New Initiative: The Naturalness of Software

Premkumar Devanbu
Department of Computer Science
UC Davis, Davis, CA, 95616
Email: ptdevanbu@ucdavis.edu

***Abstract*—This paper describes a new research consortium, studying the *Naturalness of Software*. This initiative is supported by a pair of grants by the US National Science Foundation, totaling $2,600,000: the first, exploratory ("EAGER") grant of $600,000 helped kickstart an inter-disciplinary effort, and demonstrate feasibility; a follow-on full grant of $2,000,000 was recently awarded. The initiative is led by the author, who is at UC Davis, and includes investigators from Iowa State University and Carnegie-Mellon University (Language Technologies Institute).**

I. Introduction

Of all that we do, *Communicating* is the one action that is arguably the most essentially human. Honed by millions of years of cultural and biological evolution, language and communication are an ordinary, even instinctive (See Pinker [19]) part of everyday life. I do indeed labour mightily as I write this, to create lucid prose for an august and competitive venue; but in every day use, however, I speak and write quite spontaneously and freely. Most of what I say, thus, is simple, repetitive, and effortless.

This quotidian aspect of natural, human linguistic behavior, together with large on-line corpora of utterances, modern computing resources, and statistical innovations, has led to a revolution in natural-language processing, whose fruits we enjoy everyday in the form of speech recognizition in mobile devices, and automated language translation in the cloud.

But then, there is this funny thing about programming: it is primarily, (even if sometimes unintentionally) *an act of communication*, from one human to another. Knuth said as much, 30 years ago:

> *Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do... [11]*

If one, then, were to view programming as an act of communication, is it driven by the "language instinct"? Do we program as we speak? Is our code largely simple, repetitive, and predictable? *Is code natural?*

Surely, not all code is. Just turn to any sample code in Knuth's own *Art of Computer Programming*, and see just how simple and repetitive it is! However, *that* code, especially, is carefully wrought by an expert hand to stand forth as a paradigm. Assuredly, most workaday programmers don't have the time (or even, perhaps, the talent) to routinely craft such masterpieces.

So, then, how much software is *natural*, viz., simple, repetitive and effortlessly written? If there is a lot "quotidian" code that is, indeed, simple and repetitive, can we use statistical methods from natural language processing to assist programmers? Can statistical methods help understand & characterize programmer behaviour? Can these methods be used to build models to help beginners avoid & correct common mistakes? Can algorithms automatically translate between Java and C#, as we do from English to French?

These questions are representative, to our knowledge, of an entirely new direction of research in software engineering: the study and exploitation of the naturalness of software, using large, on-line corpora of open-source and other code.

We have recently (and with much gratitude) received a total of $2,600,000 from the National Science Foundation (USA), in the form of two successive grants (an EAGER and an NSF SHF LARGE) to pursue this field. The work is led by the author, who is with the software engineering group at UC Davis (Vladimir Filkov and Zhendong Su are Co-PIs at Davis), with collaborators from Carnegie-Mellon University (William Cohen and Roni Rosenfeld at the Language Technologies Institute) and Iowa State (Tien Nguyen, who leads the Software Engineering group).

II. Project History

Our research began around 2010. We were intrigued by a paper by Gabel & Su that analyzed the "uniqueness" of code [7] in a corpus of about 500,000,000 SLOC, which found that very few fragments of code in this very large corpus were actually unique, *viz.,* never repeated. This made us wonder: if code fragments are not unique, how often do they repeat? Do they repeat in predictable ways? If so, how might be we predict such repetitions? Would it be useful to do so?

We gathered a large body of source code, in C and Java, and began to evaluate the statistics of lexical phenomena in this corpus. We used n-gram models in our work. N-gram models assign a probability to the next token based on the previous n-1 tokens. These models have a long history in natural language processing. Modern n-gram models have sophisticated ways of dealing with issues like data sparsity: *e.g.,* some words or patterns never occur during model estimation, but are then encountered in practice. We used a fairly standard version of n-gram models, over our large corpus, and attempted to determine the predictive accuracy of these models in a 10-fold cross-validation setting. Typically the measure used here is *cross-entropy*, which evaluates the "surprisingness of the code"; it's the average negative log of the probability assigned by the model to each token in a test corpus. It's measured in bits, and can be viewed as a measure of the average *information content* of each token. Typically, English comes in between 7 and 8 bits over large corpora using modern n-gram models.

To our surprise, we found that source code was not only repetitive, and predictable; it appeared to be much *more so* than natural language, coming in at around 3-4 bits. We remind the reader that this measure log-scaled, and thus, code is roughly 8-32 times *more predictable* than English! This encouraging finding led us to seek applications.

