# An Empirical Study Assessing Software Modeling in Alloy


Niloofar Mansoor
*School of Computing*
*University of Nebraska - Lincoln*
Lincoln, NE USA
nmansoor2@unl.edu

Hamid Bagheri
*School of Computing*
*University of Nebraska - Lincoln*
Lincoln, NE USA
bagheri@unl.edu

Eunsuk Kang
*Institute of Software Research*
*Carnegie Mellon University*
Pittsburg, PA USA
eunsukk@andrew.cmu.edu

Bonita Sharif
*School of Computing*
*University of Nebraska - Lincoln*
Lincoln, NE USA
bsharif@unl.edu

***Abstract*—Alloy is a declarative formal modeling language with syntax derived from notations common to object-oriented design and first-order relational logic semantics. To better understand the usability of Alloy, the paper presents the results of an empirical study with 30 participants assessing two types of modeling tasks: bug fixing and model building based on natural language specifications. The participants consisted of both novices and non-novices. Besides accuracy and time to complete tasks, we also examined the correlation between the performance of two cognitive tasks and task performance. Results indicate that overall, non-novices completed the tasks with significantly higher accuracy (54% more accurate) than novices. In the novice group, performing more actions using the Alloy analyzer led to more edits and, eventually, higher scores in the bug fixing tasks. We found that participants of all levels had much difficulty writing a model from scratch, and they did not utilize the analyzer to improve their models. On average, non-novices completed all the tasks 32 minutes faster than novices. Non-novices who performed better on the Alloy tasks had higher mental rotation scores, which indicates the importance of spatial cognition ability in solving Alloy tasks. Overall, we find that there is a definite need to improve the usability of the visualizations in the Alloy Analyzer.**

***Index Terms*—Alloy Specification Language, Bug Fixing, Empirical Study, Software Modeling, Usability**

## I. Introduction

The Alloy formal specification language [1] aids in constructing models in the software design phase and checking whether specific properties of systems hold. Its back-end tool, the Alloy Analyzer [2], performs automated analysis on models, checks assertions, and generates counterexamples to those assertions if they do not hold. Alloy has been used in a wide range of applications, such as test case generation [3], security analysis of Android [4], [5], IoT devices [6], and verification of critical properties of real systems [7], [8]. Traditionally, working with formal specification languages has required in-depth mathematical knowledge due to their complexity, and the learning curve is very steep for non-mathematicians. In dealing with formal methods and designing formal specification languages, the most important factors have been soundness and correctness, and factors like readability, usability, and comprehension were often overlooked [9]–[13].

Alloy's design seeks to alleviate this problem with its easy-to-understand syntax and use of familiar mathematical concepts such as sets. However, there is very little research on the usability and comprehension of Alloy as a language from the user perspective and how it can be best taught to novices. Krishnamurthi et al. make a case for paying more attention to human factors in formal methods and state that performing more user focused research can be beneficial for building better tools and encouraging more people to learn formal methods [10]. There has been some work on how students use Alloy Analyzer in different contexts [14], [15]. However, none of this work is focused on the comprehension of the Alloy language. This is important to study because even though the Alloy Analyzer is helpful, its use does not indicate comprehension of the actual Alloy model specifications. Moreover, almost all of the prior work is done with non-novices. There is a clear existing gap of human factor studies in the literature to understand how expertise plays a role in the comprehension of Alloy specifications. It has been shown that including varied expertise in program comprehension studies can give interesting insights on how developers think about problem solving [16].

To better understand the usability aspect of Alloy, the paper presents an empirical study on the comprehension of the Alloy language in two contexts: fixing bugs and building models. Both novices and non-novices participated in the experiment. For the bug fixing tasks, participants were given natural language specifications for problems and their corresponding Alloy models, which included buggy statements. The participants were tasked with fixing the problems within the models so they matched their specifications. To the best of our knowledge, this is the first study to assess the impact of experience on the ability to fix syntactic and semantic bugs in

Alloy models to match specifications. For the model building task, we presented the participants with a natural language specification and a blueprint of an Alloy specification for them to complete. Research has shown that cognitive skills such as spatial cognition and working memory capacity are correlated with mathematical ability [17], [18]. In software engineering studies, a moderate correlation between working memory capacity and fixing bugs was found by Baum et al. [19]. Sharafi et al. found that spatial ability and data structure manipulation are correlated [20]. We wanted to test whether such correlations exist between cognitive skills and fixing bugs/building models in Alloy. To do this, we performed two sets of cognitive tests (mental rotation [21] and operation span [22]) to explore the correlation between memory capacity and solving Alloy problems. The contributions of this paper are as follows:

- An empirical study that explores comprehension of the Alloy specification language in bug fixing tasks (syntax and semantic) and a model building task.
- Comprehension pattern differences between novice and non-novices in Alloy, which has not been studied before.
- A detailed analysis of Alloy Analyzer usage patterns during the tasks.
- Cognitive tests to investigate the relationship between working memory capacity and spatial cognition ability with software modeling, also not studied before.
- Usability guidelines to improve future Alloy releases.
- A complete replication package for verifiability and replication purposes.

Results indicate that non-novices find and fix Alloy bugs with significantly higher accuracy (54% more on average) and complete the bug fixing tasks 32 minutes faster than novices. These results show that even a few months of familiarity and working with the Alloy language can make a big difference in levels of comprehension. We found that building a model from scratch is difficult for both novices and non-novices. Many non-novices were not successful in adding the specified dynamic properties to their model. For the bug fixing tasks, we found that the number of Alloy Analyzer actions and model edits correlated, which predicted the accuracy score of novices. We found that novices and non-novices make incremental changes before running the model to check whether they can see a correct instance or fix the issues that generate counterexamples. On average, novices make more changes to the Alloy models to get to the correct specification (an average of 12 more edits for novices compared to non-novices over all the tasks). We found that spatial cognition and Alloy bug fixing ability correlated, indicating the importance of this cognitive skill in understanding Alloy's underlying mathematical concepts.

## II. Research Questions

The paper addresses the following research questions:

- RQ1: What is the difference in accuracy and speed between novices and non-novices for bug fixings tasks in Alloy models?
- RQ2: What is the difference in accuracy and speed between novices and non-novices for building Alloy models from a requirements specification?
- RQ3: What patterns do we observe in user behavior during bug fixing and model building tasks?
- RQ4: How do working memory and spatial cognition ability relate to task correctness?

The results from RQ1 help us explore the differences between the novices' and non-novices' understanding of the Alloy tasks regarding accuracy and speed in fixing bugs. RQ2 helps understand how novices and non-novices work on the model building task. Both these questions can help us understand how prior exposure to Alloy makes a difference in problem solving. RQ3 helps us understand detailed patterns of bug finding and model building in both groups via recorded snapshots of the specification every time a participant performed an action using the Analyzer. Finally, RQ4 explores the relationship between performance on the Alloy tasks, spatial cognition ability and working memory capacity, which are two different cognitive skills related to mathematical and programming abilities.

