Plat_Forms: Contests as an Alternative Approach to SE Empirical Studies in Industry

Lutz Prechelt, Ulrich Stärk
Freie Universität Berlin
Institut für Informatik
Takustr. 9, 14195 Berlin, Germany
Email: prechelt|ustaerk@inf.fu-berlin.de

***Abstract*—Plat_Forms is a set of three studies that search for process and product properties emerging from the use of specific web development platforms. Its first key property is the use of a contest format; this leads to easier, broader, and more uniform data collection compared to on-site industrial studies. As a second key property, the three instances do not merely serve to extend or corroborate each others' results, they also serve as a time series for characterizing changes in the underlying software development reality over the course of a few years – an important aspect that has received little attention in empirical software engineering research so far.**

I. Plat_Forms

Plat_Forms is a set of three quasi-experiments[1]. Its goal is to compare industrial development technologies (tools, frameworks, components) using realistic teams and development processes, but factoring out the complexity effects of industrial-sized tasks and workplace effects. The key idea of the Plat_Forms research approach is to represent the study to the outside world as a *contest*.

A. Description

The three Plat_Forms studies were carried out in 2007, 2011, and 2012. The external form of each was that of a contest. The abstract of the 2007 announcement [1] read as follows:

> "Plat_Forms" is a competition 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, performance, etc.) and internal (structure, understandability, flexibility, etc.).

We asked for (and largely got) jelled teams of professionals to apply for participation with the technology they mastered best. The teams would work as volunteers; we had promised only fun and fame, not fortunes[2]. 9 teams participated in 2007 (3 using Java, 3 Perl, 3 PHP), 16 teams in 2011 (4 Java, 1 JavaScript (non-competitively), 3 Perl, 4 PHP, 4 Ruby), and 8 teams in 2012 (1 Groovy, 3 Java, 1 .NET, 2 PHP, 1 Ruby; see discussion in Section II-D).

All teams met in one place for two-and-a-half days (first afternoon for setting up their own equipment, then two days for the actual contest) and they all worked together in one or two large rooms.

The task resembled an idealized, short prototyping project performed at a customer's site: The teams received a detailed, clear, precise written specification of what they should implement, including functional requirements, non-functional requirements, complete must/should/may priorities, and a few constraints [2]–[4]. A customer representative was available to promptly answer any requirements question. The teams worked in an unfamilar, yet largely interruption-free workplace, each team sitting together closely. They were required to release the code produced during the contest under an open source software license and to provide run-time licenses for all other software components they used.

The teams had full internet access and were allowed to use any tool, document, software component, etc. available anywhere and even feedback from external testers on their publicly set-up intermediate solutions, but were not allowed to employ additional design or implementation help beyond their three members.

At the end of the two-day timeframe, each team supplied us with

- a virtual machine running their solution,
- a source code distribution of their solution,
- their complete version archive,
- on the next day: a separate post-mortem questionnaire from each team member regarding his[3] experience with the task and with the team's development process, plus some personal background information.

In addition, we collected some process information manually by person-by-person activity type sampling with a fixed set of candidate activity types at 15-minute intervals throughout the work time.

After each contest, these data were analyzed over about 6 months by a team of three to five people.

[1] See http://www.plat-forms.org

[2] There was a small monetary price for the best team on each platform eventually in 2007 and 2011.

[3] All team members were male.

B. Research Perspective and Design Considerations

The Plat_Forms research questions is: *Which characteristic properties of product or process typically emerge from using a specific web development platform that differ from those of (some) other platforms?*

We initially defined a platform as a programming language plus the union of all web-related frameworks, tools, libraries, and components available for it. (The definition later changed, see the discussion in Section II-D.)

Despite the huge relevance of the web development domain and the intense competition of several popular platforms, no scientific empirical comparisons of web development platforms exist. Other trustworthy and objective information is also extremely scarce [5].

Therefore, we designed Plat_Forms to be exploratory and went to look for platform differences in all feasible directions at once: Whenever any observable characteristic of process or product would be similar within a platform yet meaningfully different (and again consistent) on another platform, we would consider this a platform difference. The exact notion of "similar" was not predefined: Some of the investigations (e.g. of maintainability) would be qualitative, not quantitative and even for the quantitative ones we could not hope for statistical significance with only about 3 teams per group.

