Submission to IEEE Computer

An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl

Lutz Prechelt (prechelt@ira.uka.de)
Fakultät für Informatik, Universität Karlsruhe
D-76128 Karlsruhe, Germany
+49/721/608-4068, Fax: +49/721/608-7343
http://wwwipd.ira.uka.de/~prechelt/

March 14, 2000

Summary

80 implementations of the same set of requirements are compared for several properties, such as run time, memory consumption, source text length, comment density, program structure, reliability, and the amount of effort required for writing them. The results indicate that, for the given programming problem, which regards string manipulation and search in a dictionary, "scripting languages" (Perl, Python, Rexx, Tcl) are more productive than "conventional languages" (C, C++, Java). In terms of run time and memory consumption, they often turn out better than Java and not much worse than C or C++. In general, the differences between languages tend to be smaller than the typical differences due to different programmers within the same language.

Introduction

When it comes to the pros and cons of various programming languages, programmers and computer scientists alike are usually highly opinionated. The present work provides some *objective* information comparing several languages, namely C, C++, Java, Perl, Python, Rexx, and Tcl. It has the following features:

– The same program (i.e. an implementation of the same set of requirements) is considered for each language. Hence, the comparison is narrow but homogeneous.

– For each language, we analyze not a single implementation of the program but a number of separate implementations by different programmers. Such a group-wise comparison has two advantages. First, it smoothes out the differences between individual programmers (which threaten the validity of any comparison based on just one implementation per language). Second, it allows to assess and compare the *variability* of program properties induced by the different languages.

– Several different aspects are investigated, such as program length, programming effort, run time efficiency, memory consumption, and reliability.

We will consider the languages both individually and combined into groups: Perl, Python, Rexx, and Tcl and often called *scripting languages* and will form one group called the *script group*. The name scripting language commonly refers to languages that are for instance (more or less) interpreted rather than compiled, at least during the program development phase, and that do typically not require variable declarations. The alternative are the more conventional programming languages which I will call the *non-script group*. These languages (C, C++, and Java) are more or less compiled and require typed variable declarations. We will sometimes consider C and C++ as one group and Java as another.

The number of programs considered for each language and the execution platforms are described in Table 1. See the sidebars for a discussion of setup and validity of the study. A more detailed description of this study can be found in a technical report [4].

Results

The programs were evaluated on three different input files: z1000 contains 1000 non-empty random phone numbers, m1000 contains 1000 arbitrary random phone numbers (with empty ones allowed), and z0 contains no phone number at all (for measuring dictionary load time alone).

Run time

We will first investigate the total run time and then examine the initialization phase and the search phase separately.

Total: z1000 data set. The global overview of the program run times on the z1000 input file is shown in Figure 1. We see that for all languages a few very slow programs exist, but except for C++, Java and Rexx, at least three quarters of the programs run in less than one minute. We can make several interesting observations:

The programming problem: Phonecode

All programs implement the same functionality, namely a conversion from telephone numbers into word strings as follows. The program first loads a dictionary of 73113 words into memory from a flat text file (one word per line, 938 Kilobytes overall). Then it reads "telephone numbers" from another file, converts them one by one into word sequences, and prints the results. The conversion is defined by a fixed mapping of characters to digits as follows:

```
e jnq rwx dsy ft am civ bku lop ghz
0 111 222 333 44 55 666 777 888 999
```

The task of the program is to find a sequence of words such that the sequence of characters in these words exactly corresponds to the sequence of digits in the phone number. All possible solutions must be found and printed. The solutions are created word-by-word and if no word from the dictionary can be inserted at some point during that process, a single digit from the phone number can appear in the result at that position. Many phone numbers have no solution at all. Here is an example of the program output for the phone number "3586-75", where the dictionary contained the words 'Dali', 'um', 'Sao', 'da', 'Pik', and 73108 others:

```
3586-75:  Dali um
3586-75:  Sao 6 um
3586-75:  da Pik 5
```

A list of partial solutions needs to be maintained by the program while processing each number and the dictionary must be embedded in a supporting data structure (such as a 10-ary digit tree) for efficient access.

