About Computing Science Research Methodology

penned by José Nelson Amaral with significant contributions from Michael Buro, Renee Elio, Jim Hoover, Ioanis Nikolaidis, Mohammad Salavatipour, Lorna Stewart, and Ken Wong

Computing Science researchers use several methodologies to tackle questions within the discipline. This discussion starts by listing several of these methodologies. The idea is not to classify researchers or projects in each of these methodologies or to be exhaustive. Tasks performed by a single researcher fall within different methodologies. Even the activities required to tackle a single research question may include several of these methodologies.

1 Methodologies

The following list of methodologies is intended to organize the discussion of the approach required by each of them.

Formal In Computing Science, formal methodologies are mostly used to prove facts about algorithms and system. Researchers may be interested on the formal specification of a software component in order to allow the automatic verification of an implementation of that component. Alternatively, researchers may be interested on the time or space complexity of an algorithm, or on the correctness or the quality of the solutions generated by the algorithm.

Experimental Experimental methodologies are broadly used in CS to evaluate new solutions for problems. Experimental evaluation is often divided into two phases. In an exploratory phase the researcher is taking measurements that will help identify what are the questions that should be asked about the system under evaluation. Then an evaluation phase will attempt to answer these questions. A well-designed experiment will start with a list of the questions that the experiment is expected to answer.

Build A "build" research methodology consists of building an artifact — either a physical artifact or a software system — to demonstrate that it is possible. To be considered research, the construction of the artifact must be new or it must include new features that have not been demonstrated before in other artifacts.

Process A process methodology is used to understand the processes used to accomplish tasks in Computing Science. This methodology is mostly used in the areas of Software Engineering and Man-Machine Interface which deal with the way humans build and use computer systems. The study of processes may also be used to understand cognition in the field of Artificial Intelligence.

Model The model methodology is centered on defining an abstract model for a real system. This model will be much less complex than the system that it models, and therefore will allow the researcher to better understand the system and to use the model to perform experiments that could not be performed in the system itself because of cost or accessibility. The model methodology is often used in combination with the other four methodologies. Experiments based on a model are called simulations. When a formal description of the model is created to verify the functionality or correctness of a system, the task is called model checking.

In the rest of this document we will attempt to provide advice about each of these methodologies. But first, lets consider some general advice for research. New and insightful ideas do not just happen on an empty and idle mind. Insightful research stems from our interaction with other people that have similar interests. Thus it is essential for all researchers to be very involved in their communities, to attend seminars, participate in discussions, and, most importantly, to read widely in the discipline.

It is also important to keep a record of our progress and to make notes of the ideas that we have along the way. The brain has an uncanny nag to come back to ideas that we have considered in the past — and often we do not remember what were the issues that led us not to pursue that idea at that time. Thus a good record keeping system — a personal blog, a notebook, a file in your home directory — is a very important research tool. In this log we should make notes of the papers that we read, the discussions that we had, the ideas that we had, and the approaches that we tried.

Once a student starts having regular meetings with a thesis supervisor, it is a good idea to write a summary of each of these meetings. The student may send the summaries to the supervisor or may keep it to herself. After several months, revisiting these meeting logs will help review the progress and reassess if she is on track with her plans towards graduation.

In research, as in other areas of human activities, an strategy to get things done is as important as great visions and insightful ideas. Whenever working with others, it is important to define intermediate milestones, to establish clear ways to measure progress at such milestones and to have clear deadlines for each of them. Collaborators will be multitasking and will dedicate time to the tasks that have a clear deadline.

1.1 Formal Methodology

A formal methodology is most frequently used in theoretical Computing Science. Johnson states that Theoretical Computer Science (TCS) is the science that supports the field of computing [1]. TCS is formal and mathematical and it is mostly concerned with modeling and abstraction. The idea is to abstract away less important details and obtain a model that captures the essence of the problem under study. This approach allows for general results that are adaptable as underlying technologies and application changes, and that also provides unification and linkage between seemingly disparate areas and disciplines. TCS concerns itself with possibilities and fundamental limitations. Researchers in TCS develop mathematical techniques to address questions such as the following. Given a problem, how hard is it to solve? Given a computational model, what are its limitations? Given a formalism, what can it express?

TCS is not only concerned with what is doable today but also with what will be possible in the future with new architectures, faster machines, and future problems. For instance Church and Turing gave formalisms for computation before general-purpose computers were built.

TCS researchers work on the discovery of more efficient algorithms in many areas including combinatorial problems, computational geometry, cryptography, parallel and distributed computing. They also answer fundamental questions about computability and complexity. They have developed a comprehensive theoretical frame to organize problems into complexity classes, to establish lower bounds for time and space complexity for algorithms, and to investigate the limits of computation.