In order to avoid the research becoming meaningless and given this vague operationalization, we needed a high level of comparability in the raw observation data to start from, i.e. strong control of many variables. We hence designed Plat_Forms as a quasi-experiment: a trial with non-randomized assignment of teams to platforms, but full control of the non-team-related variables. Randomized assignment is obviously infeasible: no team would be interested in participating if it knew it would likely not use its prefered platform. It would also shift the research question, as we would then get to see effects from the learnability of different platforms rather than effects from skilled routine use.

C. Discussion

If all goes well, the contest format will provide nice properties: low-threshold acquisition of companies to participate in the research[4]; a high level of control; easy and organized data collection; a large number of organizations in the resulting data set; the capability to use far larger tasks than would be possible in on-site field experiments that use task replication; consequently broader analysis and results; the opportunity to observe well-delineated team work as well as individual work.

The corresponding disadvantages are: observation of only greenfield development and detachment from the complexities induced by large existing products (with respect to both product and process); few (if any) workplace-specific effects will be visible; the danger of false positive results (fishing for significance).

[4]The contest format makes it easy for the company to understand the size of their investment, avoids disruptions of the on-site process, and reduces secrecy concerns. Also, we found the lack of shyness to publicly demonstrate their performance astonishingly low in our participant companies.

II. Year-by-Year Results and Insights

The purpose of the present article is a discussion of the Plat_Forms research approach as such, not the platforms-related results. Section II-A explains in global terms (that apply to all three executions of the contest) our experience as to the strong and the problematic aspects of the methodological approach taken by Plat_Forms.

The subsequent sections then go through the executions one by one. Each section first mentions some results of the respective execution in terms of the Plat_Forms research question and then describes the resulting insights that prompted us to make methodological adjustments for the next execution.

A. What Went Well or Not So Well

By and large, Plat_Forms was a good success. On the positive side: We had sufficiently many teams of sufficiently similar quality and we managed to uncover a number of platform differences, some of them in line with common prejudice, others against it. It required some effort to obtain those teams, but hugely less than starting an on-site collaboration with as many companies.

On the negative side:

- Problem 1: We had two mishaps with the teams.
  a.) In 2007, one Java team used immature technology resulting in big problems and low productivity that did not represent the Java platform. This team was in the contest only because we had begged them to come after we had been unable to raise a proper third Java team.
  b.) In 2011, one Perl team was not a jelled team at all; the members had found each other via a mailing list and only during the contest recognized that nobody of them was interested or skilled in HTML front-end development – they were all server-side specialists. As a result, the user-visible functionality they delivered was close to zero although they had implemented the innards for most requirements.
  Both of these particular events could in principle have been avoided, but it is difficult to prevent all types of similar stumbles completely.
- Problem 2: The evaluation effort is higher than expected. The techical heterogeneity involved due to the different platforms makes even seemingly trivial tasks (such as measuring lines-of-code consistently) laborious.
- Problem 3: Particular evaluations may even fail completely for technological reasons. In 2007 for instance, we wanted to measure dynamic coupling based on method-level who-calls-whom information collected by a runtime profiler. But one profiler returned nonsensical results (e.g. negative numbers of calls) and even though there was a profiler expert in one of the teams we were clueless how to resolve this and decided to abandon the whole evaluation.
- Problem 4: Again due to the heterogeneity, evaluating aspects that cannot be measured mechanically (e.g. maintainability) require enormous amounts of expertise.

The positive aspects mentioned above suggest the study design model of Plat_Forms (namely to collect data in a contest context) will likely be useful for some other research questions as well. As for the problems described subsequently: While problems 2 to 4 are specific to studies that compare several complex technologies, problem 1 will likely apply to most studies using a contest format.

B. 2007 Insights

The first execution found a number of expected platform differences (e.g., Java solutions tended to be larger (in terms of LOC per implemented requirement) and Perl solutions tended to be smaller than others) as well as a number of surprising platform differences. For instance, the most complex of our change scenarios for evaluating maintainability found the high-ceremony approach of the Java teams to lead to the *lowest* maintainability and the pragmatic approach used by the teams on the ill-reputed Perl platform to exhibit the highest [6, Section 13]. This difference appeared for cultural reasons, with no technical necessity. Also, our robustness tests (which served as a very rough assessment of likely security) neither found Java as the most likely secure platform nor the ill-reputed PHP as the least likely secure one. More detail is provided in a TSE article [5] and full detail of both results and evalution methods is given in a technical report [6].