Table 1: Number of programs and name/version of compiler or interpreter used for the various languages. The Java evaluation uses either the JDK 1.2.2 Hotspot Reference version or the JDK 1.2.1 Solaris Production version (with JIT), whichever was faster for each program. All programs were executed on a 300 MHz Sun Ultra-II workstation with 256 MB memory, running under SunOS 5.7 (Solaris 7). Note that the results for C and Rexx will be based on only 5 or 4 programs, respectively, and are thus rather coarse estimates of reality, but for all of the other languages there are 10 or more programs, which is a broad-enough base for reasonably precise results.

language	no.	compiler or execution platform
Tcl	10	tcl 8.2.2
Rexx	4	Regina 0.08g
Python	13	python 1.5.2
Perl	13	perl 5.005_02
Java	24	Sun JDK 1.2.1/1.2.2
C++	11	GNU g++ 2.7.2
C	5	GNU gcc 2.7.2

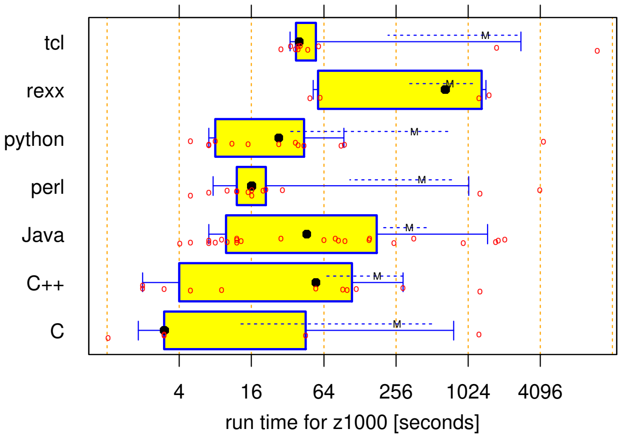

Figure 1: Program run time on the z1000 data set. Three programs were timed out with no output after about 21 minutes. The bad/good ratios range from 1.5 for Tcl up to 27 for C++. Note the logarithmic axis. ••

- The typical (i.e., median) run time for Tcl is not significantly longer than that for Java or even for C++.
- The median run times of both Python and Perl are smaller than those of Rexx and those of Tcl.
- The median run time of Java is not significantly different from any of the others (not even Rexx, where $p = 0.13$).
- Don't be confused by the median for C++. Since the distance to the next larger and smaller points is rather large, it is unstable. The Wilcoxon test, which takes the whole sample into account, confirms that the C++ median in fact tends to be smaller than the Java median ($p = 0.18$).
- The median run time of C is smaller than those of Java, Rexx, and Tcl and tends to be smaller than those of Perl and Python.
- Except for two very slow programs, Tcl and Perl run times tend to have a smaller variability than the run times for the other languages.

Remember not to over-interpret the plots for C and Rexx, because they have only few points. Note that the Rexx programs can be made to run about four times faster by recompiling the Regina interpreter so as to use a larger hash table size; the additional memory overhead is negligible.

If we aggregate the languages into only three groups (one with C/C++, one with Java, and one with scripts), we find that C/C++ is faster than Java ($p = 0.074$) and tends to be faster than scripts ($p = 0.15$). There is no significant difference between average Java and Script run times. With

Validity of this comparison

Any programming language comparison based on actual example programs is valid only to the degree to which the capabilities of the respective programmers using these languages are similar. In our case, we only need the programs to be comparable on average, not in individual cases. This section assesses program comparability threats for the 80 programs analyzed here.

The programs analyzed in this report come from two different sources. The Java, C, and C++ programs were produced in 1997/1998 during a controlled experiment [5]; all of the subjects were Computer Science master students. The Perl, Python, Rexx, and Tcl programs were produced under more variable conditions: They were created by volunteers after I had posted a "Call for Programs" on several newsgroups. These subjects are more diverse in terms of background and experience.

Programmer capabilities. It is plausible that a public call for programs may attract only fairly competent programmers and hence the script programs reflect higher average programmer capabilities than the non-script programs. However, two observations suggest that this is not a problem. First, with some exceptions, the students who created the non-script programs were also quite capable and experienced [5]. Second, a fair fraction of the script programmers have described themselves as either beginners in their respective scripting language or even as persons without a thorough programming background (e.g. VLSI designer, system administrator, social scientist).

Within the non-script group, the Java programmers tend to be less experienced in their language than the C and C++ programmers because Java was still a new language in 1997/1998. In the script group, the Perl subjects may be more capable than the others, because the Perl language appears more than others to attract especially capable people.

