DOI: 10.1145/1787234.1787271

BY LUTZ PRECHELT

Plat_Forms Is there one best Web development technology?

SOFTWARE DEVELOPERS DURING THEIR WORK face several fundamental choices with a multitude of options. There are methodological choices, where one selects among possible development processes, and technological choices regarding for instance development tools or base technologies for the software product.

While a number of solid research results are available on some methodological topics (such as review or testing techniques), there is hardly any high-quality information on how specific technologies influence development success. There are narrow benchmarking studies comparing the execution performance of, for example, dababase management systems or application servers, there are feature lists comparing the functionality of base technology products, but there are essentially no holistic studies that compare directly how competing technologies shape a realistic product and how that may influence the project. Those few such studies that exist are very small-scale in terms of the system considered, a good example being jc-cpprt,[2] and despite their small scale some also lack the methodological rigor required for high credibility, a popular example being Kelly's entertaining, yet visibly biased video.[1]

The Problem

The primary reasons for this lack of studies are the difficulty of constancy and the difficulty of relevance. There are plenty of opportunities for comparing apples and oranges (here: different technolgies used on *different* products) and this has been done many times indeed. But as we all know, we cannot learn very much this way. Comparing apples and apples (here: different technologies used for the *same* product) requires higher constancy of the observations, obtained by building the same product at least twice, which is expensive. The only solution is to scale down the size of the product—acceptable up to some point beyond which it is no longer believable that we have sufficient similarity of the product under study to the products relevant in realistic software development. The more affordable the approach to constancy, the more difficult the relevance. And further constancy problems are just around the corner: Software development critically depends on human beings and their performance is known to vary embarrassingly.[5] So we need to replicate more than just twice in order to tell human performance variations apart from technological influences, which makes it even more expensive. But using more affordable programmers (typically students) again threatens relevance: We need to be sure the programmers have sufficient (and comparable) expertise with the technologies used or else we may observe pseudo technology differences that disappear with increased skill.

A Solution

Plat_Forms is a contest designed to hit a sweet spot in the constancy-versus-relevance tradeoff for the comparison of Web development platforms, a *plat-*

form being a programming language plus frameworks and libraries for developing Web-based applications. Such comparison results would be relevant for a rather huge community of software developers, because Web applications are perhaps the most common type of medium-sized custom software being built today. Plat_Forms was announced in October 2006 as follows:

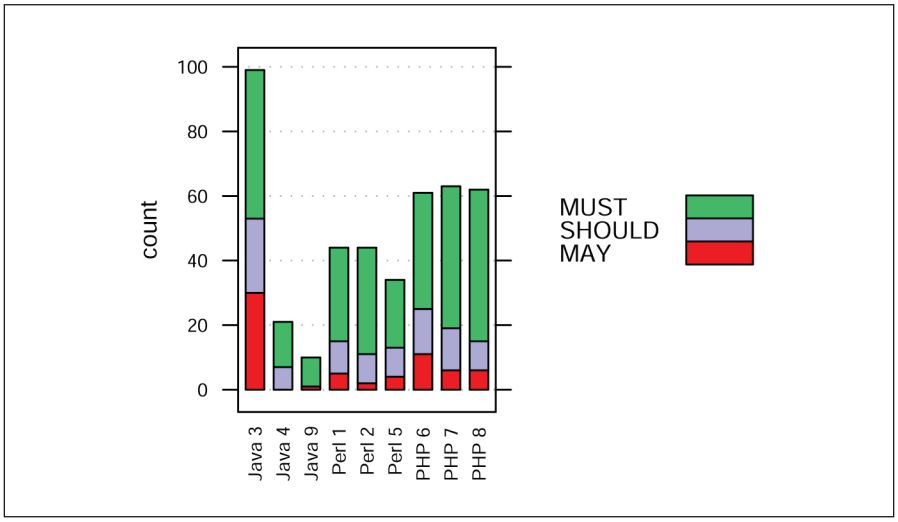
> "Plat_Forms" is a contest in which top-class teams of three programmers compete to implement the same requirements for a Web-based system within 30 hours, each team using a different technology platform (Java EE, .NET, PHP, Perl, Python, or Ruby on Rails). The results will provide new insights into the real (rather than purported) pros, cons, and emergent properties of each platform. The evaluation will analyze many aspects of each solution, both external (usability, functionality, reliability, or performance) and internal (structure, understandability, or flexibility) with scientific rigor.

