VerifyThis – Verification Competition with a Human Factor

Gidon Ernst[1], Marieke Huisman[2(✉)], Wojciech Mostowski[3], and Mattias Ulbrich[4]

[1] University of Melbourne, Melbourne, Australia
gidon.ernst@unimelb.edu.au
[2] University of Twente, Enschede, The Netherlands
m.huisman@utwente.nl
[3] Halmstad University, Halmstad, Sweden
wojciech.mostowski@hh.se
[4] Karlsruhe Institute of Technology, Karlsruhe, Germany
mattias.ulbrich@kit.edu

Abstract. VerifyThis is a series of competitions that aims to evaluate the current state of deductive tools to prove functional correctness of programs. Such proofs typically require human creativity, and hence it is not possible to measure the performance of tools independently of the skills of its user. Similarly, solutions can be judged by humans only. In this paper, we discuss the role of the human in the competition setup and explore possible future changes to the current format. Regarding the impact of VerifyThis on deductive verification research, a survey conducted among the previous participants shows that the event is a key enabler for gaining insight into other approaches, and that it fosters collaboration and exchange.

Keywords: VerifyThis · Program verification · Specification languages · Tool development · Competition

1 Introduction

The VerifyThis program verification is held in 2019 for the 8th time; earlier editions were held at FoVeOOS 2011 [6], FM 2012 [15,20], Dagstuhl (April 2014) [4], and ETAPS 2015–2018 [16–18,21], the next event takes place as part of TOOLympics at ETAPS 2019 [2]. On the VerifyThis webpage[1] the aim of the competition is formulated as follows:

– *to bring together those interested in formal verification, and to provide an engaging, hands-on, and fun opportunity for discussion, and*

[1] See http://www.pm.inf.ethz.ch/research/verifythis.html.

Author names are in alphabetic order.

© The Author(s) 2019
D. Beyer et al. (Eds.): TACAS 2019, Part III, LNCS 11429, pp. 176–195, 2019.
https://doi.org/10.1007/978-3-030-17502-3_12

– *to evaluate the usability of logic-based program verification tools in a controlled experiment that could be easily repeated by others.*

The competition will offer a number of challenges presented in natural language and pseudo code. Participants have to formalize the requirements, implement a solution, and formally verify the implementation for adherence to the specification.

There are no restrictions on the programming language and verification technology used. The correctness properties posed in problems will have the input-output behaviour of programs in focus. Solutions will be judged for correctness, completeness, and elegance.

What we would like to emphasise up-front is that VerifyThis is different from most other competitions of formal method tools at TOOLympics[2] at ETAPS 2019, see the TOOLympics proceedings [2] (this volume) for more details on each of them. Typically, the other events run a (large) number of benchmarks as batch jobs to determine the winner from values that are obtained from the invocations (like the runtimes or the number of successes) in a fully mechanised way.[3] Moreover, often they target both proving and *disproving* examples. There are also other TOOLympics competitions in which software verification tools are compared: SV-COMP [3] and RERS [23], which are both off-site events focussing on automatically checkable system properties that do not require user input.

In contrast, VerifyThis deliberately takes the user into the loop, and only considers proving correctness. VerifyThis challenges are developed under the assumption that there is currently no technique available out there that can run the problem in a widely accepted input specification format out of the box. Part of the challenge – and in many cases also the key to a successful solution – is to find a suitable logical encoding of the desired property, and to come up with smartly-encoded sufficiently strong annotations, i.e., specification engineering. Understanding the problem is essential for solving the challenges, the human factor can thus definitely not be removed.

In this paper, we discuss the current set-up of the competition, and our experiences from the past editions (Sect. 2). In addition, we also critically reflect on the current organisation, and discuss whether it still matches the competition's aims. For this purpose, we have investigated feedback and experiences from earlier participants (Sect. 3). From the participants' feedback and our experiences, we conclude that VerifyThis indeed is an engaging and fun experience. However, it is less clear whether the current setup indeed evaluates the capabilities of the tools used, or if also other things are measured. Therefore, in Sect. 4 we make several suggestions of possible changes to the setup that could make the measuring aspects of the competition more precise.

[2] https://tacas.info/toolympics.php.

[3] A notable exception are the *evaluation-based rewards* of the RERS [23] competition where submitted approaches and solutions are reviewed and ranked by the challenge organisers.

2 Previous Editions

The format of the competition has been rather stable since its first edition (see [19] for the reflections of the organisers after the first VerifyThis edition), with fine-tuning changes made whenever it was felt that this was appropriate. In this section we discuss: who are the organisers, how do we define the challenges, who are participating, what side events do we organise, and what are the results of the competition.

Organisers. The first editions of VerifyThis were run by the same group of organisers (Marieke Huisman, Rosemary Monahan, and Vladimir Klebanov (until 2014), and Peter Müller (since 2015)). Since 2016, this part of the organisation has changed a bit. The original organisers created a steering committee, which invites a new pair of organisers every year. They work in close collaboration with one or more steering committee members to define the challenges, and are fully responsible for judging the solutions. There are several advantages to the approach: it ensures that there are sufficient fresh ideas, it avoids a repeated bias on a single technique, it widens the community, and it allows the steering committee members to also participate themselves. The two organisers are always selected with the following criteria in mind: they should be familiar with the area of program verification; at least one of them should have been participating in an earlier edition of VerifyThis; and they should be from different groups, in order to involve the community as a whole as much as possible.

Challenges. To involve the community in the competition since 2012 a call for challenges has been published widely – and the submitted challenges regularly form the basis for one of the challenges set during the competition.