The first one we tried was admittedly a bit daring: we thought that perhaps we could improve on the sophisticated token-suggestion engine in Eclipse by enhancing it with an n-gram model. We found that in fact, n-gram models did improve the native engine, using a fairly simple regimen of blending its suggestions from higher-probability ones offered by the n-gram model. In retrospect, it's reasonable to expect that these two complement each other : Eclipse suggests tokens that *are allowed to appear* in a given context; the n-gram model, by comparison, suggests tokens *that have most frequently appeared* in similar contexts. This paper was published in ICSE 2012 [8], and to our knowledge was the first to explicitly observe the "naturalness of software", and lay out a research vision of work that exploited and studied this phenomenon.

In the remaining sections, we present the main thrusts of this major new research initiative, including results to date. To our knowledge, there are active groups (in addition to our immediate collaborators at Iowa State & CMU) at University of Edinburgh, ETH (Zurich), TU Delft, University of Wisconsin, Utah State, and University College London, all pursuing various aspects of this research, and we've provided citations to their contributions as available; however, in a fairly limited fashion, since this is *not intended to be a survey paper*.

III. Research Thrusts

We summarize the various activities currently underway in this area (including ones we are engaged in) under these sections below.

A. Statistical Models of Code

It's been well-established in the area of natural language & speech processing that better statistical models (where improvements are measured in reduced entropy) lead to better performance: reduced error rates, faster results, *etc* [13], [21].

While off-the-shelf n-gram models performed quite well in our early experiments in 2012, it was clear from the outset that further gains could be achieved by tuning models to the special properties of source code. Nguyen *et al.* introduced a model enriched with *sememes* which included information about types, as well as roles (method, identifier, *etc.*); it also included a topic model to capture locally relevant vocabulary and usage. This model performed better than the base n-gram models [18]. This result suggests that there is extra regularity in code that can be captured using types. More recently, we have developed a model that exploits the high-levels of local specificity and endemicity of vocabulary and usage in software, which largely arises from modularization, and the extreme locality of the scope of identifier names. This in a way addressed the problem noted in [2], that identifiers tend to be hard to predict. By continually updating the language model with a strong "local" component based on recently seen patterns, our model exploited the "localness" of software to significantly enhance model performance [23].

While models have been at the lexical level, there have been some recent efforts to model the syntactic [14] phenomena in large corpora using deep neural networks.

B. Porting & Translation

Automated statistical translation has been one of the success stories of statistical natural language processing. These tools are based on models trained using *aligned corpora* where parallel texts in two different languages are available. The question naturally arises, can we successfully train a translation engine to translate between, say Java and C#, using parallel implementations in the two languages? Nguyen *et al.* [16], [17], built a lexical translation engine that translated between Java and C#. It used training corpora based on several projects that had parallel Java and C# implementations, with method-level alignments. A notable synergistic finding in this project was the ability of statistical methods to detect API mappings across different languages and platforms: using aligned code, it is possible to learn likely mappings of API calls across platforms [15]. More recent work has moved beyond lexeme-based translation to learning phrase-based translation [10].

C. Studying the "Naural Linguistics" of Code

The discipline of linguistics (*c.f.* Human Languages) has (*inter alia*) two notable aspects: first, the study of language itself, in terms of it's properties, and the second, the study of other related subjects (*e.g.* psychology, sociology, anthropology) as they are manifest in linguistic behavior. If we were to take the view of programming as a form of human linguistic behavior, one could a) study the way large programs are written, and evaluate how different language features are used, and why. Allamanis & Sutton [2] were among the earliest to examine code corpora using language models, and studied the relative predictability of different modules and different token types in code; they found that utility modules are distinguishable, and also showed that identifiers are the hardest to predict.

Subsequent work by these authors have also shown that there project-specific linguistic "culture" in source code that can be distinguished by language models, and used this finding to develop a tool that can learn and enforce coding standards [1]. This finding suggests that code has aspects of "linguistic culture". We have recently been separately funded by the National Science Foundation[1] to pursue the second line of work, *viz.,* the social and human aspects of code linguistics; specifically, we aim to identify whether project members have

1. http://goo.gl/vYEeda

coding styles that "signal" to others that they are in-group rather than out-group, similar to the way in which speech patterns behavior can indicate whether a person belongs to a group.

D. Suggestions & Completions

Integrated development environments like Eclipse[2] and IntelliJ Idea[3] have long offered *code suggestions and completions* (S&C). Completions essentially are applicable ways of completing a token fragment, and suggestions are typically complete tokens applicable in the given context. One of our central goals is to enhance S&C functionality using various levels of language modeling.

We have developed an Eclipse plug-in that uses our locality-enhanced cache-based language model [23] to provide suggestions. A companion paper in the demonstrations track at ICSE 2015 describes this tool [6]. This tool provides suggestions at the lexical level. At this level, offering suggestions is a fairly straightforward matter of presenting a menu of possibilities. As models are developed to capture regularities and repetitions at the syntactic level, it remains an open question how such suggestions could be presented.