Work time reporting accuracy. In contrast to the non-script programs from the controlled experiment, for which we know the real programming time accurately, nothing kept the script programmers from "rounding down" the working times they reported when they submitted their program. Worse, some apparently read the requirements days before they actually started implementing the solution (in one case *"two weeks... during which my subconscious may have already worked on the solution."*)

However, there is evidence that the average work times are reasonably accurate for the script group, too: The common software engineering wisdom which says "the number of lines written per hour is independent of the language" holds fairly well across all languages. Even better, the same data also confirms that the programmer capabilites are not higher in the script group.

Different task and different work conditions. The instructions for the non-script group focused on correctness as the main goal; high reliability and at least some efficiency was required in an acceptance test. The instructions of the non-script group mentioned 8 other program quality goals besides the main goal of correctness. Instead of the acceptance test in the non-script group, the script group received the z1000 input and output data for their own testing. Both of these differences may represent an advantage for the script group.

Summary. Overall, it is probably fair to say that due to the design of the data collection, the data for the script groups will reflect several relevant (although modest) a-priori advantages compared to the data for the non-script groups and there are likely to be some modest differences in the average programmer capability between any two of the languages. Due to these threats to validity, we should discount small differences between any of the languages, as these might be based on weaknesses of the data. Large differences, however, are likely to be valid.

80% confidence a script will run at least 1.29 times as long as a C/C++ program and a Java program at least 1.22 times as long as a C/C++ program. The bad/good ratios are much smaller for scripts (4.1), than for Java (18) or even C/C++ (35).

Initialization phase only: z0 data set. Now we consider only reading, preprocessing, and storing the dictionary. Figure 2 shows the corresponding run time.

We find that C and C++ are clearly faster in this case than all other languages. The fastest script languages are again Perl and Python. Rexx and Tcl are again slower than these and Java is faster.

For the aggregate grouping we find that, compared to a C/C++ program, a Java program will run at least 1.3 times as long and a script will run at least 5.5 times as long (at the 80% confidence level). Compared to a Java program, a script will run at least 3.2 times as long.

Search phase only. Finally, we may subtract this run time for the loading phase (z0 data set) from the total run time (z1000 data set) and thus obtain the run time for the actual search phase only. Figure 3 shows the corresponding run times. We find the following:

- Very fast programs occur in all languages except for Rexx and Tcl and very slow programs occur in all languages without exception.
- The median run time for Tcl is longer than that for Python, Perl, and C, but shorter than that of Rexx.
- The median run times of Python are smaller than those of Rexx, and Tcl. They even tend to be smaller than those of Java ($p = 0.13$).

Plots and statistical methods

The main evaluation tool will be the multiple boxplot display, see for example Figure 1. Each of the "lines" represents one subset of data, as named on the left. Each small circle stands for one individual data value. The rest of the plot provides visual aids for the comparison of two or more such subsets of data. The shaded box indicates the range of the middle half of the data, that is, from the first quartile (25% quantile) to the third quartile (75% quantile). The "whiskers" to the left and right of the box indicate the bottom and top 10% of the data, respectively. The fat dot within the box is the median (50% quantile). The "M" and the dashed line around it indicate the arithmetic mean and plus/minus one standard error of the mean.

For quantitatively describing the variability within one group of values we will use the *bad/good ratio*: Imagine the data be split in an upper and a lower half, then the bad/good ratio is the median of the upper half divided by the median of the lower half. In the boxplot, this is just the value at the right edge of the box divided by the value at the left edge. In contrast to a variability measure such as the standard deviation, the bad/good ratio is robust against outliers.

Most interesting observations can easily be made directly in the plots. To be sure, I have also performed statistical tests (please skip the rest of this subsection if you are not interested in these details): Medians are compared using a one-sided Wilcoxon Rank Sum Test (Mann-Whitney U-Test). The result of each test is a p-value, that is, a probability that the observed differences between the samples are only accidental and no difference (or a difference in the opposite direction) between the underlying populations does indeed exist. I will usually not give the p-value itself, but rather say "...is larger than..." if $0 < p \leq 0.10$ or "...tends to be larger than..." if $0.10 < p \leq 0.20$. If $p > 0.10$ there is "no significant difference".

At several points I will also provide confidence intervals, either on the differences in means or on the differences in logarithms of means (that is, on the ratios of means). These confidence intervals are computed by Bootstrapping. They will be chosen such that they are open-ended, that is, their upper end is at infinity. Bootstrapping is described in more detail in [3, 5].