There is a wide variety of program verification tools used by the participants, and no particular input programming (or specification) language has been set. Therefore, problems are either presented in a standard well-known programming language or in pseudo code, and no obligatory formal specification is given, neither in logics nor in a particular specification language. If a natural language specification is given, it is formulated as precisely as possible, showcasing the problem with exemplary situations. Good challenges move the participants out of their comfort zone: they do not immediately know how to solve it, and will have to think about how to use their tool to actually solve the challenge.

Challenges are inherently "real". If a person is expected to look into a problem and understand it, the problem cannot be a generated routine that only exposes a challenge for verification tools, but it must have a sensible purpose beyond verification. Typical problems are algorithmically challenging routines, which are (possibly simplified) real-world snippets from larger code bases.

The competition typically consists of three challenges and the participants have 90 minutes to work on each one. The first is usually a relatively simple warm-up challenge – often involving a computation on the elements of an array. The other two challenges are typically more involved. Often one of them is about a complicated heap-based data structure that for example requires reasoning about operations on a binary tree. Since 2015, the third challenge typically deals with

concurrency – however, as not all tools participating in the competition support reasoning about concurrency, the challenge is always set up in such a way that it also has a sequential version. As an illustration of the kind of effort required at VerifyThis, a solved, automatically provable by most tools solution to the warm-up challenge from the FoVeOOS'11 competition [6] is shown in Fig. 1.

The maximum element property of the following array traversing procedure has to be shown:

```
int max(int[] a) {
  int x = 0, y = |a| - 1;
  while(x != y)
    if(a[x] <= a[y]) x++; else y--;
  return x;
}
```

where $|\cdot|$ stands for array length. Under the assumption (precondition) of a non-null and non-empty input array a, i.e. $a \neq \text{null} \wedge |a| > 0$, the procedure correctness assertion (postcondition) can be expressed as $\forall_{0 \leq i < |a|}\ a[i] \leq a[r]$, where r is the procedure result. The required `while` loop invariants to show this property are $0 \leq x \leq y < |a|$, $\forall_{0 \leq i \leq x}\ a[i] \leq a[x] \vee a[i] \leq a[y]$ and $\forall_{y < j < |a|}\ a[j] \leq a[x] \vee a[j] \leq a[y]$ with the $y - x$ termination measure. Teams express the procedure and specification in their tool's specific notation, in particular, the loop invariants can take different equivalent forms, many of which are more compact, yet might be more difficult to read, see [6] for the complete range of solutions.

Fig. 1. Search by elimination VerifyThis challenge from FoVeOOS'11 competition.

At the end of the 90 minutes, all teams are asked to submit their solutions (also if they are only partial) to the organisers by email. These are the versions that will be judged. However, teams sometimes also send a more complete version later, as a kind of evidence how close they were to the full solution. This happens in particular if somebody completes the challenges in the break right after the challenge was finished.

The full collection of earlier challenges (with links to polished solutions) is available from the VerifyThis webpage. This collection also serves as a benchmark set (beyond the competitions) in the program verification community, in particular because it enables comparison in verification efforts and approaches for different verification tools.

Participants & Tools. Over the years, the number of participants in VerifyThis has grown slightly. The very first editions of the competition had about 6 to 8 teams participating; the more recent ones had 10 to 12 teams participating. Most teams are "developer teams", i.e., their members are actively working on the development of the tool (sometimes even during the competition). However, we have also had several non-developer teams participating, and in particular Dafny [28] is widely used. We specifically encourage participation of students/PhD candidates. The most remarkable participation was a Dafny team at

ETAPS 2016 which was formed by Bachelor students from the Technical University of Eindhoven (where ETAPS was located that year). They had read about the competition, and then taught themselves the basics of Dafny to participate in the competition. Many of the participants joined the competition multiple times: in general they find the competition quite engaging, and will try to come back the next year.

Most of the tools are deductive program verifiers, which have explicit support for imperative programming constructs (in contrast to theorem provers for mathematical logic) and explicit support for assertion languages of various flavours. There are major differences in the way proofs are developed and checked, in the degree of automation, and the programming and specification features. Nevertheless, the common aim of these deductive tools is full-functional correctness proofs. We have also had several tools used in the competition that fall outside of this classification, such as the bounded model checkers CIVL [32] and CMBC [27], the model checker mCRL2 [9], the interactive theorem prover Isabelle [30], and the termination prover AProVE [13].

Table 1 below gives an overview of all the tools that participated in the competition, the number of times a team participated using the tool, and how many times a team using the tool actually won a first prize or first student prize.

Side Events. As VerifyThis is an on-site competition, it means that it also provides an opportunity for the program verification community to meet and exchange ideas, establish and improve personal contacts, and to see, experience and learn from each other's tools. To encourage this exchange, we organise several side events around the competition.

Since several years, before the competition itself starts, we therefore have an invited tutorial on one of the program verification tools. So far, we have tutorials about Dafny (Rustan Leino), Why3 (Jean-Christophe Filliâtre), and Viper (Alexander J. Summers). We encourage the presenter to explain the main characteristics of the tool, and to provide a challenge for the audience, so they get hands-on experience with the tool. This tutorials are open to non-competition participants as well, though typically it attracts only a few extra attendees.

Furthermore, on the evening of the competition, we organise a dinner for all participants, where they can talk about their experiences during the day. Usually, almost all participants join for the dinner, and there is a good, bonding atmosphere.