E. Analysis & Tools

The predictability of code suggests semantic properties of code are also predictable. In the extended version of our ICSE 2012 paper [9] we hypothesized that:

> semantic properties (of code) are *usually* manifest in *superficial* ways that are computationally cheap to detect, particularly when compared to the cost (or even infeasibility) of determining these properties by sound (or complete) static analysis.

We described (§VI.D) the possibility of estimating a statistical model over a corpus of code, well-annotated with static analysis products; and then using this translation approach to provide maximum a-posteriori (MAP) probability guesses as to likely semantic properties, given easily detected "surface" features. A recent paper has realized this approach, using Conditional Random Fields, for the task of guessing reasonable names and likely type annotations in Javascript programs [20].

F. Assistive Technologies

A common theme in assistive technologies, which enable users with motion impairments to use computer devices, is the need to get useful signals from constrained movements (such as limited hand movements with just one or two degrees of freedom) or from noisy signals (*e.g.* speech, or jittery movements). The user's intended input can be considered a sample drawn from a distribution conditioned on the given noisy input, and the task of the assistive device is to identify the intended input given a noisy or ambiguous signal from the user. This is essentially the celebrated "noisy channel" model of Shannon, with the attendant Bayesian formulation; in such settings a good predictive language model can provide useful likelihood estimates to help disambiguate possible transcriptions.

Dasher [24] is a device for letter input, based on a predictive hexagram model of letters in English. It works remarkably well, since the hexagram-based entropy of letters in English is quite low, in the order of a few bits. The user chooses the next letter from a streaming menu of letters that fly at him/her, based on previous six letters input. Dasher has been demonstrated to be quite effective, allowing quite rapid input of English text using limited motion in 2 dimensions. In English, token entropy is too high to be used in this setting, and the user has to choose from a streaming menu of letters, rather than words. In code, however, the token entropy is low enough that it is possible to actually choose tokens. We built a demonstration version of this approach, which can be viewed online[4].

More recently, we have learned about an NSF-funded project using corpus statistics of software corpora to build assistive (speech-recognition) systems for code input[5]

G. Corpus Curation

One of central animating themes of the Software Naturalness project is to use statistical NLP methods, estimated over very large code bases, to assist software developers. Our work thus follows in a long line of research that attempts to bring "big data" to software engineering (which some of late have taken to branding as "Big Code"). Other large software datasets of and about open-source software have been made available over the years, including the PROMISE repository[6], the FLOSSMole[7], Git Torrent[8] and so on.

Our goal is to create a very large, multi-lingual corpus of source code and derived artifacts, specifically for statistical modeling efforts. The NLP community has a lot of experience with this: the Brown [5] and Gutenberg[9] corpora are widely used. These corpora have been used extensively in the study of language models and refinements [4]. In addition, the Penn Tree Bank [12] provides a large corpus of English sentences *parsed*, and annotated to elucidate grammatical structure. This corpus has been an invaluable dataset for the development and evaluation of syntax-based statistical models of natural language. Furthermore, for work on language translation, the development of *aligned* datasets, which align sentences in different languages is very useful. Examples include Europarl[10] drawn from European Parliament proceedings, and the analogous Canadian Hansard[11]. These aligned corpora are invaluable data for estimating conditional distributions of the form $P(\mathcal{F} \mid \mathcal{E})$ where $\mathcal{F}$ and $\mathcal{E}$ are equivalent utterances in $\mathcal{F}$rench and $\mathcal{E}$nglish.

2. http://eclipse.org
3. https://www.jetbrains.com/idea/
4. http://naturalness.cs.ucdavis.edu
5. http://goo.gl/ja4F9L
6. http://promisedata.org
7. http://flossmole.org
8. https://github.com/bibanon/bibanon/wiki/Gittorrent
9. https://www.gutenberg.org/
10. www.statmt.org/europarl/
11. http://en.wikipedia.org/wiki/Hansard

For the "Naturalness" venture, it would be most useful to have a large, curated corpus of source code, fully annotated with any kind of information that programmers might deem relevant. This would certainly comprise automatically derivable informations, such as lexical, syntactic, types, scope, data flow, and static analysis products; and meta-data, such as revision history and defect reports. It could include dynamic information such as execution traces, profiles, and test coverage. In addition, it could also include information that is only derivable manually, such as *alignments* between the code of multi-platform systems. Thus Nguyen *et al.* [17], [16] used method-level alignments between Java and C# to learn automatic translation rules.

The Qualitas [22] and Sourcerer [3] are both useful collections of curated source code. We have gone a bit further, constructing a large corpus of *abstract syntax trees* of over 50 Java projects, comprising several million lines of code[12]. Work is currently under way to build a similarly large corpus for C and other languages. We hope to shortly add annotations corresponding to warnings from PMD[13]. Part of this project's goal is to extend this Naturalness project corpus, including other types of automatically derived data described above, as well as manual annotations when available.