Note that due to the validity caveats of the study these quantitative statistical inference results merely indicate trends; they should not be considered precise evidence.

– The median run times of Perl are smaller than those of Rexx, Tcl, and Java.

– Although it doesn't look like that, the median of C++ is not significantly different from any of the others.

The group-aggregated comparison indicates no significant differences between any of the groups. However, with 80% confidence the run time variability of the scripts is smaller than that of Java by a factor of at least 2.1 and smaller than that of C/C++ by a factor of at least 3.4.

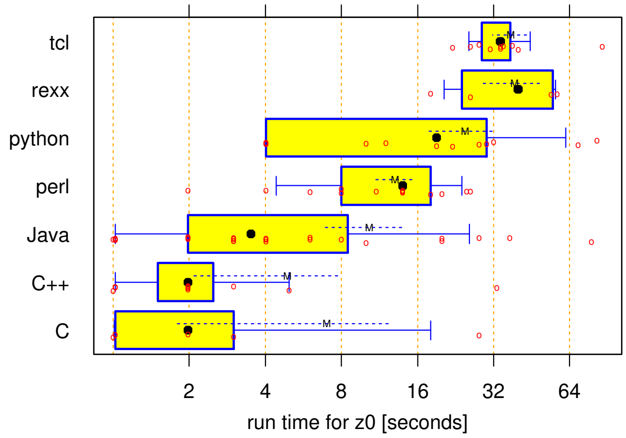

Figure 2: Program run time for loading and preprocessing the dictionary only (z0 data set). Note the logarithmic axis. The bad/good ratios range from 1.3 for Tcl up to 7.5 for Python. ●●

Memory consumption

Figure 4 shows the total process size at the end of the program execution for the z1000 input file. Several observations are interesting:

– The most memory-efficient programs are clearly from the C and C++ groups.

– The least memory-efficient programs are the clearly the Java programs.

– Except for Tcl, only few of the scripts consume more memory than the worse half of the C and C++ programs.

– Tcl scripts require more memory than other scripts.

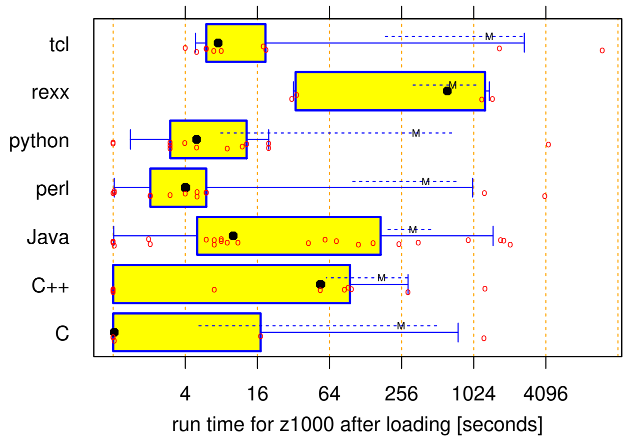

Figure 3: Program run time for the search phase only. Computed as time for z1000 data set minus time for z0 data set. Note the logarithmic axis. The bad/good ratios range from 2.9 for Perl up to over 50 for C++. ●●

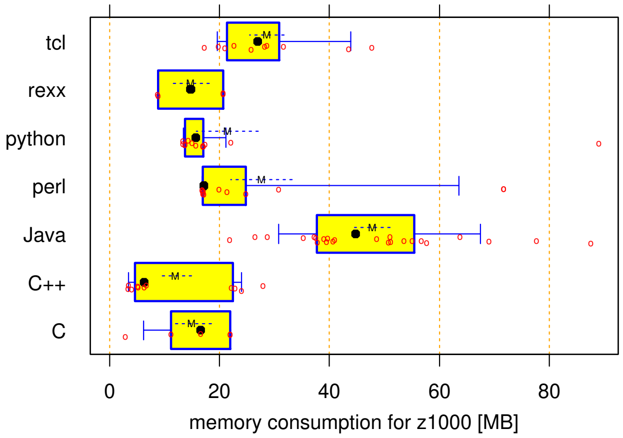

Figure 4: Amount of memory required by the program, including the interpreter or run time system, the program itself, and all static and dynamic data structures. The bad/good ratios range from 1.2 for Python up to 4.9 for C++. ●●

- For Python and Perl, the relative variability in memory consumption tends to be much smaller than for C and in particular C++.
- A few (but only a few) of the scripts have a horribly high memory consumption.
- On the average for the group-aggregated view and with a confidence of 80%, the Java programs consume at least 32 MB (or 297%) more memory than the C/C++ programs and at least 20 MB (or 98%) more memory than the script programs. The script programs consume only at least 9 MB (or 85%) more than the C/C++ programs.

