Human Languages in Source Code: Auto-Translation for Localized Instruction

Chris Piech
Stanford University
Stanford, CA, USA
piech@cs.stanford.edu

Sami Abu-El-Haija
USC Information Sciences Institute
Marina del Rey, CA, USA
haija@isi.edu

ABSTRACT
Computer science education has promised open access around the world, but access is largely determined by what human language you speak. As younger students learn computer science it is less appropriate to assume that they should learn English beforehand. To that end, we present CodeInternational, the first tool to translate code between human languages. To develop a theory of non-English code, and inform our translation decisions, we conduct a study of public code repositories on GitHub. The study is to the best of our knowledge the first on human-language in code and covers 2.9 million Java repositories. To demonstrate CodeInternational's educational utility, we build an interactive version of the popular English-language Karel reader and translate it into 100 spoken languages. Our translations have already been used in classrooms around the world, and represent a first step in an important open CS-education problem.

Author Keywords
human-language; translation; source-code; github

1. INTRODUCTION
Reading and writing comments, method names and variable names, are crucial parts of software engineering. As such, programs have both a human language, the language of identifiers and comments, in addition to the source-code language (eg Java or Python). This has meant that non-English speakers are often second-class citizens when learning to program [21]. In this paper we present a tool for translating a program from one human language to another, to assist in code education, which could reduce the barrier to computer science education for non-English speakers.

The main contributions presented in this paper are:

1. Analysis of 1.1M non-English code projects on GitHub (Sec. 2).

2. CodeInternational[1]: A tool which can translate code between human languages, powered by Google Translate (Sec. 3).

3. Validation of CodeInternational by evaluating the translation of 1,000 randomly chosen projects from GitHub (Sec. 5).

4. Use of CodeInternational to automatically translate the popular Karel textbook into 100+ languages. We further extend the textbook to parse and run KarelJava code in any language; we report adoption by classrooms around the world (Sec. 4).

Our human-language code translator was inspired by a desire to make programming more accessible [6]. An accurate and useful translator would enable faster localization of instruction materials and it would allow learners (as well as practitioners) to translate code that they are working with.

As programming becomes more of a requisite common knowledge skill, we expect coding education to become open-access to everyone. One barrier to this goal is human language. English is currently the modal language of programming instruction perhaps given that the keywords of most of the popular languages, Java, JavaScript etc, are in English (even including Python and Lua, invented in the Netherlands and Brasil respectively). However, a majority of the world, estimated in 2008 at 80%, cannot "use" English for communication and substantially more do not speak English as their L1 language (the technical term for one's arterial language, aka, mother tongue) [11]. Should the more than 6 billion non-English speakers learn to program in their native language or in English? This question is debated, which we address in the discussion.

We take the position that whether or not code instruction is in English, if students do not speak English as their L1 language, their code education would benefit from the ability to translate Code between their preferred language and English.

1.1 Related Work
To the best of our knowledge, automatic translation of code between human languages did not appear in literature, making us hypothesize: it is either difficult, or had remained ignored. Nonetheless, we summarize related work, motivating our contribution.

L@S '20, August 12–14, 2020, Virtual Event, USA.
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7951-9/20/08 ...$15.00.
http://dx.doi.org/10.1145/3386527.3405916

[1] https://compedu.stanford.edu/codeinternational.html

Translation of Text automatic translation of natural language has recently achieved high accuracy and is used in highly sensitive contexts [26, 19, 14]. At the time of writing this article, Google Translate uses Neural Machine Translation [2] to translate pairwise between languages and has become incredibly accurate, at least for languages common on the web [35]. Further research has been done on transliterating text [24, 1]. However, current state-of-the-art methods for text translation fail at translating code. Directly running a translation algorithm on code would fail to distinguish between code syntax and identifiers, would not recognize terms embedded in identifiers e.g. with camel case `getElementAt`, and could produce code with one identifier name having different translations on separate lines. As such, current automatic text translation, if ran directly on code, would produce malfunctional code.

Code Instruction in Non-English In 2017, Dasgupta and Hill published seminal work outlining the importance of learning to code in one's own language. They conclude that "novice users who code with their programming language keywords and environment localized into their home countries' primary language demonstrate new programming concepts at a faster rate than users from the same countries whose interface is in English" [12]. Since then, there has been a large set of papers expanding on the barriers for non-native English speakers. Guo et al survey over 800 non-English students learning who report on the many challenges that come with not understanding English while coding. [20] reinforced by [13, 23]. This has led to preliminary work into translating compiler errors [29] and advocation for language-free block programming [3]. However, while language-free programming is a great step forward for younger students, it doesn't address the needs of CS1 students who program in common programming languages like Python or Java. While all of this work motivates our contribution, none has attempted an automatic solution to the problem, making *crowd-translation* a viable alternative [10].

Mining Github To understand the patterns of code that students and practitioners use, we analyze public repositories on GitHub. Other researchers also analyzed GitHub, sometimes via the dataset and tools provided by [18], including work on social diversity of teams [34] and affiliation influence on code popularity [5]. This has led to a set of best practices for navigating the promises and perils of mining GitHub [22]. A growing number of students are using GitHub in software engineering courses [16] which makes it a valuable resource for understanding code of the general population, including students.