## III. Related Work

The two studies most related to ours are Li et al. [15] and Danas et al. [14], who performed empirical studies on the Alloy language in novices. Li et al. [15] explored how the Alloy tool is used in practice by beginners by logging some of the user interaction with the Alloy tool when students were building Alloy models. The students are asked to build Alloy models, which can indicate their language comprehension. In contrast, our study is focused on exploring comprehension by using both bug fixing and model building tasks, which can give us more detailed information about the participants' comprehension. Unlike their study, we focus on both novices and non-novices.

Danas et al. [14] performed studies on both students and Mechanical Turk participants to explore how different types of outputs of the Alloy Analyzer model finder are used in practice. They explored principled output forms (such as minimal and maximal forms), provenance, and unsatisfiable cores. Their goal was to see how the different types of outputs help users understand and debug Alloy models.

Our work is complementary to both of these studies. We focus on how users find and fix different kinds of bugs (syntactic and semantic) in Alloy models based on their natural language specifications. This can indicate how comfortable the participants are in understanding the Alloy syntax and language and how they work with the Alloy Analyzer. This is one of the first works we know of that includes both novices (N=17) and non-novices (N=13) in an empirical study on Alloy. The comprehension model and patterns of using the Alloy Analyzer can be observed in both groups and be compared to the findings of Li et al. [15]. To our knowledge, this is also the first study to explore the relationship between specific cognitive abilities and comprehension of a lightweight formal language such as Alloy. Exploring these relationships

TABLE I: Experiment Overview

| | |
|---|---|
| Goal | Study Alloy specification language comprehension |
| Independent Variable | Experience (novice and non-novice) |
| Tasks | Cognitive: Operation Span, Mental Rotation<br>Alloy Tasks: Syntactic Alloy Error, Semantic Alloy Bug, Model Building |
| Dependent Variables | Accuracy, Speed, Usage of Analyzer, Number of Analyzer Actions, Number of Edits |

can give us insight into what skills are more indicative of better comprehension of specification languages.

## IV. Experimental Design

We describe the experimental design of the study, including participants, tasks, study instrumentation, and measures. A complete replication package [23] is available.

### A. Experiment Overview

The goal of our controlled experiment [24], [25] is to explore comprehension of the Alloy specification language in novices and non-novices in evaluating syntactic and semantic bug fixing and model building. We designed two different task categories: cognitive tasks and Alloy tasks. The cognitive tasks measure working memory capacity and spatial recognition ability (RQ4) to determine if these abilities play a role in bug fixing and model building performance. The Alloy tasks are designed for the participants to locate and fix syntactic and semantic bugs in the Alloy models and build models based on a specification. We measure comprehension using accuracy, speed, number of Analyzer actions, and number of edits. An overview of the experiment is shown in Table I. The university's institutional review board approved the study.

### B. Participants and Experience

We recruited 30 participants from different universities and institutions worldwide. Each potential participant was sent an email inviting them to participate in the study. If they accepted the invitation, they were assigned an ID for the pre and post questionnaires and were sent the study package and the consent form via email. When the participants submitted the study, they were compensated with a $10 Amazon e-gift card.

Our participants had different levels of expertise in Alloy, ranging from beginners who had recently started learning the language to experts who had been working with the language for years. Participants were recruited through emails to class mailing lists, posts on Alloy messaging boards, and professional contacts. They were asked to fill out a demographic questionnaire before the start of the study, which asked them about their age, gender, affiliation, degree, native language, and proficiency in English. There were 19 male participants and 11 female participants. Eleven participants were between the ages of 21-25, nine were between 26-30, seven were between 31-35, two were 36-40, and one was over 40 years old. Twenty-nine participants were either pursuing or had Computer Science, Computer Engineering, or Software Engineering degrees, and one was pursuing an Industrial and Labor Relations degree. Three participants were either pursuing a Bachelor's degree or held one. The rest of the participants were either pursuing a graduate degree or held Masters or Doctorate degrees.

We asked the participants to self-report their experience level, as it has been established [26] that self estimation is a reliable measurement for programming experience. Participants completed a post-questionnaire (after they completed the study to avoid any imposter syndrome bias) that asked them to rate their programming skills, design skills, knowledge of set theory, first-order logic, and object-oriented programming skills. They were also asked to rate their comprehension level of Alloy syntax and their level of comfort using the Alloy Analyzer. The post-questionnaire showed us that some participants were more familiar with Alloy despite only using it for less than a year, specifically participants who were using Alloy for research. We decided not to group these participants with novices, as they had a deeper understanding of Alloy due to extensive use. Thus, we defined non-novices in Alloy as *having more than one year of experience* or *having less than one year of experience but having familiarity with the language and rating their comfort level in understanding Alloy syntax higher or equal to 3 out of 5*. With this criteria, our novice group consisted of N=17 and non-novices of N=13.

### C. Tasks

The first category of tasks are the cognitive tasks. The two cognitive tasks were the Operation Span Task [22], [27] and the 3D Mental Rotation Task [21], which measure working memory and spatial cognition ability, respectively. Prior research [19], [20] has shown a correlation between cognitive tasks and software comprehension tasks. We aimed to explore whether these correlations exist for bug fixing and model building tasks in Alloy.

We used a Python version of the Operation Span task [28] for our study. This task shows a number of letters to the participant, with a distractor math task between each letter, and asks them to recall all the letters they have seen in order. The final calculated score by the application (partial-credit unit score) is between 0 and 1, with a score of 1 indicating that the participant has recalled all the letters correctly. Each participant completed four practice trials and 12 task trials.

We implemented a Java desktop application for the 3D Mental Rotation task [29]. The task shows an image of a 3D object to the participant and asks them to choose the correct rotations of the 3D object from four different images presented to them. For this task, each participant completed five practice trials and 20 task trials. Since the participants had to choose two correct rotations of the 3D object, we gave them 1 point for each correct choice they made. With this scoring criteria, the max score for this task would be 40.