Finally, the next day the judges (usually, the organisers who set the challenges) talk with all teams privately to evaluate their solutions. The versions submitted by email form the basis for the discussion, and participants are given the chance to explain their formalisation and which parts of the challenges they have solved. Judges ask for clarifications and general questions (cf. Sect. 4.2). Experience has shown that for the judges these discussions are very helpful for understanding the solutions and the taken approaches, and thus for judging them. As teams use different tools, without the explanation, the solutions are much harder to understand and assess, and the judges might miss aspects of the solutions.

Table 1. Overview of tools with teams participating in VerifyThis[a]

Tool	**# of teams participating**	**# of prizes won** **O**verall/**S**tudent/**F**eature
AProVE [13]	1	
AutoProof [35]	1	
CBMC [27]	1	
CIVL [32]	3	F:1
Dafny [28]	12	
ESC/Java [8]	1	
F* [34]	1	
Frama-C [24]	2	
Isabelle [30]	1	O:1
jStar [10]	1	
KeY [1]	7	
KIV [11]	5	S:3
mCRL2 [9]	2	F:1
MoCHi [26]	1	
PAT [33]	1	
Spark/Ada/GNATprove [14]	1	F:1
VCC [7]	1	
VerCors [5]	5	
VeriFast [22]	4	O:2
Viper [29]	2	F:1
Why3 [12]	9	O:2, S:4, F:1

[a]Please note that prizes have not been awarded every year, and sometimes two prizes have been awarded in a single category.

In parallel, the participants meet among themselves and present their solutions amongst each other. As all participants have been intensively thinking about the same problem the day before, these discussions really help to gain insights into how other program verification tools work, and their relative strengths and weaknesses. This session occasionally is also attended by other conference participants.

Competition Results. In most editions of VerifyThis prizes have been awarded (see Table 1 for an overview). The prizes that are usually awarded are:

- best overall team,
- best student team, and
- distinguished tool feature.

Occasionally, the judges have decided to award a second prize in some category, or to hand out two prizes (this happened in particular in the category of

distinguished tool feature). Thanks to our sponsors, we usually have been able to hand out not only a certificate, but also a financial reward. No further order on the participating teams is given.

In addition, in some years we have also had a prize for the best submitted challenge, or the tool used by most teams. However, even though related to the competition, these prizes are not for the competition effort itself, and are not further discussed here.

Judging is done by considering the following aspects of the submitted solution:

- How close is the solution to a complete solution, i.e., how much work will it be to finish verification of the code w.r.t. the implementation?
- Did the team capture all the relevant properties to be verified in the specification?
- How understandable and accessible are the specifications?

In general, the judges do not penalise the use of auxiliary annotations such as loop invariants or intermediate assertions. Because of the time constraint, a tool requiring many auxiliary annotations, already has a drawback. Often the judges find it relatively straightforward to decide about a winner (and are relieved that no further ordering on teams is required). In some cases, the decision required more discussion, and careful re-examining of the submitted solutions.

3 The Impact of VerifyThis

In order to asses the impact of VerifyThis we conducted an online survey among all previous participants. The survey consisted of three parts: (1) General questions, such as number of times participated, the current position, participation as student and/or developer, (2) an assessment of recent advances and the state-of-the-art of deductive verification tools relative to several categories of tool qualities and features, and (3) the participant's personal take-away from the competition, including the impact it had on his/her research and career, as well as feedback to the organisers. The questionnaire is included in Appendix A.

For the second part, we asked the participants more specifically for their opinion about: which progress in recent years they considered most important, which aspects could have the most impact if they were improved, and how this is reflected in the development of the participants' own verification tools. We were interested specifically in the following categories, with an additional possibility of submitting free form responses.

- Expressiveness and ease of use of specification languages
- Proof automation and guidance
- Integration with static analysis techniques (e.g. invariant inference)
- Verification debugging and counterexample generation
- Specification and proof refactoring

Position	# responses	developer
PhD/MSc student	5	0
Postdoc	3	3
Academic	6	5
Industry	2	2
Overall	16	10

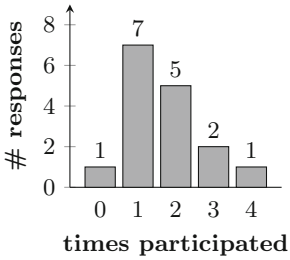

Fig. 2. Background of participants who took part in our questionnaire

Results. We received 16 responses from the approximately 80 previous participants that we contacted. Figure 2 shows the distribution of their current positions, respectively, whether they are tool developers, and how many times they have participated. Note that there is one response of a person having not participated in the competition itself (but presumably in the side events).

Figure 3 shows the responses on the current state of verification tools w.r.t. the five categories, ranked on a scale with four items. Based on the responses, the participants agree that advances in all of these categories have been made, and significantly so in expressiveness, automation, and debugging. However, no participant felt that a major breakthrough had been achieved in any of the categories. Additional remarkable improvements that were mentioned in the free-form responses were proof support for safety and liveness properties, the automation of separation logic, and integration of tools into development environments.

Regarding potential impact if major breakthroughs *were* to be achieved, the most common answer was proof automation, followed by debugging capabilities, and further advances in the expressiveness of specification languages. Integration of static analysis into deductive tools was typically considered of minor importance. The free-form answers furthermore mentioned ease of use and graphical interfaces, maturity and predictability of tools, integration into development process and existing codebases. One answer suggested to address different properties separately, i.e., separate functional specifications from canonical concerns such as memory safety and race-freedom.

Participants who are also developers indicated a number of improvements to their tools related to all of the above categories, partly in response to the experience of the competition. The majority of completely new features was related to expressiveness of specification languages, and one mention of each proof automation and debugging, respectively. One free-form answer mentioned, however, that major investments into all of the categories are planned.

