A Multi-Site Joint Replication of a Design Patterns Experiment using Moderator Variables to Generalize across Contexts

Jonathan L. Krein, Lutz Prechelt, *Member, IEEE,* Natalia Juristo, Aziz Nanthaamornphong, Jeffrey C. Carver, *Senior Member, IEEE,* Sira Vegas, Charles D. Knutson, *Member, IEEE,* Kevin D. Seppi, and Dennis L. Eggett

Abstract—**Context.** Several empirical studies have explored the benefits of software design patterns, but their collective results are highly inconsistent. Resolving the inconsistencies requires investigating moderators—i.e., variables that cause an effect to differ across contexts. **Objectives.** Replicate a design patterns experiment at multiple sites and identify sufficient moderators to generalize the results across prior studies. **Methods.** We perform a close replication of an experiment investigating the impact (in terms of time and quality) of design patterns (Decorator and Abstract Factory) on software maintenance. The experiment was replicated once previously, with divergent results. We execute our replication at four universities—spanning two continents and three countries—using a new method for performing distributed replications based on closely coordinated, small-scale instances ("joint replication"). We perform two analyses: 1) a *post-hoc* analysis of moderators, based on frequentist and Bayesian statistics; 2) an *a priori* analysis of the original hypotheses, based on frequentist statistics. **Results.** The main effect differs across the previous instances of the experiment and across the sites in our distributed replication. Our analysis of moderators (including developer experience and pattern knowledge) resolves the differences sufficiently to allow for cross-context (and cross-study) conclusions. The final conclusions represent 126 participants from five universities and twelve software companies, spanning two continents and at least four countries. **Conclusions.** The Decorator pattern is found to be preferable to a simpler solution during maintenance, as long as the developer has at least some prior knowledge of the pattern. For Abstract Factory, the simpler solution is found to be mostly equivalent to the pattern solution. Abstract Factory is shown to require a higher level of knowledge and/or experience than Decorator for the pattern to be beneficial.

Index Terms—Design patterns, software maintenance, moderator variables, multi-site, joint replication, controlled experiment.

1 Introduction

Software practitioners have ascribed numerous benefits to the use of design patterns [1], [2], [3], [4], [5]. For instance, by reducing code coupling, many patterns enable developers to add functionality without modifying existing code. Patterns also simplify communication by providing standard terminology for complex concepts. Further, as standardized representations, patterns can facilitate architectural reuse, aid inexperienced developers, improve program comprehension, and encourage best practices. Ultimately, design patterns are thought to reduce development time, increase software quality, and reduce maintenance costs—but do they really? And if they do, then under what circumstances are the various benefits realized?

Since Gamma *et al.*'s 1995 book [5], numerous empirical studies have explored design pattern claims (e.g., [6], [7], [8], [9], [10], [11]). However, these studies do not provide a consistent answer. For example: Prechelt *et al.* [10] conclude that sometimes software may still be easier to maintain even if design patterns provide *un*necessary flexibility. Conversely, Wendorff reports that the "uncontrolled use of patterns" caused, in their case, "severe maintenance problems" [11, p. 77]. In general, we find support both for (e.g., [8], [9], [10]) and against (e.g., [6], [7], [11]) design patterns, for each of several related attributes—modifiability, maintainability, understandability, quality, and so forth. In response to these and other contradictory findings, the authors of a recent (2012) systematic literature review conclude, "We could not identify [aside from two marginal caveats] firm support for any of the claims made for patterns in general" [12, p. 1213], [13].

Apparently, in the case of design patterns, contextual variation drives complex tradeoffs. The study of design patterns is thus confronted by a *problem of generalizability*—i.e., the tendency of many effects to be unstable across

- *J.L. Krein and C.D. Knutson, Department of Computer Science, Brigham Young University, Provo, UT USA and Ironwood Experts, LLC, Alpine UT USA. E-mail: jonathankrein@byu.net, knutson@cs.byu.edu.*
- *L. Prechelt, Institut für Informatik, Freie Universität Berlin, Germany. E-mail: prechelt@inf.fu-berlin.de.*
- *N. Juristo, Computing School, Technical University of Madrid, Spain and University of Oulu, Finland. E-mail: natalia@fi.upm.es.*
- *A. Nanthaamornphong, Department of Communication and Information Technology, Prince of Songkla University, Phuket Campus, Thailand. E-mail: aziz.n@phuket.psu.ac.th.*
- *J.C. Carver, Department of Computer Science, University of Alabama, Tuscaloosa, AL USA. E-mail: carver@cs.ua.edu.*
- *S. Vegas, Computing School, Technical University of Madrid, Spain. E-mail: svegas@fi.upm.es.*
- *K.D. Seppi, Department of Computer Science, Brigham Young University, Provo, UT USA. E-mail: kseppi@byu.edu.*
- *D.L. Eggett, Department of Statistics, Brigham Young University, Provo, UT USA. E-mail: theegg@stat.byu.edu.*

Manuscript received. . . ; revised. . . ; accepted. . . . Date of publication. . . ; date of current version. . . .
Recommended for acceptance by. . .
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org and reference IEEECS Log Number TSE-0000-00-0000.
Digital Object Identifier no. 00.0000/TSE.0000.0000000.

varying contexts, so that considerable knowledge of *moderator variables* (*moderators* for short) is required before generalizable statements can be made. By *moderator*, we mean any explanatory variable that interacts with another explanatory variable in predicting a response variable [14, p. 624], [15].[1]

Referring to this problem in sociology, Diesing notes, "We cannot test one hypothesis about flatworms, or students, or therapy patients... until we have learned many things about the interacting factors affecting their behavior in an experimental situation" [16, p. 338]. Moreover, "it has been argued that the amount of progress in any discipline can be indexed by the degree to which its theory and research have considered the role of moderators" [14, p. 624]. Thus, identifying moderators is critical to the research process, and until we have done so, we cannot trust our conclusions. Also, as Diesing notes, "replication is a test; but it is also part of a larger search and discovery process" [16, pp. 337–338]. Thus, we should not be surprised to find disagreement in early replications, but instead should view the situation as an opportunity to explore moderators.

Given the contradictory results among design pattern studies, identifying relevant moderators is prerequisite to generalizing across contexts. Further, as Diesing explains [16, pp. 335–339], explication of moderators *requires* empirical replication. Thus, in this paper we present a *joint replication* on design patterns, conducted at four sites, involving three countries and two continents, the contextual breadth of which is well-suited to the exploration of moderators.

By *joint replication* we mean a multi-site study, performed by multiple research teams whose efforts are coordinated, yet the researchers at each site act independently in performing their own replication. Research teams explicitly communicate about important aspects of the experiment, including adopting a common definition of the experiment. However, each team gathers participants, collects data, and performs initial data analysis separately, after which the datasets are then merged and analyzed together.[2]

A multi-site, joint replication of this sort has never been done before in software engineering. In general, joint replication is a new type of method that adds to the conventional research methods currently in use (e.g., see [18]). As we show in this paper, it enables evaluation of moderators in greater detail than previously possible. In fact, a key benefit of joint replication is the contemporaneous collection of a broad range of contextual data [17, p. 174].

For the joint replication, we closely replicate a seminal experiment investigating the impact of design patterns on software maintenance (abbreviated as "PatMain"). The original PatMain study, performed in 1997 by Prechelt *et al.* [10] and published in 2001, was the second controlled experiment ever undertaken to study design patterns (the first being "PatDoc" [19]). The experiment was replicated once previously, in 2004 by Vokáč *et al.* [20], with divergent results. In addition to being one of the first design pattern studies, we chose PatMain because it is sufficiently small scale in terms of participant time and it involves relatively simple metrics that can be mostly collected automatically.

We conducted the joint replication as part of the 2011 *Workshop on Replication in Empirical Software Engineering Research* (RESER) [21], [22], [23]. Initially eight research teams expressed interest, of which four submitted data:

- Brigham Young University (BYU), Utah, USA
- Freie Universität Berlin (FUB), Germany
- The University of Alabama (UA), Alabama, USA
- Universidad Politécnica de Madrid (UPM), Spain

The four teams submitted brief reports to the workshop describing their individual results [24], [25], [26], [27]. This paper, which builds on those reports, contributes a combined analysis of the four datasets, including the collection of contextual data, analysis of moderators, assessment of the original hypotheses, and synthesis of the results with the prior two PatMain studies.

From this point forward, we refer to the original PatMain study (by Prechelt *et al.*) as E_orig, the first replication (by Vokáč *et al.*) as E_repl, and the joint replication (our work) as E_joint. To reference these experiments generically, we use the term *study*, as in, "the three PatMain studies." To generically reference sub-replications of E_joint (i.e., BYU, FUB, UA, UPM), we use the term *site*, as in "the four E_joint sites."

1.1 Objectives and Contributions

In this study we aimed to:

1) Replicate the PatMain experiment at multiple sites.
2) Assess the heterogeneity of the results.
3) Investigate potential moderators which may be inhibiting generalizability of the results.
4) Address the original PatMain hypotheses.
5) Generalize the results across PatMain studies.

The primary objective of the study (relative to design patterns) was to generalize across the PatMain series of studies. In pursing this objective, we make three contributions to the literature. First, we identify industry-relevant moderators (e.g., developer experience), each of which is empirically/statistically shown to influence the effect of design patterns. Second, we resolve significant contradictions in the results across the three PatMain studies. Thus, the final conclusions generalize across a broader set of contexts than has previously been achieved in the study of design patterns. Third, we distill the PatMain conclusions into a set of concise statements, as well as a generalized figure, which can be readily applied to future research and/or to practice.

We also make contributions to the literature with respect to methodology. Thus, this study has produced two general deliverables: 1) a set of practical findings relative to design patterns (summarized above and discussed at length in this paper), and 2) a new method for generalizing across replications. Due to space constraints, we defer in-depth discussion of the latter topic to a separate paper [28]. In the present paper, we discuss methodology only to the extent necessary to justify and show the novelty of the design pattern results (in that they rest on a stronger-than-typical empirical foundation—i.e., the joint replication is not just another design patterns experiment).

1. See Section 2.4 for further explanation of the term *moderator*. Also note, throughout the paper, we use the terms *explanatory* and *response* instead of *independent* and *dependent* to refer to variables because, as we show, the explanatory variables are not statistically independent (which is precisely why we need to study moderators).

2. Joint replication is similar to multi-site randomized controlled trials (RCTs) in social work research. For information on multi-site RCT methods, see Solomon *et al.* [17, pp. 173–176].

1.2 Structure of this Article

In Section 2, we describe our research methods. In Section 3 we present results and discussion. In Section 4, we describe threats to validity, and finally, in Section 5, we conclude. *All appendices are included in the supplemental materials for this paper, which can be downloaded from the publisher's website.* In addition to appendices, the supplemental materials also include: a lab package for E_orig (also available at [29]); raw data for E_repl (obtained from Marek Vokáč and included with permission); and a lab package for E_joint.

Since *replicability* and *generalizability* are key concerns of this study, we include a considerable amount of supplemental information in the appendices. The appendices are provided to assist future replicators and/or to facilitate transparency; none are necessary in order to understand the conclusions and general validity of the study.

2 Research Methods

In this section, we describe our research methods, including the PatMain experiment, data and metrics, statistical models, and moderator analysis.

2.1 The PatMain Experiment

In this section, we outline the three PatMain studies: E_orig, E_repl, and E_joint. We also describe the PatMain programs, tasks, and hypotheses that were the same for all three studies. To facilitate traceability, we reuse (as much as possible) the terms, acronyms, and format of the prior two studies (E_orig and E_repl).

2.1.1 The Original Study (E_orig)

In this section, we describe E_orig (by Prechelt *et al.* [10]).

Motivation and Research Question. Motivation for E_orig was twofold. First, the authors sought to empirically validate design pattern claims, which at the time were only anecdotally grounded. Second, they noticed a complexity tradeoff: while design patterns provide flexibility, they can make changes more difficult by complicating potential solutions. To explore this tradeoff, the authors proposed the following research question [10, p. 1135, paraphrased]: *For a given problem, if using a design pattern is "overkill" (i.e., the pattern provides more functionality or flexibility than necessary), will the resulting solution be more difficult to maintain than if a simplified solution were implemented instead?*

Experiment Design. E_orig included four modestly-sized (300–700 LOC), well-documented, C++ programs, implementing five Gamma *et al.* [5] design patterns:

- Stock Ticker (ST): *Observer*
- Boolean Formulas (BO): *Visitor*, *Composite*
- Communication Channels (CO): *Decorator*
- Graphics Library (GR): *Abstract Factory*, *Composite*

Each program was implemented in two variants: one employing design patterns (PAT), the other using a simplified, alternative design (ALT). The simplified design discarded all patterns not required for the given program. Observer, Visitor, Decorator, and Abstract Factory were completely eliminated in the ALT variants; Composite was retained in part in both the BO and GR programs. Participants were not informed of the PAT/ALT distinction; they were only told, "the experiment tests the usefulness of patterns" [10, p. 1135]. The experiment was administered on *paper* and included 2–3 tasks for each program. Some tasks required modifying code (i.e., *coding tasks*); others tested comprehension of the code (i.e., *comprehension tasks*).

The experiment began with a pre-test (PRE) involving two of the four programs (one PAT, one ALT). A patterns training course was then administered, followed by a post-test (POST), involving the remaining two programs (again one PAT, one ALT). Since design patterns were new at the time of the experiment, the training course was a key incentive to participate. The experiment lasted two days—pre-test in the morning of day 1, training in the afternoon of day 1 and morning of day 2, post-test in the afternoon of day 2. All participants received all four programs. Program and variant orderings were alternated. Experience and pattern knowledge were pre-surveyed to facilitate stratification of the random assignment to groups.

This design resulted in six explanatory variables: *program* (with levels ST, BO, CO, GR); *task* (with levels 1, 2, 3); *variant* (with levels PAT, ALT); pattern knowledge (abbreviated as *patKnow*, with levels PRE, POST); program *order* (with levels first, second); and *subjectID*. The experiment assessed two response variables: *time* (measured in minutes) and *correctness* (i.e., quality of the solution, measured on a 4-point scale: no fault, minor problem, not-so-minor problem, major problem). Hypotheses addressed the impact of *variant* and pattern knowledge (*patknow*) on *time* and *correctness*.

Participants. The 29 participants were all volunteers, software professionals from the consultancy firm sd&m in Munich, Germany. The median industry experience was 3.5 years (mean 4.1), including 2 years (mean 2.4) with object-oriented programming. Fifteen (52%) of the participants had prior industry experience with patterns.

2.1.2 The First Replication (E_repl)

In this section, we describe E_repl (by Vokáč *et al.* [20]).

Motivation and Research Question. The goal of E_repl was to increase "the experimental realism and, thereby, the applicability of the results" [20, p. 150]. In particular, the participants in the replication worked on computers rather than on paper. Otherwise, the authors strove for a close replication.

Experiment Design. The design of E_repl involved four key changes: 1) participants worked on computers, in a standardized environment, rather than on paper; 2) correctness was assessed on a 5-point scale (requirements misunderstood, wrong answer, right idea, almost correct, correct); 3) the original programs were adjusted to facilitate compilation; and 4) the participants were assigned by their consultancy firms to participate, rather than being volunteers. Otherwise, E_repl closely duplicated E_orig's design; the patterns course was even taught by the same instructor (Walter Tichy) using the same materials.

Participants. The 44 participants included 39 software professionals from 11 consultancy firms and 5 graduate students. The median industry experience was 4 years (mean 6.6), including 2 years (mean 2.4) with object-oriented programming. Seventeen (39%) of the participants had prior industry experience with patterns.

2.1.3 The Joint Replication (E_joint)

In this section, we describe our replication, E_joint.

Motivation and Research Question. Three motivations prompted E_joint: 1) a joint replication had never been done before in software engineering, so we anticipated significant learning with respect to methodology; 2) we were interested to see how homogeneous the results would be across sites; and 3) we wanted to address the issue of contradictory results among design pattern studies. In general, we wanted to test a new method for performing distributed replications based on closely coordinated, small-scale instances. Like E_repl, we maintain the original research question and hypotheses.

Experiment Design. We strove for a close replication. All changes were either improvements carried over from E_repl, or they were administrative changes designed to encourage participation and to facilitate consistency across sites.

Like E_repl, the participants worked on computers, rather than on paper. We administered the experiment via a web portal, through which the participants downloaded source code and uploaded solutions. The portal recorded work times by tracking the time spent on each page. The portal also managed experiment groups and administered the questionnaires. For detailed information on the portal, see Appendix A.

Using a web portal provided four benefits: 1) it lowered the barrier to entry for replicators; 2) it facilitated uniformity across sites; 3) it allowed participants to take the experiment on their own time, thus reducing scheduling constraints; and 4) it allowed participants to work with their own tools in their own environments. The downside was that we had less control over what the participants did during the experiment (e.g., take phone calls). However, few participants reported interruptions or protocol deviations, and for those that did, we apply data corrections, as described in Section 2.2.