The second category of tasks were Alloy tasks. The first set of Alloy tasks were bug fixing tasks contained in three models. The second set consisted of a single task asking participants to build an Alloy model according to a natural

TABLE II: Semantic bugs in Alloy models' predicates

| Model | Semantic Bugs | |
|---|---|---|
| | Fix a predicate | |
| | Original Specification | Altered Specification |
| grade.als | s **!in** a.assigned_to | s **in** a.assigned_to |
| balancedBST.als | **all nl: n.left.*(left + right) \| nl.elem** <n.elem<br>**all nr: n.right.*(left + right) \| nr.elem** >n.elem | **some n.left =>n.left.elem** <n.elem<br>**some n.right =>n.right.elem** >n.elem |
| | (HasAtMostOneChild[n1] &&<br>HasAtMostOneChild[n2]) =><br>(let diff = minus[Depth[n1], Depth[n2]] —<br>-1 <= diff **&&** diff <= 1) | (HasAtMostOneChild[n1] &&<br>HasAtMostOneChild[n2]) =><br>(let diff = minus[Depth[n1], Depth[n2]] —<br>-1 <= diff \|\| diff <= 1) |
| farmer.als | (one item : from - Farmer \| {<br>from' = from - Farmer - item - **from'.eats**<br>to' = to + Farmer + item }) | (one item : from - Farmer \| {<br>from' = from - Farmer - item<br>to' = to **- to.eats** + Farmer + item }) |

language specification. We chose three Alloy models from the GitHub repository by Wang et al. [30], located at [31] for the bug fixing tasks. The three models we chose are *grade.als*, *balancedBST.als*, *farmer.als*. The Grade model describes a gradebook designed to include constraints about the graders and classes, the BST model specifies a balanced binary search tree, and the Farmer model seeks to solve the classic River Crossing Puzzle [32]. Due to their various levels of complexity, we label these models as easy, medium, and difficult, respectively. We also provide natural language specifications explaining what problems these Alloy specifications are modeling. The participant was instructed to fix the model in a way that would adhere to the natural language specification. We used some of the bugs in ARepair's [30] buggy models but introduced some other bugs to fit our task types: syntactic bugs and semantic bugs. In introducing each syntactic bug, one line in an Alloy model was changed. These bugs elicit errors from the Alloy Analyzer that would help the user in detecting and fixing them, and fixing the bugs requires a level of understanding of Alloy syntax. We also introduced semantic bugs into the models. We changed either facts or constraints within a signature to modify the constraints of the model. And finally, we also changed some predicates in the models to change their meanings. The semantic changes resulted in the Alloy Analyzer showing incorrect instances or counterexamples to assertions, and the participants were expected to find and correct these bugs. Table II shows the semantic bugs in Alloy models' predicates. The rest of the tasks can be found in the replication package.

The second set of Alloy tasks was on model building. After working on the bug fixing tasks, the participants were asked to build an Alloy model to describe a Linked List. The natural language specification described the structure and constraints of the linked list. The specification and the partial model containing a blueprint of a linked list and its functions provided to the participants can be found in the replication package [23].

The participants were always asked to do the cognitive tasks first and then work on the two sets of Alloy tasks, but the order of the tasks was randomized within each category (Alloy or Cognitive). Since the models had different levels of difficulty, we permuted the order of the tasks to control for order effects (3! Alloy tasks $\times$ 2! Cognitive tasks), ending up with 12 different variations of the study. Each participant had to complete two cognitive tasks, fix 10 Alloy bugs in total, and a subset (who volunteered) were asked to work on the model building task. The participants were not aware of how many Alloy bugs there were. The only instructions given to them were to make sure the Alloy model conforms to the natural language specification.

### D. Dependent Variables

We modified the Alloy Analyzer and instructed the participants to use it while working on the tasks. The modified Analyzer logged snapshots of the open Alloy specification file every time the user executed a command. The logged timestamped user actions are as follows.

- *Execute:* Runs the most recent or the first written command if no command has been executed so far. The commands can be either "assert" for generating counterexamples to an assertion or "run" for generating instances of a predicate. The command "run" can be combined with "show" to show an instance of the model.
- *Show Instance:* Displays the most recent instance or counterexample.
- *Show MetaModel:* Displays a *meta model* of the currently open Alloy specification [1], which shows the relationships between different elements (e.g., signatures) of the specification as an object model.

We derived the following dependent variables.

- *Accuracy:* Accuracy for bug fixing tasks is calculated by assigning 1 point to each correct bug fix and half a point for localizing a bug. There were cases where a participant would localize the line of the bug but could not correct it. The maximum score a participant could receive from all the tasks was 10 points. Tasks for models Grade and Farmer had a maximum of 3 points, and tasks for the Balanced BST model had a maximum of 4 points. Accuracy for the model building task is determined by how participants solved the subproblems of the tasks. In the model building task, we asked the participants to build an Alloy model of a linked list, which included specifying the properties of connectivity of all nodes and ensuring that no node points to the linked list head. We also asked the participants to write predicates for adding and removing nodes from a linked list and different assertions to ensure the aforementioned properties hold. We also

gave points to the participants for building the correct signatures and relationships in the model.
- *Speed:* Speed is calculated by looking at the start time and finish time of the tasks.
- *Number of Analyzer Actions:* We used the information from the logs to calculate how many times they performed an action (Execute, Show Instance, etc.) on each model to see instances or check assertions.
- *Number of Edits:* We used the log information to calculate the number of times the participants edited each model in the bug fixing task.

### E. Study Instrumentation

We sent emails to potential participants inviting them to participate in the study. Once they accepted the invitation, another email was sent, which contained the link to the study package. The participants did the study remotely in the location of their choice. The study package included the cognitive and Alloy tasks, a modified version of the Alloy Analyzer, a tutorial on Alloy for participants to remember the language syntax and structure, a sample task on Alloy with the correct answers, and a ReadMe file detailing the steps to do the study. We also asked the participants to fill out pre and post questionnaires to gather demographic and self-reported experience data. Participants were instructed to read through the ReadMe file that walked them through the steps of the study. The participants were not given a time limit to complete the tasks, but they were asked to do the study in one sitting and without interruption. Finally, they had to submit the study package, containing all their changed files and the generated logs, back to the researchers. Note that the study was not conducted via a web browser since we wanted to take full advantage of all the Alloy Analyzer functionalities (not available on web).

## V. Experimental Results

### A. Pre-processing

We created a master file that included each participant's cognitive and Alloy tasks data. For the operation span task, we gathered the automatically graded scores of the working memory tasks from the generated files. For the mental rotation task, a Python script was written to grade the result files. The automated nature of grading these two tasks eliminates the possibility of errors in grading. For the Alloy bug fixing task data, a Python script was written to show the differences between the Alloy logs generated by each action by creating HTML files highlighting the differences between each run. We used the highlighted differences to grade each submission by looking at the submitted version of the model next to the original version that included the bugs. One of the authors ran each of the submissions to make sure they passed all the checks. We did not auto grade Alloy tasks via a script since there were multiple ways of fixing a bug in some cases. The manual nature of running all the submissions shows that the bug was either fixed or not fixed, leaving no subjective nature to the grading. When looking at the differences between the original and submitted files, if a change was made to a buggy line, we consider this as bug localization. Some participants also commented on the lines with "bug detected". Section IV-D lists the scoring criteria.

Aside from the submitted models from each participant, a number of files were generated by the Analyzer if the participant performed an action. These files include the snapshot of the Alloy model at the time of action, the type of action performed (execute, show instance, show metamodel), and the timestamp of the snapshot. We refer to these snapshots as "logs". Due to technical difficulties, we could not process 3 participants' log files. We also wrote a Python script to extract the action sequence and time spent between each action. We used the difference finder to go through all the logs submitted by the user for each model to show the differences between the snapshots after each action was performed. For every set of logs we had (for different participants and models), we generated an HTML file highlighting the steps they took to localize and fix the Alloy bugs. From these HTML files, we acquired information about the number of edits. We used JASP [33] to run the statistical tests.

