Qu'est ce qu'un programme? Perspectives Historiquges et Philosophiques
What is a program? Historical and Philosophical perspectives

Abstract

What is a computer program? This seemingly simple question, which lies at the heart of computer science, has no simple answer today, neither in academia nor in industry. Nonetheless, the responses one gives to it affect very real problems, such as the issue of responsibility when a given piece of software fails. The aim of this project is to revisit that fundamental question from a more historical and philosophical angle. The project starts from a basic characterization of "program" along three different "modalities": a physical, a formal and a socio-technical modality. Any program is rooted in these three modalities and any understanding of "program" needs to incorporate them. This project wants to understand what a "program" is by focusing on the formal modality (mathematical, logical and linguistic properties), how it accommodates the two other modalities and how they come together in a programming practice.

I. Objectives and scientific hypotheses

What is a computer program? This seemingly simple question, on a notion which lies at the heart of computer science, has no simple answer today, neither in academia nor in industry. The responses one gives to it affect very real problems: Who is responsible if a given piece of software fails;[1] whether copyright or patent law applies to "programs",[2] or whether a "program" is correct or not[3]? So far, there are no clear answers to any of these concrete problems and so the need for more foundational research is not just a theoretical one.

A particular challenge of the notion of program today is the diversity of existing approaches and answers. Is a program a piece of symbolic text which requires logico-mathematical analysis? Or is it rather a configuration of physical entities, like electrons and magnetic charges, that reside in the digital circuits of a computer? What is the difference, if any, between algorithm and program? Is a program an object? If yes, is it a technological (and concrete) object, or, as many people describe it, a "liminal" one, bridging the formal and the abstract? How can the relation between these two interpretations be explained? Is the meaning of a program its execution (cf. operational semantics) or can it be captured adequately enough by its input/output behavior (cf. denotational semantics)? Is it possible to verify the correctness of a program and, if yes, what does it *really* mean to do so? Can we talk about programs without code? What is the difference between software and programs? And so on. This variety of questions indicates that there is a broad range of positions to be considered and no easy way to see the wood for the trees.

The diversity of approaches to the notion of program reflects a deeper characteristic, namely the fact that "program" has modalities. As a symbolic text, it has a mathematical and linguistic modality – a formal modality for short; as something that is stored *and* executed by a machine, it has a physical modality; and as something that is made and used by people it has a socio-technical modality. As such, our notion of program is rooted in these three modalities and any understanding of "program" needs to incorporate each of them and account for their relations. From this hypothesis, this project **aims at developing the first coherent analysis and pluralistic understanding of "program" and its implications to theory and practice.**

There are two major challenges that need to be faced:

Challenge 1: Communication gaps. An important consequence of the tripartite structure of "program" is that different competencies and methodologies are required to develop good programs and it often depends on one's own background what is considered to be a (good) program. To put it in a simplified manner, a software engineer working at Microsoft on requirements will have another viewpoint than a logician working on and with Coq. This diversity of approaches need not be a problem and can in fact be understood as a positive feature of modern computing, as long

1 For instance, recently Bringsjord claimed he would not be responsible if any of the robots that engineers built with his system would do something "wrong" because he would not be the one defining the "ethical rules" (Brinsgsjord, 2016)

2 See for instance (Newell, 1986) for an early discussion of this problem.

3 And, one might add, whether a program *is* still a program if it is considered to be incorrect.

as it does not result in a disciplinary divide. The plural nature of "program" and computing has more often than not resulted in a barrier between different communities of people though the moments of dialogue and interaction between different groups have proven most fruitful.[4] Best known is the so-called divide between the theoretical and the practical people, though it is certainly not the only one. Such divides run across the (disciplinary) history of computing and have almost become a part of its identity.[5] The result is a set of "communication gaps"[6] between different communities that contribute to the fragmentation of the computing field today. To give just one clear and almost prototypical example: the so-called software crisis from the late 60s and early 70s soon resulted into at least two camps: the theoretically-minded people led by Edsger W. Dijkstra who were convinced that a theoretical and formal approach was required to tackle the correctness problem and the more practically-minded people who did not follow a purely formal approach but would, for instance, rely more on testing and debugging. Soon, the community of researchers around Dijkstra would distance themselves from "software engineering" because it was not theory-driven. In order to develop a coherent and pluralistic understanding of "program" it is necessary to face those communication gaps.[7]