Code Conversion There is a rich literature of work to translate code between *programming languages*, such as C or C++ to Java [32, 33], or even from English to code [25]. However, the emphasis is often on maintaining efficiency, not on making code readable for students. We focus on translating the human language of code. Byckling et al [7] analyze naming conventions of identifiers based on their function (fixed, iterators, transformers, etc), and correlate the naming consistency with the students' learning experience. This motivates aspects of our translation. See Section 3.1.

2. HUMAN LANGUAGES ON GITHUB

How do non-English speakers program in a language like Java, where the keywords and core libraries are written in English? We employ a data driven approach to tell the story of non-English code and inform the decisions we made in our auto-translator. We analyzed Java repositories on GitHub, the largest host of source code in the world, where 1.1 million unique users host 2.9 million public Java projects. We downloaded and analyzed the human language used for writing comments (in Java code), naming identifiers (method and variable names), and writing *git commit* messages. We focused on Java code as it is both one of the most popular source-code languages on GitHub and in the classroom. A selection of results from this study are that:

1. Non-English code is a large-scale phenomena.
2. Transliteration is common in identifiers for all languages.
3. Languages clusters into three distinct groups based on how speakers use identifiers/comments/transliteration.
4. Non-latin script users write comments in their L1 script but write identifiers in English.
5. Right-to-left (RTL) language scripts, such as Arabic, have no observed prevalence on GitHub identifiers, implying that existing coders who speak RTL languages have substantial barriers in using their native script in code.

This is, to the best of our knowledge, the first analysis of the human languages on GitHub. See Figure 1 for an overview.

Users on GitHub do not state their L1 (arterial) language. While a subset of users optionally state their country, this is neither common nor reliable. To estimate a user's preferred language, we use the language that they use in the git commit message. To find subsets of users who speak a given language, we search for all users who write git commits in that language. We observe that, especially in personal projects, users write commit messages in their L1 language at a higher rate than comments or identifiers. To identify languages we use *Google Language Detect* which is highly accurate (more-so for common internet languages) and can identify languages with non-Roman Alphabet text which has been transliterated, for example it can detect both 算法 the Chinese characters for "algorithm" and "suanfa", the Mandarin transliteration, as Chinese[2].

Of the 1.1 million GitHub users, 12.7% wrote commit messages in non-English languages. Of those,Chinese was the most common (28.6% of non-English committers), followed by Spanish, Portuguese, French, and Japanese. More than 100 languages were detected in commit messages on public Java projects. Figure 1 contains breakdowns and the appendix contains the full list. This does not match the distribution of non-English in web content (55% English) with both major and minor languages underrepresented. For example the

[2]Google Translate provides a confidence for its language detection. We only consider positive detections with confidence > 0.5. We do not run language detection on ASCII strings less than 2 characters long. Identifiers are turned into phrases using case parsing as described in Section 3. All "positive" results are manually verified.

(a) Public GitHub Java Users

Language	Chinese	Spanish	Port.	French
% of Non-Eng.	28.6%	16.3%	13.0%	7.2%
Users (Num)	(40k)	(23k)	(18k)	(10k)

N Public Java Users on GitHub = 1,097,809

(b) Examples of non-English Java

```
// 选择文件出场动画                    S,S
void 活跃()

// dem tong so luot xem
void capNhatSoLuotXem()              T,T

// returns the closure.
// この範囲の閉包を返す.                M,E
Range closure();
```

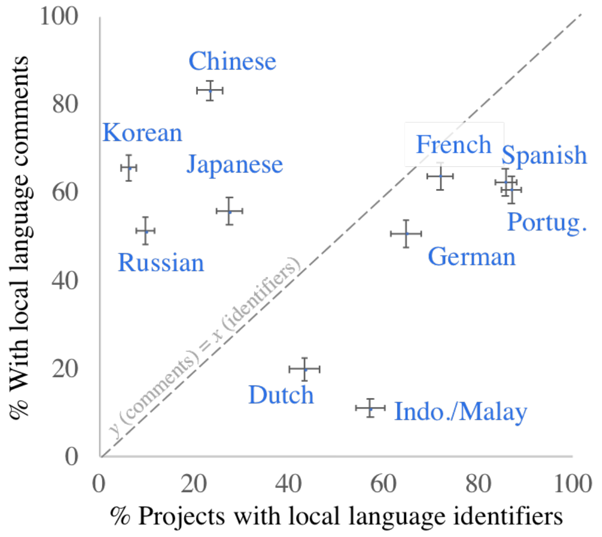
```
// Конец метода майн                 S,T
void podschetSrednego()

// 技术技能总分
Double jishuAllScore;                S,M

// @author حامد سلام
void initialize(URL url)             S,E
```

Labels (eg S,S) are CommentStye, IdentifierStyle
S = Script, T = Transliterated, E = English M = MixLang

(c) Coding patterns conditioned on language



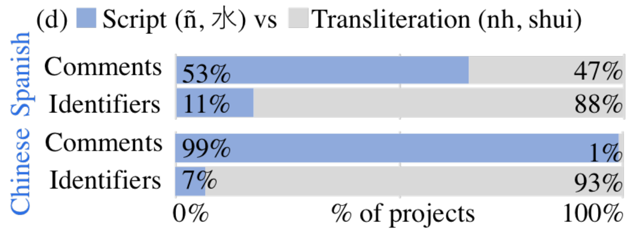


Figure 1: (a): Non-Eng languages for Java GitHub commits and their proportions (showing top four). (b) Java non-Eng example methods. (c) Use of local language in identifiers and comments conditioned on users speaking different languages. (d) Proportion of non-English projects with script vs transliteration

prevalence of Spanish on GitHub (2.1%) is about half of web-content (5.1% [31]) and further trails native speakers (7.8% of the worlds population [8]).