### B. RQ1 Results: Accuracy and Speed in Bug Fixing Tasks

Research question 1 asks about the accuracy and speed of novices and non-novices when fixing Alloy bugs. The null and alternate hypotheses are as follows.

$AH_0$ Having experience (non-novices) working with the Alloy specification language does not have an effect on the accuracy of solving the tasks.

$AH_A$ Having experience (non-novices) working with the Alloy specification language has an effect on the accuracy of solving the tasks.

$TH_0$ Having experience (non-novices) working with the Alloy specification language does not have an effect on the speed of solving the tasks.

$TH_A$ Having experience (non-novices) working with the Alloy specification language has an effect on the speed of solving the tasks.

To test the $AH$ hypothesis, we used the participants' accuracy score on the Alloy bug fixing tasks as a measure of program comprehension. For each syntactic or semantic bug, if the participant changed the buggy line, they received a score of $0.5$ for bug localization. If the participant changed the buggy line and corrected the bug, they received a score of $1$ for that task. There were overall three syntactic bugs and seven semantic bugs across the three Alloy models, and the maximum score a participant could receive was $10$.

Table III presents the descriptive statistics. Overall, non-novices performed better on the tasks, and the average accuracy score for non-novices (M = $8.615 \pm 1.024$, N = 13) is $54.17\%$ higher than the average accuracy score for novices (M = $5.588 \pm 2.386$, N = 17). We observe the same pattern in individual models and on different types of tasks as well. Figure 1 shows the box plots of the overall accuracy score in both groups. We can see that the scores are widely dispersed in the novice group, ranging from 1.5 to 8.5, whereas the

TABLE III: Descriptive Statistics for Accuracy Across the Tasks and Models

| | AccuracySyntactic | | AccuracySemantic | | AccuracyGrade | | AccuracyBST | | AccuracyFarmer | | AccuracyScore | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Novice | Non-novice | Novice | Non-novice | Novice | Non-novice | Novice | Non-novice | Novice | Non-novice | Novice | Non-novice |
| Valid | 17 | 13 | 17 | 13 | 17 | 13 | 17 | 13 | 17 | 13 | 17 | 13 |
| Mean | 2.382 | 2.923 | 3.206 | 5.692 | 2.147 | 2.731 | 1.735 | 3.308 | 1.706 | 2.577 | 5.588 | 8.615 |
| Std. Deviation | 0.740 | 0.188 | 1.937 | 0.925 | 0.880 | 0.599 | 0.970 | 0.855 | 0.902 | 0.277 | 2.386 | 1.024 |

*Note.* AccuracySyntactic and AccuracySemantic are the scores received in semantic and syntactic task types, respectively. AccuracyGrade, AccuracyBST, and AccuracyFarmer are the scores received from solving the tasks in each model. AccuracyScore is the overall score the participants received for all the tasks in all the models.

TABLE IV: Descriptive Statistics for Speed (in minutes) for Each Model and Overall

| | Grade | | BST | | Farmer | | Overall | |
|---|---|---|---|---|---|---|---|---|
| | Novice | Non-novice | Novice | Non-novice | Novice | Non-novice | Novice | Non-novice |
| Valid | 17 | 13 | 17 | 13 | 17 | 13 | 17 | 13 |
| Mean | 16.471 | 12.769 | 23.471 | 20.923 | 41.294 | 15.385 | 81.235 | 49.077 |
| Std. Deviation | 10.026 | 5.761 | 15.399 | 8.067 | 61.531 | 7.911 | 66.390 | 15.787 |

non-novice group's scores range from 6 to 9.5, with the minimum score of 6 being an outlier in this group. We were also interested in the differences between the scores of all participants in syntactic and semantic task types. We observe that the participants performed better in syntactic bug fixing tasks in general, with the average of M = $2.61 \pm 0.625$ (maximum score of 3) compared to semantic tasks, with the average score of M = $4.283 \pm 1.99$ (maximum score of 7).

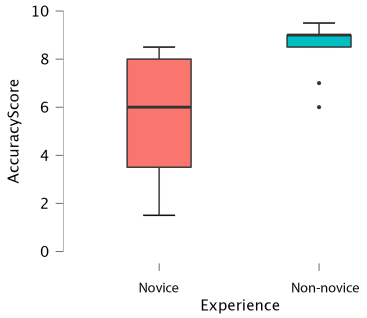



Fig. 1: Box plot of overall accuracy across Alloy tasks.

For the statistical tests, we first tested normality with the Shapiro-Wilk test. We found the data was not normal in all the groups. For the non-normal data, we chose to perform the Mann–Whitney U test, a non-parametric test to compare the accuracy scores of novice and non-novice groups. Table V shows that there are significant differences between the total accuracy scores of two groups ($p < .001$), Grade tasks ($p = .032$), BST tasks ($p < .001$), and Farmer tasks ($p = .005$). We also observed that a significant difference could be seen between novice and non-novice groups in fixing syntactic ($p = 0.006$) and semantic ($p < .001$) bugs as well. The significant differences between the two groups give us evidence to reject the null hypothesis ($AH_0$), meaning that experience makes a difference in solving Alloy tasks.

To test the $TH$ hypothesis, we measured participants' speed in completing the bug fixing tasks in each model. Table IV shows the descriptive statistics for the speed for each specification and overall for both groups. On average, novices (Overall column, M = $81.235 \pm 66.39$, N = 17) took 32 more minutes to finish the Alloy tasks compared to non-novices (Overall column, M = $49.077 \pm 15.78$, N = 13). We can observe the same pattern in individual models as well. We ran the Shapiro-Wilk normality test for this data. The distribution of the Grade model duration was the only normal distribution, and the t-test was used. For the rest of the tasks and the overall speed, we used the Mann-Whitney U test to check for significant differences between the groups, but the test did not show any significant differences. This indicates a lack of evidence to reject the null hypothesis (Grade t-test $p$ = 0.24, BST Mann-Whitney U $p$ = 0.85, Farmer Mann-Whitney U $p$ = 0.18, Overall Mann-Whitney U $p$ = 0.28).

**RQ1 Finding:** Non-novices performed significantly better in all task types than novices. Participants received higher scores on syntactic tasks compared to semantic tasks. We found that, on average, non-novices finished the bug fixing tasks 32 minutes faster than novices.