I conclude that the memory consumption of Java is typically more than twice as high as that of scripts, and scripts are not necessarily worse than a program written in C or C++, although they cannot beat a parsimonious C or C++ program.

An observation on the side: Common wisdom suggests that algorithmic programs have a time/memory tradeoff: Making a program faster will usually require more memory. Within our given set of programs, this rule holds for all three non-script languages, but the opposite rule tends to be true for script languages: Those scripts that use more memory actually tend to be slower (rather than faster) than the others.

Program length and amount of commenting

Figure 5 shows the number of lines containing anything that contributes to the semantics of the program in each of the program source files, e.g. a statement, a declaration, or at least a delimiter such as a closing brace.

We see that non-scripts are typically two to three times as long as scripts. Even the longest scripts are shorter than the average non-script.

At the same time, scripts tend to contain a significantly higher density of comments ($p = 0.020$), with the non-scripts averaging a median of 22% as many comment lines or commented lines as statement lines and the scripts averaging 34%.

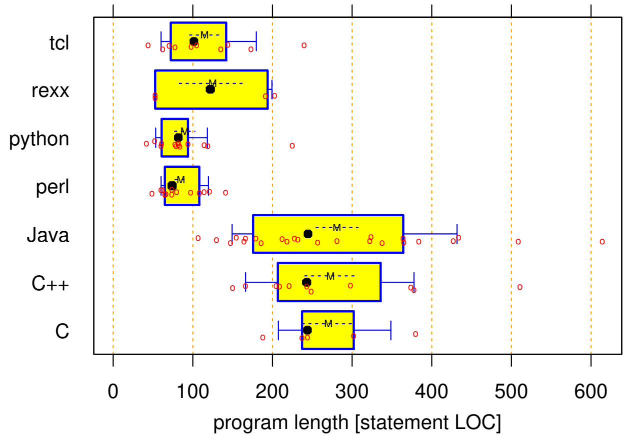

Figure 5: Program length, measured in number of non-comment source lines of code. The bad/good ratios range from 1.3 for C up to 2.1 for Java and 3.7 for Rexx. ●●

Program reliability

With the z1000 input file, 5 programs (1 C, 1 C++, 1 Perl) produced no correct outputs at all, either because they were unable to load the large dictionary or because they were timed out during the load phase. 2 Java programs failed with near-zero reliability for other reasons and 1 Rexx program produced many of its outputs with incorrect formatting, resulting in a reliability of 45 percent.

If we ignore the above-mentioned highly faulty programs and compare the rest (hence excluding 13% of the C/C++ programs, 8% of the Java programs, and 5% of the script programs) by language group, we find that C/C++ programs are less reliable than both the Java and the script programs. These differences, however, all depend on just a few defective programs and should hence not be overgeneralized. On the other hand, since these differences show the same trend as the fractions of highly faulty programs mentioned above, there is good evidence that this ordering of reliability among the language groups in the present experiment is real. Remember that the advantage of the scripts may be due to the better test data available to the script programmers.

Now let us compare the behavior for the more evil-minded input file m1000, which even allows for phone numbers that do not contain any digits at all, only dashes and slashes. Such a phone number should result in an empty encoding, but one does not usually think of such inputs when reading the requirements. Hence the m1000 input file tests the robustness of the programs.

Most programs cope with this situation well, but half of the Java programs and 4 of the script programs (1 Tcl and 3 Python) crash when they encounter the first empty phone number (which happens after 10% of the outputs), usually due to an illegal string subscript or array subscript. 13 of the other programs (1 C, 5 C++, 4 Java, 2 Perl, 2 Python, 1 Rexx) fail exactly on the three empty phone numbers, but work allright otherwise, resulting in a reliability of 98.4%.

Summing up, it appears warranted to say that the scripts are not less reliable than the non-scripts.