Another change involved reducing the scope of the experiment by eliminating the training course and the post-test, thus reducing the experiment from four programs to two. The purpose of the scope reduction was, as with the web portal, to lower the barrier to participation. Thus we traded breadth of protocol for increased coverage of contextual variables and greater diversity in the population sample. Of the four programs, we excluded ST and BO (ST because it is the least complex and we wanted to make sure the results would not be washed out by an overly simplistic task; BO because the prior two studies both found the Visitor pattern to be overly difficult.[3])

Instead of assessing pattern knowledge via the training course, we used a pre-questionnaire. As with the other materials of our replication, we adopted the pre-questionnaire from the original experiment (where it had been used to facilitate stratification of the random assignment to groups). The pre-questionnaire required participants to estimate their knowledge of 17 patterns on a 7-point ordinal scale: 1=*never heard of it*, 2=*have only heard of it*, 3=*understand it roughly*, 4=*understand it well*, 5=*understand it well and have worked with it once*, 6=*understand it well and have worked with it two or three times*, 7=*understand it well and have worked with it many times*. Except for the Reactor pattern, which was originally defined by Schmidt *et al.* [30, pp. 179–214], the 17 patterns were all standard Gamma *et al.* [5] design patterns (for a complete list, see Appendix G). Admittedly, if participants are too homogeneous, a survey may fail to detect an effect even when one exists. On the other hand, if participants are initially too knowledgeable, a training course may also fail to detect an effect. Thus both approaches have limitations.

Lutz Prechelt translated the materials of the original experiment from German to English, which translation was used at all four sites. Lutz also modified the questionnaires to collect additional data. To test the translation, we administered the experiment to three native English-speaking computer science students not previously affiliated with the study; the test participants had no problem understanding the instructions and questionnaires. For discussion of how the use of English may have impacted the sites differently, see Appendix S (specifically the subsections on language barriers and clarity of task instructions).

Other changes included: 1) The web portal offered three language options: C++, Java, and C# (Martin Liesenberg and Christian Bird provided the Java and C# translations of the original C++ programs). Despite having three options, all participants worked in Java. At BYU and UA they were instructed to do so; at FUB and UPM they did so by preference. 2) Although we did randomly assign participants to groups, we did not stratify that random assignment; instead, we controlled for developer experience and pattern knowledge via covariates in the statistical analysis. Group assignments were made by randomly assigning IDs to participants, which IDs were generated by the web portal and corresponded to the four treatment groups, as follows ($group\# = ID \ \% \ 4$):

- Group 0: GR-PAT, CO-ALT (14 participants)
- Group 1: CO-ALT, GR-PAT (12 participants)
- Group 2: GR-ALT, CO-PAT (12 participants)
- Group 3: CO-PAT, GR-ALT (15 participants)[4]

Our protocol preserves the variables from E_orig except for a few changes. Concerning explanatory variables: we limit *program* to two levels (CO, GR); we assess pattern knowledge (*patknow*) via a survey instead of a training course; and we add explanatory variables to represent developer experience (*devExp*) and *site* (BYU, FUB, UA, UPM). All other explanatory variables (*task*, *variant*, *order*, and *subjectID*) are as described previously for E_orig. Concerning response variables: *time* is measured in seconds rather than minutes, and *correctness* is measured on E_repl's 5-point scale (described in Section 2.2). Additionally, despite the standardized web portal, the four sub-replications still differed in a few minor respects—e.g., BYU's experiment spanned 3 weeks, whereas

3. For example, of E_repl, Vokáč *et al.* comment, "... Visitor was so difficult that even after a course that gave the instructor excellent feedback (grade better than 4 out 5), most subjects either ignored the pattern or were confused by it." [20, p. 172].

Additionally, note that pattern difficulty (or complexity) is not the same concept as that of "overkill" from the research question described in Section 2.1.1. "Overkill" refers to a pattern's complexity *relative* to the problem it aims to solve, rather than to its absolute complexity. Thus, although the Visitor pattern is structurally more complex than Decorator and Abstract Factory, it is not necessarily more suited to answering questions about overkill. By eliminating the visitor pattern we aimed to tackle the question of overkill with respect to Decorator and Abstract Factory first.

4. Treatment groups are modestly imbalanced due to several no-shows and eight cases of unusable data, as discussed in Section 2.2. For group sizes listed by site, see the data file in the lab package.

TABLE 1
Overview of the CO/GR programs, tasks, and hypotheses. For additional details, see Appendix D.

Program	Description	Tasks and Hypotheses (based on E_orig's published report)
Comm. Channels (CO) Key Pattern: *Decorator*	Wrapper library for establishing connections and transferring packets of data. Provides a facade to a system library. Supports optional logging, compression, and encryption. Not very complex. Simple communication primitives with similar interfaces. PAT: ∼360 LOC, 6 classes. Uses a Decorator scheme to add logging, compression, and encryption functionality to a bare channel. ALT: ∼310 LOC, 1 class. Uses flags and if-sequences for turning functionality on and off; the flags are set when creating a channel.	1: *Enable error correction (which is already implemented in the underlying system library) to be added to communication channels.* The ALT variant's functionality is localized, so it will be easier to understand. However, the PAT variant's functionality is encapsulated, so it will be easier to modify (e.g., to add a new primitive, simply add a new Decorator class). Since the latter influence should be stronger, **the PAT variant will be preferable, especially at higher levels of pattern knowledge.** 2: *Determine under which conditions a reset() call will return the 'impossible' result; also, create a channel that performs compression and encryption.* In the ALT variant, state changes are localized and so easier to track than in the PAT variant. Further, creating a channel with the ALT variant requires one statement, whereas PAT requires determining the correct nesting of Decorators. **The PAT groups will take longer and commit more errors.**
Graphics Library (GR) Key Pattern: *Abstract Factory*	Library for creating, manipulating, and drawing graphical objects on output devices. The output device is selected in a central class (generator). Object implementations depend on the selected output device. Objects can be grouped and manipulated like objects themselves. Data structure is partly recursive; more complex than CO. PAT: ∼650 LOC, 13 classes. Uses Abstract Factory for the generator classes, and Composite for hierarchical object grouping. ALT: ∼640 LOC, 11 classes. Uses a single generator class with switch statements for the different devices. Uses a quasi-Composite for object grouping—allows one level of object grouping, but no nested grouping.	1: *Add a new output device.* PAT requires adding a new factory class and extending the factory selector method. ALT requires enhancing the switch statements in all methods of the generator class. Both variants require adding two concrete product classes. Since the volume of changes is similar, the main difference should depend on comprehension. With its localized switch statements, **ALT will be easier to understand, at least for participants with low pattern knowledge. Also, pattern knowledge will help both groups deal with the Composite, though the PAT participants may profit more since they also interact with Abstract Factory.** 2: *Determine whether a given sequence of statements will result in an x-shaped figure.* This is a comprehension test on Composite, where the key is to recognize that references, and not copies of objects, are stored in an object group. The structure of both variants is similar in the region of interest, so **the ALT and PAT groups will not significantly differ. However, the task will require less time at higher levels of pattern knowledge for both variants, due to the Composite pattern.**

UA's was completed within 3 days. For a complete list of the differences, see Appendix B.

Participants. The four sites independently solicited participants. Participants were expected to have a good working knowledge of C++, C#, or Java. Design pattern knowledge was not strictly required. All four teams enlisted students. The student demographic is useful in the case of PatMain because both prior studies employed professionals.

The 53 participants were all solicited from software engineering courses. All were computer science majors or equivalent—including 27 undergraduate and 26 graduate (MS or PhD) students. The median industry experience was 0 years (mean 1.5). More than half of the participants reported understanding (at least roughly) most of the patterns surveyed. However, only one pattern (Observer) was reported by a majority of participants as having ever been used. For most of the participants, their implementation experience with patterns was the result of coursework. Thus, the participants had broad *exposure* to patterns, but little *practical* (and almost no industry) experience with them.

Concerning sites, we find three notable differences: 1) the BYU and FUB participants were mostly undergraduates, whereas the UA and UPM participants were entirely graduate students; 2) the BYU participants reported having used more programming languages than did those at any of the other three sites; and 3) the UA and FUB participants reported greater *exposure* to patterns than did the BYU and UPM participants. For further details on these and other demographics, see Appendix C.

2.1.4 The CO/GR Programs, Tasks, and Hypotheses

E_orig and E_repl both tested all four PatMain programs (ST, BO, CO, GR). E_joint, however, only tested CO and GR. Accordingly, we only consider CO and GR from this point forward. Table 1 summarizes the CO/GR programs, tasks, and hypotheses, which were the same for all three studies.

2.2 Data and Metrics

In this section, we describe the joint dataset. We also provide additional information about explanatory and response variables. Variables not appearing in this section are described sufficiently above. For summary statistics on experiment variables, see Appendix E.

2.2.1 Joint Dataset

Of the 61 participants to take the experiment, we had to discard all data for 8. For a list of the affected participants, see Appendix F. Problems included: failure to complete any of the tasks and failure to adequately record breaks. Two participants also submitted the same data for both programs, and one quit after completing only the CO program. We ended up with data from 52 participants for the CO program and 51 for the GR program (53 total).

We provide an annotated copy of the joint dataset in the lab package. We describe its schema in Appendix G and

the annotation process in Appendix H. Annotation involved scanning by column for outliers and by row for participants who deviated from the instructions. Of the 91 fields in the dataset, we exclude 53 from the statistical analysis. Some fields we exclude because they represent meta- or qualitative data. Others we exclude because, given only 53 participants, we have to limit the number of parameters we estimate in the statistical models (i.e., to avoid over-fitting, multicollinearity, loss of precision, etc. [31]). We also need to reduce the complexity of the models in order to enable theoretical interpretation of the results, which interpretation is necessary in order to generalize the findings across studies. For a list of the unused variables, some of which may be useful in other studies, see Appendix I.

Also note, E_orig defined three tasks for CO, but only two each for ST, BO, and GR. Consequently, the authors of E_repl combined CO tasks 2 and 3 to produce "a more symmetrical experimental design" [20, p. 179]. This approach is reasonable because CO tasks 2 and 3 are similar and address similar hypotheses. Like E_repl, we also combine CO tasks 2 and 3 by summing the task times and averaging the correctness scores. *From this point forward, we treat CO tasks 2 and 3 as a single task, which we collectively refer to as "CO task 2".*

2.2.2 Developer Experience (devExp)

To include in our models all of the developer experience variables we surveyed, we risk diluting statistical power [31, p. 347]. Some of the variables are also highly correlated (well above 0.7), thus leading to multicollinearity problems. Discarding data is not ideal, so instead we take an aggregate approach. By combining metrics, we conserve degrees of freedom, avoid multicollinearity, and hopefully "average out" measurement error [32].

To compose the aggregate metric, we average 4 component metrics: programming languages used, lines of code written, programming hours per week, and self-assessed programming skill. Prior to averaging, we log transform/scale each variable as needed to mitigate outliers and to prevent any single metric from dominating the average. For a detailed description of the process, see Appendix J. The result is a continuous variable ranging from 1 to 7, scaled to match the range of pattern knowledge (described below), where 7 represents high experience.

2.2.3 Pattern Knowledge (patKnow)

Participants estimated their knowledge of 17 design patterns on a 7-point ordinal scale (for a description of the scale, see Section 2.1.3). Like developer experience, and for the same reasons, we average the pattern knowledge scores to produce a single aggregate metric. Although averaging treats the ordinal scale as an interval scale, it should reasonably differentiate the participants.

In addition to the aggregate score, we also tried measuring *patKnow* based on the participants' knowledge of individual patterns (i.e., using only the scores concerning the Decorator and Abstract Factory patterns for the CO and GR programs, respectively). However, using individual (rather than aggregate) scores had little impact on the results (as tested in a preliminary analysis of the BYU data [25]). Thus we prefer to use a single, more general metric for pattern knowledge since such a metric is both consistent across programs and more easily comparable with the prior two PatMain studies.

2.2.4 Time

The web portal's page timings indicate that some participants spent considerable time on the download pages. Since the downloaded files were all less than 25 KB, the download process should have been quick, even for slow Internet connections. Further, the affected participants did not indicate having taken any breaks. Thus, the affected participants likely began working before proceeding to the task description pages—e.g., unzipping files, loading code into IDEs, and reading code.

If some participants performed lengthy tasks on the download page that others performed only on the task description page, then the only way to ensure consistent measurement across participants is to sum the download, work, and upload page timings. Moreover, since the true download/upload times should be trivial compared to the work times, and since all participants had to transfer files, aggregating the download/upload times with the work times should not mask or bias the results. *Consequently, we sum download, work, and upload page timings for all coding tasks (CO and GR task 1s).*

2.2.5 Correctness

To ensure consistency, we had all tasks graded by the same two people, neither of whom were previously affiliated with this study. The two graders were both graduate researchers with professional experience in software development, including one in software testing. For the coding tasks, the graders worked together in a pair-programming style arrangement. They initially reviewed the solutions for each task, from which they decided to use (essentially) the same 5-point scale used in E_repl:

1) *Requirements misunderstood* (0%)—the solution is completely wrong; it appears that the participant did not understand the requirements.
2) *Wrong answer* (25%)—the participant appears to have understood the requirements, but did not produce the correct solution, even conceptually.
3) *Right idea* (50%)—the solution conceptually addresses the requirements, but is incomplete or contains an error and does not compile.
4) *Almost correct* (75%)—the solution conceptually addresses the requirements and compiles, but is incomplete or contains a functional error.
5) *Correct* (100%)—the solution is completely correct; it compiles and meets the stated requirements.

For the short-answer tasks, the graders chose a binary rubric (*incorrect* = 0%, *correct* = 100%). They chose the binary rubric because, in their words, "people did not appear to be guessing; they seemed to either know the answer or not." Each grader then evaluated half of the participants, after which they compared results to ensure consistency. The graders graded holistically, considering everything a participant said, rather than just searching for a specific answer.

2.3 Statistical Models

To address the original PatMain hypotheses, we use frequentist models, which we designed prior to viewing the data. For

the *post-hoc* analysis of moderators, we use both frequentist and Bayesian models.

We add Bayesian models to the moderator analysis because the Bayesian approach provides several key advantages which allow us to form conclusions where the frequentist models are otherwise inconclusive (see Section 2.3.1). However, we do not use the Bayesian models to assess the original PatMain hypotheses because, per our formulation, they are specifically adapted to analysis of large interactions. In other words, although they provide valuable information about moderators where frequentist models fail, they are unnecessarily inefficient when it comes to assessing the original PatMain hypotheses. Moreover, we selected and designed the Bayesian models directly in response to our observations of the data. Thus, the Bayesian models suffer from additional threats to *conclusion validity* (see Section 4.2).

Below, we explain why the Bayesian approach is ideal for moderator analysis. We then describe the frequentist and Bayesian models in detail. Finally, we explain how to interpret the results for each type of model. Source code (R and SAS) for all models is provided in the lab package.

2.3.1 Why Bayesian Models?

First, *the Bayesian models allow us to directly compare probabilities for competing hypotheses, which means we can form conclusions even when statistical power is low.* In frequentist statistics, a p-value is the probability of obtaining data at least as extreme as those observed, assuming a null hypothesis is true. In contrast, Bayesian models yield *posterior* probabilities. A posterior probability is the conditional probability that a hypothesis is true, given the data. Being a probability about the truth of a hypothesis, rather than about the likelihood of the data, we can directly compare posterior probabilities to determine which hypotheses are most likely [33].

The ability to compare probabilities is especially useful for the moderator analysis, which requires modeling high-order interactions. First, adding interactions to any model dampens statistical power by spreading the data over more parameters [31]. Second, the frequentist models yield insignificant, *but very large* effect estimates for many of the interactions we test. Thus, the frequentist results for the moderator analysis are inconclusive; the data are simply too few relative to the variance and model sizes. Conversely, using Bayesian methods, even though the statistical power is minimal, we can still identify likely moderators by comparing posterior probabilities for the appropriate hypotheses.

Second, *the Bayesian models provide samples from the joint posterior distribution, which means we can use fewer parameters to estimate the same set of interactions, thereby conserving statistical power.* We estimate the Bayesian models using Gibbs sampling [33], which, as a Markov Chain Monte Carlo (MCMC) sampling algorithm, generates samples from the joint posterior distribution of all parameters in the model. Given posterior samples, we can use marginalization to compute, from a single high-order interaction, results for all lower-order interactions and component terms [33]. Conversely, in mixed models analysis, lower-order interactions and terms must be estimated by separate parameters [31].

Third, *the Bayesian models are subject to a different set of statistical assumptions, which means they provide validation for the freqentist models (and vice versa)* [31], [33], [34].

TABLE 2
Frequentist statistical models.