TABLE V: Mann-Whitney U Test Results for Accuracy in Two Groups

| | W | p | Rank-Biserial Correlation |
|---|---|---|---|
| AccuracyScore | 18.500 | $< .001$ | $-0.833$ |
| AccuracyGrade | 63.000 | 0.032 | $-0.430$ |
| AccuracyBST | 23.500 | $< .001$ | $-0.787$ |
| AccuracyFarmer | 46.500 | 0.005 | $-0.579$ |
| Syntactic | 52.000 | 0.006 | $-0.529$ |
| Semantic | 18.000 | $< .001$ | $-0.837$ |

*Note.* For the Mann-Whitney test, effect size is given by the rank biserial correlation. W is The Mann-Whitney statistic (W-Value) is the sum of the ranks of the first sample

### C. RQ2 Results: Accuracy and Speed in Model Building

Of the thirty participants who participated in the study, only sixteen received a model building task to create an Alloy specification for a Linked List. We did not send the model building task to the rest of the participants because we received feedback that building models from scratch is complex and very time consuming for the participants. To answer the research question about accuracy in model building, we checked their submitted model to see whether it satisfied the requirements of a linked list and whether it showed a correct instance. We gave the participants a partial Alloy model, which included the blueprint of two predicates (*add* and *remove*) and some signatures that contained incomplete relations (Listing 1). We graded the accuracy of the subproblems we expected the

TABLE VI: Participants' Performance in the Model Building Exercise. (0=incorrect, 0.5=partially correct, 1=correct.)

| Participant ID | Signatures | Insert Node | Remove Node | Acyclic Property | Connectivity | Show Instance | Acyclic Assertion | No Pointer to Head Assertion | Number of Actions |
|---|---|---|---|---|---|---|---|---|---|
| N3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | - |
| N4 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 71 |
| N5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | - |
| N6 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 37 |
| N7 | 0.5 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| N12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 |
| N15 | 0.5 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 32 |
| N17 | 0.5 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 31 |
| E4 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| E5 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | - |
| E6 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 36 |
| E7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 |
| E8 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 13 |
| E9 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | - |
| E10 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 75 |
| E11 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 68 |

participants to solve. Table VI describes the accuracy scores of the novice and non-novice group participants (Novice: N3-N17, Non-novice: E4-E11). We gave each participant a score of 1 for each subproblem if it was entirely correct, a score of 0.5 for the signatures if they were partially correct, or a score of 0 for incomplete or incorrect answers. We ran the models to see the generated instances of linked lists to confirm their correctness, and two of the authors graded each subproblem and met to dispute disagreement on the scores.

Out of all the non-novice participants, only one could not complete the signatures. Three novice participants could not complete the signatures and relations inside of them correctly, three completed half of the relations correctly, and two completed the signatures. None of the novice participants could successfully write the *Insert Node* or *Remove Node* predicates. In contrast, two non-novices completed *Insert Node* predicate, and three non-novices correctly wrote *Remove Node*. Furthermore, three novices and four non-novices correctly ensured the connectivity property. Seven non-novices and two novices wrote the correct predicate to show an instance of a linked list. We also checked how the novice and non-novice participants wrote assertions to verify the properties in their models (detailed in Table VI).

The difference between the speed of the two groups was not statistically significant, but on average, novices spent less time working on the model building task (M = 37.12±20.87, N = 8) compared to non-novices (M = 45±24, N = 8).

**RQ2 Finding:** Overall, we observed that the model building task was difficult for even the non-novices, especially writing predicates to create dynamic properties such as adding and removing a node from the linked list. Non-novices did better in ensuring the acyclic property, connectivity, showing the instance, and the assertions to verify the model.

### D. RQ3 Results: Behavior Patterns

The null and alternate hypotheses for RQ3 are as follows.

$PH_0$ Having experience (non-novices) working with the Alloy specification language does not have an effect on the behavioral patterns of problem solving in Alloy.

$PH_A$ Having experience (non-novices) working with the Alloy specification language has an effect on the behavioral patterns of problem solving in Alloy.

```
1  sig LinkedList {
2    head: lone Node
3  }
4  sig Object {}
5  sig Node {
6    data: ...
7    next: ...
8  }
9  //This predicate should insert a valid item to the list
10 pred add(l: LinkedList, l': LinkedList, new: Node){}
11
12 pred remove(l: LinkedList, l': LinkedList, new: Node){}
13
14 //Acyclic property
15 //Connectivity between all nodes
16 //show instances of the linked list
17 //assert whether the acyclic property holds
18 //assert that no node points to the list head
```

Listing 1: Blueprint for Model Building Task

To address RQ3, we explored the data we gathered from the participant logs to find patterns in the number of actions and the number of edits in the bug fixing tasks, as well as for the model building task. We believe that the number of actions and edits are useful quantitative measures of using the Analyzer and the work patterns of the participants. The Analyzer provides an interactive environment for the participants and gives us data on the number of actions (execute, show model, show metamodel) performed. Table VII shows the average number of actions and edits performed by the participants. The data shows that on average (rounded up to the nearest whole number), novices performed more actions (M = $58 \pm 51$, N = 16) compared to non-novices (M = $31 \pm 23$, N = 11). The same pattern can be observed in the Grade, Farmer, and BST logs as well. Since the data was not normally distributed, we used the Mann-Whitney U test but did not see a statistically significant difference between the number of actions across each model and overall.

Figures 2a, 2b, 2c show the categorical scatter plots of the sequence of actions performed by participants. We can see that as the tasks get more difficult (Difficulty: Grade $<$ BST $<$ Farmer), the number of actions performed by non-novices is reduced compared to the number of actions by novices. We can also see that "Execute" is the most popular action overall between both groups. Interestingly, participants used "Show Instance" while working on the BST model the most, to see valid instances of the balanced binary search tree. We

TABLE VII: Descriptive Statistics For Number of Logs (Number of Performed Actions) and Edits

| | GradeLogsNum | | BSTLogsNum | | FarmerLogsNum | | TotalLogs | |
|---|---|---|---|---|---|---|---|---|
| | Novice | Non-novice | Novice | Non-novice | Novice | Non-novice | Novice | Non-novice |
| Valid | 16 | 10 | 15 | 11 | 16 | 11 | 16 | 11 |
| Missing | 1 | 3 | 2 | 2 | 1 | 2 | 1 | 2 |
| Mean | 9.375 | 8.800 | 26.333 | 15.818 | 24.750 | 7.909 | 58.813 | 31.727 |
| Std. Deviation | 8.921 | 6.909 | 26.351 | 15.823 | 32.460 | 5.467 | 51.763 | 23.946 |
| | GradeNumberOfEdits | | BSTNumberofEdits | | FarmerNumberofEdits | | NumberOfEdits | |
| | Novice | Non-novice | Novice | Non-novice | Novice | Non-novice | Novice | Non-novice |
| Valid | 16 | 10 | 15 | 11 | 16 | 11 | 16 | 11 |
| Missing | 1 | 3 | 2 | 2 | 1 | 2 | 1 | 2 |
| Mean | 6.500 | 9.200 | 17.600 | 16.091 | 20.875 | 7.273 | 43.875 | 31.727 |
| Std. Deviation | 5.633 | 7.525 | 18.212 | 16.802 | 30.785 | 6.389 | 39.673 | 24.816 |

observe that most of the actions are performed close to the one before. We also noticed that changes were mostly incremental: participants only changed one line and performed an action.