Github does not present a random sample of programs written in the world, and we consider the relevant confounds this introduces. To that point, we believe the under-representation of certain languages is a form of Survivorship Bias. It suggests that users have found barriers to entry towards joining the GitHub community. Those barriers could derive from the English dominance of programming languages, code instruction, or the github interface.

2.1 Non-English in Java

The use of non-English in identifiers and comments is large for the population of users who we define as non-English "speakers" (those who use non-English in their git-commit messages). 90% of users who use a non-English language in the commit messages also use that language in their comments or as identifiers. We note that, in Java, identifiers can be written in any script.

Surprisingly, the patterns of non-English usage differs substantially when we condition on users "speaking" different languages. For example, among the detected Spanish speakers, 87.2% percent of users write identifiers in Spanish. On the other hand, among Chinese users, only 23.3% of users write code with Chinese identifiers (either in Chinese script or ASCII). Figure 1c shows coding patterns conditioned on users speaking different languages. For each language we plot the percent of projects with identifiers in the language, against the percent of projects with comments in the language. Languages naturally cluster into three categories: (1) **Major-Euro-Latin:** languages with high use of non-English identifier including Spanish, German and French (2) **Non-Latin:** languages in non-latin scripts including Russian and Chinese which have low use of non-English identifiers and (3) **English-Comment:** Programmers write their comments in English (> 70% of projects only have English comments). This group contains many smaller and non-European languages like Dutch and Bahasa Indonesia. ~50% of projects in this group still uses their L1 language in identifiers.

The use of identifiers in local language (as opposed to English) is very clearly split on whether languages use the Latin alphabet. On average 82% of projects from users speak languages with different scripts like Chinese, Korean, or Russian have only English identifiers, compared to 12% of projects from Latin alphabet users ($p < 0.0001$). The percentage of projects with only English comments is roughly correlated to the English Proficiency Index [17] of the corresponding countries ($\rho = 0.42$ $p < 0.01$).

2.2 Trans*literation* on GitHub

Transliteration is the process of transferring a word from the alphabet of one language to another (eg नमस्ते दुनिया -> namaste duniya). We observed that most Java code with human languages that have non-ASCII scripts like Kanji, Devanagari, or even Spanish accents like ñ, will have been "transliterated" into ASCII.

The Java Language Specification states that, “letters and digits (in identifiers) may be drawn from the entire Unicode character set, which supports most writing scripts”. This specification is not widely known, and even if Java supports non-ASCII , there can be complexities of file encodings across different operating systems.

We find that regardless of L1 language most users transliterate identifiers: among L1 Chinese speakers, 93% of projects have identifiers which are only written in ASCII. Similarly in Spanish 88% of projects have only ASCII identifiers. As a concrete example, in GitHub Java code "numero" is 3.8x more common than "número". Among comments languages differ greatly: 99% of Chinese projects have non ASCII comments compared to only 53% of Spanish. As an example, a comment preceeding a method specifies in script that it is calculating the Fibonacci sequence, however, the method name (an identifier) is transliterated “//斐波那契” however the code uses a transliteration of the phonemes in the script “public int feibonaqie(int n)”. This is a common pattern: Within comments, 计数 chinese for count), is 4.0x more common than jishu, the transliteration. However in identifiers jishu is 4.8x more common. The difference in transliteration patterns between Chinese and Spanish suggests a different intent: in Spanish transliteration is used to avoid file encoding errors, in Chinese it is to prevent a mix of scripts among identifiers.

2.3 Right-to-Left Languages on GitHub

One question that we did not have a solid pre-conception for was: *How do Java users who speak languages with right-to-left (RTL) scripts like Arabic, Urdu or Hebrew, write code?*

18,961 users on GitHub report their country as one where a RTL script (Arabic or Hebrew) is the primary script. Those users have 8,060 public Java repositories of which only 50 repositories (0.6%) have Arabic or Hebrew script (excluding string literals). Of those repositories, only a single Java file had a single identifier written in Arabic and none in Hebrew. It is extremely rare for methods or identifiers to be a mix of RTL and LTR.

3. CODE INTERNATIONAL

The GitHub analysis is coherent with the contemporary narrative: there are perhaps hundreds of millions of learners who will not speak English as their L1 language. For those learners, teachers need a tool to translate code so they can give examples with less congitive load. Similarly students need a tool to understand the non-English code they encounter. Finally, to a growing extent, English speakers will begin to interact with code written in other languages.