Another result from the survey is that verification challenges serve as benchmarks or regression tests of the tools, with five answers indicating 9 or more challenges to be used in this way, and seven answers indicating between 3 and 6 challenges used.

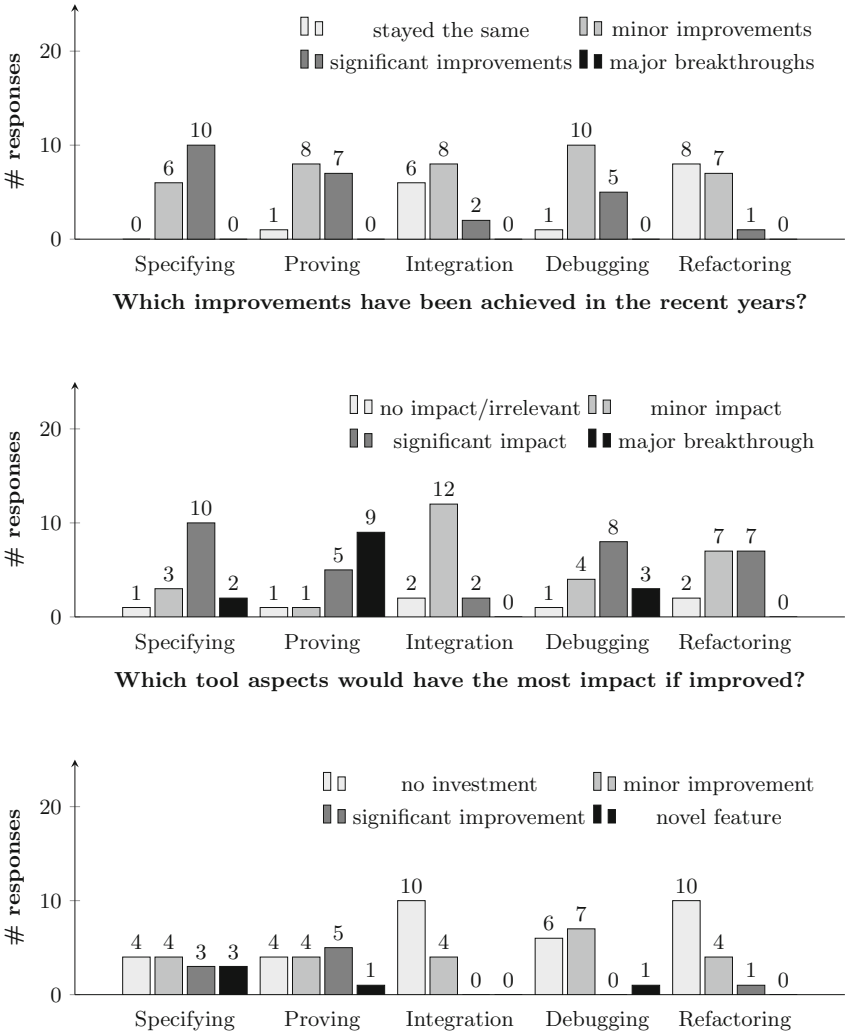

Fig. 3. Participants' assessment on the current state of deductive verification tools

All participants of the survey stated that they had enjoyed solving the challenges, and almost all indicated they particularly liked the exchange between colleagues and learning about how other tools tackle challenges. The participants were less excited about the Jury discussions (9 answers) and the presentation sessions (7 answers with a suggestion that these should be more formally organized). An additional free-form response appreciated the publications associated with the competitions that summarize the results and discuss the solutions.

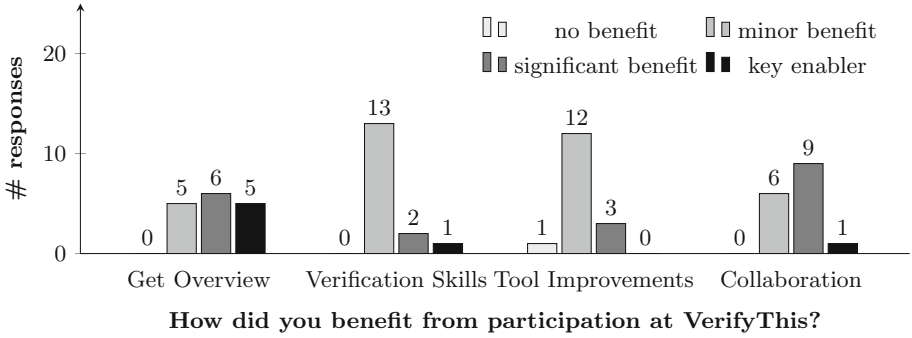

Fig. 4. Participants' personal benefit from the VerifyThis events

Participating in the competition lead to personal take-away regarding the following aspects: getting an overview and learning about state-of-the-art techniques and tools, improving one's own ability in specification and verification, improving one's own tools for day-to-day use, and establishing academic contacts or collaborative research. The results are shown in Fig. 4: The most common and greatest benefit, as indicated by the participants, is thereby to obtain a better insight into other approaches.

We have received several suggestions to improve future instalments. Some answers were related to potential off-line participation (reminiscent of VSComp [25]), potentially for a subset of the challenges to facilitate participation. One response suggested to release a "prepare for this" exercise beforehand, and one response suggested to release a more difficult off-line challenge. There was the suggestion to release partial information on the nature of the challenges in advance, i.e., which tool and library features would be helpful, to ease preparation. We were also encouraged to increase the variety of verification problems.

Discussion. The feedback from the participants sheds some light into the mostly academic perspective on the state-of-the-art and recent advances in deductive verification tools.

The response rate of 20% was less than what we had hoped for. However, clearly many of the younger participants of the earlier instalments are likely to have completed their degrees and thus moved to another institution or industry.