We could not find any specific patterns in the differences between the number of edits in different models. Overall, we observe that novices make more edits than non-novices on average (Novice: M = $43.87 \pm 39.67$, N = 16, Non-novice: M = $31.72 \pm 24.81$, N = 11), but the Mann-Whitney U test did not show any significant difference between them ($p$ = 0.4).

Additionally, we wanted to know whether performing more actions correlated with the number of edits in both of the groups. We looked at the correlation between the number of edits for each model, the number of actions on each model, and the overall number of actions and edits. We found that for both groups, and for each individual model and overall, the number of actions correlated positively with the number of edits. This indicates that seeing an instance of the model helped the participants make edits. Finally, we ran the linear regression model with bug fixing accuracy as our dependent variable and the number of actions and edits as our independent variable in both novice and non-novice groups. The regression model was statistically significant in predicting the outcome variable, meaning that the number of actions and edits had a positive effect on the novice group's overall score ($p$ = 0.046, regression equation: Accuracy = $4.192 - 0.07(NumberOfEdits) + 0.032(TotalLogs)$). We could not find this relationship in the non-novice group.

We also examined whether the experience had an effect on the number of actions (Table VI) participants performed in the model building task, but we did not find any significant differences in the number of actions between the two groups (Mann-Whitney U $p$ = 0.916).

**RQ3 Finding:** We found that on average non-novices make fewer edits than novices in bug fixing tasks. The number of actions performed by novices is, on average higher than by non-novices. We observed that participants used the "Execute" action the most, and they made small and incremental edits before executing the commands again. The number of actions and edits correlated with bug fixing accuracy for novices.

### *E. RQ4 Results: Working Memory and Spatial Cognition*

Research question 4 asks how working memory and spatial cognition ability relate to task correctness. The null and alternate hypotheses are as follows.

$CogH_0$ There is no relationship between working memory and accuracy, and no relationship between mental rotation skills and accuracy.

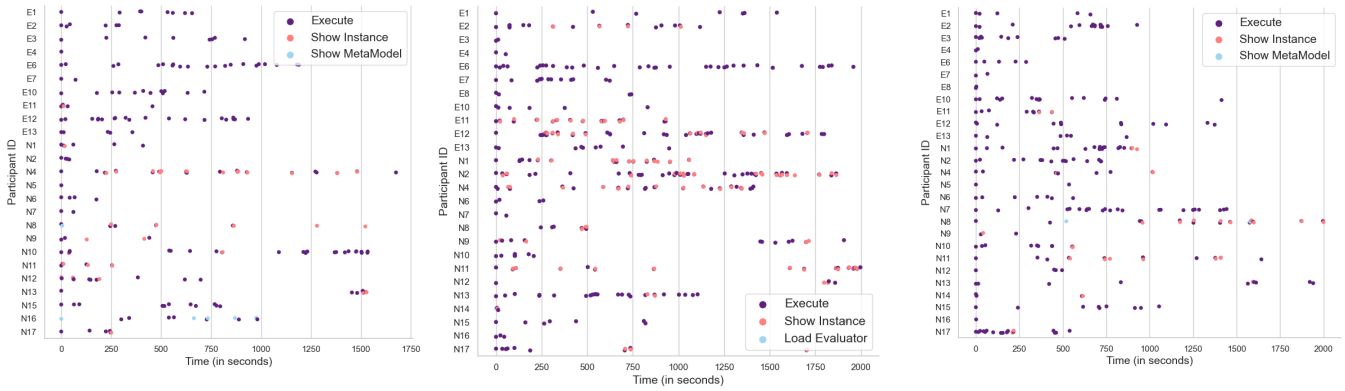
$CogH_A$ There exists a relationship between working memory and accuracy, and mental rotation skills and accuracy.

The data was not normally distributed, so we ran Spearman's correlation to assess the relationship between the operation span task score and the overall accuracy score in Alloy bug fixing tasks. The correlation was not statistically significant ($r_s$ = 0.103, $p$ = 0.588). Next, Spearman's correlation was run to assess the relationship between mental rotation task score and overall accuracy score in Alloy tasks. There was a positive correlation between the two variables, which was statistically significant ($r_s$ = 0.367, $p$ = 0.046). This finding allows us to reject the null hypothesis and accept the alternate hypothesis that there is indeed a relationship between mental rotation task and bug fixing task accuracy. We also examined the relationship between these cognitive skills and the participants' model building scores. We first added all the scores of the subproblems together to get one single model building score for each participant. We could not find statistically significant correlations between the overall model building score and cognitive task scores. We examined the relationship between the cognitive task scores and the score of each of the subproblems in model building, and we only found one significant correlation between building Signatures and the Mental Rotation Task ($r_s$ = 0.555, $p$ = 0.026).

**RQ4 Finding:** The statistically significant positive correlation between the bug fixing task accuracy and mental rotation score suggests that people with such skills might be better suited to understand Alloy models.

## VI. Threats to Validity

*Internal Validity:* The Alloy community is a relatively small community. The models that were presented in this study can be considered educational Alloy models. It is possible that some of the non-novices who have had experience with Alloy might have seen these models before while learning or teaching Alloy. We injected 1 to 2 line bugs into the models ourselves. Hence the models used were sufficiently different from models found on the web. To have everyone at the same baseline to start, we asked all the participants to go through




(a) Grade Action Sequence (b) BST Action Sequence (c) Farmer Action Sequence

Fig. 2: Analyzer Action Sequences across two groups (E1-E13: Non-novices, N1-N17: Novices)

the Alloy tutorial we sent them. They were also presented with comments on the models that could help them find the bugs. We also asked the participants not to look online for answers, but since the study was remote we did not have any control over this factor. To mitigate this threat, we took every precaution to make sure the instructions given to the participants were clear.

*External Validity:* The Alloy user population is smaller than the general developer population, making it extremely difficult to recruit participants. A few users dropped from the study because they did not understand Alloy and could not solve the tasks. We did not include their data in our analysis. Finding non-novices was also challenging because Alloy is mainly used in academic settings, and finding experts who were willing to partake in the study was difficult. Despite this, we secured 13 participants who knew and used Alloy before through extensive advertising and 17 who were willing to read the tutorial and learn the language before completing the study.

*Construct Validity:* All dependent variables were chosen carefully to ensure they represented what we sought to measure. Even though we automated most of the log analysis, we manually validated them to mitigate any errors in calculation.

*Conclusion Validity:* The unpaired Mann-Whitney test was used to compare averages of two independent groups which is suitable for small samples that are not normally distributed.

