A Usability Evaluation of Interactive Theorem Provers Using Focus Groups

Bernhard Beckert, Sarah Grebing(✉), and Florian Böhl

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
{beckert,sarah.grebing,boehl}@kit.edu

Abstract. The effectiveness of interactive theorem provers (ITPs) increased such that the bottleneck in the proof process shifted from effectiveness to efficiency. While in principle large theorems are provable, it takes much effort for the user to interact with the system. A major obstacle for the user is to understand the proof state in order to guide the prover in successfully finding a proof. We conducted two focus groups to evaluate the usability of ITPs. We wanted to evaluate the impact of the gap between the user's model of the proof and the actual proof performed by the provers' strategies. In addition, our goals are to explore which mechanisms already exist and to develop, based on the existing mechanisms, new mechanisms that help the user in bridging this gap.

1 Introduction

Motivation. The degree of automation of interactive theorem provers (ITPs) has increased to a point where complex theorems over large formalisations for real-world problems can be proven effectively. But even with a high degree of automation, user interaction is still required on different levels. On a global level, users have to find the right formalisation and have to decompose the proof task by finding useful lemmas. On a local level, when automatic proof search for a lemma fails, they have to either direct the proof search or understand why no proof can be constructed and fix the lemma or the underlying formalisation. As the degree of automation increases, the number of interactions decreases. But the remaining interactions get more and more complex as ITPs are applied to more and more complex problems.

When proving theorems, the automated proof search often leads the proof into a direction that differs from the way a human would conduct the proof. To interact with the theorem prover in a meaningful way during the proof process, users have to understand the prover's strategy and the state of proof construction and, thus, have to bridge the gap between their own model of the proof search and the current proof state of the tool. Open goals in partial proofs are the result of syntactic transformations that may not be intended to make it easy for humans to understand them. The intention of the transformations is rather

This work is part of the project Usability of Software Verification Systems within the BMBF-funded Software Campus. Florian Böhl was funded by MWK grant "MoSeS".

C. Canal and A. Idani (Eds.): SEFM 2014 Workshops, LNCS 8938, pp. 3–19, 2015.
DOI: 10.1007/978-3-319-15201-1_1

to get the automated proof search closer to a complete proof. Therefore, users need to understand the prover's strategy and often have to look at *intermediate proof states*, resulting from rule applications onto the original proof obligation, to comprehend the current state.

Although it is easy to accept that there is a gap between a human user's model of the proof resp. proof search and the actual automated proof search, it is rather unclear how large its impact on interactive theorem proving is for typical proof obligations. Nevertheless, the following is a central hypothesis for our work, which we wanted to test during the usability evaluation:

> Bridging the gap between the user's model of the proof state and the state of the theorem prover at interaction points is *the* paramount and prominent challenge for efficient and effectively usable general theorem provers.

In addition, we are interested in evaluating which tools or mechanisms are already present in today's provers that help to bridge the gap and how to extend existing mechanisms to help the user in understanding the proof states.

Our contribution in this work is that we conducted an experiment using the survey method focus groups to get a first evaluation of whether our hypothesis is true and to gain answers to our two questions: (a) Which mechanisms of this kind are already used in theorem provers? (b) What mechanisms are missing?

Survey method. We have carried out two experiments, where we applied the focus group method [10,16] to two different ITPs: the tactical theorem prover Isabelle/HOL [18] and the interactive program verification system KeY [7].

Focus groups are a qualitative survey method typically used in an early stage of the usability engineering process [12,17]. Based on their results, (prototypical) mechanisms for improving usability can be developed, which can then be evaluated with methods such as usability testing and user questionnaires to quantitatively measure increases in usability. While focus groups explore the subjective experience of users, they are designed to eliminate experimenter-bias and to provide more objective results. The number of participants required to get significant results is much smaller than for quantitative evaluations, which makes focus groups well-suited for the relatively small user base of ITPs.

Background. Our work is part of the BMBF-funded Software Campus programme. We apply various methods known from the field of human-computer-interaction (HCI) to ITPs, including focus group discussions, usability testing, and user experience questionnaires. Since expertise from both fields (ITP and HCI) is required, we cooperate with user experience experts from DATEV eG who are well-versed in the ergonomic evaluation of standard software.

Structure of this paper. Section 2 briefly reviews related work on usability evaluations of ITPs. The focus group method is introduced in Sect. 3. In Sect. 4 we present the results of the experiments and relate them to our hypothesis. Section 4.5 presents our results regarding mechanisms and tools for understanding the proof state. We conclude and discuss future work in Sect. 5.

2 Related Work

The ITP community has noticed the need to evaluate and improve usability, but so far structured usability evaluation methods have rarely been applied to ITPs.

In previous work [5], we have performed a questionnaire-based evaluation of the KeY system based on Green and Petre's Cognitive Dimensions questionnaire [9] to get a first impression of the user's perception and to develop first hypotheses about the usability of the KeY system. Beyond that Kadoda et al. [14] evaluated proof systems using Green and Petre's Cognitive Dimensions questionnaire to develop a list of desirable features for educational theorem provers.