Model	Program	Response Variable	Explanatory Variables
CO_time	CO	time_ln (*cont*: ≥ 0)*	see below
CO_correctness	CO	correctness (*cont*: 0–100)	see below
GR_time	GR	time_ln (*cont*: ≥ 0)*	see below
GR_correctness	GR	correctness (*cont*: 0–100)	see below
Blocking Variable (random effect)			
subjectID		Accounts for multiple observations per participant.	
Covariates (fixed effects)†			
order		Program order (*cat*: 1 = first, 2 = second)	
task		Program task (*cat*: 1 = coding, 2 = comprehension)	
site		Sub-replication (*cat*: BYU, FUB, UA, UPM)	
devExp		Developer experience (*cont*: 1–7)	
patKnow		Pattern knowledge (*cont*: 1–7)	
time_ln* or correctness		In a given model, we use whichever variable is not the response variable. Controls for correlations between time and correctness.‡	
Main Effect and Interactions (fixed effects)			
variant		Program variant (*cat*: PAT, ALT)	
patKnow × variant, variant × task, patKnow × task, patKnow × variant × task			

cont = continuous variable; *cat* = categorical variable.
*Normalized via log transformation.
†We tested one other covariate, representing java familiarity, but found it to have almost no impact. See Appendix K for details.
‡E.g., achieving higher correctness simply by working longer.

2.3.2 Frequentist Models

Table 2 summarizes the frequentist models. For this analysis, we use *linear mixed models* [35], an extension of multiple linear regression, which adds the ability to represent blocking (or grouping) variables as random effects. Because the CO and GR programs differ, we analyze their results in separate models. We also run separate models for each response variable. To control for participants who achieve higher correctness simply by working longer, we include *time* as a covariate in the *correctness* models (and vice versa). Also, like E_orig and E_repl, we model (via interaction effects) the impact that *patKnow* has on *variant*, which is necessary to address the original hypotheses.

To normalize the data, we log transform *time*—after which the data conform to the standard assumptions of mixed models analysis, including normality, multicollinearity, and heteroscedasticity. For a detailed assessment of model assumptions, see Appendix L. To maximize statistical power, we tune each model using a standard covariate pruning technique; the technique is essentially backward stepwise regression, but modified to avoid fishing for significance [31, p. 345]. For specific details, see Appendix M. All p-values are two-sided.

2.3.3 Bayesian Models

For the Bayesian analysis, we construct additive-effects models as shown by Felt [34, ch. 4]. By *additive-effects*, we mean models in which the explanatory variables are assumed to have linear (i.e., additive) influences on the response variables. Due to space constraints, we summarize only the Bayesian models below. For a primer on Bayesian analysis, as well as a detailed walkthrough of our models, see [28].

Whereas Felt uses a single variance parameter, we include four in each model, one for each task. We estimate the task

variances separately because some of the tasks require more time than others and longer tasks typically manifest greater variance. Additionally, we represent all explanatory variables as categorical effects. Doing so avoids linearity assumptions and, in that regard, provides validation for the frequentist models. As a drawback, using categorical effects necessitates more parameters in the models than would be needed to represent simple linear relationships. However, given the exploratory purpose of the Bayesian models, we do not want to rely on linearity assumptions in this case.

We divide each continuous variable's range into two parts, *low* and *high*, representing roughly the bottom and top halves of the data, respectively. For a complete list of the divisions, see Appendix N. Note that we could use more than two levels per variable, thus allowing the models to fit more complex relationships. However, adding additional levels causes a multiplicative increase in the number of parameters required to model interactions, which ultimately spreads the data too thin to obtain useful results. In fact, with even three levels per variable, most of the interactions involve at least one level that is completely unrepresented in the data. Conversely, with two levels per variable, all levels of all interactions are adequately represented.

Like the frequentist analysis, we run separate models for each response variable. However, because the Bayesian models allow us to easily estimate separate variances for each task, we model the CO and GR programs together. Since *time* is skewed high and cannot be negative, we model it as a gamma distribution; since *correctness* is a percentage, we model it as a beta distribution. The data conform to the assumptions inherent in the type of models we construct, including multicollinearity and heteroscedasticity. For a detailed assessment of model assumptions, see Appendix L.

For each response variable, *time* and *correctness*, we run 6 models (denoted T1–T6 and C1–C6, respectively). All models include the same set of variables, matching those for the frequentist analysis (shown in Table 2), plus an additional variable representing *program*. Each model also includes one interaction effect. If a variable is included in the interaction, then it is not included elsewhere in the model.[5] The models only differ by which variables are included in the interaction. Table 3 shows, for each model, the interaction being tested.

Concerning prior distributions, we enlisted a qualified external researcher to estimate all priors using data from E_orig. We gave our helper only two constraints (both suggestions of Felt [34]): First, we instructed him to center all priors—with the exception of the variances and base offset—at zero, thus assuming no effect by default (i.e., the null hypothesis). Second, we instructed him to select broad priors in order to minimize their impact on the results. Broad priors are ideal for *post-hoc* analysis, for which the objective is to generate data-driven hypotheses [34]. For a list of the exact priors, see Appendix O.

2.3.4 Results Interpretation—Frequentist vs. Bayesian

In frequentist statistics, p-values are significant when small—i.e., to reject a null hypothesis, the data must be unlikely under the assumption of that hypothesis. For posterior probabilities, however, large values are significant—e.g., 0.95 represents a 95% chance that the associated hypothesis is true, given the data. Further, we typically require very small margins of error in frequentist statistics (e.g., $\alpha = 0.05$); otherwise the results are inconclusive. Conversely, for posterior probabilities, significance depends on the context of the problem. For example, in our analysis, a posterior probability of 0.75 means we expect the associated hypothesis to hold in 75% of similar cases. Under such an interpretation, even relatively low probabilities can be meaningful.

5. Unlike frequentist statistics, Bayesian estimates for low-order interactions and terms can be computed via marginalization from high-order interactions. Thus, a variable need not be modeled both within an interaction and as a standalone effect.

TABLE 3
Bayesian model interactions.

Model(s)	Interaction
T1, C1	program × variant × task × site
T2, C2	program × variant × task × patKnow
T3, C3	program × variant × task × devExp
T4	program × variant × task × correctness
C4	program × variant × task × time
T5, C5	program × variant × task
T6, C6	program × variant

2.4 Moderator Analysis

In this section, we define what we mean by *moderator*, describe how we identify candidate moderators for qualitative and/or quantitative assessment, and explain how we map continuous moderators to general categories for comparison across studies.

2.4.1 What is a Moderator?

By *moderator*, we mean any explanatory variable that interacts with another explanatory variable in predicting a response variable [14, p. 624], [15]. For one variable to "moderate" another does *not* mean that it dampens the other's effect—rather, it means that an interaction exists, such that the latter's effect varies in response to the former. If unaccounted for, the variance induced by a moderator can mask the effect of the variable with which it interacts. Further, if disjoint subsets of a moderator's range are represented in two different studies, then the two studies can produce inconsistent or even contradictory results.

Additionally, when we say *moderator*, we are referring to a phenomenon inherent in nature—i.e., *one variable moderates another's effect on some outcome independent of whether either is experimentally measured or statistically modeled*. Any variable (or more loosely, factor or influence), previously known or unknown, measured or not, can be a moderator. Our goal was to discover the most influential moderators (among those that interact with the main effect), sufficient to explain cross-site variance and to produce generalized conclusions. Thus, we investigated as many potential moderators as possible, *including several that were not measured as part of the experiment*.

2.4.2 Identifying Candidate Moderators

To identify candidate moderators for analysis, we used a relaxed, grounded-theory process involving coding, memoing, and the forming of categories (similar to that described by Charmaz [36]). For source data we used the workshop

reports and datasets from the four sub-replications of E_joint, the reports from E_orig and E_repl, and email conversations between E_joint researchers discussing their experiences. The process yielded a list of candidate variables, from which we selected for analysis those that met the following two criteria: 1) the variable seemed theoretically meaningful, and 2) we had data available that could reasonably represent the variable. *Many of the variables we identified were not considered prior to this analysis, so they do not appear in Table 2.*

The variables selected for analysis include: student vs. professional status, *devExp*, *patKnow*, motivation, task difficulty, *correctness*/*time*, program *order*, perceived time limits, cultural variation, IDE preferences, language barriers, clarity of task instructions, and compilation/testing expectations. We recognize that *correctness* and *time* are response variables, not moderators. However, in their role as covariates (see Table 2), they can reveal insights about moderators, so we include them in the analysis. For measured variables, we statistically assess the extent to which each moderates the main effect. For other variables, we glean what we can from qualitative data. We document all variables explored, even those that turned out to be innocuous.

Lastly, since moderator analysis is *post-hoc*, it inflates the chances of a type 1 error—i.e., testing relationships that were not specified *a priori* increases the risk of incorrectly concluding an effect exists. *Thus, our conclusions relative to moderators are data-driven conjectures which must be further investigated.*

2.4.3 Moderator Categories (low/high)

Some of the moderators we explore are continuous variables. However, in the results discussion, we generalize them in terms of *low* and *high* categories. For the frequentist models, which are linear, *low* and *high* correspond to the min and max values found in the data for the given variable. We use min/max data values rather than the limits of the variables' scales in order to avoid extrapolation (e.g., in the case of *patKnow*, high = 5.412 rather than 7). For the Bayesian models, which make no linearity assumptions, the terms *low* and *high* roughly correspond to the bottom and top halves of the data for each variable, as described in Section 2.3.3.

Concerning pattern knowledge, *low* and *high* correspond to the prior two studies' designations of PRE and POST—meaning before and after the patterns training course. By making this comparison, we are not asserting that PRE and POST map directly onto E_joint's categories of *low* and *high*—especially since the participants' prior pattern knowledge differed across all three studies. Instead, assuming a linear effect, we expect the transition from PRE to POST to be comparable to that from *low* to *high*.

3 Results and Discussion

In this section, we assess the heterogeneity of the results for E_joint, investigate candidate moderators, and address the original hypotheses. To facilitate future meta-analysis, we provide a concise listing of all statistical results in Appendices Y and Z.

3.1 Assessment of Heterogeneity

In this section, we show that E_joint manifests the problem of generalizability—i.e., the main effect varies across sites.

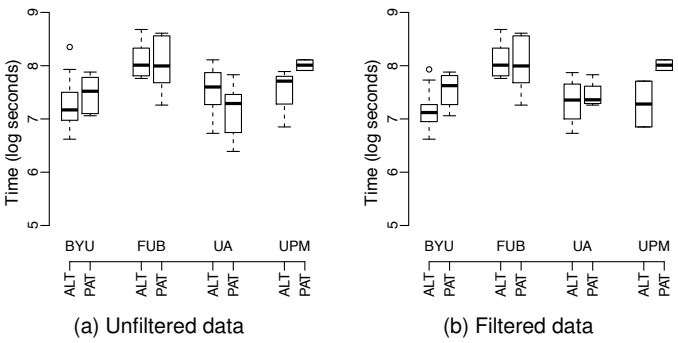


Fig. 1. Time data for CO task 1, showing ALT versus PAT displayed by site. Max whisker range is 1.5 IQR.

TABLE 4
Frequentist model p-values for *site* and *variant*. p-values less than or equal to 0.05 are bolded.

	Unfiltered		Filtered	
Model	*site*	*variant*	*site*	*variant*
CO_time	**<0.001**	0.925	**<0.001**	**0.019***
CO_correctness	**0.003**	0.095	**0.008**	0.523
GR_time	**0.019**	**0.016**	**<0.001**	**0.025**
GR_correctness	NS	0.322	NS	0.245

NS = not significant—i.e., the exact value is not available since the variable was removed during model tuning due to lack of significance. For a description of model tuning, see Appendix M.
*Min p-value for *variant* within the *patKnow* × *variant* interaction.

We demonstrate this problem via both the frequentist and the Bayesian models. We also explore data filtering as a method to mitigate the problem. The purpose of this section is to validate that moderator analysis is needed to make sense of E_joint's dataset.

3.1.1 Heterogeneity in E_joint

Fig. 1a depicts the main effect, program *variant*, compared across the four sites (before filtering). Notice that the sites significantly differ—in two cases PAT takes more time than ALT, in another case no effect can be seen, and in the remaining case the effect is reversed. We find similar divergences on all tasks, for both *time* and *correctness* (the remaining plots are shown in Appendix Q). *Thus, with respect to variant, the results do not generalize well across the four sites.*

In the (unfiltered) frequentist models, this problem shows up as *site* being considerably more significant than *variant* (see Table 4). Similarly, in the Bayesian models, we find (for all tasks) that the effect of *variant* is more significant at individual sites than when generalized across sites (*u*T1,C1:46–54,791–826).[6] For example, on CO task 2, the marginal

6. We use references of this form to map specific assertions back to the Bayesian probabilities on which they are based. The references are necessary because the Bayesian analysis produces more data than we have room to list in the main paper. The references provide an audit trail, as well as a more complete working example for future replicators. The paper is written such that the references can be ignored. The reference notation is as follows: *u* and *f* signify unfiltered versus filtered data—i.e., Tables 59 and 60 in Appendix Z. T# and C# indicate a specific *time* or *correctness* model—i.e., one of the models T1–T6 or C1–C6, which are represented as columns in the tables. Lastly, a colon indicates a list of row numbers—e.g., *u*T1:12 signifies Table 59 (unfiltered data), column T1, row 12. Abbreviations resulting from this notation include: *u*T1, *u*C1, *f*T1, *f*C1, *u*T2, *u*C2, *f*T2, *f*C2, *u*T3, *u*C3, *f*T3, *f*C3, etc.

(i.e., generalized) probability that PAT took longer than ALT is only 0.58—compared to, for example, 0.86 at UPM (uT1:51,807).

Clearly, cross-site variance is inhibiting isolation of the main effect—otherwise, aggregating data across sites would yield an increase in statistical significance. Thus, in addition to being a replication of PatMain, E_joint evokes the broader problem of generalizability that confronts the PatMain series of studies. Moreover, E_joint reproduces that problem within the context of a single, controlled experiment. Thus, much of the variance observed across sites can reasonably be attributed to meaningful variables rather than to experimental artifacts. This setup is ideal for studying moderators.

3.1.2 Heterogeneity in E_repl and E_orig

E_repl, which involved participants from 11 companies, encountered problems with variance similar to E_joint. As Marek Vokáč describes, "[Our participants] came from some of the major (even international) consultancy companies, and they were paid quite well for their efforts" (email, Oct. 14, 2012). Nevertheless, the analysis required "an expert from the statistics section of the Math faculty" because "the 'usual' methods were not enough to extract a good signal from the fairly noisy data" (email, Oct. 16, 2012). Conversely, all of E_orig's participants came from a single company, and variance was not a significant concern in that case.

3.1.3 Filtering to Reduce Heterogeneity

E_repl used two types of data filtering to reduce variance, which we refer to as *observation* and *participant* filtering. Observation filtering—i.e., discarding all data points with a *correctness* score below 75% (*time* models only)—is too course grained to effectively reduce cross-site variance in E_joint (for further details, see Appendix P). Participant filtering, however, does account for at least some of the cross-site variance.

Concerning participant filtering in E_repl, "Four subjects had consistently low-quality solutions. Inspection. . . revealed that their C++ proficiency was so low that it would significantly mask any other effect" [20, p. 162]. Consequently, Vokáč *et al.* discarded all data for these participants, concluding that no bias was introduced since the PAT and ALT variants were equally affected. However, Vokáč *et al.* did not specify a process for identifying underqualified participants.

Therefore, to filter participants in E_joint, we determined a filtering threshold based on the participants' average task *correctness*. By averaging across tasks, we were able to factor out *variant*, such that the choice of threshold would not bias the final results. At a threshold of 25 percentage points (unanimously selected by 4 independent reviewers), 10 participants were excluded (6 UA, 3 BYU, 1 UPM, 0 FUB). These participants were likely underqualified and/or insufficiently motivated. For further details on the filtering process, including a list of the 10 participants excluded, see Appendix Q.

Ultimately, the filtering increased statistical precision by reducing cross-site variance, but without significantly altering the main effect estimates (e.g., see UA in Fig. 1b; see also Table 4). The filtered models also make more sense. First, *patKnow* becomes significant in three of the four frequentist models (see Table 6), consistent with the prior two studies. Second, the *patKnow* × *variant* interaction becomes significant in the CO_time model (p-value = 0.029), which is consistent with the notion that pattern knowledge should be more meaningful for the PAT variant than for the ALT variant.

However, the filtering did not significantly affect either of the *correctness* models or the GR_time model (see Table 4); *site* is also still significant in the same models as before, and the Bayesian results (fT1,C1:46–54,791–826) still show *variant* to be more significant within sites than across sites—though the disparity is reduced. *Thus, although the participant filtering reduces cross-site variance, it does not fully explain such variation.*

3.2 Assessment of Moderators

In this section, we investigate candidate moderators. Note that we use both the frequentist and the Bayesian models for this discussion; also, *all conclusions from this point forward are limited to: 1) the patterns tested in the CO/GR programs (Decorator and Abstract Factory), 2) maintenance activities, 3) maintainers that were not the original implementers, and 4) programs for which the full functionality of the patterns was not initially needed.*

3.2.1 How to Read Table 5

Table 5 shows results for the Bayesian assessment of moderators. Rows represent candidate moderators; columns represent interactions. The *candidate*-columns (i.e., c-columns) represent *variant* × *program* × *task* × *candidate* interactions, where *candidate* is the candidate moderator for the given row; the $\neg c$-columns represent *variant* × *program* × *task* interactions, where the given candidate has been marginalized out. The $\neg c$-columns provide a baseline against which to compare the c-columns. The baseline values within each $\neg c$-column differ because the candidates were tested in separate models. The data were simply too few to test all of the interactions via a single model. Thus, we can quantitatively assess which variables are likely moderators, but we cannot quantitatively assess which moderators are most influential.

