Software Engineering and autonomous biological scientific discovery

Filip Kronström, filipkro@chalmers.se

August 31, 2023

For many years, yeast has been an organism of great interest for the biological research community. This research is partly motivated by this organism's importance, e.g., when brewing beer or wine, and for the production of biofuels. Another important factor is that, thanks to its common, although very distant, ancestry with us humans it can be used as a model organism for humans. As such it can help us understand gene functions or act as an early target for drug trials.

One of the keys for improving our understanding of any biological system is to perform lots of experiments. However, the set of possible experiments one can perform is incredibly large and most of the possible experiments are not very informative. This is where the work of my research group comes in. By combining artificial intelligence and laboratory robotics with computational biology and experimental biology we are developing a system performing autonomous cycles of biological experiments. This is done by letting a computer form hypotheses based on the currently available information, such as computational models or information in public databases, and design experiments to test them. After incorporating the results from the experiment this cycle of autonomous hypothesis led experiments are repeated. In the short term, we have laboratory equipment to perform a handful of experiments simultaneously, but eventually the plan is to expand and have hundreds or thousands of experiments running in parallel. The aim is that by, at least partly, removing the human from this scientific process reduce the costs and improve the speed, reproducibility, and quality of the experiments and science conducted.

My work is primarily focused on how to represent the knowledge and information in this system, and how learning can be improved through reasonable knowledge representations. It also involves development of software for our machines, both low level embedded software controlling, e.g., pumps and valves, but also higher level interfaces to the machines.

Robert's lectures

Requirements engineering

To design or develop any product, be that software or a physical product, you first specify the requirements needed to solve the problem. In a way this is quite similar to research. You (or sometimes your PI) have a high level, usually quite abstract

problem. To actually implement something solving this you need to break it down to concrete subproblems with requirements for their solutions.

One part of the work we do in our group is to develop a new experiment platform together with a group at Vanderbilt University who do the hardware design and manufacturing. This means our high level thoughts about the machine and what we want to be able to do with it needs to be translated to concrete requirements and specifications for them to design a product around.

Pragmatic Software Engineering "Rules"

I think it is very useful to be reminded about some concrete advise on how to write good code. Although most of us probably have heard it before I doubt a majority is really adhering to it when doing our research. Personally, starting a new project I always try to keep my code base well organised, but somewhere halfway through I usually give up on this ambition. Then, at the end of the project you have to spend all this unnecessary time cleaning code and repositories. One thing I have noticed is that this tends to be a bigger problem when I am working alone on projects. Working together with others forces you to work harder on maintaining organisation.

As a PhD student you easily end up thinking you do not have time to stick to good practice. However, over the four-five years we are doing this we would most likely save time doing it properly from the beginning instead. More importantly I think we need to keep in mind that we actually are students and that one important skill for most of us to acquire actually is to write good code.

Guest lectures

Boundary mining

I like when tedious tasks can be automated. Hence, I found the boundary mining using program "derivatives" to be interesting. Especially the fact that it worked using such a simple and crude distance metric as `strlendist`.

A somewhat similar approach can be used to find interesting parts of the experiment parameter space in our research field. Areas where small changes in inputs have big effects on the measurements probably contain useful information about the observed system.

AutoML

By automating and streamlining the design and training of ML models AutoML can greatly simplify and speed up the rate at which decent models for a specific task can be found. In Calefato's lecture this was mainly discussed in relation to sentiment analysis, but the good thing with AutoML is that as long as you have a dataset it simplifies the training of models, no matter the domain.

For us this can be useful as predictive models are an important part of choosing experiments, but we can't spend to much time engineering every individual one.

Two topics

Human-Computer Interaction

Human-computer interaction (HCI) covers the broad field of interfaces and interactions between human users and any kind of computer system. This system could be anything between a traditional software or application running on a computer or a phone, and a cyber physical system like an autonomous car or an industrial robot. No matter the system, a central aspect is to be able to intuitively communicate its current and upcoming state to the human, such that the user has a feeling for what will happen next and the effect of its actions. For a simpler system like an application running on a smartphone, the main goal might be to make it more intuitive and enjoyable to use. For a system like selfdriving cars, which we want to largely occupy the same spaces as humans, I think it is one of the central problems to solve. Today most of us have a feeling for what cars are about to do, since other humans are driving them, that is not necessarily the case when they are controlled by computers.