Aitken and Melham [1–3] evaluated the interactive proof systems Isabelle and HOL using recordings of user interactions with the systems in collaboration with HCI experts. During the proof process the users were asked to think aloud and after the recordings the users were interviewed. The goal of this work was to study the activities performed by users of interactive provers during the proof process to obtain an interaction model of the users. They propose to use typical user errors as usability metric and they compared provers w.r.t. these errors. Also, suggestions for improvements of the systems have been proposed by the authors based on the evaluation results, including, besides others, improved search mechanisms and improved access to certain proof-relevant components.

Jackson et al. used co-operative evaluation methods on the CLAM Proof Planner [13]. Users were asked to perform predefined tasks while using the "think-aloud technique" to comment on what they were doing.

Vujosevic and Eleftherakis used questionnaires and interviews to explore why Formal Methods Tools are not used in industry [20]. Their work includes evaluations of usability aspects of several formal methods tools, such as the Alloy Analyzer. For improving the interface of the prover NuPRL, a self-designed questionnaire was used to evaluate the users' perceptions of the interface [11].

Similar to our findings, Archer and Heitmeyer [4] also realized the gap between the prover's and the user's model of the proof. They have developed the TAME interface on top of the prover PVS to reduce the distance between manual proofs and proofs by automation. TAME is able to prove properties of timed automata using so called *human-style reasoning*. Proof steps in TAME are intended to be close to the large proof steps performed in manual proofs. The authors have developed strategies on top of the PVS strategies that match more closely the steps performed by humans. The goal is to provide evidence and comprehension of proofs for domain but not proof experts.

Lowe et al. describe in their work [15] their approach to building a co-operative theorem prover and describe some undesirable features of ITPs focussing on feedback of the system. They have implemented the BARNACLE interface for the CLAM prover which allows explanations for failing preconditions, which should make proofs more comprehensible for the users.

Ouimet identified different issues, e.g., large proof size and number of proof steps, that have to be addressed in order to have a widespread use of theorem provers in [19] and evaluated the system ESC/Java against these issues. The issues were identified by examining a large case study conducted at Motorola.

3 Survey Method: Focus Groups

Focus group discussions are a qualitative method to explore opinions of users about specific topics or products, e.g., in market research. In the field of human-computer interaction (HCI) they are used to explore user perspectives on software systems and their usability in an early stage of the usability engineering process [12,17]. As already mentioned in the introduction, they provide the subjective experience of the users and require only a small number of participants (five to ten). The duration of the discussion groups is around one to two hours and it is guided by a moderator who uses a script to structure the discussion. Focus groups have three phases: Recruiting participants, performing the discussion and post-processing. In the following we will briefly give an insight into the script which was used to guide the discussion. The full description of the setup and script can be found in [6].

Script for the discussions. The main questions and tasks in the script were the same for both conducted focus groups as we wanted to compare the results. Adaptations of the questions and presented mock-ups to the specifics of the two systems were the main differences. As a warm-up task, we asked about typical application areas of the systems and about their strengths and weaknesses related to the proof process. In the main part of the discussion, we had two topics: (1) Support during the proof process and (2) Mechanisms for understanding proof states. As a cool-down task, we asked the participants to be creative and imagine their ideal interactive proof system. The full scripts with all questions for our experiments are available at http://formal.iti.kit.edu/~grebing/SWC.

4 Evaluation of the Focus Groups and Analysis Results

4.1 The User's and the Tool's Model of the Proof Process

ITPs are used to aid users in proving complex theorems in many areas of computer science and mathematics. For using such systems, the user needs to have a certain level of experience in proving theorems. In general, the user has a concept or plan of how to prove the desired theorem. We call this concept *user's model of the proof*. This can either be already a whole proof plan or just first ideas on the proof process. This model also includes an assumption about the theorem prover's strategies as we do not consider the proof plan for a pen and paper proof as being the user's model, but the proof plan for how the user would prove the problem using a theorem prover.

One big difference between the user's model of the proof and the current partial proof is that the proof steps in the model are coarser and have an intuitive (summing up) semantic for the user (such as "simplification of the proof obligation"), whereas the prover's steps are more fine-grained and are a syntactic manipulation of the proof state. While an intuitive semantic for each rule application exists (as given by the rule's author), a sequence of consecutive rule applications in the system may not have a clear intuitive semantic for the user.