The best practical advice for new researchers in the area of formal research methods is to practice solving problems and to pay attention to detail. The general advice for researchers in computing science, know the literature, communicate with colleagues in the area, ask questions, think, applies to formal method research as well. Problem solving can be risky but also very rewarding. Even if you don't solve your original problem, partial results can lead to new and interesting directions.

The skills and the background knowledge that formal method researchers find useful include: problem-solving, mathematical proof techniques, algorithm design and analysis, complexity theory, and computer programming.

1.2 Experimental Methodology

The Computing Science literature is littered with experimental papers that are irrelevant even before they are published because of the careless fashion in which the experiments were conducted and reported. Often the authors themselves could not reproduce the experiments only a few weeks after they rushed to obtain the experimental results for a mid-night conference deadline. Here is some general advise to help preventing you from producing worthless experimental papers.

1.2.1 Record Keeping

Good record keeping is very important in experimental work. Computing-Science researcher's record keeping tend to be surprisingly lax. Because experiments are run on computers, inexperienced researchers have a tendency to think that they can rerun the experiments later if they need to. Thus they tend to not be as careful as they should be about labeling and filing the results in ways that will make it possible to retrieve and check them later. Sometimes it is even difficult for a researcher to reproduce his own experiments because he does not remember in which machine it was run, or which compiler was used, or which flags were on, etc.

The reality is that computers are very transient objects. The computer that is in the lab today is likely to not be available in this configuration in just a couple of months.

Thus experimental computing science would greatly benefit if each experimental computing scientist would treat her experiment with the same care that a biologist treats a slow-growing colony of bacteria. Annotating, filing, and documenting are essential for the future relevance of an experimental scientist's work.

1.2.2 Experimental Setup Design

Speed is an important factor during the exploratory phase of an experimental work. Thus this phase usually proceeds with less care than it should. Once this exploratory phase is over, a researcher should stop and document the findings and carefully describe the experimental setup, as well as the characteristics of the hardware and software that will be used for the evaluation phase. What are the questions that the experimental work is expected to answer? What are the variables that will be controlled? What variables may affect the results of the experiment but are not under the control of the researcher? What measures will be taken to account for the variance due to these variables — will the results be statistically significant? Is the experiment documented in a fashion that would allow other researchers to reproduce it?

1.2.3 Reporting Experimental Results

When reporting the results of an experimental evaluation, it is important to state clearly and succinctly, in plain English, what was learned from the experiments. Numbers included in a paper or written into a report should be there to provide an answer or make a point. Graphical or table representations should be carefully selected to underscore the points that the researcher is making. They should not mislead or distort the data. Analyzing data based only on aggregation is very dangerous because averages can be very misleading. Thus, even if the raw data is not presented in a paper, the author should examine the raw data carefully to gain further insight into the results of the experiments. The numerical results presented in tables and graphs should be accompanied with a carefully written analytical discussion of the results. This discussion should not simply repeat in words the results that are already shown in the tables and graphs. This discussion should provide insight on those results, it should add knowledge that the researcher gained and that is not in those numbers. Alternatively this discussion should attempt to explain the results presented.

1.3 Build Methodology

Whenever a research question leads to the building of a software system, the researchers involved should consider the following set of good practices:

Design the software system . No matter how simple the system is, do not allow it to evolve from small pieces without a plan. Think before you build. Most importantly, consider a modular approach - it simplifies testing. Testing is also simplified by choosing text-based data and communication formats. Defining small interfaces increases flexibility and reuse potential.

Reuse components . Are some needed software components already (freely) available? If yes, using such components can save time. When deciding which components to reuse consider the terms of use attached with them. The components that you reuse in the system can have implications on the software license under which the new system can be distributed. For instance, if a component distributed under the GNU Public License (GPL) is used in a software system, the entire system will have to be distributed under GPL.

Choose an adequate programming language . Often researchers want to use a programming language that they already know to minimize the time invested on learning a new language. However it may pay off to learn new languages that are more adequate for the building of an specific system. Important factors to consider when selecting a programming language include: required run-time speed (compiled vs. interpreted languages), expressiveness (imperative vs. functional vs. declarative languages), reliability (e.g. run-time checks, garbage collection), and available libraries.

Consider testing all the time . Don't wait to test the entire system after it is built. Test modules first. Keep a set of input/output pairs around for testing. This way future changes can be tested when they are introduced. Consider building an automated testing infrastructure that compares the program's output on a set of input data with correct outputs and also measures run time. Set this automated testing infrastructure to run automatically daily/weekly to notify the builders about troublesome changes immediately.

