 'Political Animal', in *Politics*, n.d., 1253a, http://www.perseus.tufts.edu/hopper/text?doc=Perseus:abo:tlg,0086,035:1:1253a.

References

'Abstraction Noun - Definition, Pictures, Pronunciation and Usage Notes | Oxford Advanced Learner's Dictionary at OxfordLearnersDictionaries.Com'. Accessed 18 March 2021. https://www.oxfordlearnersdictionaries.com/definition/english/abstraction?q=abstraction.

Aristotle. 'Political Animal'. In *Politics*, 1253a, n.d. http://www.perseus.tufts.edu/hopper/text?doc=Perseus:abo:tlg,0086,035:1:1253a.

Bruegge, Bernd, and Allen H. Dutoit. 'V-Model of Software Development'. In *Object-Oriented Software Engineering Using UML, Patterns, and Java: Pearson New International Edition*, 3rd ed., 624. Pearson Education Limited, 2013.

Carboneras, Carles, and Guy M. Kirwan. 'Black Swan (Cygnus Atratus)'. Birds of the World, 4 March 2020. https://birdsoftheworld.org/bow/species/blkswa/cur/introduction.

Emmer, Michele. 'M.C. Escher, Hand with Reflecting Sphere'. In *M.C. Escher's Legacy*, 182. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

Euclid. 'In Right-Angled Triangles the Square on the Side Subtending the Right Angle Is Equal to the Squares on the Sides Containing the Right Angle.' In *Elements*, 1:Proposition 47. Oxford: Clay Mathematics Institute Historical Archive, 2008. https://www.claymath.org/library/historical/euclid/files/elem.1.47.html.

Faulk, William B. *Pythagoras Proof Animated*. 20 November 2014. Animation. https://commons.wikimedia.org/wiki/File:Pythagoras-proof-anim.svg.

Ferriss, Tim. 'The Tim Ferriss Show Transcripts: Naval Ravikant on Happiness, Reducing Anxiety, Crypto Stablecoins, and Crypto Strategy (#473)'. Accessed 18 March 2021. https://tim.blog/2020/10/15/naval-transcript/.

Hofstadter, Douglas R. *Gödel, Escher, Bach: An Eternal Golden Braid*. 20th Anniversary Edition. New York: Basic Books, 1999.

American Conference Institute. 'Mr. Denis Calabrese'. Accessed 18 March 2021. https://www.americanconference.com/speakers/mr-denis-calabrese/.

Peter Attia - Reverse Engineered Approach to Human Longevity, 2017. https://www.youtube.com/watch?v=vDFxdkck354.

Schmitter, Philippe C. 'The Design of Social and Political Research'. *Chinese Political Science Review* 1, no. 4 (1 December 2016): 577–609. https://doi.org/10.1007/s41111-016-0044-9.

Seneca. 'On Discursiveness in Reading'. In *Moral Letters to Lucilius*, translated by Richard M. Gummere, 1:Letter 2. London; New York: William Heinemann ; G. P. Putnam's Sons, 1917. https://en.wikisource.org/wiki/Moral_letters_to_Lucilius/Letter_2.

Strogatz, Steven. ‘Einstein’s First Proof’. *New Yorker*, 19 November 2015. https://www.newyorker.com/tech/annals-of-technology/einsteins-first-proof-pythagorean-theorem#:~:text=Einstein%20became%20particularly%20enamored%20of,own%20mathematical%20proof%20of%20it.&text=You%20may%20once%20have%20memorized,%2B%20b2%20%3D%20c2.

Tanenbaum, Andrew S. ‘Contemporary Multilevel Machines’. In *Structured Computer Organization*, 6th ed., 5–9. Always Learning. Boston, Mass. ; Munich: Pearson, 2013.

Wagemann, Claudius, Achim Goerres, and Markus B. Siewert. *Handbuch Methoden Der Politikwissenschaft*. Wiesbaden: Springer VS, n.d.