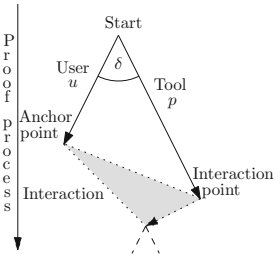

Fig. 1. Model of the proof process

In Fig. 1 we have sketched our idea of the relation between the actual proof performed by the prover's search strategy (p) and the user's proof model (u). At the beginning of the proof process the user's model is identical with or close to the proof obligation in the proof system. However, the more the automatic strategies of the prover try to prove the proof obligation (arrow p), the more the actual proof state in the system differs from the user's model (arrow u). As the user has to guide the prover by interacting with it, the user has to understand the process of the prover and relate the actual proof state to the user's model. For this relation the user has to inspect the current proof state (*interaction point*) and find a corresponding state in the own model (*anchor point*). After the user interacts with the prover, the proof of the system below the interaction point is proceeding to some extent into the direction of the user's model, reducing the gap.

In some cases, no useful anchor point may exist. Then the user has to follow and understand the automatic proof construction and, in doing so, construct a new model u that is identical with or an abstraction of p. In contrast, if the user only applies rules manually and there is no automatic proof search, then p is identical to u (in case the user fully understands the effect of the applied rules).

In the standard case, however, where there is a gap between u and p, there should be mechanisms in the systems that help the user in relating the anchor point with the interaction point (dotted line). In general, we can identify two parameters which can differ from system to system: the size of the gap between the actual proof and the user's model (δ), and the mechanisms that help to relate the user's model and the current proof state to aid the user in comprehending the proof state (dotted line between anchor and interaction point).

Apart from the gap it could be that the user does not have a clear model of the proof or even none at all. Here the gap, as described is not applicable. In this case the user uses the automation of the prover without any model in mind in order to use the resulting proof state to concretize the own fuzzy model and therefore the user has to comprehend the resulting proof state.

4.2 The Participants of Our Focus Group Discussions

We conducted two focus groups, one for the Isabelle system and one for the KeY system. To categorize the participants, we draw a distinction between tool knowledge and domain knowledge. Most of them were at expert or intermediate level w.r.t. domain knowledge. With respect to tool expertise, the Isabelle group consisted of five participants: one less experienced, two intermediate, and two expert users. The KeY group consisted of seven participants: one less experienced, two intermediate, and four expert users.

4.3 Targets of Evaluation

In the following we will briefly introduce the two systems under evaluation with the focus on those parts that were mentioned by the participants. Here, we start with the application areas of the systems as given by the participants.

KeY system. The KeY system is an interactive verification system for programs written in Java annotated with the Java Modelling Language (JML). As such it is mostly used for the verification of Java programs w.r.t. a formal specification (usually a functional specification but also, for example, information-flow properties). KeY is also used for teaching and demonstrating formal methods, and as verification condition generator for other systems. KeY has an explicit proof object, i.e., all intermediate proof states can be inspected by the user. KeY uses a sequent calculus for Java Dynamic Logic [8]. Its user interface shows proofs as a tree, the nodes of the tree contain intermediate proof goals (i.e., sequents). Each node N is annotated with the rule that was applied to some formula in N's parent node to construct N.

Isabelle. Isabelle is a theorem prover for higher-order logic. As mentioned by the participants, it is especially used for the formalization, verification and execution of algorithms, for proving in general and for the development of formal models. It has an implicit proof object, i.e., not all intermediate proof states are shown to the user, only goal-states where the system stops its automatic strategies. These automatic strategies are called *methods*, however the participants used the term *tactics*, therefore we use this term throughout the paper. Isabelle' proof tactics are basically sets of rules or lemmas that can be applied to the goal state. In this paper, the *auto* tactic will often be mentioned, which applies a large number of rule sets automatically, and the *simp* tactic, which applies rules that simplify the goal-state. Within Isabelle also different tools can be invoked that generate counterexamples (e.g., *nitpick*, *quickcheck*) or that invoke SMT solvers to find a (sub-)proof (e.g., *sledgehammer*).

4.4 Strengths and Weaknesses of the Targets of Evaluation

Here, we discuss the strengths and weaknesses of the systems with respect to the proof process as mentioned by the participants. Interestingly, some characteristics of the systems that were first named as a strength lead to lively discussions in later phases, which often brought up negative aspects of the same characteristics.

Strengths. First, we discuss results of the focus groups w.r.t. the strengths of the systems, which are summarized in Table 1.

KeY System. The group on KeY agreed that the expressiveness of the system is an important strength. The participants like how the Java Modeling Language can be used to annotate Java code. They appreciated that a proof with the KeY system always follows a certain structure, that this structure is visualized

Table 1. Strengths of the two systems according to the participants. The labels indicate whether a characteristic is linked to our (M)odel of the proof process (see Sect. 4.1) or rather to (O)ther aspects of interactive theorem proving (the classification is our own and not the focus group's).