As for insights, problem 1a) mentioned above made us recognize that having only three teams per platform was too risky: If any team got into team-specific (rather than platform-related) problems, the remaining two would be too few to diagnose consistent platforms differences. We decided to allow four teams per platform in subsequent executions of Plat_Forms in order to be able to exclude one from the analysis in pathological situations.

C. 2011 Insights

The second instance of Plat_Forms four years later had some results that confirmed previous ones (e.g. the size-related results), some that were new (e.g. Ruby, not present in 2007, was the most productive platform), and some that documented general trends (e.g. the robustness results had improved throughout all platforms). Some of the new ones were very interesting, e.g. the Ruby teams spent more time writing automated tests than the teams of all other platforms, while the Java teams spent more time testing manually than the teams of all other platforms. More detail is provided in [7].

The central insight was related to the very core of Plat_Forms, the definition of "platform". Doubt had been nagging already in 2007 and now it became undeniable: The programming language is not a good criterion for classifying web development platforms. Within each of the most popular languages (Java, PHP), there were frameworks with very different structure and different consequences for development even in 2007 and this trend had continued to 2011. At the same time, many web frameworks had continued to adopt similar design ideas and philosophies *across* languages and those ideas led to similar outcomes. For instance, one (and only one) PHP team could keep up with the Ruby teams in terms of productivity; this team (and only this) used the Symfony framework, which is inspired by the ideas underlying Ruby-on-Rails.

Summing up, Plat_Forms 2011 showed that the original design of the study had reached its limits and needed an overhaul.

D. 2012 Approach

Our original approach for Plat_Forms 2012 was to de-emphasize the role of the web development frameworks by asking for the implementation of a scalable RESTful web service (using Amazon Web Service technology and without any HTML front-end) rather than a complete web application. We announced the contest in this form and came out essentially empty-handed: Only 5 teams applied for participation; many others stated they did not feel competent enough with the new cloud technologies.

We canceled the contest and developed a different route instead: Classify platforms by the structural properties of and development styles suggested by the various frameworks. This means that frameworks using different languages can end up in the same platform (e.g. Groovy/Grails, PHP/Symfony, and Ruby/Ruby-on-Rails all end up in a Rails-like class) and frameworks using the same language can end up in different platforms.

For this classification we scanned 476 different web development frameworks, reviewed 156 of them, and analyzed 44 of those: we tabulated many key properties, selected those few properties that have plausibly grave effects, and created multiple different classifications (usually only binary). This work is ongoing.

As a consequence, we dropped the requirement to have three teams per language in the 2012 contest, which was lucky because we received only 12 applications for participation, of which only 8 eventually showed up in the contest (after 24 applications in 2011).[5]

The evaluation of the 2012 solutions has not progressed far enough to present specific results, but despite the low number of only 8 teams, many of the various binary framework classifications will work out in the sense as to yield classes with at least three representatives so as to allow evaluation.

III. The Time Series Aspect of Plat_Forms

So far most families of studies in software engineering repeat a study several times very similarly, each time varying one important aspect (such as the specific set of subjects, the general type of subjects, or the particular task to be solved) in order to corroborate or differentiate existing findings to broaden their generalizability. Plat_Forms is also of this type:

[5]Despite many attempts to find out, we are still not sure how this so-much-lower number came to be; it appears to be a combination of a too-short repetition rhythm (only 12% of the 2011 teams applied in 2012, versus 33% of the 2007 teams that had applied in 2011; all teams had reported they had enjoyed the contest in both years) and simply bad luck.

One purpose of the 2011 contest was certainly corroborating or invalidating the 2007 results.

However, Plat_Forms adds another aspect: The changes of the underlying software engineering reality over time. The time series of studies spanned by Plat_Forms' subsequent executions served to better understand the trends that have changed web development overall between 2007 and 2012 as well as how these trends are reflected in specific platforms.

At the level of detail possible in this short article, we have already described (in Sections II-C and II-D above) what this time series aspect did to Plat_Forms in terms of Plat_Forms' research question.

From a more general research method perspective, the platforms define the groups in a quasi-experiment and we find (1) drift in the composition of each individual group as well as (2) of the overall notion of what defines those groups.

Both of these kinds of drift make research more complicated: (1) The within-group drift makes it harder to corroborate previous results as it will often be unclear whether diverging results in a later sub-study are contradicting the previous result or merely describe a change of the underlying reality.