Related to my research I think this is a very interesting and important topic. Science has always been about us trying to understand this universe better. Hence it is crucial that findings from a "robot scientist", at some level, can be communicated to humans. It could for instance be interesting to explore how LLMs could be used to explain discoveries in natural language.

Another aspect of HCI connected to my field I find intriguing is the is the use of humans in the loop for experiment selection. Gilad Kusne et al. recently published their work on closed loop experiment selection for material science [1]. Along with a fully automatic experiment selection, they also allow a human researchers to help guiding the process. For this the system's internal information about the material, along with its confidence, is communicated through live visualisations. Fully autonomous system performing truly ground breaking research might require a lot of work to achieve, but I think this kind of human-machine symbiosis can prove efficient with relatively low effort already today.

Writing software to control robots and making them accessible for researchers with a completely different background has also forced me to think about another aspect of HCI. For someone with a computer science-like background a command-line interface might feel like the most efficient and simple way to interact with a system. This is most likely not the case for most others, for instance experimental biologists. This illustrates one realisation you get from working in an interdisciplinary project. I think this is useful for most researchers, primarily since it forces you to see things from the perspective of someone outside your bubble (which, to be fair, is a big majority of the rest of the world).

Sustainability

The next topic I want to discuss is sustainability of research related software, primarily in terms of its longevity and reusability. The FAIR (findability, accessibility, interoperability, reusability) guiding principles about data have been adopted by a wide research community to simplify the sharing of data. Lamprecht et al. argue they should also be adopted for research software [2], mainly by providing the software at

a persistent location along with extensive metadata and information about what is needed and how to run it. Personally I find it very helpful to have code I can run and look at in parallel to reading a paper, yet I am surprised by the amount of times this is, for some reason, not possible.

Within a research lab there might very well also be lab specific software that is not necessarily meant to be published. For instance, we have quite a lot of internal software for our lab robots. When developing such software its sustainability is also very important, but in a way that is more (I imagine) similar to software developed for products at companies. You would hope it works perfectly well once it is in place, but of course this is rarely the case. In most cases this software will also require maintenance. Since there is a constant staff turnover in a research lab such a code base needs to be documented and developed in a way making it maintainable by anyone, which is not necessarily the primary feature you normally think about as a researcher.

Finally, something I personally know I need to improve is the sustainability of software I write to explore a topic. This is quite a central part of my research process, you try things out and sometimes they work, sometimes they do not. However, even if they don't work you never know when you want to revisit them. Going back to something you did months or even years ago is not trivial unless you put some effort into, for example, documentation.

Discussion about future

As a result of automation playing a greater part in various fields so will also software engineering. I think this will be most obvious in fields that have traditionally not been so software heavy, such as experimental biology.

However, in industries where software engineering is already playing a big part we have recently seen big layoffs. These have not necessarily been a result of AI/ML methods replacing human labour, but considering the capabilities shown by LLMs the last year maybe this trend will continue. For sure it will change the way a lot of software engineering will be done. More emphasis will probably be put on prompt engineering and validation of programs generated by machines. Of course the speed of this adoption will vary depending on the application, e.g, safety critical and high security application will likely stick to "traditional" software engineering longer.

In general I can definitely see positive outcomes of a higher degree of automation through AI/ML in society. For instance I am convinced it can help generate better solutions to many of our big problems (e.g. climate change, drug development), while also allowing us more time to spend doing what we actually want to do. At the same time I think it can cause even more even bigger problems unless it is deployed in a sensible way. The main problem I see is that power and resources will be even more centralised to a few actors and that these actors to a higher degree will be companies instead of institutions being controlled in democratic ways. I don't know what needs to be done to avoid this scenario, but I believe it is important to think about it sooner rather than later.

References

[1] A. G. Kusne et al. On-the-fly closed-loop materials discovery via bayesian active learning. *Nature Communications*, 11(1):5966, 2020. Number: 1 Publisher: Nature Publishing Group.

[2] A.-L. Lamprecht et al. Towards FAIR principles for research software. *Data Science*, 3(1):37–59, 2020. Publisher: IOS Press.