KeY	Isabelle
· Expressive specification language (O)	· Underlying language very intuitive (M)
· Proof can be inspected in detail (M)	· Helpful community (O)
· KeY tries to simplify open goals (M)	· Large public library of theorems (O)
· High degree of automation for simple problems (O)	· Automatic tactics and tools ease proof process (M)
· All proofs follow a similar structure (M)	· Proofs can be modularized (M)
· Intuitive presentation of proof by using macros and proof tree (M)	· Flexible w.r.t. use of top down or bottom up approach (O)
· Allows user-defined rules (M)	· Code export for testing the model (M)
· Support of JML (O)	· User-adjustable syntax (M)

in form of the proof tree, and that this tree can be inspected at an arbitrary level of detail. Macros, which group rules similar to tactics in Isabelle, ease the interaction process and help to give the proof the direction intended by the user. According to the participants, the KeY system can solve easy problems without any or with only very little interaction. Furthermore, KeY supports user-defined rules. These rules can be of help during the proof process.

Isabelle. The group on Isabelle considers the underlying proof input language Isar to be one of the system's main advantages. It allows for proofs to be structured and presented in a standard textbook style that is very intuitive for humans. The large user community of Isabelle is considered to be an important strength. It provides a growing (and already quite extensive) library of theorems available to everyone. Furthermore, the community is a good resource of knowledge and friendly towards beginners. Isabelle provides a variety of tools that help during the proof process, e.g., *sledgehammer* and *nitpick*. The system can be used for a top-down as well as for a bottom-up proof approach.

Weaknesses. The results of the focus groups w.r.t. weaknesses of the systems, i.e., room for improvements are shown in Table 2. For this brief overview, we omit some of the more technical remarks by participants that are not related to the general proof process in our opinion. For example, regarding KeY there were complaints about an unstable proof loading mechanism and memory leaks. Some Isabelle users complained about specific features of jEdit – a widespread editor for Isabelle proofs.

KeY System. Interestingly, several characteristics of KeY that were named as strengths by the focus group were also identified as areas with potential for

Table 2. Weaknesses of the two systems according to the participants. The labels indicate whether a characteristic is linked to our (M)odel of the proof process (see Sect. 4.1) or rather to (O)ther aspects of interactive theorem proving (the classification is our own and not the focus group's).

KeY	Isabelle
· Necessity of repeated trivial manual interactions (M)	· Finding the right tactic for a proof state is a non-trivial explorative task (M)
· Not possible to get practically usable counterexamples (M)	· Unexpected inference of types leads to unintuitive errors (M)
· Proof tree too detailed (M)	· Bloated formulas (M)
· Interaction on low-level logic formulas required (M)	· No insight into automatic tactics; unintuitive (M)
· Unintuitive mapping between formula and program (M)	· Messy downward compatibility for older proofs in newer system versions (O)
· Performance of automatic strategy (O)	· No support for proof refactoring (O)
· Practical scalability (O)	· Library: important mathematical foundations are missing (O)

improvement. The proof tree – whose existence was perceived as a strength of KeY – was considered to be too detailed. Some stated that linking proof states to Java code would be helpful. Interaction on the low-level logic formulas is necessary, sometimes trivial and tedious. Manual interaction often has to be repeated in similar situations. There are no useful tools to generate counterexamples.

Isabelle. According to the participants, an important downside of Isabelle is that the process of choosing the right tactics and tactic parameters to conduct a proof is not always intuitive. If a tactic cannot be applied successfully in a situation it is hard to find the reason. A technical problem is that type inference sometimes leads to very unintuitive errors. Additionally, formulas belonging to different properties that could be checked (and thus presented) independently are all combined in a single goal state which increases the size of the formula (e.g., invariants encoding type information for functions).

An often recurring task when working with Isabelle is to refactor proofs towards better understandability, however, tools for refactoring are missing. While the public library of theorems was also mentioned as a strength, a weakness is that some important mathematical foundations are still missing, i.e., in some theories lemmas are still missing.

Observations and Relation of Results to Our Model. Here, we relate results of the focus groups to our model of the proof process (Sect. 4.1) and to our hypothesis. We evaluate the characteristics (Tables 1 and 2) w.r.t. to three challenges an ITP has to solve:

(A) Keeping the gap small. In general, mechanisms that help to keep the gap between the tool's proof state and the user's mental model small are seen as strengths of the systems – unintuitive behavior of the tools in the proof process is often mentioned as a problem. Several strengths of KeY help to keep the gap small: Proofs follow the same structure, macros help to guide the proof into the expected direction (similar to tactics which were mentioned as a strength of Isabelle), and users can introduce new rules that match their intuition (these rules have to be proven correct). Both tools allow the proof to be modularized (in Isabelle it can be split up into lemmas, in KeY into contracts) – this allows structuring the proof as a sequence of statements intuitive for humans. Some KeY users stated that they use the automatic proof search only if it closes a branch as otherwise the resulting state is too unintuitive to continue interactively.