Each probability in Table 5 represents the max significance of *variant* to occur at any level of the associated interaction. If a variable moderates *variant*, then by definition, *variant*'s effect must vary across the levels of the moderator. Consequently, the significance of *variant* will always be greater for at least one level of the moderator than it is when the moderator is marginalized out. Thus, if a moderator is significant, $c \gg \neg c$. For example, the top-left probability in Table 5, 0.62, is the max significance of *variant* to occur among the four levels of *program* × *task* (unfiltered data, model T1). The next probability to the right, 0.86, is the max significance of *variant* to occur among the 16 levels of *program* × *task* × *site*. Since the max significance of *variant* substantially increases when *site* is added to the interaction, *site* represents a likely moderator.[7]

As mentioned previously, *site* does not help us to generalize because it is not a software-domain variable; thus, we need to identify other, more meaningful moderators. Also,

7. By *significance*, we mean the maximum of p(ALT>PAT) versus p(ALT<PAT), where p is the posterior probability. Using the greater of the two is appropriate because, for binary comparisons, non-significance is at 0.5. Thus, values such as 0.25 and 0.75 are equally significant. Further, the directionality of the ALT/PAT comparison is irrelevant. All that matters is the degree to which each moderator increases the significance of *variant* when added to the interaction.

TABLE 5
Bayesian interaction results—moderator assessment. Probabilities are shown in black; effect estimates in gray. Probabilities greater than 0.75 are bolded. See Section 3.2.1 for a description of how to read and interpret this table. For a visualization of the data, see Appendix R.

time Models	Unfiltered		Filtered		*correctness* Models	Unfiltered		Filtered	
	$\neg c$	c	$\neg c$	c		$\neg c$	c	$\neg c$	c
site	0.62	**0.86**	0.67	**0.88**	*site*	0.72	**0.81**	0.73	**0.87**
(T1)	196	653	309	842	(C1)	10.5	18.2	11.7	21.0
patKnow	0.64	0.67	0.71	**0.85**	*patKnow*	0.71	**0.92**	0.73	**0.86**
(T2)	153	169	298	504	(C2)	8.7	17.3	8.5	13.7
devExp	0.66	**0.90**	**0.76**	**0.89**	*devExp*	0.70	**0.85**	0.67	**0.89**
(T3)	237	561	398	600	(C3)	7.5	12.7	6.9	15.8
correctness	0.68	**0.83**	**0.81**	**0.88**	*time*	**0.81**	**0.93**	**0.82**	**0.85**
(T4)	218	408	404	821	(C4)	10.0	18.0	10.9	12.8

TABLE 6
Frequentist model p-values for *devExp* and *patKnow*. p-values less than or equal to 0.05 are bolded.

Model	Unfiltered		Filtered	
	*devExp**	*patKnow*	*devExp**	*patKnow*
CO_time	0.076	NS	NS	**0.007**†
CO_correctness	0.101	NS	NS	**0.049**
GR_time	<**0.001**	NS	NS	**0.001**
GR_correctness	NS	NS	NS	NS

NS = not significant—i.e., the exact value is not available since the variable was removed during model tuning due to lack of significance. For a description of model tuning, see Appendix M.
*E_repl also tested an experience covariate—a "pre-qualification score" [20, p. 157]—which was an aggregate metric similar to *devExp*. It was not found to be significant, but that result was obtained only after filtering low-scoring participants.
†Min p-value for *patKnow* within the *patKnow* × *variant* interaction.

notice that the results for *site* in Table 5 are consistent with the assessment of heterogeneity in Section 3.1. Thus, Table 5 is a shorthand method for showing a moderator's influence. For each variable listed in the table, we could have provided a lengthy assessment similar to that given for *site* in Section 3.1.

Finally, each effect estimate in Table 5 represents the magnitude of the difference between ALT and PAT (in seconds or percentage points) associated with the corresponding probability. We provide these estimates to show the practical significance of the moderators.[8]

3.2.2 Students vs. Professionals

We consider the student-professional distinction first because it represents a notable experimental difference between the three PatMain studies. It is possible that E_joint's use of students caused its results to differ from those of the prior two studies. For instance, the participant filtering, which brought E_joint's results into greater alignment with the prior studies, may have distilled from the student data a more professional-like sample. After all, the median experience in E_joint was much lower than in E_orig and E_repl (0 years, compared to 3.5 and 4), and accordingly, E_joint had to filter more participants (10/53 versus 0/29 and 4/44).

Likely, our use of students did involve a higher percentage of underqualified participants. However, the prior two studies both used professionals, with similar experience, and yet they still differed in their results. Also, as previously mentioned, E_repl encountered variance problems similar to E_joint, whereas E_orig did not. *Thus, the student-professional distinction is too simplistic to align well with the cross-site variance; accordingly, it is not an effective variable (from a methodological standpoint) by which to generalize the PatMain results.* From this point forward, we explore other variables, several of which likely underlie the student-professional distinction.

3.2.3 Developer Experience

As a covariate, *devExp* is significant in the frequentist models only before filtering (see Table 6). A 1-unit (or approx. 17%) increase in *devExp* corresponds with a *time* decrease of 10% for CO (80% confidence interval (CI): 3–17) and 30% for GR (80% CI: 22–37),[9] as well as a *correctness* increase of 7 percentage points for CO (80% CI: 2–13).

As a moderator, we make four observations about *devExp* (see Table 5): 1) *devExp* strongly increases the significance (or predictive capability) of *variant* when the two are interacted; 2) *devExp* interacts with *variant* for both response variables; 3) filtering reduces the interaction for *time*, but not for *correctness*; and 4) filtering does not completely eliminate the interaction for either response variable. Thus, *devExp* moderates *variant*, both before and after filtering. Accordingly, we conclude that *generalizing across sites will likely require contextualizing the conclusions with respect to developer experience; generalizing across studies (as opposed to sites) may also require a standardized experience assessment*.

Concerning impact, we find that *variant* is most significant when *devExp* is low (*u*T3,C3:277–296 and *f*T3,C3:277–296). This is true for nearly all tasks, before and after filtering, and for both response variables. Also, when *devExp* is low, PAT tends to take longer and score lower. Thus, *using Decorator or Abstract Factory during maintenance instead of a simpler solution is likely detrimental to inexperienced developers (in the general case)*. Conversely, when *devExp* is high, PAT tends to have little impact. Thus, using Decorator or Abstract Factory during maintenance instead of a simpler solution *may* have little or no impact in industry (in the general case). However, based on the analysis presented in Section 3.3 (in which we compare data *across* the three PatMain studies), we find that the positive effect of devExp on the utility of patterns continues to increase as experience increases from students to professionals. Thus, the more likely conclusion is that *using Decorator or Abstract Factory during maintenance instead of a simpler solution is preferable in industry (in the general case), as long as the developers have more experience than our student participants*. Note that the threshold of experience needed for a specific pattern to be beneficial varies from pattern to

8. As described in Footnote 7, the directionality of the ALT/PAT comparison is irrelevant when assessing a moderator's influence. Thus, we use effect magnitudes—i.e., we drop negative signs.

9. For the frequentist models, we normalized *time* by log transformation prior to analysis. Thus, the *time* models are linear on the log scale, but exponential on the original scale. Accordingly, a 4-unit increase in *devExp* does not equal, e.g., an impossible 120% decrease in *time* for the GR program. Instead, the stated percentage decrease must be repeatedly applied for every 1-unit change in *devExp*, such that *time* asymtotically approaches zero as *devExp* increases.

pattern; as we show in Section 3.3, the threshold is higher for Abstract Factory.

3.2.4 Pattern Knowledge

In contrast to *devExp*, *patKnow* is significant as a covariate only after filtering (see Table 6). Thus, removing low-scoring participants enables detection of the *patKnow* effect by mitigating the effect (or interference) of *devExp*. After filtering, a 1-unit (or approx. 17%) increase in *patKnow* yields an average *correctness* increase of 11 percentage points for CO (80% CI: 4–17), as well as a 30% *time* decrease for GR (80% CI: 20–39). *patKnow* is also significant in the CO_time model, but in that case its marginalized effect is not meaningful because the *patKnow* × *variant* interaction also becomes significant (p-value = 0.029). The *patKnow* × *variant* interaction is depicted in Fig. 2. The figure shows that *patKnow* affects only the PAT variant. We discuss the frequentist interaction results for *patKnow* further in the next section, at which point we address the original PatMain hypotheses.

The *patKnow* × *variant* interaction is strongly supported in the Bayesian models (see Table 5): 1) *variant* increases in significance when interacted with *patKnow*; and 2) like *devExp*, *patKnow*'s moderating influence applies to both response variables (though for *time*, only after filtering). Thus *patKnow* moderates *variant*. This result is not surprising since the original hypotheses anticipated an interaction between *patKnow* and *variant*. It makes sense that knowing more about patterns would help participants cope with the PAT variant more so than with the ALT variant. Thus, similar to *devExp*, we conclude: *generalizing across sites and studies will likely require contextualizing the conclusions with respect to pattern knowledge, and may also require a standardized pattern knowledge assessment*.

Concerning impact, *patKnow* is similar to *devExp* (*u*T2,C2: 167–186 and *f*T2,C2:167–186). First, *variant* is most significant when *patKnow* is low. Second, when *patKnow* is low, the PAT variant tends to take longer and score lower. Third, when *patKnow* is high, *variant* is largely insignificant. Thus, we conclude: *using Decorator or Abstract Factory during maintenance instead of a simpler solution is likely detrimental to developers with little knowledge of patterns (in the general case); conversely, it may be preferable in industry (in the general case), as long as pattern knowledge among professional developers is greater, on average, than among our student participants*. Similar to *devExp*, the threshold of knowledge needed for a specific pattern to be beneficial varies from pattern to pattern; again, as we show in Section 3.3, the threshold is higher for Abstract Factory.

3.2.5 Motivation

We find evidence that motivation explains cross-site variance both within E_joint and across the three PatMain studies. First, according to Lutz Prechelt, the FUB participants were "true volunteers," who not only stuck around despite two reschedulings, but who were also "helping a fellow student" whose bachelor thesis depended on their participation. Thus, the primary motivations at FUB were likely intrinsic. Accordingly, the FUB participants spent far more time on the experiment than any other site (47% more time on average than the next highest site); also, of the 10 participants filtered, *none* were from FUB. In contrast, the primary incentive at

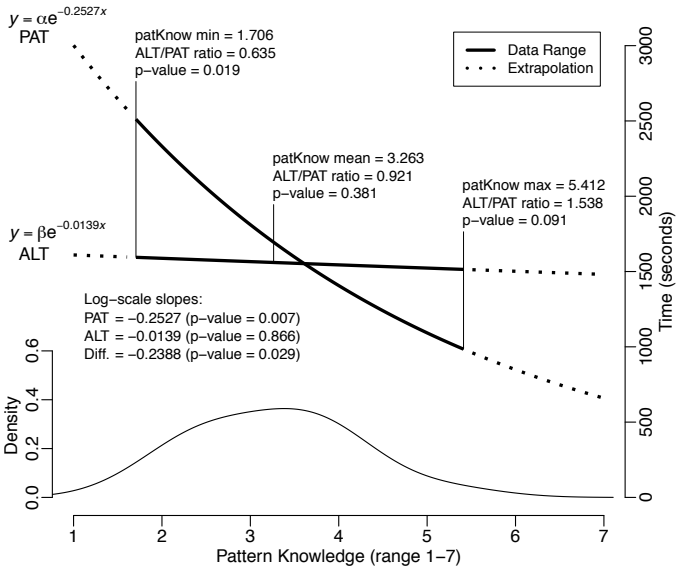


Fig. 2. Frequentist results for the *patKnow* × *variant* interaction (CO_time model, filtered data). Since *time* was logged prior to analysis, the back-transformed results are exponential. Intercepts ($\alpha = 3865.3$, $\beta = 1632.6$) are relative to *site*=BYU, *order*=1, *task*=1, *correctness*=60. Other settings for these variables scale the y-axis, but do not alter the ratios or p-values shown.

BYU and UA was course credit, and in both of those cases many participants had to be filtered. Moreover, the majority of the filtered participants (6 of 10) were from UA—the only site to *require* participation as a class assignment.

Motivation also explains why E_repl required filtering, whereas E_orig did not, even though both studies used professional consultants. For E_orig, the CEO invited the study (which was conducted during normal business hours), participation was voluntary, and the primary incentive was to learn about design patterns (which at the time were still relatively new). Conversely, E_repl's participants were *selected* to participate by their consultancy firms. Thus, although the participants in both studies were essentially paid for their time, it would seem that E_orig's participants were more intrinsically motivated. If so, motivation correlates not only with cross-site variance in E_joint, but also with the fact that E_orig was the only PatMain study to not need filtering.[10]

Prior to the moderator analysis, the possibility of motivation being a moderator did not occur to us, so we did not collect data on the variable as part of the experiment. Of the variables considered so far, motivation is the most in need of further investigation. At the very least, we can conclude that *motivation could strongly influence study outcomes, inasmuch as it affects statistical variance; however, we cannot determine, based on the available data, whether motivation directly moderates the effect of design patterns*. Accordingly, *we recommend that future design pattern studies report on motivation*. Reports should describe both the formal incentives and any other variables that may have motivated or demotivated the participants. A

10. E_orig was also the only PatMain study to use a paper-based format, so the format could explain the differences in filtering across studies. However, because the format was kept constant within E_joint, it cannot explain the differences in filtering across sites. Thus, motivation is the more likely explanation for the observed variance.

post-experiment survey may be useful to gather such data. We also suggest experimentally controlling motivation in some studies (e.g., by testing multiple types of incentives).

3.2.6 Other Variables

We analyzed several additional variables, many of which correlate with cross-site variance. In most cases, the data are insufficient to statistically test for a moderating effect. We summarize our findings below. For the detailed discussions, see Appendix S.

- *task difficulty*: When a task exceeds a certain threshold of difficulty, relative to a developer's experience and/or motivation, the use of patterns appears to have no effect on his/her performance.
- *correctness/time*: These variables positively correlate, meaning the participants likely achieved higher scores by working longer. Also, the variables statistically interact with *variant* before filtering, but not after. Since the filtering targets (in part) undermotivated participants, the interactions suggest that motivation could moderate *variant*.
- *program order*: Performance tended to improve by a small margin on the second program (i.e., lower times and higher correctness), thus indicating a learning (or maturation) effect. The learning effect appears unrelated to design patterns, so we correct for it via statistical modeling.
- *perceived time limits*, *cultural variation*, *IDE preferences*, *language barriers*, *clarity of task instructions*, *compilation/testing expectations*: These variables correlate with cross-site variance, but the data are insufficient to determine (even informally) whether they moderate *variant*.

3.2.7 Summary of Results

We find evidence that both developer experience and pattern knowledge influence cross-site variance. We also find evidence that these variables moderate the effect of design patterns. Thus, both variables will likely have to be better understood and controlled in order to fully resolve the problem of generalizability. Pattern knowledge was anticipated to be a moderator prior to E_orig; developer experience, however, has not previously been considered as such.

Additionally, we have identified motivation as a strong candidate for explaining cross-site and cross-study variance. Our data are insufficient to statistically test whether it moderates the effect of patterns, but we find indirect evidence for it as a moderator via the *correctness* and *time* covariates. At the very least, motivation could strongly influence study conclusions, inasmuch as it influences statistical variance.

Lastly, we have documented several variables in Appendix S (summarized above) that correlate with cross-site variance. Although our data are insufficient to statistically test whether most of these variables moderate *variant*, some of them may prove to be important in future studies.

3.3 Assessment of the Original Hypotheses

Table 7 shows the final results compared across the three studies. To preserve the integrity of Table 7, the results for E_joint are taken exclusively from the frequentist models, which we designed prior to viewing the data and which have not been altered by the moderator analysis. We integrate the moderator analysis into the discussion, however, to resolve contradictions between the PatMain studies.

Like E_repl, our results are based on the filtered data (described in Section 3.1). The filtering, which targets underqualified and undermotivated participants, increases statistical precision without significantly altering the main effect estimates. Also, *the conclusions in this section are limited to: 1) the patterns tested in the CO/GR programs (Decorator and Abstract Factory), 2) maintenance activities, 3) maintainers that were not the original implementers, and 4) programs for which the full functionality of the patterns was not initially needed.*

We discuss each of the four tasks in turn. Recall that *coding tasks* required modifying the code, whereas *comprehension tasks* tested comprehension of the code.