Work time and productivity

Figure 6 shows the total work time for designing, writing, and testing the program as reported by the script programmers and measured for the non-script programmers.

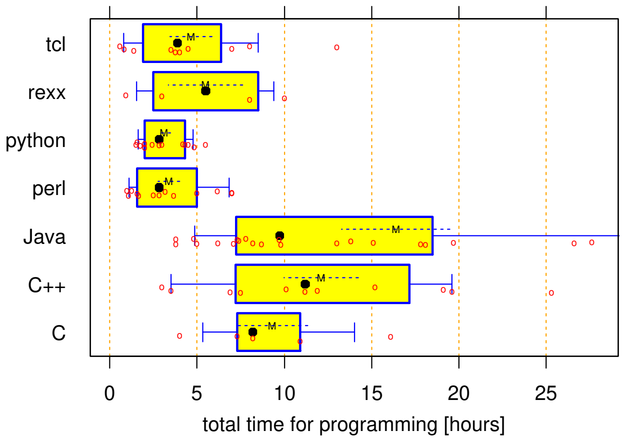

Figure 6: Total working time for realizing the program. Script group: times as measured and reported by the programmers. Non-script group: times as measured by the experimenter. The bad/good ratios range from 1.5 for C up to 3.2 for Perl. Three Java work times at 40, 49, and 63 hours are not shown. ••

As we see, scripts (total median 3.1 hours) take less than half as long as non-scripts (total median 10.0 hours). Keep in mind the validity threats discussed above, which may have exaggerated this difference.

Validation. Fortunately, there is a way how we can check two things at once, namely the correctness of the work time reporting and the equivalence of the programmer capabilities in the script versus the non-script group. Note that both of these possible problems, if present, will tend to bias the script group work times downwards: we would expect cheaters to fake their time to be smaller, not larger, and we expect to see more capable programmers (rather than less capable ones) in the script group compared to the non-script group if there is a difference.

This check relies on an old rule of thumb, which says that programmer productivity measured in lines of code per hour (LOC/hour) is roughly independent of the programming language. Several widely used effort estimation methods explicitly assume that productivity in lines of code per hour is independent of programming language, for instance Boehm's CoCoMo [1] and Capers Jones' programming language table for function point estimation [2].

The validation of our work time data based on this rule is plotted in Figure 7. Judging from the reliably known productivity range of Java, all data points except maybe for the top three of Tcl and the top one of Perl are quite believable.

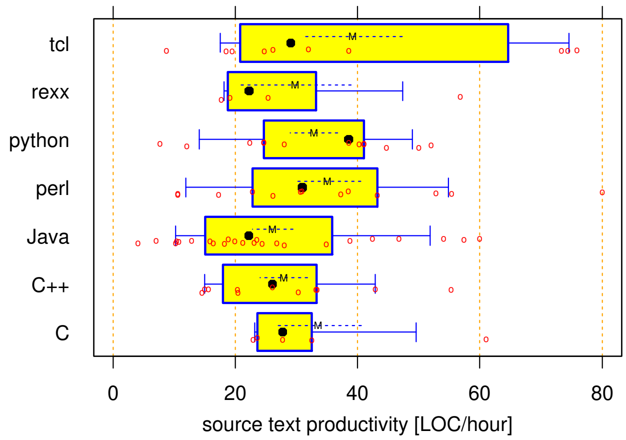

Figure 7: Source text productivity in non-comment lines of code per total work hour. The bad/good ratios range from 1.4 for C up to 3.1 for Tcl. ••

None of the median differences are statistically clearly significant, the closest being Java versus C, Perl, Python, or Tcl where $0.07 \leq p \leq 0.10$. Even in the group-aggregated view with its much larger groups, the difference between C/C++ and scripts is not significant ($p = 0.22$), only Java is less productive than scripts ($p = 0.031$), the difference being at least 5.2 LOC/hour (with 80% confidence).

This comparison lends a lot of credibility to the work time comparison shown above. The times reported for script programming are probably either not at all or only modestly too optimistic, so that a work time advantage for the script languages of about factor two holds. The Java work times appear to be a bit pessimistic, probably due to the lower language experience of the 1997/1998 Java programmers.

Program structure

If one considers the designs chosen by the authors of the programs in the various languages, there is a striking difference.