(B) Bridging the gap. Understanding a given proof state is an important challenge for users of both systems during the proof process. Consequently, mechanisms and characteristics of the systems that help the user's understanding are considered to be important strengths. Here, Isabelle provides a couple of useful tools (quickcheck and nitpick to name two). Furthermore, the intuitive structure of the underlying language Isar is named as an important strength. Correspondingly, the absence of suitable mechanisms for certain situations is an important weakness. For example, our participants criticized that KeY does not provide a useful tool to generate counterexamples. Such a tool is necessary to detect whether the prover is stuck because further user input is needed or the property does not hold and no proof exists. While there are tools to generate counterexamples for Isabelle, the counterexample representation could be improved in the eyes of some participants in case proof obligations contain functions. Currently it is difficult to find the part of a proposition that is not provable.

(C) Supporting Interaction. Finally, as soon as users have a sufficient understanding of the proof state, they need to interact with the tool in an effective way. In this area there still seems to be a lot of room for improvement for both tools. The participants of the KeY focus group criticized that the interaction often has to be performed not on the annotation level but on low-level logic formulas. Furthermore, low-level steps have to be repeated by hand in similar situations. The Isabelle users were unhappy about the tedious task of finding the correct tactic to continue.

Conclusion. We observe a strong connection between the named strengths and weaknesses and our model of the proof process from Sect. 4.1. More than half of the mentioned characteristics can be associated with concepts introduced by the model. Furthermore, the results support our hypothesis that bridging the gap between the user's model of the proof and the ITP's proof state is very important during the proof process.

4.5 User Support During the Proof Process

We divided the part of the discussion about the proof processes into two parts, namely the *global* proof process (finding the right formalization and decomposing the proof task) and the *local* proof process (proving a single lemma or theorem). The participants were asked to describe their typical proof process respectively, and to name feedback mechanisms that the systems provide. Our expectations were that existing prover support and mechanisms to aid the user are adapted to the respective abstraction levels of the two processes.

4.6 State-of-the-Art in User Support

Global proof process. For both, KeY and Isabelle, the participants described a similar proof process: it starts with the formalization of the system/problem and its main properties. Users considered the modeling task to be among the most time-consuming ones. However, system feedback in this phase is restricted to syntactical and simple consistency tests. Instead, feedback causing the user to revise the model on the global level results from the *local* proof process. It is not surprising that there is only little user support for the global process, as the tasks often require creativity and depend on the particular problem.

Local proof process. In the local proof process, the users are guided by their individual impression of the complexity of open goals/proof obligations. If the user considers the obligation to be"easy enough", he or she tries a fully automatic strategy. Otherwise, or if the automation fails, the user tries to prove the obligation interactively. In this case there are two options: structured proofs (Isar/macros) or proof exploration (manual application of rules resp. tactics).

The case where the problem is considered to be easy and is tried to be proven automatically fits our model: It is the case where the user's proof plan has only one step leading to the proof state "proof complete". In the other case, proof exploration corresponds to the user having only a partial proof model, or a set of different models from which the appropriate one has to be determined. In terms of Fig. 1, we observe multiple arrows originating from the proof obligation.

Both KeY and Isabelle aid the user by providing search mechanisms or suggestion mechanisms for proof rules resp. lemmas: As stated by the participants, Isabelle supports the user in finding the right proof technique with a search mechanism for theorems in the library. KeY offers different search mechanisms and suggests applicable rules for a user-selected formula.

System feedback for the local process. In the local processes the systems give different kinds of feedback, e.g., counterexamples, open or closed goals, and (partial) proofs. Some of these are explicit (e.g., message boxes), others are implicit in a changed proof state.

The main difference between both tools is that KeY provides the full path to the open goals as proof tree, while no explicit tree is available in Isabelle.

Which part of the system (e.g., sequent, proof tree, formalization) is inspected by the user to decide on how to continue the proof depends on the problem, but we also learned that different users use different information.

From an abstract perspective the approach of inspecting the proof state, especially in KeY, corresponds to top-down analysis of the proof: the focus moves from the specification to single goals/sequents. At the beginning of the proof process, the specification is inspected more often and the shape of the proof tree plays an important role. Later in the process, the branches in the proof tree and the sequents in the open goals become more important. Also, problem complexity influences whether the sequents of the open goals are helpful or not.

In Isabelle, the strategy *try* (that carries out the complexity estimation in a simple form) and other tools and tactics (e.g., *sledgehammer*, *quickcheck*, *nitpick*, *auto*) give feedback about the goal-state. If the tactics cannot find a proof, the resulting goal-states have to be inspected by the user. However, Isabelle does not provide information about the used rules or lemmas leading to an open goal. As stated especially in the Isabelle group, it is a matter of experience to decide how proof search should proceed.

The comments on the feedback mechanisms of the proof systems support our hypothesis: the user has to understand the system's proof. The different proof artifacts are inspected and the user tries to recognize certain familiar shapes, for which he or she knows from experience how to continue in the proof process.

Proof granularity in the local process. One part of our hypothesis is that the granularity of the automatic strategies as presented to the user does not match the granularity in the user's proof model.