Since the answers to the three tool-related questions had the same format, we can attempt to investigate how well current research is aligned with those aspects that are thought to be critical. Of the 66 data points for the question on improvements made to the tools, there were 34 indications of "no investment" to a particular aspect, and of these the majority of 26 answers is related to integration with static analysis, debugging, and proof maintenance. While the latter two features have been identified to have critical importance in industrial context [31], they seem to be less important in academic verification projects, which are often at a smaller scale w.r.t. the software being built, as well as the team involved. Integration with static analysis, on the other hand, is arguably

a less active research area, and from personal communication we can report scepticism on the usefulness of e.g., automatically inferred invariants.

Similarly, considering the *motivation* of improving the tools' capabilities, almost all of the answers relate to *personal benefit* to the developers (and academic users), i.e., related to solving the competitions better and to support ongoing research. Only three (free-form) answers were related to other stakeholders (industry, customers, non-experts).

The personal impact of VerifyThis was overwhelmingly positive, with 7 replies indicating that participation was the key enabler for the respective category. We would also remark that 7 of 11 participants mentioned that improvements to their tools were inspired by features of other tools observed at the competition.

Outlook. The more general question with respect to the impact of VerifyThis is in which way can VerifyThis be understood as a controlled experiment? Which measurements can be taken for a systematic better assessment of the potential and improvements of modern verification tools over time? Given the diversity in approaches, tools, and levels of experience of the participants in relation to their (relatively) small numbers of participants and the great effort to develop challenges and solutions, it is not likely that events like VerifyThis can arrive at statistically sound scientific conclusions any time soon.

However, it is possible to keep track of some descriptive measures, similarly to the data obtained by this survey, as a proxy that would provide an ongoing and semi-rigorous evaluation that is independent of the individual challenges and VerifyThis instalments. We therefore plan to conduct similar surveys on a routine basis as part of the competition event. This will provide a more thorough, up-to-date, and ongoing assessment of the field, in addition to the results reported here.

4 The Human Factor

The feedback from Sect. 3 is very encouraging and suggests that VerifyThis has succeeded as a community event, i.e., having achieved its first goal. However, it is much less clear in which sense the current format of the competition including the evaluation and summarization done in the corresponding publications, constitutes to an experimental assessment of the usability of verification tools, i.e., the quality of user guidance, and feedback in case of failed verification attempts, to tackle real verification problems? How do we even measure this?

As mentioned before, the crucial aspect in this discussion is that VerifyThis takes the human into the loop. In fact, there are several ways in which the outcome of a task depends on the person(s) performing the task, i.e., where the *human factor* becomes visible, namely, through the abilities of the participants during the competition to solve the challenge using their respective tools, as well as through the ability of the judges to compensate for the varying tool contexts and the need to be objective about the quality of (often partial) solutions.

4.1 The Human Factor in the Competition

Most competitions in the area of formal methods are *unsupervised*, i.e., fully automated tools are run on a batch of challenges without human interaction and the ranking is determined from the results that they produce (and, possibly, their runtimes/memory consumption). VerifyThis is a supervised competition since challenges are not submitted to a fully automatic analysis.

One aspect in the success of solving a particular problem, at a high-level, is the experience of a competition participant with respect to the problem's characteristics (e.g., whether it involves pointer structures, concurrency, ...). This determines how hard or easy one may find it to come up with suitable invariants, for example, or to employ clever approaches that lend themselves to an elegant solution, mathematically.

As an example, even a seemingly trivial property like sortedness can be formalised in different ways, either stating that any element is not greater than its successor ($a[i] \leq a[i+1]$ for all $i < |a| - 1$), or stating that any element is not greater than all succeeding elements ($a[i] \leq a[j]$ for all $i < j < |a|$). Note that in order to derive the second formulation from the first one, an explicit induction is needed, and hence the second one is strictly more "powerful" when one may assume, e.g., a sorted input. Depending on the challenge, choosing the right encoding may be the enabling key to a successful verification. In general, finding the ideal encoding, the ideal function contract or the ideal loop invariant can require a considerable amount of creativity and ingenuity.

Another aspect is that such intuition must be formalized into a concrete representation of the specification within the confines of the deductive verification tool. This task is usually more than a straightforward logical encoding of natural language properties. Not only could logical choices (as the one above) critically affect whether the automation can find a proof (at all resp. within a reasonable time limit). Even benign things like the order of conjuncts can make a significant difference. As a consequence, effective use of a verification tool may require significant and detailed knowledge of the internal mechanics of the tool itself and the verification infrastructure it is built on.

The central question regarding the goal of the VerifyThis competition is, hence, whether it is

- a competition in which humans compete about their capabilities to perform difficult verification tasks verification, *or*
- a competition in which the capabilities, strengths or weaknesses of the participating verification tools come to light.

The conciliatory answer to this is that VerifyThis combines both, as these characters are inherently entangled by the nature of the field itself: Deductive program verification for challenging, algorithmic problems with heavyweight properties is far from begin a push-button technology–and probably always will be for sufficiently complex challenges. Human and tool must play together to succeed. Moreover, in all but trivial cases, a challenge will not be solved in one go, but

requires an iterative process towards the final solution. The design of the VerifyThis competition reflects these aspects and thus mirrors reality in this respect. The human factor is not added *per se* as an on-top feature to the competition, but arises as an integral part of the specification and verification process. Furthermore, the human factor brings to light the qualities of a verification tool in the interactive process. For instance,

- usability and intuitiveness, in particular of the provided error messages,
- degree of automation,
- responsiveness (how easy is it to try a slightly changed specification),
- facilities to debug failed verification attempts, e.g. by producing counterexamples for failing specifications,
- the quality of counterexamples and their presentation,
- and the quality of a tool's specification libraries

all manifest themselves through the human factor. To measure these aspects, the human operator needs to be involved in the process and its evaluation. The in-vitro character of the competition emphasizes the human factor since it takes much experience to successfully interact with the tool under the tight time constraints.