3.3.1 CO Task 1 (Decorator, Coding Task)

The results for this task significantly contradict across the three PatMain studies. However, given our findings from the moderator analysis, we can resolve most of the contradictions. First, the moderator analysis found that the PAT variant was most beneficial (or least harmful) when pattern knowledge was high. On CO task 1, we see this effect manifest in both E_repl and E_joint (Table 7, rows 2–3). Second, if we rank the studies by their participants' average pre-experiment pattern knowledge (in decreasing order: E_orig, E_repl, E_joint), we see that the benefit of the PAT variant decreases across the studies, following the order of decreasing pattern knowledge (Table 7, rows 2–3, 8–9). Thus, although the results significantly differ across the studies, they agree when considered in reference to the moderating effect of pattern knowledge. As predicted by the moderator analysis, *the time (and possibly also correctness) benefits of the Decorator pattern positively correlate with pattern knowledge on this coding task.*

Further, notice that the pattern training had little impact in E_orig—i.e., before training, the PAT effect was −63%, compared to −55% after (Table 7, rows 2–3). This trend makes sense given that, of the three PatMain studies, E_orig's participants had the most prior pattern knowledge. However, according to E_orig's published report, only 52% actually had prior pattern knowledge. Thus, not only does pattern knowledge positively correlate with the benefits of the Decorator pattern (as shown above), but *it appears that only minimal prior knowledge of the Decorator pattern is needed in order to realize a substantial benefit on this coding task*. In fact, the only case in which ALT was significantly better than PAT on this task occured for the *least knowledgable* E_joint participants—i.e., *students* with almost no practical pattern experience whatsoever.

Ultimately, given an understanding of pattern knowledge as a moderator, the original hypothesis for CO task 1 is confirmed. *The Decorator pattern is indeed preferable on this task, especially at higher levels of pattern knowledge—the only caveat being that if pattern knowledge is too low, the time and correctness benefits may be negated*.

3.3.2 CO Task 2 (Decorator, Comprehension Task)

For this task, the three studies mostly agree. First, PAT took significantly *longer* than ALT in all three cases when pattern knowledge was *low* (Table 7, row 15). Second, when

TABLE 7
Comparison of results across the three PatMain studies.*

Hypothesis Statement	Concrete Hypotheses	Baseline & Expectation		E_orig	Reanalysis of E_orig†	E_repl	E_joint‡	
CO Task 1, Coding The PAT variant will be preferable, especially at higher levels of pattern knowledge.	$t : P < A$	A	$-$	-38% ($<.001$)	negative	-	$+10\%$ ($.344$)§	1
	$t : PH < AH$	AH	$-$	-35%	-55% ($<.05$)	-49% ($<.05$)	-35% ($.091$)	2
	$t : PL < AL$	AL	$-$	-41%	-63% ($<.05$)	$+13\%$ ($>.05$)	$+58\%$ ($.019$)	3
	$t : H < L$	L	$-$	-3%	-	-	-36% ($.090$)§	4
	$t : PH < PL$	PL	$-$	$+8\%$ ($.29$)	$+10\%$ ($>.05$)	-49% ($<.05$)	-61% ($.007$)	5
	$t : AH = AL$	AL	0	-1% ($.46$)	-17% ($<.05$)	$+13\%$ ($>.05$)	-5% ($.866$)	6
	$c : P > A$	A	$+$	positive	positive	positive	-5 pp ($.523$)	7
	$c : PH > AH$	AH	$+$	positive	$+43$ pp ($<.05$)	$+15$ pp ($<.05$)	INS	8
	$c : PL > AL$	AL	$+$	positive	$+43$ pp ($<.05$)	$+13$ pp ($<.05$)		9
	$c : H > L$	L	$+$	no diff.	no diff.	no diff.	$+39$ pp ($.049$)	10
	$c : PH > PL$	PL	$+$	no diff.	0 pp ($>.05$)	0 pp ($>.05$)	INS	11
	$c : AH = AL$	AL	0	no diff.	$+3$ pp ($>.05$)	-3 pp ($>.05$)		12
CO Task 2, Comprehension The PAT groups will take longer and commit more errors.	$t : P > A$	A	$+$	$+72\%$	positive	positive	$+10\%$ ($.344$)§	13
	$t : PH > AH$	AH	$+$	$+50\%$	$+65\%$ ($>.05$)	$+9\%$ ($>.05$)	-35% ($.091$)	14
	$t : PL > AL$	AL	$+$	$+91\%$	$+130\%$ ($<.05$)	$+117\%$ ($<.05$)	$+58\%$ ($.019$)	15
	$c : P < A$	A	$-$	negative	negative	-	-5 pp ($.523$)	16
	$c : PH < AH$	AH	$-$	-	-15 pp ($>.05$)	-4 pp ($>.05$)	INS	17
	$c : PL < AL$	AL	$-$	-	-35 pp ($<.05$)	$+13$ pp ($>.05$)		18
GR Task 1, Coding ALT will be easier to understand, at least for participants with low pattern knowledge; pattern knowledge will help both groups, though the PAT participants may profit more.	$t : P > A$	A	$+$	$+17\%$ ($.10$)	positive	-	$+41\%$ ($.025$)	19
	$t : PH > AH$	AH	$+$	$+19\%$	$+30\%$ ($>.05$)	$+40\%$ ($>.05$)	INS	20
	$t : PL > AL$	AL	$+$	$+11\%$	$+35\%$ ($>.05$)	-17% ($>.05$)		21
	$t : H < L$	L	$-$	-21% ($.021$)	negative	positive	-73% ($.001$)	22
	$t : PH < PL$	PL	$-$	-17% ($.17$)	-20% ($>.05$)	$+62\%$ ($<.05$)	INS	23
	$t : AH < AL$	AL	$-$	-23% ($.031$)	-22% ($<.05$)	$+2\%$ ($>.05$)		24
	$c : P < A$	A	$-$	-	-	-	-10 pp ($.245$)	25
	$c : PH < AH$	AH	$-$	-	$+21$ pp ($<.05$)	-3 pp ($>.05$)	INS	26
	$c : PL < AL$	AL	$-$	-	$+3$ pp ($>.05$)	$+34$ pp ($<.05$)		27
	$c : H > L$	L	$+$	-	-	-	NS	28
	$c : PH > PL$	PL	$+$	-	$+10$ pp ($>.05$)	-9 pp ($>.05$)	INS	29
	$c : AH > AL$	AL	$+$	-	-11 pp ($>.05$)	$+25$ pp ($<.05$)		30
GR Task 2, Comprehension ALT and PAT will not significantly differ; the task will require less time at higher levels of pattern knowledge for both variants.	$t : P = A$	A	0	-21% ($.085$)	negative	negative	$+41\%$ ($.025$)	31
	$t : PH = AH$	AH	0	-26%	-20% ($>.05$)	-39% ($>.05$)	INS	32
	$t : PL = AL$	AL	0	-20%	-30% ($>.05$)	-9% ($>.05$)		33
	$t : H < L$	L	$-$	-21% ($.091$)	negative	positive	-73% ($.001$)	34
	$t : PH < PL$	PL	$-$	-26%	-17% ($>.05$)	$+11\%$ ($>.05$)	INS	35
	$t : AH < AL$	AL	$-$	-20%	-28% ($>.05$)	$+66\%$ ($>.05$)		36
	$c : P = A$	A	0	-	no diff.	-	-10 pp ($.245$)	37
	$c : PH = AH$	AH	0	-	0 pp ($>.05$)	-31 pp ($<.05$)	INS	38
	$c : PL = AL$	AL	0	-	$+5$ pp ($>.05$)	$+1$ pp ($>.05$)		39

*Due to ambiguity in the original hypotheses, E_orig and E_repl tested slightly different things. To facilitate comparison of their results, we define concrete hypotheses, where:
t, c = *time*, *correctness* response variables.
P, A = PAT, ALT variants.
H, L = high, low pattern knowledge.

Results for each row are computed relative to the baseline. For example, on the first row, -38% means that E_orig estmiated PAT to cause a 38% reduction in time relative to ALT. The expectation $(+, -, 0)$ represents the hypothesized direction of the results relative to the baseline (up, down, or no effect).

Non-numeric entries indicate that no statistical results were reported, but data were given about the direction of the effect. A dash (-) means no data were given at all about the effect. Gray entries had to be extracted from plots; their magnitudes are subject to a margin of error (about ± 5).

pp = percentage points (0–100%).

(I)NS = (interaction) not significant—i.e., the exact values are not available because the variable (or interaction) was removed during model tuning due to non-significance. For details on model tuning, see Appendix M.

p-values are provided in parentheses where available; all p-values are two-sided. As discussed in Appendix T, the statistical methods are sufficiently similar between the studies that we can directly compare the numerical results.

The CO and GR task 2 hypothesis statements do not address all twelve of the possible concrete hypotheses. Results for the remaining combinations are shown in Appendix U.

†E_repl reanalyzed E_orig's data. This column summarizes the results of that reanalysis.

‡In E_joint, the *task* interactions are all insignificant. Thus, the results shown for the two tasks in each program are the task-independent results repeated as necessary.

§These values are taken from the CO_time model (filtered), as defined in Section 2.3.2, but with all interactions dropped.

pattern knowledge was *high*, none of the *time* effects were statistically significant (Table 7, row 14). Third, the *correctness* effects were nearly all insignificant, regardless of pattern knowledge (Table 7, rows 17–18). Thus, we conclude that, *for developers with little pattern knowledge, the ALT variant is preferable (at least in terms of time, but possibly also correctness) on this comprehension task*. We also tentatively conclude that, *for developers with high knowledge, the PAT variant is probably no worse than ALT (in terms of both time and correctness)*. Thus, the hypothesis for CO task 2 is confirmed for developers with low pattern knowledge, but tentatively contradicted for those with high pattern knowledge.

Also, as with task 1, pattern knowledge positively correlates with the benefits of the Decorator pattern. However, a greater minimum level of knowledge is required on this *comprehension* task for PAT to be beneficial, than was needed on the *coding* task (task 1). Additionally, we recognize that large estimates, even though statistically insignificant, may still represent meaningful effects (e.g., +65%; Table 7, row 14). Thus, we label the second conclusion for this task as "tentative." Greater confidence requires either larger sample sizes or reduced within-study variance.

3.3.3 GR Task 1 (Abstract Factory, Coding Task)

For this task, the *time* effect mostly agrees across the studies—PAT takes *more time* than ALT (Table 7, rows 20–21). However, the results are statistically significant for only one of the three studies, E_joint. Coincidentally, E_joint's participants had the *least* developer experience of all the PatMain studies. Further, the one estimate showing a reverse effect (PAT takes *less time* than ALT) occurred in E_repl—i.e., for the participants with the *most* developer experience. Since the moderator analysis identified developer experience as a likely moderator, we tentatively conclude: *For developers with little professional experience, ALT will likely take less time on this coding task than PAT, but with sufficient experience, the reverse may be true.* However, given that E_repl's participants had a median of 4 years (mean 6.6), the level of developer experience needed for PAT to outperform ALT is likely high. Possibly, for developers with greater pattern knowledge (than E_repl's participants), the minimum level could be less.

As for *correctness*, the estimates are mostly small and insignificant (Table 7, rows 26–27). Thus, Abstract Factory likely has little impact on *correctness* for this coding task (in the general case). However, two estimates, one for E_orig and one for E_repl, are large (+21 and +34 percentage points; rows 26 and 27, respectively). Also, the results for pattern knowledge are scattered and inconsistent (Table 7, rows 23–24, 29–30). Since neither of these divergences are explained by the moderator analysis, further investigation is needed.

For now, we tentatively conclude that *PAT likely does not impact correctness in general on this coding task, but in some (as yet unknown) cases, it may promote higher correctness*. Concerning pattern knowledge, we conclude: 1) *pattern knowledge is not always helpful on this coding task (in terms of both time and correctness)*; 2) *the conditions under which it is helpful are still unknown*; and 3) *when it is helpful, it does not appear to help the PAT group more than the ALT group*. Thus, the original hypothesis for GR task 1 is (tentatively): partially confirmed for *time*, rejected for *correctness*, and at least partially rejected for pattern knowledge.

3.3.4 GR Task 2 (Abstract Factory, Comprehension Task)

On this task, the *time* and *correctness* estimates are mostly insignificant (Table 7, rows 31–39), which supports the original hypothesis that ALT and PAT will not differ. However, we again encounter the problem of insignificant, but large effect estimates (e.g., +66%; Table 7, row 36), which means we cannot confidently conclude the null hypothesis. We also find significant contradiction in estimates for both *time* and *correctness*, for which the moderator analysis provides no resolution. In the few cases of statistical significance, especially those of E_joint, PAT took longer and scored lower than ALT. Thus, without further investigation of moderators, we can only tentatively conclude: *In general, PAT likely does not significantly impact time or correctness on this comprehension task; however, in at least some contexts (possibly those of low developer experience), PAT probably does have a harmful influence*; also, *in at least some (as yet unknown) contexts, developers do take less time on this task at higher levels of pattern knowledge, but in other contexts they do not*. Accordingly, the original hypothesis for GR task 2 is (tentatively): partially confirmed for *time*, *correctness*, and pattern knowledge, but also partially rejected for pattern knowledge.

3.3.5 Summary of Results

We can summarize the results for each program as follows:

- *Communication Channels (CO), Decorator*.
  - *Expectation*: Delocalization of functionality should make the PAT variant easier to modify, but more difficult to analyze and call.
  - *Result*: Using the Decorator pattern instead of a simpler solution is preferable during maintenance, as long as the developer has at least some prior understanding of the pattern. Given even minimal pattern knowledge, the PAT variant is easier to modify; given sufficient knowledge, code comprehension is not negatively affected.
- *Graphics Library (GR), Abstract Factory*.
  - *Expectation*: Architectural similarities between PAT and ALT should cause only minor differences in the results; where differences occur, ALT should outperform PAT due to the comprehensibility of its more localized structure.
  - *Result*: Due to several unexplained divergences between the three studies, coupled with the incidence of insignificant, but large effect estimates, the GR results are tentative. Nevertheless, the results suggest that a simplified solution is often equivalent to or better than using Abstract Factory—though given sufficient developer experience (4+ years), the reverse may be true.

Overall, using a pattern where a simpler solution would be possible can be advantageous during maintenance, but only if the developer performing the maintenance has a sufficient understanding of the pattern (and/or a sufficient level of developer experience); also, the critical level of knowledge (or experience) required for the pattern to be helpful appears to be higher for Abstract Factory than for Decorator (Fig. 3 depicts the general form of these conclusions). Thus, as Vokáč *et al.* state, "each design pattern. . . has its own nature, so that it is not valid to characterize patterns as useful or harmful in general" [20, p. 191].

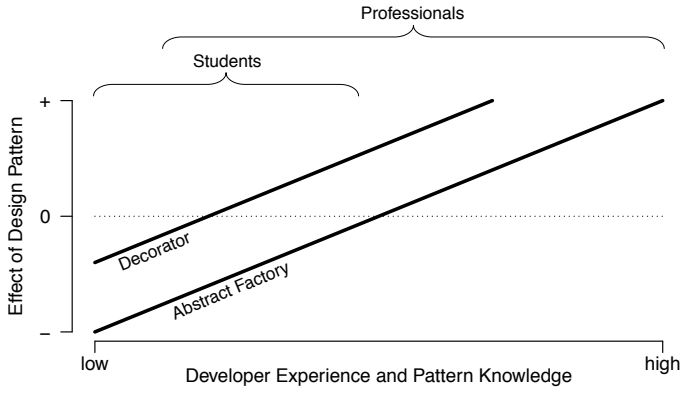


Fig. 3. Relative impact that developer experience and pattern knowledge have on the effect of Decorator and Abstract Factory, generalized across all three PatMain studies (E_orig, E_repl, and E_joint). A positive effect for a design pattern (+) means that the pattern leads to lower work times and higher quality solutions.

4 Threats to Validity

In this section, we discuss threats to validity. We follow a standard validity framework consisting of four parts [37], [38]:

- *Construct Validity*: The extent to which the protocol, treatment operationalizations, and metrics accurately represent the concepts under study.
- *Conclusion Validity*: The extent to which we can infer relationships in the data, particularly considering the statistical methods used.
- *Internal Validity*: The extent to which we know that the treatment caused the observed changes.
- *External Validity*: The extent to which the results can be generalized to other situations and people.

Due to space constraints, minor threats to validity are presented in Appendix W. *Note that internal validity only appears in Appendix W.*

4.1 Construct Validity

To operationalize *work environment*, we used a web portal, which not only allowed us to asynchronously administer the experiment, but also allowed the participants to:

1) work in their own environments, rather than in an unfamiliar experiment room with unfamiliar tools;
2) be externally interrupted;
3) potentially cooperate;
4) create time-recording ambiguities by using the browser back-button.