Documentation is crucial in any software system. Good computer programs must be well documented. Supervisors, outside users, and fellow students who may extend the system in the future need to be able to understand the code without much trouble. Even when there is a single developer there are advantages to use of a version control system, such as Concurrent Versions System (CVS). CVS gives the developer, and anyone that needs casual access to the code and documentation, easy access to the set of current files from multiple locations. Moreover, CVS allows access to previous versions in case of changes that introduce bugs.

Once the software system is functional, researchers should compare its functionality and/or performance with that of existing systems to verify that the claim(s) that they want to make about the system still hold. Often, the runtime/space requirements dependent on input size are reported on commonly used test sets. Architecture-independent measures, such as the number of nodes visited on a graph per unit of time, should be reported based on wall-clock time or actual memory consumption to simplify comparison with other systems. Results should be reported using statistics - such as percentiles (e.g. quartiles min 25% median 75% max) - that don't depend on unjustified distribution assumptions.

1.4 Process Methodology

Process methodologies are most useful in the study of activities that involve humans. Examples of such activities in Computing Science include the design and construction of software systems — large or small, the design and evaluation of human-computer interactions, and the understanding of cognitive processes. More recently the creation of interactive games has been studied extensively. This activities often involve studies with human subjects.

1.4.1 Software process

The construction and evolution of a large-scale software system is challenging for many reasons, including complexity, large teams, long lifetimes, changing needs, quality trade-offs, and emerging technology. One aim of software engineering research is to devise methods or processes to put more discipline into how such systems are developed and ultimately raise their quality. We can study how systems are currently put together or managed, and discover proven designs, repeated patterns, or recurring strategies. We might discover strong trends that could be useful predictive indicators about some system property, like defect density. Alternatively, we can step back and abstract the system to better reason about some aspect, like resource usage, and devise a prescriptive software process to prevent certain classes of defects from occurring in the first place.

Such generalizations can be codified in various procedural ways — such as best-practice guides, pattern languages, application frameworks, process models, and development tools — to be applied by the software engineering practitioner. Indeed, many scientific techniques when applied to software can provide useful fodder for a process or tool to be applied in reality. Examples of techniques include the development of models of software structure and behavior; experiments of a system with users and developers; and the development of a proof-of-concept or useful component. Experience with these techniques can lead to processes for model-driven development, methods to enable improved collaboration, and tool-based assistance for both. Processes and tools themselves may evolve together as we better understand the context where they are situated.

There are still difficult social, cognitive, and technical barriers to the adoption of a new process or tool in an existing setting. A tool that either is conceptually complex, forces a heavy-handed approach, requires significant training, or ignores legacy issues could be doomed to failure. In-the-field case studies may be needed to convincingly demonstrate real impact. In scoping out a research problem in software engineering, one needs to be aware of issues beyond the technical ones, without being too bogged down in non-essentials. Important is having clear assumptions, a well-stated, verifiable research question, and a suitably designed evaluation. The definition of a research problems resembles the definition of a software module: it should have a clear interface, a well-defined function, and meaningful tests.

1.4.2 Methodological Issues

In designing a study to evaluate Human-Computer Interfaces (HCI) or to assess how a software engineering tool or methodology, it is important to appreciate that there is a large body of literature on effective ways to design surveys and to conduct structured interviews. Such literature should be consulted even if the survey is exploratory in nature and not aimed at hypothesis testing. For instance, psychologists describe "anchor effects" that may impact how a person chooses a rating on a rating scale to make a preference judgment. The psychometric literature should be explored before designing a study that employs surveys or interviews. Natural sources of information range from the web to meeting with someone from the Psychology Department for advice, especially if the data produced by the survey is a crucial part of the research. The University of Alberta has a Population Research Lab that offers consultation on the design of reliable surveys. It may be useful to contact someone in that centre (there may be a fee, so discuss this with your supervisor).

Once results are collected from a properly design survey or study, a researcher should ensure that appropriate statistical techniques are used for the analysis of these results.

1.4.3 Cognitive Modeling

Another area of Computing Science that centers around processes and human subjects is the study of cognition. Cognitive Scientists develop computer models of hypothesized cognitive processes, just as physicists develop computer models of hypothesized physics processes, or meteorological scientists develop computer models of hypothesized atmospheric processes. In all cases, there are observed empirical data, and the game is to develop a process model that accounts for the observed data, i.e., a set of proposed processes that operate in particular ways to produce the data of interest. Interesting and important models address a general issue that may lead to the understanding of underlying mechanisms that produce the existing data.