IV. Conclusion

Software is a natural product of human effort, and shares many statistical properties with natural language. This phenomenon promises a wealth of applications, leveraging on prior work on statistical methods in natural language; it also offers the promise of using new scientific methodologies to investigate the human & social aspects of software, drawn from approaches in cognitive and social linguistics. The NSF in the US have recently funded a large, multi-disciplinary, multi-institution initiative to pursue these directions.

Acknowledgment

The author gratefully acknowledges support from the U.S. National Science Foundation, grants NSF EAGER #1247280;, and more recently, a NSF SHF LARGE Collaborative grant, # 1414172. Co-PIs. on this grant include Zhendong Su and Vladimir Filkov at UC Davis, William Cohen, and Roni Rosenfeld at CMU, as well as Tien Nguyen at Iowa State. The author acknowledges other early contributors to this work, specifically Abram Hindle of Univ. of Alberta and Earl Barr of University College, London.

References

[1] M. Allamanis, E. T. Barr, and C. Sutton. Learning natural coding conventions. *arXiv preprint arXiv:1402.4182*, 2014.

[2] M. Allamanis and C. Sutton. Mining source code repositories at massive scale using language modelling. In *MSR*, pages 207–216, 2013.

[3] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes. Sourcerer: a search engine for open source code supporting structure-based search. In *Companion to the 21st OOPSLA*, pages 681–682. ACM, 2006.

[4] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th ACL annual meeting*, pages 310–318, 1996.

[5] W. N. Francis and H. Kucera. Brown corpus manual. *Brown University Department of Linguistics*, 1979.

[6] C. Franks, Z. Tu, P. Devanbu, and V. Hellendoorn. Cacheca: A cache language model based code suggestion tool. In *ICSE Demonstration Track*, May 2015.

[7] M. Gabel and Z. Su. A study of the uniqueness of source code. In *FSE*, pages 147–156. ACM, 2010.

[8] A. Hindle, E. Barr, M. Gabel, Z. Su, and P. Devanbu. On the Naturalness of Software. In *ICSE*, pages 837–847. IEEE, 2012.

[9] A. Hindle, E. Barr, M. Gabel, Z. Su, and P. Devanbu. On the Naturalness of Software. http://macbeth.cs.ucdavis.edu/nature.pdf, 2012. [Extended Version, Online; accessed 08-Feb-2015].

[10] S. Karaivanov, V. Raychev, and M. Vechev. Phrase-based statistical translation of programming languages. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, pages 173–184. ACM, 2014.

[11] D. E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[12] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[13] A. McCallum, R. Rosenfeld, M. Mitchell, and A. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings, ICML*. IEEE, July 1998.

[14] L. Mou, G. Li, Z. Jin, L. Zhang, and T. Wang. Tbcnn: A tree-based convolutional neural network for programming language processing. *arXiv preprint arXiv:1409.5718*, 2014.

[15] A. T. Nguyen, H. A. Nguyen, T. T. Nguyen, and T. N. Nguyen. Statistical learning approach for mining api usage mappings for code migration. In *Proceedings, ASE Conference*, pages 457–468. ACM, 2014.

[16] A. T. Nguyen, T. T. Nguyen, and T. N. Nguyen. Lexical statistical machine translation for language migration. In *Proceedings, SIGSOFT FSE*, pages 651–654. ACM, 2013.

[17] A. T. Nguyen, T. T. Nguyen, and T. N. Nguyen. Migrating code with statistical machine translation. In *Companion Proceedings of ICSE*, pages 544–547. ACM, 2014.

[18] T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen. A statistical semantic language model for source code. In *Proceedings, ESEC/FSE*, pages 532–542. ACM, 2013.

[19] S. Pinker. *The Language Instinct: The New Science of Language and Mind*, volume 7529. Penguin UK, 1994.

[20] V. Raychev, M. Vechev, and A. Krause. Predicting Program Properties from "Big Code". In *POPL*, 2015.

[21] R. Rosenfeld. Incorporating linguistic structure into statistical language models. *Philosophical Transactions of the Royal Society, Series A*, 358(1769):1311–1324, April 2000.

[22] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. The qualitas corpus: A curated collection of java code for empirical studies. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, pages 336–345. IEEE, 2010.

[23] Z. Tu, Z. Su, and P. Devanbu. On the localness of software. In *Proceedings SIGSOFT FSE*, pages 269–280. ACM, 2014.

[24] S. A. Wills and D. J. MacKay. Dasher-an efficient writing system for brain-computer interfaces? *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 14(2):244–246, 2006.

12. http://naturalness.cs.ucdavis.edu/ast.html
13. http://pmd.sourceforge.net.