How can the competition and the challenges be designed to control the influence of the human factor?

Ideally, one would like to separate the abilities of the human expert from the usability and performance of a tool when assessing the solution of a challenge. Due to the mentioned entanglement, this is difficult. Even worse, missing experience or unfamiliarity with a particular part of the verification system or type of specification, may be a showstopper for a team during the competition time. Several ideas for the design of the competition have emerged that would allow one to control the role of the human in the process, in particular by reducing its impact.

Reduce the need for human creativity: If crucial proof-guiding annotations (e.g. invariants) cannot be found, a solution to a challenge may become stuck in early stages. To mitigate this factor, the challenge description could contain logical formulations of such annotations. These hints could also be provided in a closed envelope, to be opened at the discretion of the team only, or half way through the time available for the challenge. This challenge scheme where part of the solution is given away, suggests itself particularly for the warm-up challenge where the solution is usually not so particular to the applied verification technology.

As an alternative, instead of an algorithm-driven challenge, we could provide a specification and ask to provide a verifiable implementation.

Reduce the need for experience: Experience with program verification in general and with a particular verification tool have a prominent impact on the results of the competition. To lessen this effect, one of the challenges could be solved by ad-hoc teams composed during the competition. This has the potential to bring together different experience levels and tool expertises, and would also provide a great opportunity for knowledge transfer.

Decouple tools from their users: Verification tools may have a tendency to be (over)fitted to the specification and verification style of their developers. To lighten this bias, we could do a cross-validation experiment, where teams are asked to reproduce a solution of another team, in their own specification methodology using their own technique.

Another possible cross-validation experiment is to reserve one of the challenges as a competition of tool A vs. tool B (judged separately). This can be incorporated into the tutorial session, where both these tools could be presented but the audience is leveraged for a more systematical evaluation. Such an effort could also be done off-line, similarly to the Isabelle competition[4].

4.2 The Human Factor in the Judging

There is a second human factor involved with supervised competitions: Judging cannot be automated to the same degree as it can with unsupervised competitions. For the latter, ranking schemes can still be biased for particular tools or approaches, but at least the criteria are defined a priori. Manually crafted solutions are usually not comparable by pre-definable metrics, and require careful examination. Therefore, for the judges, the most intensive activity of the competition with substantial time urgency is the evaluation process to arrive at the prize decision: the complete judging for all the teams and their solutions takes just one (long) day. This activity is certainly receptive to the judges' subjective views and tastes, and thus another human factor.

The judges have to consider all the possible specification and verification aspects in the solutions – parts that have been done, parts that could have been done, and parts that were only completed to a certain degree, as well as the automation level and tool support aspects. At the same time, the teams being interviewed concentrate on the best and completed parts of their solution. Both sides also tend to have a technology specific view – the teams look at the solution and possible improvements from the point of view of their tool and method, while the judges, even though staying impartial, would have their own expertise and tool bias. This is especially true considering that the judging committee is now different every competition instance and coming with their own expertise, expectations, and often first time experience approach.

Defining objective criteria: In this context, one of the ideas that we would like to implement in the future instalment of the competition to reduce the biases and to optimize the judging process is a challenge solution form that the teams should fill in along with the submission. The form would include generic questions about the solution completeness, e.g.,

- "Have you specified the main functional property?",
- "To what degree were you able to prove it?",
- "Have you specified/proved the termination/memory safety/non-interference/... properties?",

[4] See https://competition.isabelle.systems.

- "Are the proofs automatic? If not, what is the user interaction effort?",
- "Is the incompleteness of your solution due to insufficient proof guidance (e.g., too weak invariants), or due to tool or method shortcomings?",
- "Estimate how much time you would need to complete the task?", etc.

Systematic judging process: Such questions would also give the teams the chance to preliminary self-evaluate the solution before the discussion and prepare some answers up-front. To not occupy the challenge solving time, this form can be easily filled in between the challenge closing time and the judging, nevertheless it should be obligatory.

A structured interview after the competition also helps to mitigate the human factor and use it to our advantage: By explicitly querying about the usability and interaction support of the tool (e.g., guided by the usability issues listed in Sect. 4.1), both weaknesses and strengths can be learned by inspecting the impact of the human factor during the competition. This feedback can then again help developers to improve the user experience of their tools. One question that was typically asked previously during judging was "which tool feature did you find most helpful", in order to determine the corresponding prize.

Another possibility is to integrate the judging and the team presentations into a single event. This opens up the opportunity to involve all participants in the judgement process through consensus (e.g. a voting or scoring scheme), thereby avoiding potential bias of the judges on the competition's outcome.

These suggestions can help in answering questions related to completeness of solutions and usability of the tool. It still remains difficult to check whether a given solution does in fact formalize the requirements adequately, i.e., whether it is a *correct* solution. Answering this question is highly non-trivial as it involves not only understanding the specification language of the tool, but also its meta-theory and verification approach and what is, semantically, implied by proving a particular statement. An example for this was last year's third concurrency challenge, which involved a lock-free data structure [17]: How fine-grained is the concurrency model of the tool? How do the synchronization primitives work? Such aspects can be illuminated in the dialogue between the judges and the participants only.