The first three factors (1–3) each increased the realism of the *work environment*. However, factors 2–4 introduced the potential for measurement error in the response variables (particularly *time*). Concerning interruptions, few participants reported any problems. For those that did, their task timings were adjusted accordingly. Concerning cooperation, we conducted an extensive investigation of the participants' solutions, but found no evidence of sharing—the only two dubious similarities between participants eventually turned out to have a convincing technical explanation. As for the browser back button, the final comments report almost no problems, but we nevertheless aggregated the timings for the download, work, and upload pages to accommodate atypical work orderings. Thus, although the three factors may have increased variability, we do not expect them to have significantly affected measurement of the response variables, or to have otherwise systematically biased the results.

In assessing *developer experience*, we collected multiple experience metrics from which we ultimately created a single aggregate metric, *devExp* (as described in Section 2.2). Unfortunately, one of the individual metrics (years professional experience) appeared to have been variously interpreted with respect to whether part-time work counts as "professional experience." Given that the participants were students, with little experience in general, this ambiguity could have significantly affected the results within E_joint (for further details, see Appendix I). Consequently, we used the professional experience metric only in the form of summary statistics to compare E_joint with E_orig and E_repl (which comparisons do not pose a concern). In other words, we excluded professional experience entirely from the *devExp* metric. Nevertheless, *devExp* is still a reasonable metric for the students of E_joint given the other component metrics involved (e.g., lines of code, programming hours, etc.).

Since we had no idea beforehand that *motivation* might impact cross-site variance, we did not formally operationalize or measure it as part of the experiment. Consequently, our analysis of *motivation* is based on *ad hoc*, largely qualitative data. As such, our conclusions about *motivation* should not be accepted as established fact, but rather should be considered as justification for future work.

Lastly, note that *correctness* is measured on a five-point ordinal scale following the same rubric originally used by E_repl (see Section 2.2 for details). However, we inevitably treat it as an interval scale (as did E_repl) by, in our case, mapping it onto percentages for statistical analysis (0, 25, 50, 75, 100%). As Vokáč et al. explain with respect to E_repl, "[I]t was impossible to estimate [a logistic regression] model by GEE or GLM, because the methods break down when all observations for certain combinations of the explanatory variables have the same value" [20, p. 160]. In our case, we used mixed models, for which the problem identified by Vokáč et al. is not a concern. Nevertheless, being a close replication, we preferred to follow E_repl's operationalization of *correctness* as closely as possible so that the two sets of results would be easily comparable. Statistically speaking, given 1) the nature of our models, 2) the fact that *correctness* is ordinal (as opposed to categorical), and 3) the fact that our final conclusions are based on trends in the data rather than on magnitudes, inferring the intervals is not a significant concern.

4.2 Conclusion Validity

The moderator analysis is a *post-hoc* analysis. *Post-hoc* analyses are useful for exploring experiment instability. However, they inflate the chances of a type 1 error, the chances of incorrectly concluding an effect exists. Thus their findings must be tested in future studies. Analyzing moderators also requires modeling large interactions, which reduces statistical power. To mitigate this problem, we used Bayesian methods, which allow us to directly compare probabilities for competing hypotheses to determine which are most likely.

Concerning the results in Table 7, data dredging (i.e., fishing for significance) is not a problem; those results rely on pre-planned statistical models to address *a priori* hypotheses. Instead, the primary concerns for Table 7 are extraneous variance within studies and heterogeneity of results across studies—which problems the moderator analysis was designed to help resolve. However, inasmuch as the final conclusions from Table 7, presented in Section 3.3, rely on the moderator analysis, they are also tentative.

For the Bayesian analysis, we enlisted a qualified external researcher to estimate prior distributions. We instructed him to choose broad priors in order to minimize the weight of those priors on the final results. In general, choosing broad priors leads to broader posteriors, but for a *post-hoc* analysis, sacrificing some precision is an acceptable tradeoff—i.e., a bias toward type 2 errors is appropriate given that the *post-hoc* nature of the analysis inflates the chances of a type 1 error.

In addition to choosing broad priors, we also could have asked more than one external researcher to independently select priors. Given multiple estimates for priors, we could have performed a sensitivity analysis to verify our expectation that the priors have little influence on the results. Not having done this, we recognize it here as a limitation and recommend it for future studies conducting similar analyses.

4.3 External Validity

Our conclusions are limited to the patterns tested in the CO/GR programs (Decorator and Abstract Factory), maintenance activities, and programs for which the full functionality of the patterns was not initially needed. The maintainers were also not the original designers/implementers, and real programs are typically larger and less well commented than the PatMain programs.

E_joint's participants also had little developer experience and pattern knowledge. However, being based on all three PatMain studies, the final conclusions represent a fairly broad range of developers.

Additionally, for both E_repl and E_joint, the replicating researchers interacted considerably with prior experimenters, which means the likelihood of shared bias is high among all three studies. For a description of the cross-study interactions, see Appendix V. Shared bias is not necessarily a bad thing. It helps in the early stages of investigation in order to reduce unexpected variance across experiments. However, it does indicate a weakness in external validity, which can only be addressed by eventually replicating the study with little or no cross-study interaction—other than the use of published reports and (possibly) lab packages.

Ultimately, given that we have successfully generalized the results across all three PatMain studies—which collectively represent 58 students and 68 professionals from 17 institutions and 4+ countries—the most significant threats to external validity are not population related; rather, they concern the size and complexity of the software programs being tested, as well as the fact that the developers worked in isolation, rather than in team environments.

As discussed in Appendix S, we found evidence in E_joint that program complexity could impact the outcome of the PatMain experiment. Although inconclusive, the data suggest that if the difficulty of a problem exceeds a certain threshold (relative to a developer's experience and/or pattern knowledge) then design patterns will have little impact on work time and solution correctness. However, we do not know whether this observation would hold in industry, much less whether it would be experimentally repeatable. Quite possibly, the reverse is true—that the real value of design patterns is only manifest for big software of the magnitude found in industry. To answer these questions, additional studies are needed explicitly targeting the issue of program complexity.

Concerning the issue of programming environments, it is possible that isolated developers respond differently to design patterns than developers working in a team. For instance, via the sharing of knowledge, team dynamics may compensate for negative effects of design patterns. On the other hand, team dynamics may also attenuate design pattern benefits, thus causing patterns to have less impact overall. Additional studies are needed.

5 Conclusions

In this section, we present conclusions. We discuss moderators, design patterns, and finally, future work.

5.1 Moderators

We find that both *developer experience* and *pattern knowledge* moderate the effect of design patterns, such that a higher level of either tends to enhance the benefits of patterns (or reduce their harm) during maintenance. We also find indirect evidence for *motivation* as a moderator. At the very least, we can tentatively conclude that lack of motivation increases statistical variance, which in turn can confound the comparison of results across studies. Whether and to what degree motivation impacts the effect of patterns outside the experimental setting is still unclear.

5.2 Design Patterns

Based on the two moderators for which we have quantitative data—developer experience and pattern knowledge—we were able to fully resolve conflicts for one of the two patterns studied (Decorator) and partially for the other (Abstract Factory). Each of the final conclusions (summarized below) generalizes across all three PatMain studies (E_orig, E_repl, and E_joint), involving 126 participants from five universities and twelve software companies, spanning two continents and at least four countries—thus covering a broader set of contexts than has previously been achieved in the study of design patterns. Such a high level of generalization would not have been possible without the moderator analysis.

1) The Decorator pattern is preferable to a simpler solution, as long as the developer has at least some prior knowledge of the pattern.
2) For Abstract Factory, the simpler solution is mostly equivalent to the pattern solution.
3) Abstract Factory requires a higher level of pattern knowledge and/or developer experience than Decorator for the pattern to be beneficial.

In general, using a pattern where a simpler solution would be possible can be advantageous during maintenance, but only

if the developer performing the maintenance has a sufficient understanding of the pattern. See Fig. 3, above, for a pictorial representation of these conclusions.

5.3 Future Work

An obvious item for future work is to further replicate the PatMain experiment with additional controls and measurements for moderators. In that case, we recommend focusing on developer experience, pattern knowledge, and motivation. However, we also recommend documenting other factors which may be influencing the outcome of the experiment. These additional factors can be cross-referenced against the list we provide in Section 3.2. Further, it may be valuable to explore relationships between moderators. For instance, it would be useful to know whether developer experience can compensate for a lack of pattern knowledge (or vice versa).

Due to the fact that measurement operationalizations tend to vary across experiments, controlling variables within studies will not, by itself, solve the problem of generalizability. To fully solve that problem, we also need to develop methods for mapping moderators across studies. To accomplish this, we recommend investigating best methods for assessing common context variables (e.g., developer experience), and then formulating standardized assessments for those variables.

In addition to further replicating the PatMain experiment, it would be interesting to review the design pattern literature for data on potential moderators. Many studies likely contain at least some traces of information on moderators, the synthesis of which may reveal useful insights. The results of such a literature review could be used to corroborate (or even generalize and extend) the findings reported in this paper.

For additional discussion of future work, see Appendix X.

Acknowledgments

We are grateful to: Martin Liesenberg for building the web portal; Ulrich Stärk for expert advice to Martin; Martin Liesenberg and Christian Bird for the Java and C# program translations, respectively; Alexander MacLean and Landon Pratt for grading all solutions to ensure consistency in the scoring across sites; Paul Felt for sharing source code and advice concerning his Bayesian models; Gilbert Fellingham for expert advice on the Bayesian analysis; Marek Vokáč for providing data and historical information about the first replication of the PatMain experiment, as well as for member checking our analysis and reporting of his experiment; RESER 2011 reviewers and participants for encouragement and valuable feedback; and all study participants for their contributions. This research has been partially supported by the grant TIN-2011-23216 (Spanish Ministry of Economy and Competitiveness), as well as by funding from Ironwood Experts, LLC.

References

[1] K. Beck, J. O. Coplien, R. Crocker, L. Dominick, G. Meszaros, F. Paulisch, and J. M. Vlissides, "Industrial experience with design patterns," in *Proc. Int'l Conf. Softw. Eng.*, 1996, pp. 103–114.

[2] F. J. Budinsky, M. A. Finnie, J. M. Vlissides, and P. S. Yu, "Automatic code generation from design patterns," *IBM Syst. J.*, vol. 35, no. 2, pp. 151–171, 1996.

[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*. Chichester, UK: John Wiley & Sons, 1996.

[4] G. Florijn, M. Meijers, and P. van Winsen, "Tool support for object-oriented patterns," in *Proc. European Conf. Object-Oriented Programming*, 1997, pp. 472–495.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[6] L. Aversano, L. Cerulo, and M. D. Penta, "Relationship between design patterns defects and crosscutting concern scattering degree: An empirical study," *IET Softw.*, vol. 3, no. 5, pp. 395–409, 2009.

[7] J. Garzás, F. García, and M. Piattini, "Do rules and patterns affect design maintainability?" *J. Computer Sci. Technol.*, vol. 24, no. 2, pp. 262–272, 2009.

[8] S. Jeanmart, Y.-G. Guéhéneuc, H. Sahraoui, and N. Habra, "Impact of the visitor pattern on program comprehension and maintenance," in *Proc. Int'l Symp. Empir. Softw. Eng. and Measurement*, 2009, pp. 69–78.

[9] T. H. Ng, S. C. Cheung, W. K. Chan, and Y. T. Yu, "Work experience versus refactoring to design patterns: A controlled experiment," in *Proc. ACM SIGSOFT Symp. Foundations of Softw. Eng.*, 2006, pp. 12–22.

[10] L. Prechelt, B. Unger, W. F. Tichy, P. Brössler, and L. G. Votta, "A controlled experiment in maintenance comparing design patterns to simpler solutions," *IEEE Trans. Softw. Eng.*, vol. 27, no. 12, pp. 1134–1144, 2001.

[11] P. Wendorff, "Assessment of design patterns during software reengineering: Lessons learned from a large commercial project," in *Proc. European Conf. Softw. Maintenance and Reeng.*, 2001, pp. 77–84.

[12] C. Zhang and D. Budgen, "What do we know about the effectiveness of software design patterns?" *IEEE Trans. Softw. Eng.*, vol. 38, no. 5, pp. 1213–1231, 2012.

[13] A. Ampatzoglou, S. Charalampidou, and I. Stamelos, "Research state of the art on GoF design patterns: A mapping study," *J. Syst. Softw.*, vol. 86, no. 7, pp. 1945–1964, 2013.

[14] N. J. Salkind, Ed., *Encyclopedia of Measurement and Statistics*. Thousand Oaks, CA: Sage Publications, 2007.

[15] R. M. Baron and D. A. Kenny, "The moderator-mediator variable distinction in social psychological research: Conceptual, strategic, and statistical considerations," *J. Pers. Soc. Psychol.*, vol. 51, no. 6, pp. 1173–1182, 1986.

[16] P. Diesing, *How Does Social Science Work? Reflections on Practice*. Pittsburgh, PA: University of Pittsburgh, 1991.

[17] P. Solomon, M. M. Cavanaugh, and J. Draine, *Randomized Controlled Trials: Design and Implementation for Community-Based Psychosocial Interventions*. Oxford, UK: Oxford University, 2009.

[18] D. I. K. Sjøberg, T. Dybå, and M. Jørgensen, "The future of empirical methods in software engineering research," in *Future Softw. Eng.*, 2007, pp. 358–378.

[19] L. Prechelt, B. Unger-Lamprecht, M. Philippsen, and W. F. Tichy, "Two controlled experiments assessing the usefulness of design pattern documentations in program maintenance," *IEEE Trans. Softw. Eng.*, vol. 28, no. 6, pp. 595–606, 2002.

[20] M. Vokáč, W. F. Tichy, D. I. K. Sjøberg, E. Arisholm, and M. Aldrin, "A controlled experiment comparing the maintainability of programs designed with and without design patterns: A replication in a real programming environment," *Empir. Softw. Eng.*, vol. 9, no. 3, pp. 149–195, 2004.

[21] J. L. Krein, C. D. Knutson, L. Prechelt, and N. Juristo, "Report from the 2nd international workshop on replication in empirical software engineering research (RESER 2011)," *ACM SIGSOFT Softw. Eng. Notes*, vol. 37, no. 1, pp. 27–30, 2012.

[22] RESER Workshop, "2nd International Workshop on Replication in Empirical Software Engineering Research," http://sequoia.cs.byu.edu/reser2011, 2011, last accessed: April 2014.

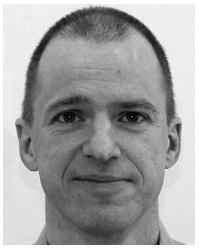
[23] Freie Universität Berlin, "Replication of 'PatMain'," http://www.inf.fu-berlin.de/w/SE/PatmainReplicationInfo, 2011, last accessed: April 2014.

[24] N. Juristo and S. Vegas, "Design patterns in software maintenance: An experiment replication at UPM," in *Proc. Int'l Workshop Replication Empir. Softw. Eng. Research*, 2011, pp. 7–14.

[25] J. L. Krein, L. J. Pratt, A. B. Swenson, A. C. MacLean, C. D. Knutson, and D. L. Eggett, "Design patterns in software maintenance: An experiment replication at Brigham Young University," in *Proc. Int'l Workshop Replication Empir. Softw. Eng. Research*, 2011, pp. 25–34.

[26] A. Nanthaamornphong and J. C. Carver, "Design patterns in software maintenance: An experiment replication at University of Alabama," in *Proc. Int'l Workshop Replication Empir. Softw. Eng. Research*, 2011, pp. 15–24.
[27] L. Prechelt and M. Liesenberg, "Design patterns in software maintenance: An experiment replication at Freie Universität Berlin," in *Proc. Int'l Workshop Replication Empir. Softw. Eng. Research*, 2011, pp. 1–6.
[28] J. L. Krein, L. Prechelt, N. Juristo, K. D. Seppi, A. Nanthaamornphong, J. C. Carver, S. Vegas, and C. D. Knutson, "A method for generalizing across contexts in software engineering experiments," *IEEE Trans. Softw. Eng.*, 2015, under review.
[29] L. Prechelt, "PatmainPackage," http://page.mi.fu-berlin.de/prechelt/Biblio/#package, 2013, last accessed: April 2014.
[30] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. Chichester, UK: John Wiley & Sons, 2000.
[31] F. L. Ramsey and D. W. Schafer, *The Statistical Sleuth: A Course in Methods of Data Analysis*, 2nd ed. Pacific Grove, CA: Duxbury, 2002.
[32] J. Surowiecki, *The Wisdom of Crowds*. New York, NY: Anchor Books, 2005.
[33] M. H. DeGroot and M. J. Schervish, *Probability and Statistics*, 4th ed. Boston, MA: Addison-Wesley, 2012.
[34] P. Felt, "Improving the effectiveness of machine-assisted annotation," Master's thesis, Dept. of Computer Science, Brigham Young University, Provo, UT, 2012.
[35] R. A. McLean, W. L. Sanders, and W. W. Stroup, "A unified approach to mixed linear models," *Am. Stat.*, vol. 45, no. 1, pp. 54–64, 1991.
[36] K. Charmaz, *Constructing Grounded Theory: A Practical Guide through Qualitative Analysis*, 1st ed. London, UK: Sage Publications, 2006.
[37] T. D. Cook and D. T. Campbell, *Quasi-Experimentation: Design & Analysis Issues for Field Settings*. Boston, MA: Houghton Mifflin, 1979.
[38] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. New York, NY: Springer, 2012.
[39] C. F. Dormann, J. Elith, S. Bacher, C. Buchmann, G. Carl, G. Carré, J. R. G. Marquéz, B. Gruber, B. Lafourcade, P. J. Leitão, T. Münkemüller, C. McClean, P. E. Osborne, B. Reineking, B. Schröder, A. K. Skidmore, D. Zurell, and S. Lautenbach, "Collinearity: A review of methods to deal with it and a simulation study evaluating their performance," *Ecography*, vol. 36, no. 1, pp. 27–46, 2013.
[40] D. E. Farrar and R. R. Glauber, "Multicollinearity in regression analysis: The problem revisited," *Rev. Econ. Stat.*, vol. 49, no. 1, pp. 92–107, 1967.
[41] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*, ser. Monographs on Statistics and Applied Probability. London, UK: Chapman & Hall/CRC, 1993.
[42] J. C. Carver, "Towards reporting guidelines for experimental replications: A proposal," in *Proc. Int'l Workshop Replication Empir. Softw. Eng. Research*, 2010, pp. 1–4.