Most of the programmers in the script group used the associative arrays provided by their language and stored the dictionary words to be retrieved by their number encodings. The search algorithm simply attempts to retrieve from this array, using prefixes of increasing length of the remaining rest of the current phone number as the key. Any match found leads to a new partial solution to be completed later.

In contrast, essentially all of the non-script programmers chose either of the following solutions. In the simple case, they simply store the whole dictionary in an array, usually in both the original character form and the corresponding phone number representation. They then select and test one tenth of the whole dictionary for each digit of the phone number to be encoded, using only the first digit as a key to constrain the search space. This leads to a simple, but inefficient solution.

The more elaborate case uses a 10-ary tree in which each node represents a certain digit, nodes at height n representing the n-th character of a word. A word is stored at a node if the path from the root to this node represents the number encoding of the word. This is the most efficient solution, but it requires a comparatively large number of statements to implement the tree construction and traversal. In Java, the large resulting number of objects also leads to a high memory consumption due to the severe memory overhead incurred per object by current implementations of the language.

The shorter program length of the script programs can be explained by the fact that most of the actual search is done simply by the hashing algorithm used internally by the associative arrays. In contrast, the non-script programs require most of the elementary steps of the search process to be coded explicitly by the programmer. This is further pronounced by the effort (or lack of it) for data structure and variable declarations.

It is an interesting observation that despite the existence of hash table implementations in both the Java and the C++ class libraries none of the non-script programmers used them (but rather implemented a tree solution by hand), whereas for the script programmers the hash tables built into the language were the obvious choice.

Conclusions

The following statements summarize the findings of the comparative analysis of 80 implementations of the phonecode program in 7 different languages:

- Designing and writing the program in Perl, Python, Rexx, or Tcl takes no more than half as much time as writing it in C, C++, or Java and the resulting program is only half as long.
- No unambiguous differences in program reliability between the language groups were observed.
- The typical memory consumption of a script program is about twice that of a C or C++ program. For Java it is another factor of two higher.
- For the initialization phase of the phonecode program (reading the 1 MB dictionary file and creating the 70k-entry internal data structure), the C and C++ programs have a strong run time advantage of about factor 3 to 4 compared to Java and about 5 to 10 compared to the script languages.
- For the main phase of the phonecode program (search through the internal data structure), the advantage in run time of C or C++ versus Java is only about factor 2 and the script programs even tend to be faster than the Java programs.
- Within the script languages, Python and in particular Perl are faster than Tcl for both phases.
- For all program aspects investigated, the performance variability due to different programmers (as described by the bad/good ratios) is on average about as large or even larger than the variability due to different languages.

Due to the large number of implementations and broad range of programmers investigated, these results, when taken with a grain of salt, are probably reliable despite the validity threats discussed in the sidebar. However, it must be emphasized that the results are valid for the phonecode problem only; generalizing to different application domains would be haphazard. It is likely that for many other problems the relative results for the script group of languages would not be quite as good as they are. I conclude the following:

- As of JDK 1.2.1 (and on the Solaris platform), the memory overhead of Java is still huge compared to C or C++, but the run time efficiency has become quite acceptable.
- The often so-called "scripting languages" Perl, Python, Rexx, and Tcl can be reasonable alternatives to "conventional" languages such as C or C++ even for tasks that need to handle fair amounts of computation and data. Their relative run time and memory consumption overhead will often be acceptable and they may offer significant advantages with respect to programmer productivity — at least for small programs like the phonecode problem.
- Interpersonal variability, that is the capability and behavior differences between programmers using the same language, tends to account for more differences between programs than a change of the programming language.

References

[1] Barry W. Boehm. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ, 1981.

[2] Software Productivity Research Capers Jones. Programming languages table, version 7. http://www.spr.com/library/0langtbl.htm, 1996 (as of Feb. 2000).

[3] Bradley Efron and Robert Tibshirani. *An introduction to the Bootstrap*. Monographs on statistics and applied probability 57. Chapman and Hall, New York, London, 1993.

[4] Lutz Prechelt. An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program. Technical Report 2000-5, Fakultät für Informatik, Universität Karlsruhe, Germany, March 2000. ftp.ira.uka.de.

[5] Lutz Prechelt and Barbara Unger. A controlled experiment on the effects of PSP training: Detailed description and evaluation. Technical Report 1/1999, Fakultät für Informatik, Universität Karlsruhe, Germany, March 1999. ftp.ira.uka.de.