(2) The drift of the group-defining notion either

- makes it harder to design an *appropriate* next instance of a study family at all (if the notion is kept as it was)
- or makes it necessary to re-evaluate the data of the previous instances (if the new notion can sensibly be applied to the old study instances)
- or breaks the coherence of the study family (if the new notion *cannot* sensibly be applied to the old study instances).

IV. Conclusion

We offer three take-home messages from our discussion of the Plat_Forms example.

A. Consider Contests as a Data Collection Approach

Phrasing a study as a contest is a trick that can be applied to many more research questions than only ours. When applied successfully, the contest format drastically reduces data-collection effort compared to on-site data collection within organizations, while keeping many properties of technologies and processes used. At the same time, it allows to broaden the set of organizations that go into the dataset far beyond what would otherwise be feasible.

The price paid for these advantages consists of ripping the work out of its normal context of products and workplace.

B. Non-Randomized Experiments Can be Superior

Software engineering researchers are used to thinking of non-randomized-but-otherwise-controlled experiments (also known as quasi-experiments) as a poor man's replacement of "proper" controlled experiments.

Plat_Forms demonstrates that this is not always an appropriate view: Whenever sociological phenomena are relevant for the practices observed in a study (and they usually are), the self-assignment of individuals to certain groups will be a relevant influence in practice and should therefore not always be suppressed in an empirical study.

Rather, the people who develop a preference for a certain group (such as a specific web development platform) will often have different talents, preferences, and attitudes than people developing a preference for a different group[6] Over time, being a member of this group will also make them develop different skills and reinforce attitudes and existing behavioral differences as exemplified by the Java/Perl maintainability difference mentioned in Section II-B. A quasi-experiment can capture and reflect such effects; a randomized experiment will suppress them.

C. Drift is Important

Finally, we suggest that software engineering research should pay more attention to the fact that the software world is constantly changing. Physicists can safely assume that the same laws and relationships hold in the physical universe today that held two decades ago — we should not. It would often be more helpful if our studies aimed at rough estimations of a trend rather than precise snapshots of a transient status quo.

Acknowledgments

Plat_Forms 2007 would not have been feasible without the help from Heise Zeitschriftenverlag and Richard Seibt and Eduard Heilmayr of the OSBF. Plat_Forms 2011 and 2012 were supported by a DFG research grant. Additional sponsoring came from Accenture, ICANS, Immobilien Scout, LF.net, Microsoft, Optaros, and Zend.

References

[1] L. Prechelt, "Plat_Forms – a contest: The web development platform comparison," Freie Universität Berlin, Institut für Informatik, Germany, Technical Report TR-B-06-11, October 2006.

[2] ——, "Plat_Forms 2007 task: PbT," Freie Universität Berlin, Institut für Informatik, Germany, Technical Report TR-B-07-03, January 2007.

[3] U. Stärk and L. Prechelt, "Plat_Forms 2011 task: CaP," Freie Universität Berlin, Institut für Informatik, Germany, Technical Report TR-B-11-10, January 2011.

[4] ——, "Plat_Forms 2012 task: Cafman," Freie Universität Berlin, Institut für Informatik, Germany, Technical Report TR-B-12-08, October 2012.

[5] L. Prechelt, "Plat_Forms: A web development platform comparison by an exploratory experiment searching for emergent platform properties," *IEEE Transactions on Software Engineering*, vol. 37, no. 1, pp. 95–108, January/February 2011.

[6] ——, "Plat_Forms 2007: The web development platform comparison — evaluation and results," Freie Universität Berlin, Institut für Informatik, Germany, Technical Report TR-B-07-10, April 2007, www.plat-forms.org. [Online]. Available: http://www.plat-forms.org/2007/documents/platformsTR.pdf

[7] U. Stärk, L. Prechelt, and I. Jolevski, "Plat_Forms 2011: Finding emergent properties of web application development platforms," in *Proc. 6th Int'l. Symposium on Empirical Software Engineering and Measurement*. ACM Press, 2012.

[6] If you do not believe this, take the rather entertaining test at http://alvar.a-blast.org/plat_forms/ and look at all teams before you decide. With respect to the above argument, the test is tongue-in-cheek but makes the point well: People can recognize developers' platform preferences far better than allowed by chance.