Challenge 2: Opacity Today, when using a computer or some other computational device, one is always using some kind of interface. Unlike the early days, direct interaction with the hardware is often not desired because, practically speaking, one cannot do what we can do today without the introduction of several layers of abstraction between the user and the electronic circuits. A consequence of this is that *any* program which we develop or use today, often hides (part of) that which lies "below" it. These layers and their complexities have been constructed historically on top of each other in different socio-technical contexts and so, to some extent, programs contain and hide the traces of their own history: each layer and set of interactions added, might assume a different understanding of "program". Or, to put it differently, a "program" in the context of the Manchester Mark I is quite different from a "program" such as Whatsapp? Usually, the different layers of abstraction are not transparent because the program and/or one of its layers contains parts which are closed to the user to "protect" him/her from making some mistake and/or to protect corporate interests. Moreover, each layer has its own character and the connections in between the many different layers are far from simple. As a result, "programs" are complex objects up to the point of being opaque. In order to develop a pluralistic and systematic analysis of program it is necessary to "break" through this opacity so that our understanding of "program" is not reduced to one particular such understanding.

The approach of this project is to face Challenges 1 and 2 using **a combined historical *and* philosophical approach.** The general aim is to retrace the different meanings of "program" through its own history and, by so doing, identify and render transparent the *different* understandings *and* practices underpinning "program". This will be done in such a way that it is possible to analyze these different understandings relative to the three program modalities and their interrelations (See I.b. for more details). In that way, it is possible to differentiate also between those meanings of "program" that are historically rooted in and shaped by all three modalities and so have crossed communication gaps against those that have not. Philosophically, the task is to identify the ontic properties that are essential to a taxonomy of the different meanings of "program", including their different practical instantiantions, and to use standard or *ad hoc* methodologies from the philosophy of science literature to offer explanations of the various type of systems in which specific occurrences of the term "program" can be located. By

4 Of course, I am presenting the reality as if it were black and white while it is actually a world of shades of grey. I am following Richard Hamming here when he stated: "*We live in a world of shades of grey, but in order to argue, indeed even to think it is often necessary to dichotomize and say ``black'' or ``white''. Of course in doing so we do violence to the truth, but there seems to be no other way to proceed. I trust, therefore, that you will take many of my small distinctions in this light -- in a sense, I do not believe them myself, but there seems to be no other simple way of discussing the matter.*" (Hamming, *1969)*

5 See (Tedre, 2015) for a socio-historical study of the disciplinary formation of computing.

6 The use of this term here goes back to the late 60s when it was considered as one major issue to be faced in order to resolve the so-called software crisis (Buxton and Randell, 1969)

7 Note that the existence of such gaps is well-known and quite some effort has already been invested in tackling this issue. For instance, computer scientists like Peter Denning and Peter Wegner identified this diversity in computing very early on (1970s) and the more recent work by Denning is a systematic approach towards giving a coherent and systematic overview of the computing discipline that takes into account this diversity (Denning and Martell, 2015).

rendering transparant and systematizing the different meanings of "program" across the divides, it becomes possible to (re)-construct fundamentals of "programs", which open rather than restrict the meaning of program. This project thus requires **historical, philosophical and computer science competencies** and is interdisciplinary by nature. This is reflected in a diverse team which will consist of **25 regular researchers, one Postdoc, a PhD student and a part-time researcher (paid per hour).**

The three different modalities of "program" have a certain ordered relation with respect to one another: the formal modality (time-independent) stands in between the physical (temporal) and socio-technical modalities (temporal). It has developed out of the need to cross the gap between humans and machines and so it implies or should imply the two other modalities, implicitly or explicitly. Moreover, the formal modality is traditionally the locus of foundational research. **The aim of this project is to understand how the formal modality accommodates the two other modalities**, viz. how its scientific aspects function also as a technique which accounts for the properties and needs of the two other modalities.[8]