Jonathan L. Krein is a Partner at Ironwood Experts, LLC and an Adjunct Professor of Computer Science at Brigham Young University (BYU). He also works as an Adjunct Researcher in the BYU SEQuOIA ("Software Engineering Quality: Observation, Insight, Analysis") Lab. He holds Ph.D., M.S., and B.S. degrees in computer science from BYU and served as co-founder and co-organizer of the International Workshop on Replication in Empirical Software Engineering Research (RESER 2010, 2011, 2013). His research interests include knowledge production in empirical software engineering research (replication, theory building, and the development of transferable best practices), Bayesian methods in Computer Science, the management of private information in software organizations, and software repositories as empirical data sources.



Lutz Prechelt received a PhD from the University of Karlsruhe for work that combined machine learning and compiler construction for parallel machines. He then moved to empirical software engineering and performed a number of controlled experiments before spending three years in management in the software industry. He is now full professor for software engineering at Freie Universität Berlin. His research interests still revolve primarily around the human factor in the software development process but his research methods are now more often qualitative than quantitative. Additional research interests concern research methods and the health of the research system.

Natalia Juristo received the PhD degree from the Technical University of Madrid in 1991. She is currently a professor of software engineering at Universidad Politecnica de Madrid. Natalia was the Director of the UPM MSc in Software Engineering from 1992 to 2002 and the coordinator of the Erasmus Mundus European Master on SE (with the participation of the University of Bolzano, the University of Kaiserslautern, and the University of Blekinge) from 2006 to 2012. Her main research interests are experimental software engineering, requirements, and testing. Natalia is coauthor of the book, *Basics of Software Engineering Experimentation* (Kluwer, 2011). She is a member of the editorial board of Empirical SE Journal. She was recently awarded a FiDiPro (Finland Distinguish Professor) research grant. She began her career as a developer in the European Space Agency (Rome) and the European Center for Nuclear Research (Geneva). She was a resident affiliate at the Software Engineering Institute in Pittsburgh in 1992.

Aziz Nanthaamornphong is a PhD student in the Computer Science Department at the University of Alabama. His primary research interests include empirical software engineering, software quality, and software engineering for computational science and engineering. He earned his MS in information technology from the Kasetsart University, Thailand. He is a student member of IEEE and the ACM. Contact him at anan-thaamornphong@ua.edu.

Jeffrey C. Carver earned his PhD degree in Computer Science from the University of Maryland. He is an Associate Professor in the Department of Computer Science at the University of Alabama. His main research interests include empirical software engineering, software quality, software engineering for computational science and engineering, software architecture, human factors in software engineering and software process improvement. He is a Senior Member of the IEEE Computer Society and a Senior Member of the ACM. Contact him at carver@cs.ua.edu.

Sira Vegas is an associate professor of software engineering with the Computing School at the Universidad Politcnica de Madrid in Spain. She received the BS and PhD degrees in Computing from the Universidad Politcnica de Madrid, Spain. Sira was visiting scholar of the European Centre for Nuclear Research (Geneva) in 1995. She was a regular visiting scholar of the Experimental Software Engineering Group at the University of Maryland from 1998 to 2000, and visiting scientist at the Fraunhofer Institute of Experimental Software Engineering in Germany in 2002. Sira has served in several Program Committees ESEM/ISESE, SEKE, CSEET and others. She has been Program Chair for ESEM07. She is/has been regular reviewer of several journals, including IEEE Transactions on Software Engineering, IEEE Software, Empirical Software Engineering Journal, and Information and System Technology (http://grise.upm.es/miembros/sira/).

Charles D. Knutson is an Emeritus Professor of Computer Science at Brigham Young University (BYU) and Managing Partner at Ironwood Experts, LLC. He is the Director of the BYU SEQuOIA ("Software Engineering Quality: Observation, Insight, Analysis") Lab and the former Director of the BYU Mobile Computing Lab. Dr. Knutson has 29 years of industry experience, including engineering and management positions at Hewlett-Packard and Novell, Inc. He was also Vice President of R&D at Counterpoint Systems Foundry, Inc. (now Sybase iAnywhere), the world's leading provider of IrDA and Bluetooth protocol stacks for embedded systems. Dr. Knutson has more than 120 technical publications in areas including mobile computing, medical informatics, and software engineering. He is a co-founder and organizer of the Workshop on Research in Empirical Software Engineering Research (RESER). He holds a PhD in Computer Science from Oregon State University, and BS and MS degrees in Computer Science from BYU.

Kevin D. Seppi is an Associate Professor at Brigham Young University. He received his BS in Computer Science from Brigham Young University in 1982 and his MS from Santa Clara University in 1987. He received his PhD from the University of Texas in 1990 on a resident study fellowship from IBM. Following his PhD he returned to IBM where he worked until 1995, at which time he joined BMC Software. In 2002, Dr. Seppi joined the Computer Science faculty at Brigham Young University, where he currently serves as director of the Applied Machine Learning Lab. His research focuses on probabilistic models and their applications, as well as on optimization.

Dennis L. Eggett is an Associate Research Professor of Statistics and the Director of the Center for Statistical Consultation and Collaborative Research at Brigham Young University (BYU). He received his B.S and MS in statistics from BYU and PhD in statistics from North Carolina State University. From 1987–1997 he was a research scientist and statistical consultant at Pacific Northwest National Laboratory (PNNL) in Richland, Washington. PNNL is a contract laboratory of the Department of Energy operated by Battelle Memorial Institute. While at PNNL, Dr. Eggett worked in areas related to waste cleanup, waste characterization, remote sensing, and the analysis of large data sets. Dr. Eggett joined the faculty of BYU in August of 1997. During that time he has served as Director of the Center. This includes consulting with on campus and off campus clients, supervising student consultants, and teaching statistical methods. He has co-authored over 140 journal papers. He has worked with researchers from many different fields ranging from nursing to engineering. His expertise include linear models, multivariate methods, experimental design, mixed models analysis, and resampling methods.

This document contains appendices for the paper, *A Multi-Site Joint Replication of a Design Patterns Experiment using Moderator Variables to Generalize across Contexts*, by Krein *et al.*, 2015.

Since *replicability* and *generalizability* are key concerns of the study, we provide a considerable amount of supplemental information in the appendices. The appendices are provided to assist future replicators and/or to facilitate transparency; none are necessary in order to understand the conclusions and general validity of the study.

The appendices are meant to be used, as needed, like a reference manual. To make the information more accessible, we cite each appendix in the main paper where it is most relevant. This document is numbered as a continuation of the main paper.

APPENDIX A
REPLICATION WEB PORTAL

The replication web portal embodies the materials, instructions, and data collection infrastructure for the PatMain experiment. The web portal allows the researcher to do the following actions:

1) Register and create an instance of the experiment.
2) Create batches of participant IDs.
3) Print little chits providing the experiment URL and login ID for each participant.
4) Monitor experiment progress.
5) Close the experiment and download the results.

The web portal supports administering the experiment in full with all four programs, or administering an abridged version with only two of the four programs (either the pair CO/GR or the pair ST/BO).

Participation via the portal involves answering questionnaires, downloading programs, and uploading solutions. The portal guides the participant through the following steps:

1) Log in to the application using an assigned ID.
2) Choose a language: C++, C#, or Java.
3) Complete two short questionnaires assessing development experience (10 questions) and design pattern knowledge (19 questions).
4) Perform maintenance tasks on two programs (one PAT and one ALT variant), including two tasks for each program (one coding and one comprehension). Coding tasks require the download and upload of source code. Comprehension tasks consist of 1–2 short answer questions.
5) At the end of each program, complete a performance self-evaluation (6 questions per program).
6) Submit final comments—in particular, listing any interruptions.

The total time required for the abridged experiment is approximately 2–3 hours. The portal records the participants' answers, uploads, and timings at each step. By assigning IDs in contiguous sequence, participants are evenly bucketed into the experiment groups. All participants see the same instruction text for a given program regardless of the assigned treatment group.

The portal's source code (written by Martin Liesenberg) is provided in the lab package as a ZIP file (jointrep_experiment_webapplication.zip, packaged by Lutz Prechelt). The ZIP file contains a README.txt and the directory layout of a Java EE WAR file (for Tomcat and MySQL). Portal screenshots and the original experiment programs are also provided in the lab package.

APPENDIX B
INFORMATION ON SITES

Table 8 provides an overview of the experiment execution and participants at each of the four sites. Note that the three-day experiment timeframe mentioned for UA includes only the official PatMain experiment protocols. All additional UA protocols (described below), were administered in a separate one-week period.

B.1 Additional Protocols

The UA replication added protocols beyond those defined by PatMain. All additions were administered to participants as a separate "pre-experiment," which the participants completed prior to the PatMain experiment. Although the two experiments were administered at different times, they involved the same participants and programs. Consequently, UA's results could systematically differ from those of BYU, FUB, and UPM.

The pre-experiment required participants to view UML diagrams of the programs (not program source code) and to answer questions. Thus UA participants likely performed better on the PatMain tasks—i.e. lower times and higher correctness scores—than they otherwise would have. However, the main effect (program *variant*) most likely remains intact. In general, previewing UML diagrams would tend to raise the overall level of program comprehension. Thus if any impact did occur, the most likely effect would be to reduce (but not reverse) the experiment effect among UA participants. Accordingly, we believe the UA data can be used in the joint analysis. See the E_joint lab package for copies of UA's pre-experiment artifacts.

Additionally, note that UA tested its participants on all four programs (CO, GR, ST, and BO), rather than on just CO and GR. However, UA issued two web portal IDs to each participant, one for the CO/GR programs and one for the ST/BO programs (rather than running all four programs in the same session). Thus, UA's experiment groups for the CO/GR programs are consistent with those of BYU, FUB, and UPM.

B.2 Design Pattern Education

At BYU, design patterns are taught by two professors (neither of which is affiliated with this study). The first professor teaches classic GOF patterns via the Gamma *et al.* book [5] and via a modestly-sized programming project (approx. 3–5 KLOC). He does not explicitly teach patterns in a generalizable way, but when asked, commented, "it's probably some of both," meaning general and specific (interview, Oct. 30, 2012). The second BYU professor also teaches GOF patterns via a modestly-sized programming project (approx. 3–5 KLOC), but does not teach patterns from the Gamma *et al.* book [5]. Instead, he uses in-class programming examples. Via these examples he first shows "bad ways of solving a problem," after which he then interactively helps the students to improve those bad solutions by incorporating design patterns (interview, Oct. 30, 2012).

In general, the teaching style at BYU stresses applying design patterns within specific implementations. Of this, the latter professor commented, "On average, I do not think most of the students could generalize the patterns well, given their limited experience in the course" (interview, Oct. 30, 2012). Nearly all of the BYU participants should have previously received training from one of these two professors. Thus both types of education are represented among BYU participants.

At FUB, all participants were recruited from a project course involving Eclipse plugin programming. In that context, they each definitely had practical contact with the Observer pattern, as well as possibly Composite, Strategy, and others. Further, most of the FUB participants had

TABLE 8
General information about the experiment and participants at each of the four sites.

	BYU	FUB	UA	UPM	All
Experiment Timeframe	3 weeks	approx. 1 week	3 days	4 days	-
Experiment Execution	asynchronous*	asynchronous*	asynchronous*	asynchronous*	-
Programming Language	Java	Java	Java	Java	-
Language Required	yes	no†	yes	no†	-
Participant Type	students	students	students	students	-
Solicitation Venue	software engineering course	software project course	software engineering course	research group‡	-
Participation	voluntary	voluntary	assignment	voluntary	-
Total Participants	22	13	18	8	61
Undergraduate Students	20	9	0	0	29
Graduate Students	2	4	18	8	32
Data Discarded as Invalid	1 undergrad	1 undergrad	4 grad	2 grad	8
Socioeconomic Background	mostly middle-class	broad	unknown	unknown	-
Nationality	mostly North American	mostly German	10 American 8 other	all South American	-

BYU = Brigham Young University
FUB = Freie Universität Berlin
UA = The University of Alabama
UPM = Universidad Politécnica de Madrid

*Participants took the experiment at a time of their own choosing.
†The participants were allowed to select their preferred language (C++, C#, or Java), but all chose Java.
‡Software engineering research group of the UPM joint replicators.

previously taken Lutz Prechelt's basic software engineering course. That course dedicates two 90-minute lectures to design patterns—specifically, composite, adapter, bridge, facade, observer, strategy, abstract factory, and builder. Later in the course, two further 90-minute lectures on software reuse cover other types of patterns—including, analysis and usability patterns (for requirements), usability architecture patterns (for design), software process patterns, and anti-patterns. Although the design patterns course does not require students to implement patterns, all of the FUB participants had practical experience with patterns in the aforementioned project course.

In general, Lutz explains, "I teach patterns as a form of packaged, reusable, and reconfigurable design ideas where the specific form of their implementation is definitely not a key element of the pattern as such. Ideally, my students should come out with a rather flexible, adaptable idea of what the use of a specific pattern looks like" (email, Oct. 9, 2012).

UA participants were all recruited from Jeffrey Carver's software engineering course. Jeff reports, "In our course, the students learn design patterns from a book. We do not do any special exercises on patterns. The students in the course each prepare a lecture on one or more patterns to teach to the class. I add information that they miss" (email, Oct. 30, 2012).

UPM participants were all graduate students, recruited from the software engineering research group. According to Natalia Juristo, "We did not teach the students about patterns. Our experimental participants were master and PhD students who said they knew design patterns, which does not mean they really do. None of the students completed their undergraduate degrees at UPM. They studied computing in other Spanish or Latin-American universities. So I do not know how many previously received training on patterns. I tend to think they have learned patterns as part of their work, but cannot be sure" (email, Oct. 30, 2012).

APPENDIX C
PARTICIPANT DEMOGRAPHICS PER SITE

Table 8 provides general information about the participants at each site. Tables 9 and 10 summarize the participants' self-reported developer experience and pattern knowledge. For each variable in Tables 9 and 10, we statistically compare sites using the Kruskal-Wallis rank sum test (which procedure compares medians, rather than means). We use a nonparametric (distribution-free) test because, in many cases, the data are not normally distributed. Histograms of the data are provided in the lab package.

C.1 Developer Experience (Table 9)

The most noticeable differences across sites (in terms of means) involve the lines of code (LOC) and professional experience questions. Statistically, the sites also differ in terms of the languages-used questions. We discuss each in turn.

Concerning LOC questions (i.e., LOC-lifetime and LOC-Java), four of the American participants (3 at BYU, 1 at UA) report unrealistically high values.[11] For example, one at BYU reports 8 million lifetime lines of code. Conversely, none of the participants at the European universities (FUB and UPM) list such extreme values. Once the unrealistic outliers are

11. For students reporting at most 5 years professional experience, we consider responses of 150 KLOC or more to be unreasonable. Some of the extreme outliers may be due to typing errors, but some respondents may have exaggerated, possibly due to disinterest with the questions. All other data for these participants appear reasonable, so we do not exclude their data from analysis.

dropped, the four sites no longer statistically differ on the LOC questions (updated p-value for LOC-Java = 0.053).

As for professional experience (i.e., working-hours-per-week and years-professional-experience), two of the UPM participants report outlier values (10 and 15 years experience, working 40 hours per week). Since UPM only contributes 6 participants, its median responses are much higher than those of the other three sites. Statistically, UPM differs from the other sites for years-professional-experience, but not for working-hours-per-week. However, with the two outliers excluded, the p-value for years-professional-experience becomes 0.203, indicating that the UPM outliers differ from all other participants, even those at UPM.