Finally, one criterion where the human factor is intentionally brought into the judging process is "elegance". While elegance affects the ranking much less than completeness and correctness of solutions, it may serve as a tie-breaker, and is often recognized by singling out certain solutions in the competition reports.

5 Conclusion

We set out to reflect on the organisation of VerifyThis, discussed the competition's format and impact to come up with several concrete ideas to improve future events.

The survey in Sect. 3 showed that VerifyThis leads to an intense exchange between participants, allowing them to gain a unique overview of the state-of-the-art and establishing academic collaboration. The personal contact between

the participants is thereby a major strength of VerifyThis. VerifyThis has also led to concrete improvements to the verification tools including a few completely novel features.

In Sect. 4 we illustrated that the human factor in the competition is inherent both in solving the challenges as well as in the judging. Human interaction (e.g., by providing a suitable encoding in the specification or by providing auxiliary annotations) is indispensable in deductive program verification of sophisticated properties. The human factor can thus not be fully eliminated from the competition – nor should it be. The discussion led to a few suggestions responding to the involved human factor: We identified a number of possible modifications of the modalities of the challenges regarding the composition of the teams and the design of the challenges. To mitigate the influence of the human factor in the judging, we suggest to aid the process by questionnaires filled out by the participants themselves.

Finally, we think it is important to widen our reach for a more diverse set of tools that implement different approaches, such as software model checkers and tools to synthesise specifications and programs that are correct by construction, as attempted in Dagstuhl in 2014 [4].

The VerifyThis competition enriches the portfolio of the TOOLympics as it differs from other competitions by explicitly incorporating the tool's user into the process.

Acknowledgement. We thank Microsoft Research, Amazon Web Services, Galois, and Formal Methods Europe for their support and generous sponsorship of VerifyThis over the last years. We thank Rosemary Monahan for suggestions to improve the competition format and feedback on the manuscript.

A Survey Questions

As part of the celebration 20's anniversary of TACAS, we are writing an article on the VerifyThis competition. In contrast to previous publications on the series (which emphasized the practical technical aspects), we would now like to focus on the higher-level perspective that relates the competition to the field, the community, as well as your personal view.

A.1 General Questions

- How many times have you participated? [1–6]
- What is your current position? [Undergraduate, PhD/MSc, Postdoc, Academic, Other...]
- Have you participated as a student? [Yes, No, Both]
- Have you participated as a tool developer? [Yes, No]

A.2 Tool Improvement

- Which improvements in deductive verification tools do you think have been achieved in the recent years?
  Scale: [Stayed the same, Minor improvements, Significant improvements, Major breakthroughs that have or will change the field]
  Categories:
  - Expressiveness and ease of use of specification languages
  - Proof automation and guidance
  - Integration with static analysis techniques (invariant inference, shape analysis, ...)
  - Verification debugging and counterexample generation
  - Specification and proof refactoring
- Which tool aspects do you think could have the most impact if they were improved?
  Scale: [No impact/irrelevant, Minor impact, Significant impact, Major breakthroughs that would change the field]
  Categories: (as above)
- Are there any other future improvements that you think need to happen?
- If you are a developer: Which changes to the tool were improved or implemented as response to the experience at the competition?
  Scale: [No investment, Minor improvements, Significant improvements, Novel feature previously not present]
  Categories: (as above)
- Are there any other future improvements that you would like to add to your tool?
- If you are a developer: What was the motivation for adding new features?
  - Missing feature required solve certain competition challenges
  - For research unrelated to the competition
  - Improvements to the verification process
  - Other: ...

A.3 Personal Take-Away

- How did you benefit from participation?
  Scale: [Did not benefit, Minor benefit, Significant benefit, Major benefit that was primarily enabled through participating at VerifyThis]
  Categories
  - Learn about state-of-the-art techniques and tools
  - Improve own ability in specification and verification
  - Improve own tool in day-to-day use
  - Establish academic contacts or collaborative research
- How many of the VerifyThis challenges from the past serve currently as a benchmark/test in the development of your tool?

– Which aspects of the event did you particularly enjoy?
  - The challenge problems & solving them
  - Presentation sessions among the participants
  - Discussions with the jury
  - Exchange with colleagues
  - Leaning how other approaches tackle things
  - Other: ...
– How could future instalments be improved?

References

1. Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., Schmitt, P.H., Ulbrich, M. (eds.): Deductive Software Verification - The KeY Book: From Theory to Practice. LNCS, vol. 10001. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49812-6
2. Bartocci, E., et al.: TOOLympics 2019: an overview of competitions in formal methods. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) TACAS 2019. LNCS, vol. 11429, pp. 3–24. Springer, Cham (2019)
3. Beyer, D.: Automatic verification of C and Java programs: SV-COMP 2019. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) TACAS 2019. LNCS, vol. 11429, pp. 133–155. Springer, Cham (2019)
4. Beyer, D., Huisman, M., Klebanov, V., Monahan, R.: Evaluating software verification systems: benchmarks and competitions (Dagstuhl Reports 14171). Dagstuhl Rep. **4**(4), 1–19 (2014)
5. Blom, S., Darabi, S., Huisman, M., Oortwijn, W.: The VerCors tool set: verification of parallel and concurrent software. In: Polikarpova, N., Schneider, S. (eds.) IFM 2017. LNCS, vol. 10510, pp. 102–110. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66845-1_7
6. Bormer, T., et al.: The COST IC0701 verification competition 2011. In: Beckert, B., Damiani, F., Gurov, D. (eds.) FoVeOOS 2011. LNCS, vol. 7421, pp. 3–21. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31762-0_2
7. Cohen, E., et al.: VCC: a practical system for verifying concurrent C. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 23–42. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03359-9_2
8. Cok, D.R., Kiniry, J.R.: ESC/Java2: uniting ESC/Java and JML. In: Barthe, G., Burdy, L., Huisman, M., Lanet, J.-L., Muntean, T. (eds.) CASSIS 2004. LNCS, vol. 3362, pp. 108–128. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30569-9_6
9. Cranen, S., et al.: An overview of the mCRL2 toolset and its recent advances. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 199–213. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36742-7_15
10. DiStefano, D., Parkinson, M.: jStar: towards practical verification for Java. In: ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, pp. 213–226. ACM Press (2008)
11. Ernst, G., Pfähler, J., Schellhorn, G., Haneberg, D., Reif, W.: KIV: overview and VerifyThis competition. Int. J. Softw. Tools Technol. Transfer **17**(6), 677–694 (2015)
12. Filliâtre, J.-C., Paskevich, A.: Why3 — where programs meet provers. In: Felleisen, M., Gardner, P. (eds.) ESOP 2013. LNCS, vol. 7792, pp. 125–128. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37036-6_8