To address this need, we designed a tool to help programmers, regardless of their spoken language, access code in many languages. The tool, which we call CodeInternational, takes-in code written in either Java or Python with comments and identifiers written in a human-language, and translates the comments and identifiers into another human-language. It supports the growing set of human languages covered by Google Translate and is adaptive to the particular context of source-code. To translate code, it first parses the code and extracts four types of tokens:

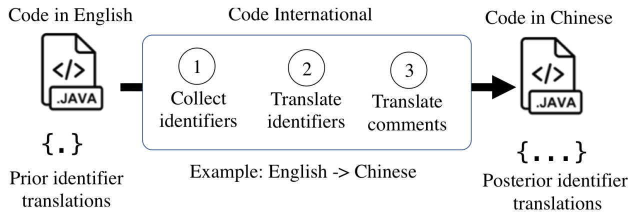


Figure 2: High-level of how CodeInternational work

- **Comments**: inline or multi-line comments. Their purpose is for the programmer to communicate to programmers (including herself) on the purpose of code sections.
- **Immutable**: consisting of language keywords (`while`, `void`, etc), and identifiers imported from libraries that are external to the code being translated (e.g. `FileReader` of `java.io`). By default this group is not translated.
- **Target identifiers**: including variable and function names that are defined in the code base undergoing translation.
- **String literals**: In some cases a user may want String literals to be translated, other times they should be unchanged.

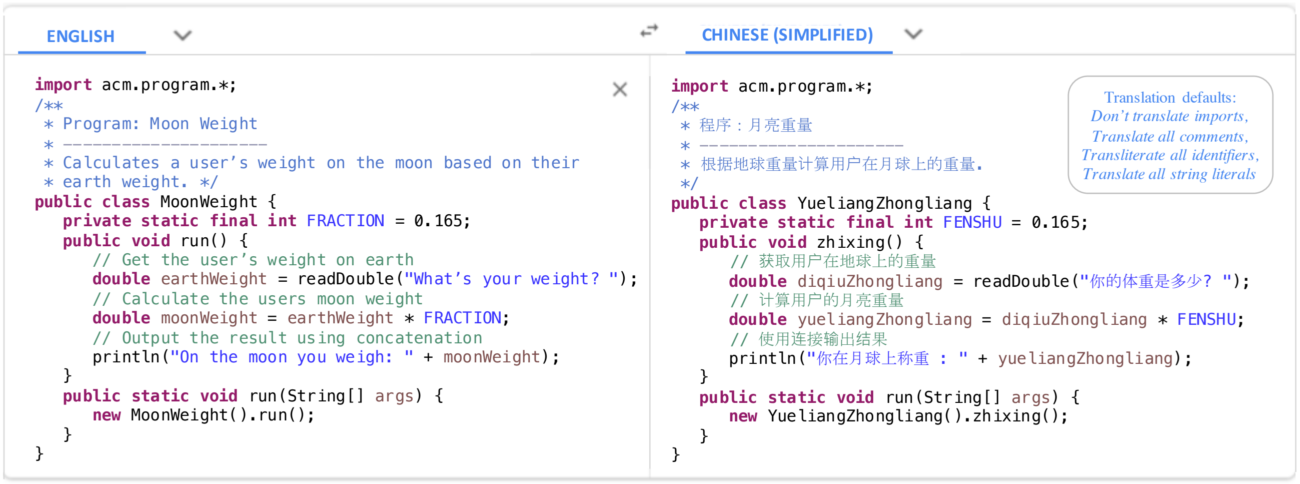
Our translation algorithm is as follows. We (1) collect all of the target identifiers defined in the codebase and (2) translate them (enforcing that if two identifiers have the same name, they are given the same translation). Once the identifiers are translated we (3) translate the comments preserving structure and references to identifiers. (4) Finally string literals are optionally translated. See Figure 2 for a highlevel depiction and Figure 3 for a concrete example. Each of these steps has surprising challenges. In this section we cover the corresponding solutions we developed. The mapping of identifier translations that the tool decides on is preserved to assist any external source which needs to refer to the newly translated identifiers (such as text in a text-book or code in a related project).

CodeInternational is implemented in Python. Tokenization is performed using a modified version of "Javalang" (for Java) and the "Parser" library (for Python). Supporting other programming languages requires a small amount of extra work.

3.1 Translating Identifiers

In order to properly translate identifiers, we consider the following:

Identifier segmentation: Translating an identifier using a tool like Google Translate does not work by default as identifiers are often composed of unsegmented words. For example: getFavoriteNumber is readable to a human as "get favorite number" but is not parsable by an online translator. We segment identifiers using naming conventions (e.g. `camelCaseVariable`, `PascalCaseClass`, `UPPERCASE_CONSTANT`). We thus segment identifiers into phrases which we feed into an automatic translator. We then recombine the translated phrase using the original casing convention. For example, to translate the method name identifier “turnAround” into Spanish: “turnAround” is segmented into “turn around”, which is translated into “media vuelta” that is



```java
import acm.program.*;
/**
 * Program: Moon Weight
 * ----------------------
 * Calculates a user's weight on the moon based on their
 * earth weight. */
public class MoonWeight {
   private static final int FRACTION = 0.165;
   public void run() {
      // Get the user's weight on earth
      double earthWeight = readDouble("What's your weight? ");
      // Calculate the users moon weight
      double moonWeight = earthWeight * FRACTION;
      // Output the result using concatenation
      println("On the moon you weigh: " + moonWeight);
   }
   public static void run(String[] args) {
      new MoonWeight().run();
   }
}
```

```java
import acm.program.*;
/**
 * 程序：月亮重量
 * ----------------------
 * 根据地球重量计算用户在月球上的重量.
 */
public class YueliangZhongliang {
   private static final int FENSHU = 0.165;
   public void zhixing() {
      // 获取用户在地球上的重量
      double diqiuZhongliang = readDouble("你的体重是多少? ");
      // 计算用户的月亮重量
      double yueliangZhongliang = diqiuZhongliang * FENSHU;
      // 使用连接输出结果
      println("你在月球上称重 : " + yueliangZhongliang);
   }
   public static void run(String[] args) {
      new YueliangZhongliang().zhixing();
   }
}
```