We found sufficient support for this idea in the Java, Perl, and PHP communities. However, due to our less-than-perfect marketing, only one team from Python and zero from Ruby and .NET applied for participation, so these platforms were not present in the contest. Eventually, we had 27 participants (from five countries) forming nine teams, three each using Java, Perl, and PHP as their platform technology in the contest which took place in January 2007. All 27 were professional software developers, all were skilled with the technology they were using, and the members of each team had worked with one another before. The teams worked without financial compensation.

The Contest

The contest took place in a single large room of more than 400 square meters. Each team brought its own hardware and software setup, had full internet connection, and was allowed to use any previously existing software in the preparation of their solution. The only thing not allowed was external coding help during the contest.

Figure 1



Completeness of solutions in terms of the number of GUI requirements implemented that are working correctly. There were 108 such requirements overall, each of them marked with priority MUST, SHOULD, or MAY. Each bar represents the solution of one team. Two reviewers first independently judged each requirement for each solution and then discussed any discrepancies until they reached agreement.

The contest started on January 25 at 9:00 hours in the morning. The participants received a detailed requirements description[3] that specified 127 fine-grained functional requirements (regarding an HTML GUI and a Web-service) and 23 non-functional requirements. The task was to build a simple community portal involving a personality test and a fairly difficult multi-criterion search functionality.

The contest ended at 15:00 hours the next day. The participants had to submit a DVD containing the VMware image of their server, a source code distribution, and their versioning archive. All teams delivered a working solution on time, but none of these solutions were complete.

In the meantime, the teams were free to work at whatever pace and in whatever style they prefered. All participants took some sleep, although most of them later found they did it less than they should have. Food and drink was provided in the contest room, the hotel was two kilometers away.

The four months of contest preparation and three days of actual contest work were followed by five months of evaluation, performed by five people. The goal of the evaluation was finding characteristics that can be attributed to the platform (rather than to a team). Such characteristics would have to be reasonably similar for the three solutions of platform A, reasonably similar for the three solutions of platform B, but different between platforms A and B. The evaluation was not driven by specific expectations, but rather looked for differences of any kind in any place. Where an evaluation criterion could not be measured objectively, two reviewers assessed it independently and then resolved any differences by discussion.

The Results

The full description of the results covers 11 chapters in the detailed result report,[4] so they are obviously too many to present them all. Let us focus on a few that are particularly interesting: functionality (completeness) of the solutions, size of their source code, the results of some modifiability tests and of some security/robustness tests, and the null results. (In case you wonder: Yes, we would have liked to do performance testing, but it not only exceeded our evaluation capacity but also would not have been very useful because of too-big functionality differences among the solutions.)

If you are interested in more detail about these results, in the results not discussed here, or in a precise description of how we arrived at any of these results, please refer to the full result report.[4]

System completeness: Figure 1 shows how many of the functional UI-requirements each team has implemented. Three observations draw attention:

First, the most complete solution

is a Java solution. Many people might find this surprising, because scripting languages (such as Perl or PHP) tend to have higher productivity at least for small tasks.[2] Some participants suggested as an explanation the fact that team 3 was the only one to use a commercial framework; all others used Open Source components and frameworks of some sort.

Second, the Java results are extremely variable. The most comprehensive solution as well as the two least complete solutions are all on the Java platform. However, this is probably not an attribute of the platform; there are identifiable reasons for the low performance of team 4 and team 9 and they are not Java-specific: team 4 fought with their server's VMware setup for almost a day and lost immense amounts of time; team 9 used a brand-new technology (RAP, Rich Ajax Platform, which brings Eclipse to the Web) that was only in alpha stage at the time of the contest and was simply too incomplete and immature for a better result.

Third, the Perl results and in particular the PHP results are remarkably uniform. The work speed differences of individual programmers are usually fairly large; a factor 2 difference can be expected between the slower and faster half of a group even of professionals.[5] Even if one considers that a small team of three may often exhibit less variation, the similarity in the performance of the three PHP teams is sensational (and at a high level, too). This may indicate a low risk of unwelcome surprises with respect to the development time required for PHP, which would be an important platform strength.