When the application of automatic strategies and tools does lead to open goals instead of a closed proof, information about used lemmas or rules is often missing. An example is the *auto* tactic: if it finds a proof, showing only a single proof step is appropriate. If it does not find a proof, it does not provide information about the concrete proof rules it applied and the resulting intermediate states (although this information is available internally). Only the remaining goal-states are presented to the user. Better feedback is provided by sledgehammer, as it displays the lemmas used in the underlying SMT proof.

Granularity of the proof and feedback of single steps also plays a role when publishing or refactoring a proof depending on the intended audience. In user-constructed proofs Isabelle allows different levels of granularity. Often proofs in Isabelle are more fine grained than proofs on paper.

In KeY, there are three different granularity levels (in this case for proof construction): (a) each rule application individually, (b) using the full automatic strategy, and (c) proof macros together with one step-simplification as middle-course. Proof macros are a preferred way of proving. However, they are not applicable in every proof situation.

In both systems, the granularity of the proof steps can be too fine-grained or too coarse, depending on the proof situation (e.g., failed proof attempts) and the purpose of the proof (e.g., publishing a proof). We conclude that there should be a compromise between the two extremes, e.g., a mechanism that allows to get

insight into the Isabelle tactics if required. For the KeY system, a mechanism would be useful that summarizes steps in the proof tree and only unfolds them on user inspection – extending existing mechanisms that collapse/unfold certain kind of proof nodes like intermediate steps or closed proof branches.

Time-consuming tasks during the proof process. We suspected that inspecting open goals resp. finding relations between different proof artifacts would be time-consuming tasks. To test this, we asked for time-consuming actions in the proof processes. As mentioned above, in the global process the modelling and specification task is time-consuming as well as the proof attempts in the local process. Additionally, when the user wants to minimize the proof attempts in the local process, the setup for the automatic strategies is time-consuming in both systems. Other time-consuming tasks that were mentioned, are the decision when to reconsider the whole model, proof refactoring (in Isabelle), and model refactoring (in KeY).

In the local process, the following time-consuming actions are related to *understanding the proof state*: analyzing open goals, finding counterexamples, identifying the cause of a failed proof, as well as systematic proof exploration (in KeY), and *find_theorems* and proof exploration by using apply scripts (in Isabelle). These answers support our hypothesis, as they provide evidence that understanding the proof state is a laborious task. Also, other costly tasks were mentioned: automatic proofs (as the user has to wait for the prover) and trivial repetitive instantiations on different branches (in KeY), as well as redoing a proof and especially finding the correct point to which to backtrack before correcting the model or specification. In Isabelle, cleaning up proofs takes time as well.

Conclusion. Our observation is that a lot of answers focused on understanding the proof state. For example, Isabelle users spend a lot of time cleaning up their proofs to make them accessible and understandable for other users. The answers related to the topic "understanding the proof state" in the part about time-consuming actions also support this observation. To conclude, the answers support our hypothesis that understanding a proof is a central and important task in theorem proving. The participants spend time on understanding the proof state in order to be able to proceed with the proof or find the cause for a failed proof attempt. Comprehending the proof state is also necessary for proof exploration, e.g., when the user only has parts of the proof process in mind or when the user does not know how to start or proceed.

4.7 Mechanisms Supporting the Comprehension of the Proof State

Prior to the discussion, we developed paper mock-ups of mechanisms for both verification tools which we believe aid the user in understanding the proof (state) and therefore help to overcome the discrepancy between the proof model of the user and the actual proof of the system. Implementing these remains for future work. These mock-ups were presented to the focus groups as a sequence of screenshots that show how to invoke the mechanism and the effect of the mechanism

in a particular proof situation.[1] Our intention was to gain feedback whether our developed mechanisms are comprehensible, serve our intended purpose (bridge or reduce the gap) and are of interest for the participants. The task for the participants was to describe the purpose and effect of the mechanism (as they saw it) and share their opinion about it.

Tracing Terms/formulas/variables. We showed two mock-ups (designs) for each system for the mechanism of tracing the origin of formulas respectively variables in an open goal: In Isabelle we showed the parent formula of an open goal with renamed variables. Additionally, the relation between the original and the renamed variables was depicted. As a second mock-up we showed a state with a number of open goals. By clicking on one of the goals, some of the used lemmas and definitions leading to that goal were shown.

For the KeY system, the starting point for both designs was the same: we selected one (sub-)formula of the sequent in the open goal. Then, for the first design, we depicted a new window showing the selected formula and its ancestors up to the original proof obligation (we summarized some of the intermediate parent formulas to not clutter up the screen). In addition, the names of the rules producing the formulas were given. The top-most parent shown was that part of the specification where the formula had its origin. In the second design we did not use a new window, instead we highlighted the parents in each inner node of the proof tree up to the root, which contains the original proof obligation.

When the groups where shown the mock-up of the mechanism for tracing formulas, the first reaction was clearly positive, particularly in the Isabelle group for the first mock-up. Almost all participants intuitively understood the mechanism. One participant reported that he simulates this mechanism by manual "reverse-renaming" in an external text editor. However, the question came up whether the additional information may be confusing or clutter the screen. It was suggested to implement the mechanism carefully, possibly using mouse-over tags and – in particular for KeY – include it into the existing GUI concept.