Figure 3: An example of using CodeInternational to translate a simple Java program from English to Chinese. Mandarin speakers will notice that when the meaning is misrepresented in English (for *Calculate the users moon weight*), then the translation could fail to capture user's intended meaning (translation says *calculate moon weight*).

then formatted into the original camelCase, "mediaVuelta". Advances in artificial intelligence for word segmentation could enable future versions of this tool to break up words, *without* a given case-segmentation (eg "turnaround").

Verb prior: The correct translation for a phrase can be ambiguous, especially without context. As an example, the method "move" translated into Spanish could be translated into a noun ("movimiento", movement) or a verb ("moverse"). For method identifiers, there is an implicit context that an action is being performed. We incorporate this context by placing a prior on the first word being a verb. Thus, for example, when we translate "move()" into Spanish we chose "moverse()" instead of "movimiento()", the noun movement, as Google suggests.

In addition to knowing the translations of methods should start with verbs, we also have a select number of reasonable tenses for the verb: infinitive (eg "toMove"), third person present (eg "moves" as in "he moves") and imperative (eg "move"). In most languages, including English, we translate verbs with a prior that they be the imperative tense. In English you would expect a method to be "getObject()" the imperative. However some languages, especially Romance languages, use the infinitive of the verb: as an example, Spanish "obtener" the infinitive of "obtain" is 200x more common on GitHub then "obtenga" the imperative.

Translating short identifiers: Short variable names that are used for mathematical symbols or as iterators should not be translated. This is especially important to pay attention to for the canonical for loop identifier "i". For example translating the code "for(int i = 0; i < 10; i++)" into Spanish should not produce "for(int yo = 0; yo < 10; yo++)" even though "yo" is the translation of the pronoun "I". We only translate identifiers which are at least two characters long. This exception has its own edge-case: CJK (Chinese, Japanese Korean) identifiers can be non-mathematical names even if only a character long.

3.2 Translating comments

Once we have finished translating identifiers, we translate the comments in a program. Translating comments has two complexities: (1) we would like to maintain the comment structure, eg if it is a block javadoc comment, we would like to reserve the column of "*"s on the left margin of the comment and (2) we want references to identifiers to be translated exactly as they were in the code.

To translate a comment we classify the structure (eg JavaDoc, BlockComment, or PythonDocString). We then strip the text, translate it, and reformat it back into the same structure. For multi-line comments, we are conscious not to increase the maximum length of a line, taking into account the wider width of CJK characters.

3.3 Translating Right-to-Left languages

Arabic, Hebrew, Farsi, and Urdu are popular right-to-left (RTL) natural languages. When translating code to RTL languages, comment can be translated (mixing RTL within the left-to-right syntax) and optionally transliterated (keeping left-to-right flow). Some of the difficulty in RTL transliteration is in distinguishing between short- and long-vowels. Further, these languages contains consonants that cannot be described using Latin alphabets, which are generally represented with numbers in the transliteration culture – e.g. 7 for ح, which is closest to Latin alphabet "h" e.g. in "A**h**mad".

When translating non-Latin scripts which are LTR, we give the user the option to transliterate identifiers and separately, to transliterate comments or not. Transliteration is currently supported in Arabic, Chinese, Hebrew, Japanese, Korean, and Russian.

3.4 Prior and posterior translations

Translations of code need to be coherent with respect to other translations of written text (or other files) that refers to the code. To that end our translator takes-in and uses a preset identifier translation map, and returns the translations it made. This system enables having humans override translations, for translating textbooks with text that references embedded code and more.

4. TRANSLATING GITHUB

How good is a translation of source-code from one human language to another? Evaluating quality of a translation is hard without a large collection of native speakers and since we are powered by Google, evaluation can devolve into evaluating how accurate Google Translation is. Such an evaluation is a moving target: Google Translation is perennially improving.

To evaluate out translator, we randomly selected 1,000 (1k) single file projects from public GitHub Java and translated them into the languages: Chinese, Spanish and Arabic. We measure (1) how often the translated code still compiles and (2) what percent of identifiers that we attempt to translate are translatable.

Of the 1k projects, 100% maintained their ability to be compiled, regardless of whether we translated or transliterated the comments or identifiers. From the 1k projects, 91% of the identifiers were able to be translated. The nine percent that were not able to be translated were mainly abbreviations (such as users who named a variable frac instead of fraction or pct instead of percent). This presents an opportunity for future work. Overall the results paint the picture of a functioning tool which is ready for use.