## VII. Discussion and Implications

Our findings for RQ1 present clear differences between novices and non-novices in accuracy and speed of working on bug fixing tasks. It implies that prior exposure to and experience with the language is important in completing Alloy tasks. Despite Alloy being more readable and easier to understand in comparison to other formal languages, it is still challenging for novices to work on Alloy tasks without much background on formal methods and the language itself. RQ2 results show that despite the differences in experience, all participants found it difficult to build an Alloy model from scratch by looking at the natural language specification. Novices had a very difficult time completing the easier subproblems. In contrast, non-novices had difficulty in completing the harder task of completing the *Insert Node* and *Remove Node* predicates. By examining the logs, we found only two participants (N17 and E10) ran the *Insert Node (add)* predicate to see what instances the Analyzer created. Despite running *add* and the analyzer showing an incorrect instance, the participant could not recognize the issue and could not correct the predicate. An example of an incorrect instance of *add* is shown in Figure 3. The instance shows that the participant did not ensure that the difference between the linked lists used in the predicate is only the new node (*Node0*). They also did not notice that *LinkedList2*, which is the first argument of the predicate, does not contain any nodes. Another common mistake in writing the *Add* predicate was that participants did not specify to the analyzer that the two linked lists in the argument list of the predicate cannot be the same, which resulted in wrong instances (included in the replication package). Our observations highlight the importance of understanding the instances and visualization in Alloy and that the participants either did not know how to get information out of the instances or they were not able to understand their mistakes and fix them.

RQ3 results show that the novices rely more on the Analyzer to find issues with the model and make more edits to fix bugs. We also observe that overall and in each task, the number of actions and edits were correlated, indicating that the instance generated by the analyzer can help the participant in deciding about their edits. Furthermore, in the novice population, we observe that accuracy is affected by the number of actions and edits. This implies that seeing the instances and interacting with the analyzer helps the novice participants solve the tasks more accurately. RQ4 results show that the comprehension of Alloy language is more related to spatial cognition ability and not as much related to working memory capacity. We can rationalize this difference by pointing out that Alloy tasks are not memory intensive tasks, and they are more related to the mathematical abilities of a person, which research shows is correlated with spatial recognition abilities [17].

The post-questionnaire results indicate that 16 users had trouble completing tasks. Out of the fourteen that said they

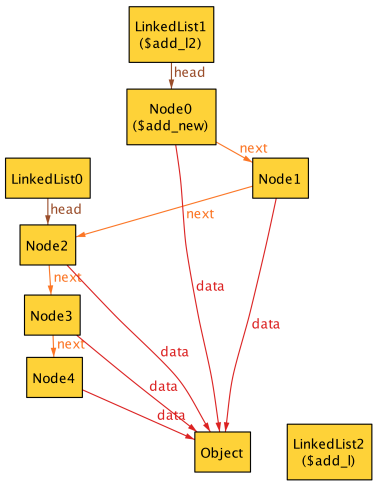



Fig. 3: An incorrect instance Add predicate in model building

did not have trouble with the tasks, eight of them were non-novices. The findings about the patterns in solving tasks, specifically the fact that the participants make incremental and small changes is consistent with the observations of Li et al. [15], who found that users perform consecutive actions on models that are only slightly different. The other study [14] is not comparable to ours because their tasks were different and dealt with which model outputs (minimality/maximality, UNSAT cores) people used the most.

**Suggested Usability Improvements:** Based on the outcome of this study, we recommend the following improvements to the Alloy Analyzer as well as the areas of focus for teaching Alloy. Given that novice users struggled more with semantic tasks (while they performed relatively well on syntactic ones), teaching systematic methods for debugging an Alloy model to locate and fix semantic bugs more quickly (e.g., identifying an over-constraint that results in unsatisfiability of a model or a missing constraint that causes an assertion failure) is recommended. An extension to the Analyzer that automates this debugging process would also be valuable. In addition, given that both novices and non-novices tend to work with the Alloy models in an incremental manner, tool enhancements that further facilitate this incremental process would also be helpful (e.g., automated compilation and execution of the model given a change; generating suggestions for which part of the model the user should inspect next). Finally, the results of RQ2 suggest that even non-novice users of Alloy struggle with inspecting the generated instances to build models. In our experience, navigating visual instance diagrams is a non-trivial task that demands a significant amount of cognitive load, especially for models with complex relations. An alternative way of visualizing Alloy instances (e.g., one that supports domain-specific visualization [34]) may help overcome this challenge. For example, looking at Figure 3, we notice that the default visualization can highlight the involved nodes better by perhaps changing their color so users have an easier time understanding that it is not a correct instance of *add*. Such details are easy to miss for novices, especially in bigger instances. The same concept applies to visualizing counterexamples where the affected nodes should be highlighted for easier comprehension. Overall, users rated their confidence as low/medium for all tasks. Perhaps these suggested changes will boost their overall confidence in finding incorrect instances.

**Implications:** Educators can make use of patterns we find in novices to better teach Alloy and help them avoid common mistakes while fixing bugs. In practice, one can choose developers in the industry who are better at spatial skills to help with Alloy debugging. Formal specification languages such as Alloy are used for many safety critical applications [7], [8], [35]. With the amount of day to day activities that depend on software running safely and securely, it is important to study how developers interact with modeling software such that we can improve them to support novice modelers by learning how the experts/non-novices behave. We still have a long way to go in this area, as is clearly evidenced by the literature. We strongly believe that more studies on formal specification languages are needed for different types of tasks. The tasks used in this paper are only the beginning of paving the way for more studies that can be conducted in this space. One way we can improve usability and tool support and adoption of software modeling tools such as Alloy is by learning (via studies such as this one) how modelers (users) interact with them. This behavior can then be used in conjunction with patterns found for novices and experts via static profiling [36].

## VIII. Conclusions and Future Work

The paper investigates how novices and non-novices perform bug fixing and model building tasks in Alloy. The results indicate that non-novices perform 54% better than novices on average and that participants perform better on syntactic tasks compared to semantic tasks. Non-novices spend less time working on the bug fixing tasks, and the participants in both groups use the action "Execute" most frequently while working on the Alloy models. The study results also show that small incremental changes are made before re-executing the model commands. The number of edits and actions performed is smaller with non-novices and predicts accuracy in the novice group. Results also show that the model building task was difficult even for non-novices. Several usability improvements in Alloy Analyzer visualizations are presented based on the study results. This study has taken the critical first step towards digesting a practice that software designers have always engaged in, leading to an understanding that promises to enable researchers, practitioners, and educators to improve rigorous software modeling. In future work, we plan to qualitatively explore the participants' patterns of problem solving and perform in-person studies to monitor closely the participants while working on Alloy tasks.

## Acknowledgments

This work is supported in part by the US National Science Foundation under Grant Numbers CNS 18-55753, CCF 18-55756, CCF 17-55890, and CCF 16-18132.

## REFERENCES


[1] D. Jackson, *Software Abstractions: logic, language, and analysis*, 2012.