Inspired by the second mechanism for Isabelle (showing the used lemmas) some participants stated that it would be useful to have a mechanism showing the path or case distinctions leading to selected open goals on demand.

The second design in the KeY group triggered a new idea: some participants suspected a filtering mechanism and discussed about filtering the sequent and the proof tree.

What Needs to Be Proven? For the Isabelle system, a mock-up was given, showing which lemmas and theorems contribute to a proof (depicted as a simple coloured graph). Unproven lemmas were coloured red, lemmas whose proofs used unproven lemmas were coloured orange, and fully proven lemmas were coloured green. The lemmas already proven were depicted with a box with an ellipsis as description. The red and orange boxes were labelled with the name of the

[1] The screenshots may be found at http://formal.iti.kit.edu/~grebing/SWC/.

lemma that still needs to be proven resp. uses unproven lemmas. The participants described the mechanism as separating the used from the unused lemmas and that it would be useful in combination with, e.g., the automatic strategy *simp*.

Most of the participants showed a positive reaction to this mechanism. Some participants would prefer a textual representation of the used and unused lemmas. The design of our mock-up can be improved in general. The level of detail should be chosen carefully in order not to clutter up the screen (e.g., fold proven lemmas with the option to unfold) and the view should be hierarchic.

What Happened During the Proof Process? For the KeY system, the mock-up showed a diff mechanism relating two nodes in the proof tree (not necessarily adjacent nodes). We designed the mock-up such that all unchanged parts of the sequent were blurred out and the relevant changes were shown directly above each other. The participants needed some time to understand the idea and the blurring was found to be confusing, as the presentation of two different sequent parts can be mistaken as belonging to the same single sequent.

One participant noticed that something similar is implemented in the KeY system already as string diff mechanism, where the diff between two sequents is shown in one new window. However, this participant also claimed that the mechanism needs improvement, which supports our idea that such a functionality should be implemented in the KeY system.

Already during the discussion, ideas for improvement came up, e.g., that the diff between two sequents should be shown in two windows adjacent to each other or above each other. Also, like in a text-diff viewer, the changes should be marked using colours or typographical presentations. And in the proof tree, the two nodes which are being compared should be marked.

In conclusion, we suggest to develop a user-configurable diff mechanism which shows the two sequents being compared in two windows. One window depicts the old sequent and one depicts the new sequent. In addition, the algorithm for comparing two sequents has to be chosen carefully and consider the tree-structure of the sequent. A string diff algorithm is not sufficient for comparing tree-shaped sequents, as certain differences are recognized in the wrong way. For example, it is wrong to assume that replacing n by `null` results from appending `ull` to n.

4.8 The Ideal Interactive Proof System

As a cool-down task, we asked the participants to name properties that an ideal interactive verification system should or should not have. Our goal here was twofold – we wanted to collect more ideas about desirable features of ITPs and evaluate our hypothesis at the same time. For the sake of brevity, we can only present some of the mentioned features here. We decided to omit comments that were of technical nature (e.g., "It should not have memory leaks.") as well as points that have already been mentioned in previous phases.

Intuitive proof process. Both groups wished that an ideal interactive proof system would produce proofs "close to what an experienced user would expect."

This perfectly supports our paradigm of reducing the gap resp. keeping the gap small between the user's model of the proof and the ITP's current proof state.

Understandable proof states. The focus group on KeY prefers more interaction in terms of the original proof obligation (e.g., specification and program) while the Isabelle group wishes for semi-automatic proof steps (instead of the fully automatic tactics). In our opinion this illustrates that too many as well as too few details have a negative effect on understandability of the ITP.

Convenient interaction. One important feature that was wished for by both groups is a good performance of the ITP. The performance can impede usability if the user has to wait too long between interaction steps.

Conclusion. In summary, participants of our focus groups asked for an ITP that (i) produces intuitive proofs, (ii) can present proof steps in an understandable way (and give counterexamples if the proof can not be closed), and (iii) provides a convenient interface for interaction.

5 Conclusion and Future Work

We conducted two focus group discussions to evaluate the usability of ITPs. Our goal was to find evidence that a gap between the user's model of the proof and the system's current proof state exists and that this gap is a central problem for the usability of ITPs. In addition, we have developed mock-ups for mechanisms that help to bridge this gap or keep it small. We have developed a first model of the proof process with the focus on the relation between the user's (partial) model of the proof process and the current proof state.

In this evaluation we have found evidence that our model of the proof process is reasonable: the model does not fully represent the complexity of interactive proof search but captures already a lot of peculiarities. Our findings also indicate that the gap between the user's model of the proof and the system's current proof state is a central problem in interactive theorem proving.