5. INTERNATIONAL KAREL

Our motivation for developing an automatic human-language code translation tool was to support education for non-English speakers. To that end, we used CodeInternational to translate a web-version of the popular Karel the Robot learns Java reader by Eric Robers [30] a textbook for *Karel the Robot*, a grid world robot invented by Richard Pattis [27] to help CS1 students learn to program. Karel has been the inspiration for assignments on platforms such as Code.org and CodeHS and is a staple of the first weeks of CS1 [4].

We translated a Karel reader in Python and Java to 100 languages. The translated web-readers are free to use and are hosted online[3]. At time of publication, the reader has been public (without advertising) and has already been used by over 3,000 people from 50 countries. With permission from Eric Roberts, we first made an eBook version of his Karel reader and simplified the English used [15]. The reader merges text and code in a seem-less fashion. Then, for each language: we (a) translated code on each chapter using CodeInternational and (b) translated the reader text such that any reference to identifiers in the example code would use the same translations. In order to have text which is consistent with the corresponding code we heavily rely on the "Posterior identifier translation map" from CodeInternational's translations.

5.1 Line-highlighting in any language

To make the Karel reader a fantastic learning experience, we made it so that each code-snippet is runnable. When run, the program ***executes the code and highlights the corresponding lines*** as the program is run, regardless of the complexity of the program's control flow. In order to line-highlight, we parse and compile the Python-Karel or Java-Karel programs using an engine written in JavaScript. Our line-highlighter builds upon the compiler described in *Informatics Education using Nothing but a Browser* [28].

Our Karel reader can run and line-highlight in any human-language that we translate into. For example our compiler can execute and line-highlight the command "moverse()" if the code is written in Spanish, "移动()" if the program is written in Chinese, "emshi()" if the program is written in Arabic, or "move()" if the Karel program is written in English. We chose to only transliterate commands for RTL scripts. Figure 4 shows three screenshots from the international Karel reader, though of course a PDF is unable to capture the ability of the reader to line-highlight code.

5.2 Usage in Classrooms

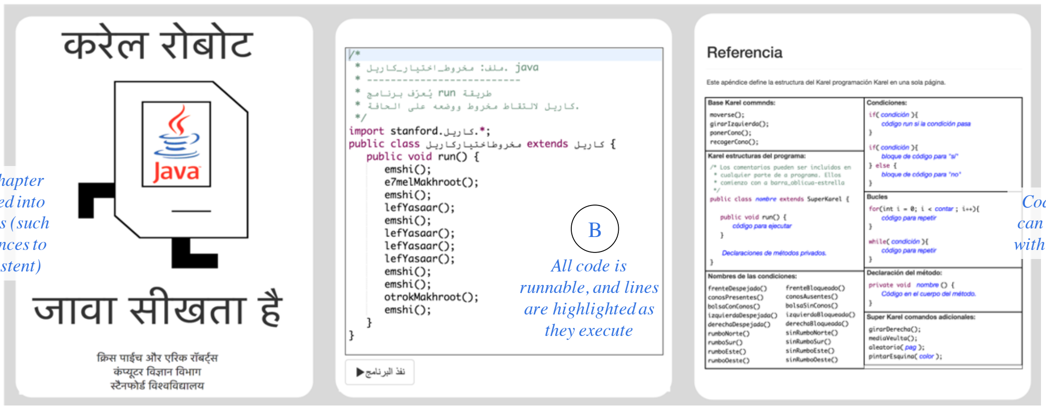
We know of three classes where the internationalized Karel eReader has been used. These classes are around the world in: Istanbul, Bogota and Prague. The eReader has been visited by >1k users in 3 months. Both the English and the non-English version of the website have a high average session duration, respectively, 9.7 and 10.1 minutes. Moreover, the tool has been used to translate the CSBridge[4] curriculum website and assignments; HTML that mixes code and description (used by 400 students / year).

6. DISCUSSION

Whether English should be used as the sole language of instruction has been debated. **Case for code instruction in English**: In order to program professionally, one will have to interact with keywords and libraries that are written for English speakers. English is *the* language of code, and it is practically required from anyone who wants to interact globally: correspond via email, read stack-overflow, watch educational videos, travel, etc. For classrooms where English is the main form of instruction, but students are not yet fluent, CodeInternational can be used to assist learning English and learning to program. Students could improve their English through coding, e.g. by placing English code against their L1 code, side-by-side. **Case for instruction on transl(iter)ated code**: On the other hand, people argue that it is beneficial for students to have much of their coding instruction in their L1 language, and doing so benefits access to CS. The primary reason for this is intuitive: the cognitive-load of learning to program is already high. Moreover, if students learn coding using their L1 language and enjoy it, they become intrinsically motivated to learn English, knowing that English would

[3]Python: `https://compedu.stanford.edu/karel-reader/docs/python/en/intro.html` and Java: `https://compedu.stanford.edu/karel-reader/docs/java/en/intro.html`

[4]`https://www.csbridge.org/`


A
All code and chapter text is translated into 100+ languages (such that text references to code are consistent)
B
All code is runnable, and lines are highlighted as they execute
C
Code translations can be interspliced with text translation
Java
▶ نفذ البرنامج

करेल रोबोट

जावा सीखता है

क्रिस पाईच और एरिक रॉबर्ट्स
कंप्यूटर विज्ञान विभाग
स्टैनफोर्ड विश्वविद्यालय

```
/*
 * ملف: مخروط_اختيار_كاريل. java
 * -------------------------
 * طريقة run يُعرّف برنامج
 * كاريل لالتقاط مخروط ووضعه على الحافة.
 */
import stanford.كاريل.*;
public class مخروطاختياركاريل extends كاريل {
   public void run() {
      emshi();
      e7melMakhroot();
      emshi();
      lefYasaar();
      emshi();
      lefYasaar();
      lefYasaar();
      lefYasaar();
      emshi();
      emshi();
      otrokMakhroot();
      emshi();
   }
}
```

Referencia

Este apéndice define la estructura del Karel programación Karel en una sola página.