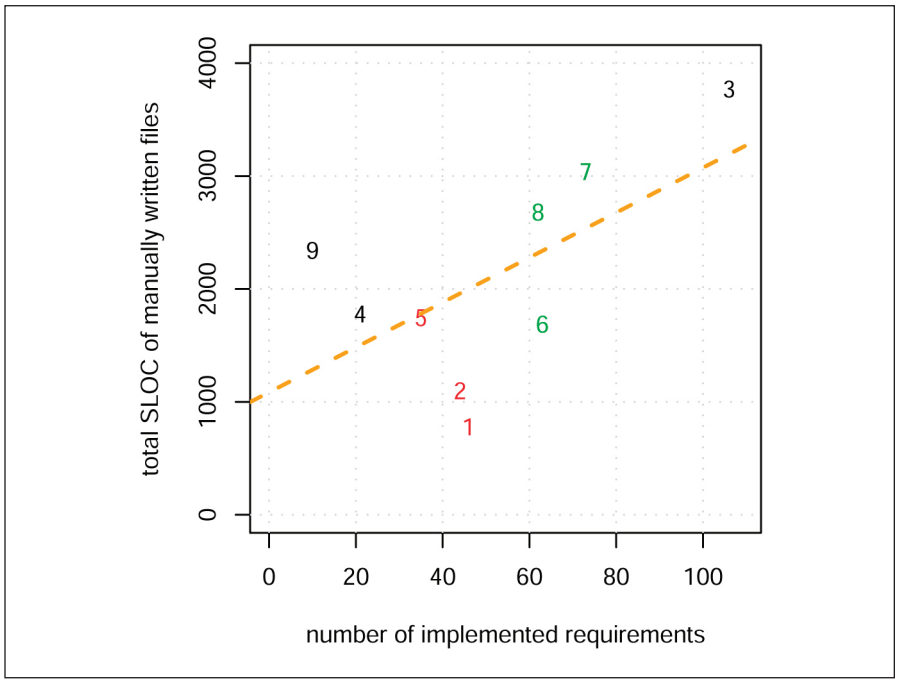
Implementation size: Figure 2 compares how much code and other files the teams have written from scratch during the contest relative to the amount of functionality they realized. Points above the trend line thus indicate comparatively verbose solutions, lines below indicate terse ones. We see that all three Java solutions are somewhat lengthy while two of the Perl solutions are rather compact — both of these results are in accordance with common expectations.

Modifiability: We investitated maintainability by looking how two simple extensions would have to be implemented for each of the solutions. The modifications required were both more extensive and more difficult-to-find for the Java solutions than for the others. The Perl solutions exhibited the simplest and most straightforward modifiability for both extensions, PHP was similar for one of the two. For the second extension, that straightforward Perl approach would have been the clearly most appropriate one even if the overall system had been much larger. A cultural difference between powerful but heavyweight Java approaches and pragmatic, lightweight Perl approaches was quite obvious.

Robustness/security: Figure 3 summarizes the results of a few simple black-box tests of robustness and security issues that we performed. In short, these tests were (left to right): handling of HTML-tags in user input (danger of cross-site scripting), handling of very long inputs, handling chinese ideograms, email address validity checking, handling of attempts at SQL injection, operation with cookies turned off. Except for two solutions that allow for cross-site scripting, the PHP results are fairly solid. The only team to come out of all tests without a failure is a PHP team. Only the Java solutions are all immune against cross-site scripting. Actual SQL injection is probably *not* possible in any of the solutions except maybe one of the Perl solutions. See the full report for details.

Other: Finally, we found a large number of null results: For many of the criteria that we checked, there were bigger differences among the solutions *within* one platform than there were when comparing the averages *across* platforms—no consistent platform differences were to be seen in these respects. I find this encouraging: It tells us that the creators of each of the platforms were competent engineers; that each platform provides a good supply of powerful and practical mechanisms and that hence the results in a particular project are far more dependent on the performance of the individuals using the platform than on the platform itself.