For instance, consider a data set containing some observed memory results such as accuracy and reaction time. It is possible to write a computer program that produces the same results given some particular input, and declare it to be a model. However, is it an interesting model? Does this model inform the community in some more general way? Are the observations obtained with this model extendable to other results (in principle), that embodies some general assumptions that themselves are testable. Otherwise, it is a one-off computer program whose existence does not really contribute to any larger story. This is true whether one models using a neural network paradigm or a higher level architecture with various assumptions about how control, memory, and learning processes interact.

The principle that there must be a bigger story behind an implementation applies broadly to Computing Science enterprises. A researcher that decides to create a program that does X should be all along concerned with the research questions that this program will answer or will raise.

Researchers develop computer models to make explicit all the assumptions that constitute a given explanation. To say that information is retrieved "on the basis of similarity" requires one to offer some computational account of "similarity". A researcher must describe how it is computed, and what other factors might affect that computation. To say that certain kinds of information are "preferred" over other sorts of information may be appropriate for a descriptive account of observed results. However, a computational model of the processes by which preference is computed requires making explicit the factors, which are often context-dependent, that determine preference. The translation of assumptions into a particular computational process typically leads to predictions that themselves are verifiable in other experiments. In fact, a computational model that does not — as a by product — leads to new testable hypotheses is not very interesting or useful to the broader enterprise of understanding a complex system.

So how does one get around writing a one-off computer program to model some particular set of results? First, decide what kind of computational paradigm to operate in. This is a personal choice. You can operate with a neural net paradigm, you can operate within a higher-level architecture paradigm. Adopt a paradigm and situate your work within the community that uses that paradigm to increase the likelihood that the new research results produced will actually contribute to some larger understanding about an issue.

1.5 Model Methodology

Modeling is the purposeful abstraction of a real or a planned system with the objective of reducing it to a limited, but representative, set of components and interactions that allow the qualitative and quantitative description of its properties. Strictly speaking, modeling is a methodological aspect of science. Modeling is not the object of the research, it is part of an arsenal of instruments used by researchers to study and understand the research's object.

The action of building a model, called *modeling*, is driven by the study that will use the model. Hence, there is no single modeling approach applicable to all systems. Rather, scientists build models that capture important aspects of a system and gloss over — either completely ignore or just approximate — the aspects that have lesser (or no) impact to their intended study. The decision of which aspects are important and which ones have lesser impact is itself part of the modeling strategy. Misleading outcomes are produced by models that eliminate what is important or that over-emphasize what is of lesser impact. Extensive arguments in the communities that depend on modeling center on decisions about the actual model used. Modeling should be seen as an evolving process that is coupled to a particular sub-discipline.

For instance, the workload — e.g. the form and pattern of requests — to a web server is usually modeled using particular distributions. These distributions are used to express time between request arrivals, the sizes of documents retrieved, the probabilistic relationship between successive item requests, and other such features. As the technologies around the World-Wide Web evolve, so does the corresponding traffic. Thus the issue what is a good model to express the traffic to web servers is by definition a moving target. The corresponding community of researchers needs to be in continuous dialog about the models used and needs to update them as the studied phenomena evolve. In some disciplines a good model is one that anticipate changes by incorporating degrees of freedom that allow it to be applied even when some aspects of the problem change over time.

Models can be expressed in diverse ways. A model can be described by text and block diagrams to specify how a system is constructed from a collection of interacting components. In Computing Science, a model is often described by a formal language designed specifically to describe particular kinds of models, or it is embedded in a computer program that simulates the behavior of a system. Formal models are usually geared toward specific qualitative and quantitative aspects of a system, while simulation is open ended. Simulation often lacks the power to make definite statements about properties of the system. For instance,the results of simulations may not be used to prove that a deadlock never develops in a concurrent system.

The best approach for new researchers in areas where models are required is to study publications that disclose the full details of the model used, or that describe models for similar systems. Unfortunately, often the presentation of models is abbreviated in papers due to page restrictions. Authors often assume that the reader is aware of usual modeling assumptions in a particular domain. A careful researcher that sees no details about the model used in a simulation based study will keep in mind that certain properties (qualitative or quantitative) of the study might be directly influenced by sloppy modeling. Such researcher will have a keen eye (developed through experience) to spot discrepancies that go hand-in-hand with sloppy modeling. It is also a good idea to reflect on, spot shortcomings, and to report them in the literature when models are found inadequate or incorrectly assume to "do their job" when they don't. Proposing a new (better) model is also a very important contribution to the community.

References

[1] D. S. Johnson. Challenges for TCS (NSF 2000). http://www.research.att.com/∼dsj/nsflist.html.