Base Karel commnds:

```
moverse();
girarIzquierdo();
ponerCono();
recogerCono();
```

Karel estructuras del programa:

```
/* Los comentarios pueden ser incluidos en
 * cualquier parte de a programa. Ellos
 * comienzo con a barra_oblicua-estrella
 */
public class nombre extends SuperKarel {

   public void run() {
      código para ejecutar
   }

   Declaraciones de métodos privados.
}
```

Nombres de las condiciones:

```
frenteDespejado()       frenteBloqueado()
conosPresentes()        conosAusentes()
bolsaConConos()         bolsaSinConos()
izquierdaDespejada()    izquierdaBloqueado()
derechaDespejado()      derechaBloqueado()
rumboNorte()            sinRumboNorte()
rumboSur()              sinRumboSur()
rumboEste()             sinRumboEste()
rumboOeste()            sinRumboOeste()
```

Condiciones:

```
if( condición ){
   código run si la condición pasa
}

if( condición ){
   bloque de código para "si"
} else {
   bloque de código para "no"
}
```

Bucles

```
for(int i = 0; i < contar ; i++){
   código para repetir
}

while( condición ){
   código para repetir
}
```

Declaración del método:

```
private void  nombre () {
   Código en el cuerpo del método.
}
```

Super Karel comandos adicionales:

```
girarDerecha();
mediaVeulta();
aleatorio( pag );
pintarEsquina( color );
```

Figure 4: Three screenshots from the Karel eReader, translated both into 100+ languages for Java and Python: Left: intro page in Hindi; Middle: code translated into Arabic with transliterated identifiers; Right: reference in Spanish.

broaden their access to learning material (learning earning a language, with no short-term motives, could be dull especially for young students). In this context CodeInternational can help students who are interacting with libraries in English. Perhaps more importantly, our tool can help teachers rapidly develop localized content that builds off English content. The alternative: manual-translation of API, code-examples and website text, can be a huge barrier to translating material. Finally, our tool builds off GoogleTranslate, which is high accurate, but charges $1 per 50,000 characters. A free version would have a huge impact on utility.

We call for **future work** from tool experts, for example, to extend popular code-editors (e.g. vim, XCode, Visual Studio, Eclipse) to integrate with our APIs. This could allow back-and-forth translation and side-by-side display. Optionally, integrating with automatic text-to-speech (e.g. [9]) could allow students learn English pronunciation of code components. Moreover, one remaining feature in automatic human-translation of code is identifier consistency: if two identifiers have specific terms in common, eg getHeight, setHeight, we would like the translation of "height" to be consistent. While they are often consistent in our work, it is not enforced. Full consistency is hard, but not impossible, with modern neural machine translation.

7. CONCLUSION

We analyze millions of non-English Java programs on GitHub to inform our understanding of patterns of human-language and make some surprising observations. We build CodeInternational, an open-source tool which can translate Java or Python code between human languages. We evaluate our tool and use it to make an internationalized Karel eReader (with runnable code) in 100+ languages. Our tool is already being used in classrooms around the world, a trend we hope to continue supporting.

Acknowledgements

We would like to graciously thank Colin Kincaid for contributing code to this translation project. We would also like to thank Olya Irzak for her inspiration. We also thank the CSBridge teachers for educating students around the world in their local language.

8. REFERENCES

[1] Arbabi, M., Fischthal, S. M., Cheng, V. C., and Bart, E. Algorithms for arabic name transliteration. *IBM Journal of research and Development 38*, 2 (1994), 183–194.

[2] Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[3] Banerjee, R., Liu, L., Sobel, K., Pitt, C., Lee, K. J., Wang, M., Chen, S., Davison, L., Yip, J. C., Ko, A. J., et al. Empowering families facing english literacy challenges to jointly engage in computer programming. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (2018), ACM, p. 622.

[4] Becker, B. W. Teaching cs1 with karel the robot in java. In *ACM SIGCSE Bulletin* (2001), vol. 33, ACM, pp. 50–54.

[5] Blincoe, K., Sheoran, J., Goggins, S., Petakovic, E., and Damian, D. Understanding the popular users: Following, affiliation influence and leadership on github. *Information and Software Technology 70* (2016), 30–39.

[6] Brinkman, B., and Diekman, A. Applying the communal goal congruity perspective to enhance diversity and inclusion in undergraduate computing degrees. In *Proceedings of the 47th ACM technical symposium on computing science education* (2016), ACM, pp. 102–107.

[7] Byckling, P., Gerdt, P., and Sajaniemi, J. Roles of variables in object-oriented programming. In *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (2005), ACM, pp. 350–355.