Figure 2

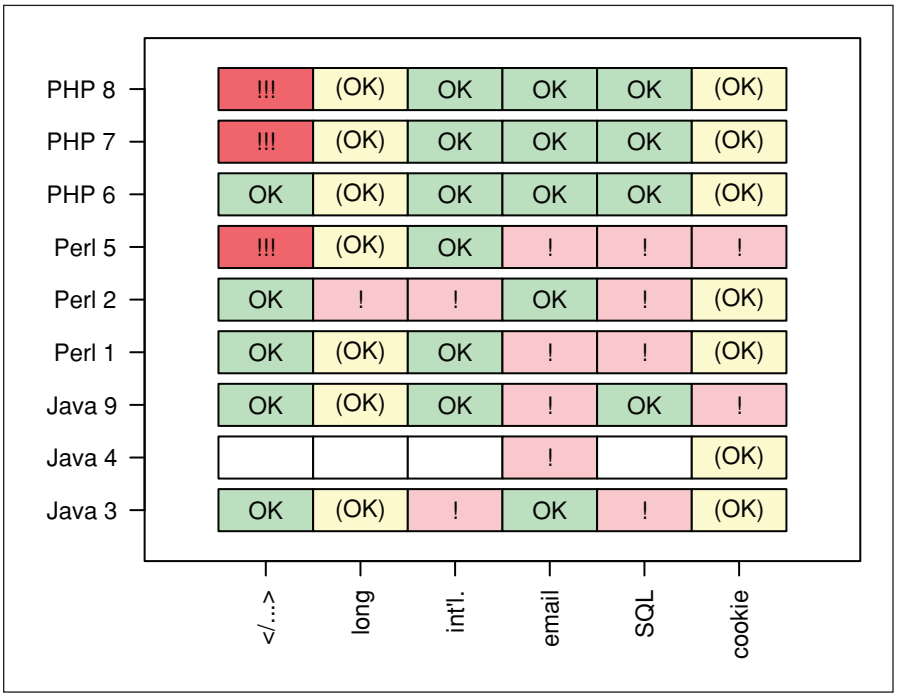


Total size of the solutions in source lines-of-code (SLOC), shown in relation to the number of functional requirements implemented (this time including the Web service requirements). Each digit represents the respective team (black for Java, red for Perl, green for PHP); the line is the overall trend line. The SLOC count ignores all files that were reused or automatically generated, even if they were modified manually, and includes only files written entirely by hand.

So What?

How can one sum up what this first Plat_Forms contest has found? First, it has confirmed some of the "I have known this for years"-type prejudices, for example, that Java solutions are large and Perl solutions small. Second, it has overthrown some negative beliefs, dearly held by ad-

Figure 3



Summary of the robustness test results for all nine teams. Each column represents one robustness test. Green color "OK" means correct, yellow "(OK)" means acceptable, light red "!" means broken, and bright red "!!!" means security risk. White areas indicate results that could not be evaluated due to gaps in functionality. Details are available in the full report.[4]

vocates of competing platforms: Is Java always unproductive? Looking at team 3's success, it is obviously not. Is Perl outdated? No. In contrast, the small size of the solutions and their good modifiability suggest that Perl may be a particularly strong platform with respect to maintainability, at least when we consider the Perl culture to be part of the platform – which is quite appropriate from a customer's point of view. Are PHP applications less secure? Not when used appropriately, as demonstrated by the clean results of team 6. The common denominator: All platforms allow for good success, but you need highly competent people to make it happen. Third, Plat_Forms has thrown up two questions that may be worth more discussion than they currently get: Are the best commercial frameworks still strictly superior to Open-Source-based approaches when it comes to development productivity (team 3's high completeness results)? Are there platform differences with respect to development productivity *risk* (the PHP teams' uniform completeness results)?

What Next?

After this successful pilot of Plat_Forms, we would like to have a similar contest next year: Hopefully with participation from all major platforms; possibly with a twist to address one of the above questions. If you know somebody who might be interested in participating or in becoming a sponsor, I'll be happy to hear from them. And in the meantime: Do not doubt your favorite platform, master it!

Referernces

1. Kelly, S. Better Web application development. NASA Jet Propulsion Laboratory, 2006, http://oodt.jpl.nasa.gov/better-Web-app.mov.
2. Prechelt, L. An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program. Technical Report 2000-5, Universität Karlsruhe, Fakultät für Informatik, (Mar. 2000); ftp.ira.uka.de.
3. Prechelt, L. Plat_Forms 2007 task: PbT. Technical Report B-07-03, Freie Universität Berlin, Institut für Informatik, (Jan. 2007); www.plat-forms.org.
4. Prechelt, L.Plat_Forms 2007: The Web development platform comparison—Evaluation and results. Technical Report B-07-10, Freie Universität Berlin, Institut für Informatik (June 2007); www.plat-forms.org.
5. Prechelt, L. The 28:1 Grant/Sackman legend is misleading, or: How large is interpersonal variation really? Technical Report 1999-18, Universität Karlsruhe, Fakultät für Informatik, (Dec. 1999); ftp.ira.uka.de.

Lutz Prechelt (prechelt@inf.fu-berlin.de) is a professor of software engineering at Freie Universität Berlin and performs empirical studies of software development.