The focus will be on **models and the abstractions on which they are based.** The notions of "model" and "abstraction" are understood here in a non-restrictive manner in order to avoid making too many assumptions in advance on what a model is/should be and so to create a bias towards a specific understanding of "program". The only assumption made is that the **model assumes a certain understanding of "program"** by referring to one or more of its modalities. Or, to put it differently, as a model rooted in a particular practice, it contains the traces of how that practice understands "program". So for instance, Landin's SECD machine – which is a virtual machine – assumes a formal model of a "machine" on which a particular type of textual "programs" can be "run". Thus, it refers to the formal modality and makes assumptions of what a "program" should look like and also "models" aspects of the physical modality. Another, quite different, example is the basic model for deep learning, namely the restricted Boltzmann machine. These "machines" model a neural network which is then (usually) "implemented" in some programming language (e.g. Java) in order to, for instance, "discover" patterns in some dataset. Clearly, while two machine models, they each have quite different assumptions about what a program should do/be and what kind of conditions it should fulfill. Note also that "model" can for instance refer to a highly formal model of computation, like, for instance, lambda-calculus, but also to a more informal type of model as we encounter them for instance in debates on software patents.[9]

II. Methodology

As explained in I.a. this project aims to develop a systematic, pluralistic and historically-anchored analysis of "program" which integrates its three modalitie, that is, its physical, formal and socio-technical modality. As such, this project runs the **risk of being too broadly defined** and so it is important to apply an approach which is focused, structured and integrative *without* becoming too restrictive.

While the focus on models and their abstractions does provide the structural backbone for this project, a more systematic approach is required which guarantees (1) that *all* three modalities of "program" and their interconnections or lack thereof will be considered and (2) a structure which allows to integrate both the historical as well as the more philosophical or systematic viewpoint. To this end, the project is structured around four main clusters, each replicating one of the modalities or their connections. They follow the historical development from foundational research in the 20s and 30s in the context of mathematics and logic to problems of reliability and trustworthiness in safety-critical systems. These are:

Cluster 1: Logic. In almost any arbitrary textbook on computer science, one will find some version or other stating that the theoretical foundations of computer science can be retraced to work that was done in the 20s and 30s by a number of logicians/mathematicians like Church, Kleene and Turing. This foundational research, which originates in a machine-less context, is the traditional historical background to study models and abstraction techniques that were developed in a "purely" formal manner but were used later to handle aspects of "program" in its formal modality. But how have these models been used in actual programming contexts and are the "classic" sources

8 A recent text by the *Société Informatique de France*, "*Enseigner l'informatique de la maternelle à la terminale*, elaborates on computer science's double nature as a science and a technology.

9 See (Con Diaz, 2016)

the only set of mathematical and logical sources to consider when the aim is to understand "program"? What kind of assumptions do we make about "program" if we assume the Turing machine model is the most appropriate model to study, for instance, issues of algorithmic complexity or problems of infinite computation? By integrating this cluster into the project, it is possible to identify and study the different transformations that occur in models that were/have been/are developed in a purely formal context and are then used in a more concrete context of, for instance, modeling a machine, modeling a program or modeling the meaning of a program. Some obvious examples are the lambda-calculus (cf. for instance Landin's work on the lambda-calculus), Turing machines (for instance, as used in research on automatic programming and program portability in the late 50s) or constructive logic and type theory. One question of interest is then how these models evolved into systems of relevance today to model complex, safety-critical systems acting in the presence of multiple points of failure. And how the semantics of "program" has changed to accommodate contemporary translations like distributed computation, cyber-physical systems or computational systems under threat.

Cluster 2: Computing Machines. Computing machines and other calculatory devices have a very long history that goes back at least until ancient Greek culture[10] and it is clear that the first high-speed, programmable and discrete machines – computers for short – have their historical roots in the calculatory and partially mechanized practices to make and use mathematical tables. This project will not focus on this longer tradition, but starts with the first computers from the late 40s. The notion of program can, in fact, be traced back historically to (one of) the first computers, viz. ENIAC[11] where it originally referred to program in its physical modality given that it was literally wired on the machine. Thus, the historical roots of the physical modality of "program" should be situated in the context of these practices and it was during the development and use of the first high-speed, programmable and discrete machines that one sees the first computer models, with their associated notion of "program" (which might be termed differently), being developed. Initially, these models were quite "pure" in the sense that they are mostly about *directly* modeling the physical structure for running "programs" often in order to improve on that physical structure. Soon, however, this modeling becomes indirect and develops, for instance, into so-called virtual machine models which are used to "run" a "program" written in a certain higher-order notation and so models developed in cluster 2 might develop into models from clusters 1, 3 and 4. Clearly, models originating in this cluster can be of varying type and used for various purposes. Very early examples are the EDVAC model (which was inspired by the McCulloch and Pitts model for neurons) or the ACE model. Later examples are models developed and studied in the context of *automata studies* which are often a mix of models from cluster 1 and 2,[12] but also models for new machine designs like, for instance, van der Poel's models for the simplest possible computers that became the basis for an actual machine, the ZEBRA.[13]