[2] "Alloy analyzer," https://alloytools.org/download.html.

[3] N. Mirzaei, J. Garcia, H. Bagheri, A. Sadeghi, and S. Malek, "Reducing combinatorics in gui testing of android applications," pp. 559–570, 2016.

[4] H. Bagheri, A. Sadeghi, J. Garcia, and S. Malek, "Covert: Compositional analysis of android inter-app permission leakage," *IEEE transactions on Software Engineering*, vol. 41, no. 9, pp. 866–886, 2015.

[5] H. Bagheri, E. Kang, S. Malek, and D. Jackson, "A formal approach for detection of security flaws in the android permission system," *Formal Aspects of Computing*, vol. 30, no. 5, pp. 525–544, 2018.

[6] M. Alhanahnah, C. Stevens, and H. Bagheri, "Scalable analysis of interaction threats in iot systems," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 272–285.

[7] J. P. Near, A. Milicevic, E. Kang, and D. Jackson, "A lightweight code analysis and its role in evaluation of a dependability case," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 31–40.

[8] N. Mansoor, J. A. Saddler, B. Silva, H. Bagheri, M. B. Cohen, and S. Farritor, "Modeling and testing a family of surgical robots: an experience report," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 785–790.

[9] M. Spichkova and A. Zamansky, "Teaching of formal methods for software engineering," in *ENASE*, 2016, pp. 370–376.

[10] S. Krishnamurthi and T. Nelson, "The human in formal methods," in *Formal Methods – The Next 30 Years*, M. H. ter Beek, A. McIver, and J. N. Oliveira, Eds. Cham: Springer International Publishing, 2019, pp. 3–10.

[11] T. Weber, A. Zoitl, and H. Hußmann, "Usability of development tools: A case-study," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2019, pp. 228–235.

[12] R. Kalantari and T. C. Lethbridge, "Characterizing ux evaluation in software modeling tools: A literature review," *IEEE Access*, vol. 10, pp. 131 509–131 527, 2022.

[13] S. Abrahao, F. Bourdeleau, B. Cheng, S. Kokaly, R. Paige, H. Stoerrle, and J. Whittle, "User experience for model-driven engineering: Challenges and future directions," in *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. Los Alamitos, CA, USA: IEEE Computer Society, sep 2017, pp. 229–236. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MODELS.2017.5

[14] N. Danas, T. Nelson, L. Harrison, S. Krishnamurthi, and D. J. Dougherty, "User studies of principled model finder output," in *International Conference on Software Engineering and Formal Methods*. Springer, 2017, pp. 168–184.

[15] X. Li, D. Shannon, J. Walker, S. Khurshid, and D. Marinov, "Analyzing the uses of a software modeling tool," *Electronic Notes in Theoretical Computer Science*, vol. 164, no. 2, pp. 3–18, 2006.

[16] N. J. Abid, J. I. Maletic, and B. Sharif, "Using developer eye movements to externalize the mental model used in code summarization tasks," in *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, 2019, pp. 1–9.

[17] M. Hegarty and M. Kozhevnikov, "Types of visual–spatial representations and mathematical problem solving." *Journal of educational psychology*, vol. 91, no. 4, p. 684, 1999.

[18] K. P. Raghubar, M. A. Barnes, and S. A. Hecht, "Working memory and mathematics: A review of developmental, individual difference, and cognitive approaches," *Learning and individual differences*, vol. 20, no. 2, pp. 110–122, 2010.

[19] T. Baum, K. Schneider, and A. Bacchelli, "Associating working memory capacity and code change ordering with code review performance," *Empirical Software Engineering*, vol. 24, no. 4, pp. 1762–1798, 2019.

[20] Z. Sharafi, Y. Huang, K. Leach, and W. Weimer, "Toward an objective measure of developers' cognitive activities," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 3, pp. 1–40, 2021.

[21] S. G. Vandenberg and A. R. Kuse, "Mental rotations, a group test of three-dimensional spatial visualization," *Perceptual and Motor Skills*, vol. 47, no. 2, pp. 599–604, 1978, pMID: 724398. [Online]. Available: https://doi.org/10.2466/pms.1978.47.2.599

[22] N. Unsworth, R. P. Heitz, J. C. Schrock, and R. W. Engle, "An automated version of the operation span task," *Behavior research methods*, vol. 37, no. 3, pp. 498–505, 2005.

[23] N. Mansoor and B. Sharif, "An empirical study assessing software modeling in alloy-replication package," 2023. [Online]. Available: https://osf.io/5p6e4/

[24] N. Juristo and A. M. Moreno, *Basics of Software Engineering Experimentation*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[25] K. Stol and B. Fitzgerald, "The ABC of software engineering research," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 3, pp. 11:1–11:51, 2018. [Online]. Available: https://doi.org/10.1145/3241743

[26] J. Siegmund, C. Kästner, J. Liebig, S. Apel, and S. Hanenberg, "Measuring and modeling programming experience," *Empirical Software Engineering*, vol. 19, no. 5, pp. 1299–1334, 2014.

[27] A. R. Conway, M. J. Kane, M. F. Bunting, D. Z. Hambrick, O. Wilhelm, and R. W. Engle, "Working memory span tasks: A methodological review and user's guide," *Psychonomic bulletin & review*, vol. 12, no. 5, pp. 769–786, 2005.

[28] T. von der Malsburg, "Py-Span-Task – A Software for Testing Working Memory Span," Jun. 2015. [Online]. Available: http://dx.doi.org/10.5281/zenodo.18238

[29] N. Mansoor, "3d mental rotation task in java." [Online]. Available: https://github.com/niloofarmansoor/3DMentalRotation

[30] K. Wang, A. Sullivan, and S. Khurshid, "Arepair: a repair framework for alloy," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 103–106.

[31] K. Wang, "Arepair: A repair framework for alloy - experiment repository." [Online]. Available: https://github.com/kaiyuanw/ARepair/tree/master/experiments/models

[32] Alloy, "River Crossing Puzzle," 2022. [Online]. Available: https://alloytools.org/tutorials/online/sidenote-RC-puzzle.html

[33] JASP Team, "JASP (Version 0.14.1)[Computer software]," 2022. [Online]. Available: https://jasp-stats.org/

[34] T. Dyer and J. W. B. Jr., "Sterling: A web-based visualizer for relational modeling languages," in *8th International Conference on Rigorous State-Based Methods ABZ*, 2021, pp. 99–104.

[35] S. Pernsteiner, C. Loncaric, E. Torlak, Z. Tatlock, X. Wang, M. D. Ernst, and J. Jacky, "Investigating safety of a radiotherapy machine using system models with pluggable checkers," in *International Conference on Computer Aided Verification*. Springer, 2016, pp. 23–41.

[36] E. Eid and N. A. Day, "Static profiling of alloy models," *IEEE Transactions on Software Engineering*, vol. 49, no. 2, pp. 743–759, 2023.