13. Giesl, J., et al.: Proving termination of programs automatically with AProVE. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS (LNAI), vol. 8562, pp. 184–191. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08587-6_13
14. Hoang, D., Moy, Y., Wallenburg, A., Chapman, R.: SPARK 2014 and GNATprove - a competition report from builders of an industrial-strength verifying compiler. STTT **17**(6), 695–707 (2015)
15. Huisman, M., Klebanov, V., Monahan, R.: VerifyThis verification competition 2012 - organizer's report. Technical report 2013-01, Department of Informatics, Karlsruhe Institute of Technology (2013). http://digbib.ubka.uni-karlsruhe.de/volltexte/1000034373
16. Huisman, M., Monahan, R., Mostowski, W., Müller, P., Ulbrich, M.: VerifyThis 2017: a program verification competition. Technical report, Karlsruhe Reports in Informatics (2017)
17. Huisman, M., Monahan, R., Müller, P., Paskevich, A., Ernst, G.: VerifyThis 2018: a program verification competition. Technical report, Inria (2019)
18. Huisman, M., Monahan, R., Müller, P., Poll, E.: VerifyThis 2016: a program verification competition. Technical report TR-CTIT-16-07, Centre for Telematics and Information Technology, University of Twente, Enschede (2016)
19. Huisman, M., Klebanov, V., Monahan, R.: On the organisation of program verification competitions. In: Klebanov, V., Beckert, B., Biere, A., Sutcliffe, G. (eds.) 1st International Workshop on Comparative Empirical Evaluation of Reasoning Systems (COMPARE 2012). CEUR Workshop Proceedings, vol. 873. CEUR-WS.org (2012)
20. Huisman, M., Klebanov, V., Monahan, R.: VerifyThis 2012. Int. J. Softw. Tools Technol. Transf. **17**(6), 647–657 (2015)
21. Huisman, M., Klebanov, V., Monahan, R., Tautschnig, M.: VerifyThis 2015. A program verification competition. Int. J. Softw. Tools Technol. Transf. **19**(6), 763–771 (2017)
22. Jacobs, B., Smans, J., Piessens, F.: Solving the VerifyThis 2012 challenges with VeriFast. STTT **17**(6), 659–676 (2015)
23. Jasper, M., et al.: RERS 2019: combining synthesis with real-world models. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) TACAS 2019. LNCS, vol. 11429, pp. 101–115. Springer, Cham (2019)
24. Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-C: a software analysis perspective. Formal Asp. Comput. **27**(3), 573–609 (2015)
25. Klebanov, V., et al.: The 1st verified software competition: experience report. In: Butler, M., Schulte, W. (eds.) FM 2011. LNCS, vol. 6664, pp. 154–168. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21437-0_14
26. Kobayashi, N., Sato, R., Unno, H.: Predicate abstraction and CEGAR for higher-order model checking. In: Hall, M.W., Padua, D.A. (eds.) 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2011), pp. 222–233. ACM (2011)
27. Kroening, D., Tautschnig, M.: CBMC – C bounded model checker - (competition contribution). In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 389–391. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_26
28. Leino, K.R.M.: Dafny: an automatic program verifier for functional correctness. In: Clarke, E.M., Voronkov, A. (eds.) LPAR 2010. LNCS (LNAI), vol. 6355, pp. 348–370. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17511-4_20

29. Müller, P., Schwerhoff, M., Summers, A.J.: Viper: a verification infrastructure for permission-based reasoning. In: Jobstmann, B., Leino, K.R.M. (eds.) VMCAI 2016. LNCS, vol. 9583, pp. 41–62. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49122-5_2
30. Nipkow, T., Wenzel, M., Paulson, L.C. (eds.): Isabelle/HOL: A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45949-9
31. O'Hearn, P.W.: Continuous reasoning: scaling the impact of formal methods. In: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, pp. 13–25. ACM (2018)
32. Siegel, S.F., et al.: CIVL: the concurrency intermediate verification language. Technical report UD-CIS-2014/001, Department of Computer and Information Sciences, University of Delaware (2014)
33. Sun, J., Liu, Y., Dong, J.S., Pang, J.: PAT: towards flexible verification under fairness. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 709–714. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_59
34. Swamy, N., Chen, J., Fournet, C., Strub, P., Bhargavan, K., Yang, J.: Secure distributed programming with value-dependent types. J. Funct. Program. **23**(4), 402–451 (2013)
35. Tschannen, J., Furia, C.A., Nordio, M., Polikarpova, N.: AutoProof: auto-active functional verification of object-oriented programs. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 566–580. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_53

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.