Cluster 3: Program notations and languages. Whereas the notion "program" originated in the ENIAC design and hardware, the problem of "planning and coding" a problem for a computer became a problem in itself soon afterwards. Already in the late 40s one can identify attempts at developing a notational approach that makes the programming of the machine more "reliable" and "easy" for the human user. This issue became more prominent as the machine design and building moved from the universities to business and commerce. It was and is in this context of developing reliable and easy-to-use methods (where reliability and use can have changing meanings) for a broadening range of "users" that we can locate the *original* problem of bridging the gap between the socio-technical and physical modality through a set of notational devices which are nowadays known as programming languages and their abstractions. Models are used in this context for a variety of purposes. They can be syntactical or semantical; they can be used to reduce error; to improve "readibility" (from the socio-technical perspective, this is about developing methods to improve human readability, see for instance work on structured programming, or the development of literate programs); to design better languages; to improve efficiency; etc. Early examples are the implicit models in the flowchart notation as developed by von Neumann and Goldstine or the so-called algebra of program compositions as developed by Curry. Other examples are Landin's use of the lambda-calculus for modeling *all* programing languages; the use of Post production systems to model the syntax of

10 See for instance the analogue Antikythera mechanism.
11 See (Grier, 1996) and (Haigh, Priestley, Rope, 2016)
12 See e.g. Wang's model which mixes features of "real" computers with the Turing machine model (Wang, 1957)
13 See: (De Mol, Bullynck and Daylight, forthcoming)

programming languages (often referred to as the Backus-Naur form); dynamic logic which is conceived as a logic of programs; etc.

Cluster 4: Software Systems. In the late 50s and early 60s already the first so-called high-level programming languages were developed. In the meantime advances in machine design, the commercialization of (digital) computation and the growing needs of the "users" of the new machines resulted in the need for something other than just programming languages. One started to realize that "programming" was not just about writing one "small" program by one person but about a large set of programs that work on different levels but need to interact with one another; that such set of programs need to fulfill a set of purposes which are not always clear from the beginning: and that such set of programs are much more error-prone than just one small program. It is in this context that the so-called software crisis is traditionally located. Today, it is the problem of large and/or complex software systems like operating, safety-critical or concurrent systems which pose a real challenge for software engineers and are often the motivation for developing new approaches which assume different program models. Also here, models serve different purposes. Models can be developed or used to improve on the efficiency, security, reliability and/or use of the system. Also one can model different parts or aspects of a system to deal with a particular issues of the (software) engineering process, or model instead the entire system. Some typical examples are: models for concurrent systems (cf pi-calculus or Goldblatt's temporal logic); models for system designs (like those expressed in UML[14]); models used in the context of formal verification methods (for instance, those that underpin the Calculus of Constructions) and models used in the context of testing and debugging. In this context, implementation of formal models of trustworthiness, security and reliability reflect the translation of notions originating in Cluster 1 and realised in the physical (Cluster 2) and socio-technical (Cluster 3) clusters.

The four above clusters can be used both as a historical *and* as an analytical tool.

It is **a historical tool** because the four clusters structure the history of "program" into four stages in which every new stage *can* (but need not) rely on models from the previous stage, but introduces also a new set of problems that require the development of new models or the reshaping of previous ones, and so provides insight into how models (and their abstractions) develop and relate to one another and how they deal (or not) with a changing historical and technological context.

It is **an analytical tool** because this structure facilitates the comprehension through explanation and formal modelling of these "origins" which are separated in time, but reflect a continuous evolution up to today. Indeed, today we can still see that models are being developed, for instance, in a/the purely formal context but then used in one of the other contexts. One relatively recent example that comes to mind are Abstract State Machines. Originally developed by Gurevich to study the notion of algorithm which is richer and more intensional than the classical notion of computable function, they evolved into a software engineering method for formal verification. As a consequence we see that models can move not just from Cluster 1 to 4 but also move back from Cluster 4 to 1. Also, models can belong to different clusters depending on the perspective (e.g. use vs. modeled "reality") and so one can start to describe a **dynamics of models** and **scalability issues** between the different clusters. So, for instance, from the socio-technical perspective, one might consider the dynamics of models or lack thereof from issues related to the valorisation of theoretical research to industry. From a formal perspective, one might detect scalability issues between the use of finite-state machines as a model used in circuit design or their implicit use by Dijkstra and others to reason about program correctness.