[8] CERVANTES, I. El espanol: una lengua viva. informe 2017. `https://cvc.cervantes.es/lengua/espanol_lengua_viva/pdf/espanol_lengua_viva_2017.pdf`, 2017.

[9] CHEN, Y., ASSAEL, Y. M., SHILLINGFORD, B., BUDDEN, D., REED, S. E., ZEN, H., WANG, Q., COBO, L. C., TRASK, A., LAURIE, B., GÜLÇEHRE, Ç., VAN DEN OORD, A., VINYALS, O., AND DE FREITAS, N. Sample efficient adaptive text-to-speech. In *arXiv.org 1809.10460* (2019).

[10] CODE.ORG. Help translate | code.org. `https://code.org/translate`, Accessed: 2019-08-27.

[11] CRYSTAL, D. Two thousand million? *English today 24*, 1 (2008), 3–6.

[12] DASGUPTA, S., AND HILL, B. M. Learning to code in localized programming languages. In *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale* (2017), ACM, pp. 33–39.

[13] DASGUPTA, S., AND HILL, B. M. How "wide walls" can increase engagement: evidence from a natural experiment in scratch. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (2018), ACM, p. 361.

[14] DE VRIES, E., SCHOONVELDE, M., AND SCHUMACHER, G. No longer lost in translation: Evidence that google translate works for comparative bag-of-words text applications. *Political Analysis 26*, 4 (2018), 417–430.

[15] ERICSON, B. J., ROGERS, K., PARKER, M., MORRISON, B., AND GUZDIAL, M. Identifying design principles for cs teacher ebooks through design-based research. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (2016), ACM, pp. 191–200.

[16] FELICIANO, J., STOREY, M.-A., AND ZAGALSKY, A. Student experiences using github in software engineering courses: a case study. In *Proceedings of the 38th International Conference on Software Engineering Companion* (2016), ACM, pp. 422–431.

[17] FIRST, E. Ef english proficiency index. *Retrieved on June 1* (2013), 2014.

[18] GOUSIOS, G. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (Piscataway, NJ, USA, 2013), MSR '13, IEEE Press, pp. 233–236.

[19] GROVES, M., AND MUNDT, K. Friend or foe? google translate in language for academic purposes. *English for Specific Purposes 37* (2015), 112–121.

[20] GUO, P. J. Non-native english speakers learning computer programming: Barriers, desires, and design opportunities. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (2018), ACM, p. 396.

[21] JENKINS, J. *English as a lingua franca in the international university: The politics of academic English language policy*. Routledge, 2013.

[22] KALLIAMVAKOU, E., GOUSIOS, G., BLINCOE, K., SINGER, L., GERMAN, D. M., AND DAMIAN, D. The promises and perils of mining github. In *Proceedings of the 11th working conference on mining software repositories* (2014), ACM, pp. 92–101.

[23] KIRKPATRICK, T. A. Internationalization or englishization: Medium of instruction in today's universities.

[24] KNIGHT, K., AND GRAEHL, J. Machine transliteration. *Computational linguistics 24*, 4 (1998), 599–612.

[25] LITTLE, G., AND MILLER, R. C. Translating keyword commands into executable code. In *Proceedings of the 19th annual ACM symposium on User interface software and technology* (2006), ACM, pp. 135–144.

[26] MOBERLY, T. Doctors choose google translate to communicate with patients because of easy access, 2018.

[27] PATTIS, R. Karel j robot.

[28] PIECH, C., AND ROBERTS, E. Informatics education using nothing but a browser. In *Proceedings of the IFIP Conference on ICT and Informatics in a Globalised World of Education, Mombasa, Kenya* (2011).

[29] REESTMAN, K., AND DORN, B. Native language's effect on java compiler errors. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (2019), ACM, pp. 249–257.

[30] ROBERTS, E. Karel the robot learns java. *Department of Computer Science Stanford University* (2005).

[31] SURVEYS, W. T. Historical trends in the usage of content languages for websites. (Accessed on August 30, 2019) `https://w3techs.com/technologies/history_overview/content_language`, 2019.

[32] TEREKHOV, A. A. Automating language conversion: a case study. In *Proceedings IEEE International Conference on Software Maintenance. ICSM 2001* (2001), IEEE, pp. 654–658.

[33] VARGAS, B. D. System for translating programming languages, Mar. 18 2008. US Patent 7,346,897.

[34] VASILESCU, B., SEREBRENIK, A., AND FILKOV, V. A data set for social diversity studies of github teams. In *Proceedings of the 12th Working Conference on Mining Software Repositories* (2015), IEEE Press, pp. 514–517.

[35] WU, Y., SCHUSTER, M., CHEN, Z., LE, Q. V., NOROUZI, M., MACHEREY, W., KRIKUN, M., CAO, Y., GAO, Q., MACHEREY, K., KLINGNER, J., SHAH, A., JOHNSON, M., LIU, X., KAISER, L., GOUWS, S., KATO, Y., KUDO, T., KAZAWA, H., STEVENS, K., KURIAN, G., PATIL, N., WANG, W., YOUNG, C., SMITH, J., RIESA, J., RUDNICK, A., VINYALS, O., CORRADO, G., HUGHES, M., AND DEAN, J. Google's neural machine translation system: Bridging the gap between human and machine translation.