We have also encountered related topics, such as counterexample generators and finding the correspondence between the current proof state and the program (in the KeY system) that clearly show that our model does not capture all the details of proving yet and therefore for future work this model will be extended. We have also discovered other usability issues in the systems not related to our hypothesis. These are often either technical or relate to other topics, e.g., performance of the automatic strategies. We believe that attention has to be drawn to these as well to enhance the user experience for ITPs.

We have presented functionalities that should help to bridge the gap or reduce the gap concentrated on providing the user insights into what happened during the automatic proof search. The participants reacted positively towards the

mechanisms and provided feedback for improvements or new ideas, such as user defined filter mechanisms for the proof tree in KeY.

For future work we will extend the proposed mechanisms and prototypically implement them in the KeY system and perform usability tests to evaluate our solutions. Additionally, we plan to extend the model to take into account that there are also different proof strategies for one proof and it is often user-dependent which proof style is used for a proof.

Acknowledgements. We thank the participants of our focus group discussions on the usability of KeY and of Isabelle and, in particular, the two moderators for their great work. In addition, we thank our project partners from DATEV eG for sharing their expertise in how to prepare and analyse focus group discussions.

References

1. Aitken, J.S., Gray, P., Melham, T., Thomas, M.: Interactive theorem proving: an empirical study of user activity. J. Symb. Comp. **25**(2), 263–284 (1998)
2. Aitken, J.S., Melham, T.F.: An analysis of errors in interactive proof attempts. Interact. Comput. **12**(6), 565–586 (2000)
3. Aitken, S., Gray, P., Melham, T., Thomas, M.: A study of user activity in interactive theorem proving. In: Task Centred Approaches To Interface Design, pp. 195–218. GIST Technical. Report G95.2, Department of Computing Science (1995)
4. Archer, M., Heitmeyer, C.: Human-style theorem proving using PVS. In: Ait Mohamed, O., Muoz, C., Tahar, S. (eds.) LNCS. Springer, Heidelberg (1997)
5. Beckert, B., Grebing, S.: Evaluating the usability of interactive verification systems. In: Proceedings, 1st International Workshop on Comparative Empirical Evaluation of Reasoning Systems (COMPARE), Manchester, UK, June 30, 2012, CEUR Workshop Proceedings, vol. 873, pp. 3–17. CEUR-WS.org (2012)
6. Beckert, B., Grebing, S., Böhl, F.: How to put usability into focus: using focus groups to evaluate the usability of interactive theorem provers. In: Benzmüller, C., Woltzenlogel Paleo, B. (eds.) Proceedings, Workshop on User Interfaces for Theorem Provers (UITP), Vienna. EPTCS, July 2014 (to appear)
7. Beckert, B., Hähnle, R., Schmitt, P.H. (eds.): Verification of Object-Oriented Software: The KeY Approach. LNCS, vol. 4337. Springer, Heidelberg (2007)
8. Beckert, B., Klebanov, V., Schlager, S.: Dynamic logic. In: Beckert et al. [7], chapter 3, pp 69–175
9. Blackwell, A., Green, T.R.: A cognitive dimensions questionnaire (v. 5.1.1) Feb 2007. www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDquestionnaire.pdf
10. Caplan, S.: Using focus group methodology for ergonomic design. Ergonomics **33**(5), 527–533 (1990)
11. Cheney, J.: Project report - theorem prover usability. Technical report, 2001. Report of project COMM 641. http://homepages.inf.ed.ac.uk/jcheney/projects/tpusability.ps
12. Ferré, X., Juzgado, N.J., Windl, H., Constantine, L.L.: Usability basics for software developers. IEEE Softw. **18**(1), 22–29 (2001)
13. Jackson, M., Ireland, A., Reid, G.: Interactive proof critics. Formal Aspects Comput. **11**(3), 302–325 (1999)

14. Kadoda, G., Stone, R., Diaper, D.: Desirable features of educational theorem provers: a cognitive dimensions viewpoint. In: Proceedings of the 11th Annual Workshop of the Psychology of Programming Interest Group (1996)
15. Lowe, H., Cumming, A., Smyth, M., Varey, A.: Lessons from experience: making theorem provers more co-operative. In: Proceedings 2nd Workshop User Interfaces for Theorem Provers (1996)
16. Morgan, D.L.: Focus groups. Annu. Rev. Sociol. **22**(1), 129–152 (1996)
17. Nielsen, J.: Usability Engineering. Morgan Kaufmann Publishers Inc., San Francisco (1993)
18. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002)
19. Ouimet, M., Lundqvist, K.: Formal software verification: model checking and theorem proving. Technical report, March 2007
20. Vujosevic, V., Eleftherakis, G.: Improving formal methods' tools usability. In: Eleftherakis, G. (ed.) 2nd South-East European Workshop on Formal Methods (SEEFM 05), Formal Methods: Challenges in the Business World, Ohrid, 18–19 Nov 2005. South-East European Research Centre (SEERC) (2006)