Each of the clusters more or less corresponds to one of the three modalities. Cluster 1 refers to the formal and Cluster 2 to the physical modality. Cluster 3 then, with its focus on programming *languages and notations* originates in the need for developing a notation that works not just for the machine but also for the human and so focuses on the socio-technical modality. Cluster 4 then brings together all three modalities as a means to control computing systems.

Models will be studied as they originate and evolve within and between the different clusters. The main objective then becomes to offer **a historicized genealogy, taxonomy and dynamics of models**. The purpose is not to offer a structural or family tree-like viewpoint[15] on models but rather to provide an **in-depth diachronic and**

14 UML stands for the Unified Modeling language.

15 As has, for instance, already been done to some extent for programming languages. See: http://rigaux.org/language-study/diagram.html for several graphical genealogies.

chronological analysis of models from multiple perspectives. This means that for every model studied we will not just focus on its relation with respect to other models but also consider the **actual practices** underpinning the development of a given model as well as the **theoretical assumptions and limitations** for that model. In this way, this project allows to reflect and integrate also the practices in which the models are rooted and in which, if any, they have been applied and how this has affected the model. This permits to follow, as it were, the historical trace of a set of models (and the implicit understanding of "program" they subsume) as they move and are reshaped within or across different clusters. The results from this study will be materialized in **a searcheable information network**. This network will allow its users to look at the different models studied within the project from different perspectives. For instance, it will be possible to zoom-in on one particular model, and generate its genealogy or its dynamics or, using the categories developed in the taxonomy, it will be possible to explore classes of models and their genealogies (See II.b for more details).

It is not realistic to study *all* possible models within the framework of this project which means there will certainly be gaps. This is not a problem *per se* as long as the most important models are studied and so identified. These models will be identified during a preparatory phase of this project (see Task B.1.1).

III. Originality, significance and state of the art

The aim of this project is to develop a coherent analysis of "program" which integrates the three program modalities, relying on historical and philosophical methods. As a consequence, this project engages with research in the history of computing, the philosophy of computing and (foundations of) computer science. For an extended bibliography, see the Appendix.

With respect to history of computing The history of the history of computing goes back to the late 70s and was, initially, an initiative coming from a number of computer scientists and programmers who felt the need for a historical record. Today, much of professional history of computing has moved to the history departments. The *Special Interest Group Computers, Information and Society* (SIGCIS) is probably the most important international community. It organizes its annual workshop during annual meeting of The Society for the History of Technology and so has a particular methodological focus on history of technology (rather than history of science). By consequence, the focus today is more on the sociological, institutional and industrial history than on the actual practices that underpin these histories (see (Campbell-Kelly, 2007). The disadvantage of this development is that, currently, historical work on computer science is scarce and research which dives into the formal, engineering and other details of actual computing practices is usually considered to be too internalistic and so uninteresting. In the last few years, some researchers have re-engaged with the technical histories underpinning (computational) practices, and, more broadly, the shaping of the computing discipline.[16] The current project fits well with these more recent developments but without ignoring the social and institutional dimension.

On a national level, the series of conferences *Colloques l'Histoire de l'Informatique, des Réseaux et des Télécommunications* (organized between 1988 and 2004) has been a very important initiative bringing together historians and (mostly) computing professionals. Since 2013, there is also the monthly research séminaire *Histoire de l'informatique et du numérique* organized at CNAM in the framework of the project *Vers un musée de l'informatique* (towards a computer science museum). The efforts and works by *P.-E. Mounier-Kuhn* (CNRS, Centre Koyré) on the French history of computer science have laid the basis for research in history of computing in France. At CNAM, there are several researchers, like *Camille Paloque-Berges* (HT2S-Cnam) and *Loïc Petitgirard* (HT2S-Cnam), working on the history of computing and the recent collaborative project *La genèse d'un laboratoire de recherche en informatique (1968-1988)* is an indication of the increased interest in France for the history of computer science. Also in France, the work by *Maarten Bullynck* (Université de Paris 8/Sphère, Université Paris Diderot), *Marie-José Durand-Richard* (Sphère, Université Paris Diderot) and the PI

16 Examples are (Mahoney, 2011) (Priestley, 2011) (De Mol, Carlé and Bullynck, 2014) (Haigh, Priestley, Rope, 2016) (Mounier-Kuhn, 2015)

of this project focuses more on the mathematical, logical and engineering practices underpinning certain developments.[17]

With respect to philosophy of computing The philosophy of computing is still a young, underdeveloped field. Only in the last 10 years or so, a serious discussion about the ontology and epistemology of computing has surfaced at the intersection of logic, computer science, philosophy. On an international level, the philosophy of computer science was for a large part shaped by *Raymond Turner*. He was the organizer of the track on the Philosophy of Computer Science for the Annual IACAP[18] conference.[19] Moreover, Turner wrote the important entry on *Philosophy of Computer Science* (Turner, Angius, 2017) for the Stanford Encyclopedia of Philosophy which basically defined the field. In his work, Turner develops a broader notion of program wich takes into account the semantics, implementation and specification of programs (Turner, 2014). On the national level, *Jean-Baptiste Joinet* (Université Jean Moulin Lyon 3) organized the first séminaire in France devoted to the philosophy of computer science titled *Philosophie de la logique, de l'informatique et de leurs interfaces* and several of his students (PhD and Master)[20] have focused on the philosophy of computer science. More recently, Baptiste Mélès (CNRS, Archives Henri Poincaré) has written extensivey about the philosophy of computer science. He is also one of the organizers of the séminaire *Codes Sources* (co-organized with R. Fournier-S'niehotta and L. Tabourier) which aims at a deep reflection on computer science based on a study of source codes. Finally, we mention the séminaire *Histoire et Philosophie de l'Informatique* (L. De Mol, A. Naibo, M. Pegny, Sh. Rahman).

With respect to computer science Within computer science and the computing sciences at large, there are some initiatives that call for a deeper, interdisciplinary and, possibly, historically-achored reflection on the field. The efforts from Peter Wegner and later Peter Denning to develop the computing field as an interdisciplinary field that integrates both mathematical, engineering and scientific aspects should certainly be mentioned here (Wegner, 1976; Denning et al, 1989; Denning and Martell 2015). The Dagstuhl séminaire on the *History of Software Engineering* (Brennecke and Keil-Slawik, 1996) was intended specifically at bringing together historians and software engineers to reflect on the foundations of software engineering. More recently, there was also the publication of Matti Tedre's book *The science of computing. Shaping a discipline* which is the first systematic study of the disciplinary formation of the computing field from an interdisciplinary perspective and sketches the development of computing around three fundamental debates showing, in this way, at least three different aspects of computing (its scientific, mathematical and engineering aspects). In recent years, one can see an increased interest in more historical and reflective work on computing. For instance, *Communications of the ACM* has published several papers from professional historians in the last few years and since a couple of months there is a regular *ACM blog* from Robin Hill on the philosophy of computing.

The current proposal is the first research project which offers a broad, deep and systematic analysis of one of the core topics of computing, namely the notion of "program", by bringing together these different research communities and their respective methods. The backbone of the current analysis is represented by events and publications organised by the HaPoC Commission like the *Symposia on History and Philosophy of Programming,* with the last edition in 2016 specifically devoted to Operating System; a *Symposium on Proofs, programs, procedures: formal and epistemic issues* organised in 2013; a session at IACAP2014, titled *What are programs, algorithms, machines and how do we understand their languages?*

The project intends to bring these initial steps into a mature academic and scientific analysis much needed for the field.

---

17 See for instance: (De Mol, Carlé and Bullynck, 2014) and (De Mol and Durand-Richard, forthcoming)

18 IACAP stands for International Association for Computing and Philosophy.

19 Until 2012 when the format of the conference was reshaped.

20 Maël Pégny did his Ph.D. on the so-called physical Church-Turing thesis; Wendy Hammache, who is finishing her Ph.D. works on the notion of type and its history within logic and computer science; Yannis Hausberg, a master student is preparing a mémoire for Master 1 with the provisional title *Processus computationnels : le calculable et le prédictible.*