Thus, outliers account for most of the statistically significant differences in developer experience that we observe across sites. Excluding unrealistic outliers, only two significant cross-site differences remain: 1) Two of the UPM participants report significantly more professional experience than any other participant in the study, and 2) BYU participants report using more programming languages than participants at the other three sites (see languages-used-lifetime and languages-used-often). The BYU participants' advantage in language experience may be a product of their university's teaching style, or it may reflect a cultural tendency to exaggerate on survey questions. Coincidentally, BYU participants are also the primary contributor of unrealistic responses to the LOC questions.

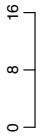
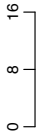
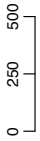
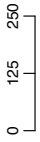
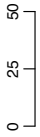
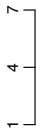
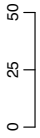
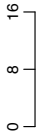
C.2 Pattern Knowledge (Table 10)

Statistically, the four sites differ on 14 of the 18 patterns surveyed. However, the patterns-used-lifetime variable, which describes *actual pattern use*, is not statistically significant. Additionally, only four of the statistically significant patterns involve a median exceeding 4.0 (i.e., "understand it well"), and the largest median among all patterns is only 5.5 (i.e., between "understand it well and have worked with it once" versus "understand it well and have worked with it two or three times"). Thus, cross-site differences in pattern knowledge almost exclusively concern exposure to pattern concepts rather than experience implementing patterns.

Many of the individual patterns display similar box plot arrangements—for instance, Command, Composite, and Decorator. With a little effort, we can identify several potentially meaningful arrangements. However, since we measured 17 patterns, these groupings may simply be due to random chance. To test the possibility, we count the number of unique arrangements. Ordering sites by median, allowing means to break ties, we find 10 unique arrangements among the 17 patterns. If the pattern variables share dependencies, we would expect fewer unique arrangements than predicted by random chance. With 24 possible permutations, the probability of obtaining 10 or fewer by random chance is 0.085. Thus we cannot reject the null hypothesis that the observed groupings are due to random chance, though the p-value is suggestive.

We also examine the order of sites overall, to see which ones tend to report higher pattern knowledge. We assign points to each site based on its rank for each of the 17 pattern knowledge variables—where rank 1 (i.e., highest pattern knowledge) is worth 3 points, rank 2 is worth 2 points, and so forth. As before, rank is determined by median, allowing means to break ties. For example, the rank order for the Command pattern is FUB, UA, BYU, UPM. Given this

TABLE 9
Summary statistics for the developer experience pre-questionnaire. For a description of each variable, identified by field number, see Appendix G. The count of participants for each column is given in parentheses. For each box plot, the median value is labeled, the mean is shown as a black dot, and the max whisker range is 1.5 IQR. All p-values are two-sided; p-values less than or equal to 0.05 are bolded; p-values are based on the Kruskal-Wallis rank sum test comparing sample medians across sites (calculated with R 2.15.2).

	BYU (21)	FUB (12)	UA (14)	UPM (6)	All (53)	*sig.*
Languages Used Lifetime* (field 7)	 10.00	6.50	7.00	8.00	8.00	**0.001**
Languages Used Often (field 9)	 4.00	2.00	3.00	2.50	3.00	**0.007**
Lines of Code Lifetime*† (KLOC) (field 10)	 50.00	25.00	35.00	85.00	35.00	0.329
Lines of Code Java‡ (KLOC) (field 11)	 12.50	5.75	4.50	7.00	9.00	**0.010**
Programming Hours Per Week (field 12)	 15.00	18.50	10.00	30.00	15.00	0.131
Programming Skill (scale 1–7)§ (field 13)	 3.00	2.00	4.00	3.50	3.00	0.261
Working Hours Per Week (field 15)	 0.00	10.00	0.00	25.00	0.00	0.054
Years Professional Experience (field 16)	 0.00	1.00	0.00	3.50	0.00	**0.029**

BYU = Brigham Young University
FUB = Freie Universität Berlin
UA = The University of Alabama
UPM = Universidad Politécnica de Madrid
sig. = significance (two-sided p-value)
* "Lifetime" means the number of languages (or LOC) the participant reports having *ever* used (or written).
† Two outliers not shown: BYU=8000 and UA=1000.
‡ One outlier not shown: BYU=4000.
§ 1=high skill, 7=low.

TABLE 10
Summary statistics for the design pattern knowledge pre-questionnaire. For a description of each variable, identified by field number, see Appendix G. The count of participants for each column is given in parentheses. For each box plot, the median value is labeled, the mean is shown as a black dot, and the max whisker range is 1.5 IQR. All p-values are two-sided; p-values less than or equal to 0.05 are bolded; p-values are based on the Kruskal-Wallis rank sum test comparing sample medians across sites (calculated with R 2.15.2).†

	BYU (21)	FUB (12)	UA (14)	UPM (6)	All (53)	*sig.*
Patterns Used Lifetime* (field 19)	8.00	6.50	5.00	4.00	6.00	0.077
Abstract Factory‡ (field 20)	3.00	3.00	3.00	2.50	3.00	0.454
Adapter (field 21)	3.00	3.50	4.00	2.00	3.00	**0.010**
Bridge (field 22)	1.00	2.00	3.50	2.00	3.00	**<0.001**
Chain of Responsibility (field 23)	2.00	2.00	4.00	2.50	3.00	**0.002**
Command (field 24)	4.00	5.00	4.00	2.50	4.00	**0.032**
Composite‡ (field 25)	3.00	5.00	4.00	2.00	4.00	**0.008**
Decorator‡ (field 26)	3.00	4.50	4.00	2.00	4.00	0.086
Factory Method (field 27)	4.00	2.50	3.00	2.50	3.00	**0.002**
Flyweight (field 28)	1.00	3.50	3.50	1.50	2.00	**<0.001**
Mediator (field 29)	1.00	2.00	4.00	2.00	2.00	**<0.001**
Memento (field 30)	1.00	2.00	4.00	2.00	2.00	**<0.001**
Observer (field 32)	5.00	5.00	4.00	3.50	5.00	0.104
Proxy (field 33)	2.00	5.50	4.00	2.50	4.00	**<0.001**
Reactor (field 34)	1.00	2.00	1.00	1.00	1.00	**<0.001**
Strategy (field 35)	2.00	3.00	4.00	2.00	3.00	**0.002**
Template Method (field 36)	2.00	2.00	3.50	2.00	3.00	**0.009**
Visitor (field 37)	5.00	3.00	4.00	1.50	4.00	**<0.001**

BYU = Brigham Young University
FUB = Freie Universität Berlin
UA = The University of Alabama
UPM = Universidad Politécnica de Madrid
sig. = significance (two-sided p-value)
*"Lifetime" means the number of patterns the participant reports having *ever* used.

†Individual patterns (fields 20–37) are self-assessed on an ordinal scale (1–7): 1=*never heard of it*, 2=*have only heard of it*, 3=*understand it roughly*, 4=*understand it well*, 5=*understand it well and have worked with it once*, 6=*understand it well and have worked with it two or three times*, 7=*understand it well and have worked with it many times*. Note that differentiation between levels 2, 3, and 4 is subjective and may be sensitive to cultural influence. Since these values are frequent in the responses, cross-site comparisons may be problematic.
‡The design pattern is included in E_joint.

scheme, UA scores 41% of the total possible points, followed by FUB with 32%, then BYU with 17%, and UPM with 10%. Thus for an overwhelming number of patterns, UA and FUB report greater pattern knowledge than BYU and UPM.

Incidentally, this effect explains why we find fewer unique permutations than expected.

APPENDIX D
DETAILED PROGRAM/TASK DESCRIPTIONS

The text in this section is quoted from E_orig's published report [10, pp. 1140–1142], with minor modifications to adapt it to the present context. The text is also similar to that given in E_repl's published report [20, pp. 173–176], which mostly paraphrases the original. The program metrics listed below (LOC and class counts) vary across the three studies. This is due to: 1) E_repl adjusted the programs from the original paper-based format to facilitate compilation, and 2) in E_joint, the programs were translated from C++ to Java. The metrics given in the text are those for E_orig; metrics for E_repl and E_joint are provided in footnotes.

The programs used by E_orig and E_repl included C++ header files, in which system components were declared. Since the system components were not technically part of the programs, the header files were excluded from the program metrics. Some testing code was also provided for the GR program, but that code was not provided in the form of a class. However, for E_joint—which administered the experiment in Java—both the header files and the testing code had to be translated into classes. To make E_joint's metrics comparable to those of the prior studies, we ignore the system classes entirely and consider the GR testing class only for LOC counts (not for class counts).

D.1 Decorator: Communication Channels (CO)

Communication Channels is a wrapper library. A communication channel establishes a connection for transparently transferring arbitrary-length packets of data. One can turn on additional logging, data compression, and encryption functionality. The library does not implement the functionality itself, but only provides a facade to a system library. However, this application of the Facade pattern is irrelevant to the experiment.

The PAT variant, which comprises 365 LOC in six C++ classes,[12] is designed with a Decorator for adding functionality to a bare channel. Logging, data compression, and encryption are implemented as decorator classes. The ALT variant, which comprises 318 LOC in a single C++ class,[13] uses flags and if-sequences for turning functionality on or off; the flags can be set when creating a channel. Communication Channels is the only program where the ALT variant has a structured (as opposed to object-oriented) design.

D.1.1 Work Task 1

"*Enhance the functionality of the program such that error-correcting encoding (bit redundancy) can be added to communication channels.*" The error-correcting functionality is already provided by a system class, so the participants only had to integrate that functionality into the program as a new wrapper. The PAT participants had to add a new Decorator class, while the ALT participants had to make additions and changes at various points in the existing program.

We expect two influences of the Decorator on the participants' behavior. First, the ALT variant is easier to understand because its behavior is not delocalized as in the multiple decorator classes. This would lead to the conclusion that the ALT groups should be faster than the PAT groups, especially for participants with low pattern knowledge. Second, a counter-influence results from the structure of the Decorator: The functionality is encapsulated in classes and one need hardly care about mutual influences. In particular, in the ALT variant, the participants have to ensure they add the new functionality at the correct places in the program for proper sequencing of the various switchable functionalities; this will consume time and may lead to mistakes. We expect the second influence to be stronger than the first and, hence, the PAT variant to be preferable, especially at higher levels of pattern knowledge.

D.1.2 Work Task 2

A communication channel has different states (namely, `opened`, `closed`, or `failed`) and its operations have different result codes (`OK`, `failure`, or `impossible`). Work task 2 called to "*determine under which conditions a reset() call will return the 'impossible' result.*" To do this the participants had to find the spots where the states were changed. In the PAT variant, these spots are spread over the decorator classes. Program understanding is gained in the first task, so only the new details relevant for this task need to be learned. We expect this task will be easier for the more localized ALT variant with respect to both time and correctness.

Additionally, the participants were asked to "*create a channel object that performs compression and encryption.*" The ALT participants had to create only a single object (one statement), giving parameters for the functionality flags, while PAT participants had to determine the proper nesting of the decorators to get the required functionality in the requested order. We expect the PAT groups will take longer and commit more errors.

D.2 Composite, Abstrt. Factory: Graphics Library (GR)

Graphics Library contains a library for creating, manipulating, and drawing simple types of graphical objects (lines and circles) on different types of output devices (alphanumeric display, pixel display). In a central class (generator), the output device is selected. Depending on the device, the corresponding types of graphical objects are created. Some basic objects (lines and points) are implemented identically for all devices. However, for complex objects, like circles, implementation depends on the specific device. Furthermore, graphical objects can be collected into groups, that can be manipulated like individual objects.

The patterns used in the PAT variant of this program are Abstract Factory (for the generator classes) and Composite (for hierarchical object grouping). The ALT variant of the program uses switch statements, implemented in a single generator class, to select and instantiate the appropriate classes for each output device. The ALT variant also uses a quasi-Composite to implement object grouping. The only difference is that groups are not treated as objects themselves,

12. [CO-PAT] E_repl = 404 LOC, 6 C++ classes; E_joint = 311 LOC, 6 Java classes (including 1 Java interface).

13. [CO-ALT] E_repl = 342 LOC, 1 C++ class; E_joint = 267 LOC, 1 Java class.

as in the Composite. For instance, a group B is included in another group A by adding each element of B individually to A—i.e., there is no hierarchical group nesting.

The Graphics Library program has a smaller structural difference between its PAT and ALT variants than does the Communication Channels program. The PAT variant comprises 682 LOC in 13 C++ classes;[14] the ALT variant comprises 663 LOC in 11 C++ classes.[15]

D.2.1 Work Task 1

"*Add a third type of output device (plotter).*" Participants in the PAT groups had to introduce a new concrete factory class and extend the factory selector method. Participants in the ALT groups had to enhance the switch statements in all methods of the generator class. Both groups had to add two concrete product classes.

The time for finding the changes is expected to be almost equal for the PAT and ALT groups. Thus, we anticipate the main difference in time required for this task to be caused by program understanding. We expect the simpler ALT variant to be easier to understand, at least for participants with low pattern knowledge. Additionally, we expect that pattern knowledge will help both groups due to the Composite structure in both variants. However, the PAT participants may profit a little more from pattern knowledge since they also interact with the Abstract Factory pattern.

D.2.2 Work Task 2

Determine whether a specific sequence of operations will result in an x-shaped figure. This work task is a small comprehension test concerning the Composite structure. The key to the answer for both groups is finding out that only references to graphical objects (not copies of objects) are stored in an object group. The structure of both variants is quite similar in the region of interest, so we do not expect to observe significant differences between the ALT and the PAT groups. However, we do expect pattern knowledge to have an impact—participants with low pattern knowledge are expected to be slower than those with high knowledge because the latter will be more familiar with the Composite pattern.

APPENDIX E
SUMMARY STATISTICS FOR KEY VARIABLES

Tables 11 and 12 summarize key experiment variables. For each variable, we statistically compare sites using the Kruskal-Wallis rank sum test (which procedure compares medians, rather than means). We use a nonparametric (distribution-free) test because, in several cases, the data are not normally distributed. Histograms of the data are provided in the lab package.

APPENDIX F
UNUSABLE DATA

In 8 cases, we exclude all of a participant's data from analysis as unusable. These data still appear in the data file provided in the lab package (with appropriate annotations), but are completely ignored for analysis.

- *10354*: The participant left both CO task 2 and 3 blank. The timings for those tasks are also both zero seconds, which requires clicking passed the tasks without even reading them. Additionally, the participant reports "many interruptions" during the experiment, but does not provide sufficient information for us to correct his or her timings.
- *11088*: The participant quit the experiment before completing any program tasks.
- *31563*: The participant held a false assumption that the experiment was to be no more than 2 hours, as stated in the participant's comments.
- *36737*: The participant reported having to repeat the experiment due to website problems. Many of his or her timings are extremely short (less than 10 seconds). Presumably, the participant resubmitted previously competed solutions.
- *61820*: Instead of listing programming languages when requested, the participant responded as though s/he did not know any languages or completely misunderstood the question—e.g., "I don't know" and "None that I know of." Also his or her CO/GR task 1 timings are both extreme outliers, totalling 7.75 hours, which implies long, unrecorded breaks. Further, the participant completed the 19-question pattern knowledge survey in only 13 seconds. Given that his or her responses covered an array of categorical selections, such a low time suggests random choices.
- *63358*: Instead of listing programming languages when requested, the participant responded as though s/he did not know any languages—e.g., "Yes, I did. But not that much I've experience on this as I never worked in a software company [*sic*]" and "I worked in the telecom

TABLE 11
Summary statistics for key explanatory variables (see Section 2.2 for a description of each variable). The count of participants for each column is given in parentheses. For each box plot, the median value is labeled, the mean is shown as a black dot, and the max whisker range is 1.5 IQR. All p-values are two-sided; p-values less than or equal to 0.05 are bolded; p-values are based on the Kruskal-Wallis rank sum test comparing sample medians across sites (calculated with R 2.15.2).

	BYU (21)	FUB (12)	UA (14)	UPM (6)	All (53)	*sig.*
Developer Experience (scale 1–7)*	4.39	4.42	3.65	4.38	4.34	0.158
Pattern Knowledge (scale 1–7)*	2.76	3.44	3.65	2.09	3.24	**0.001**

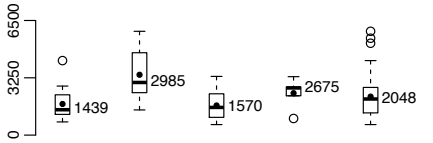
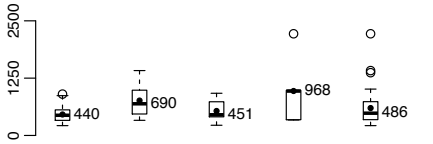
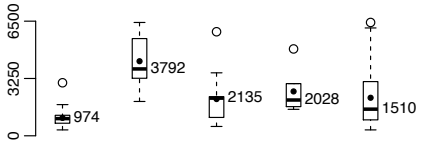
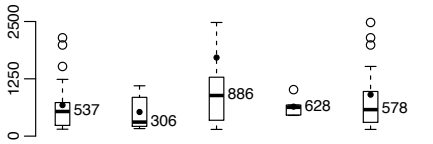
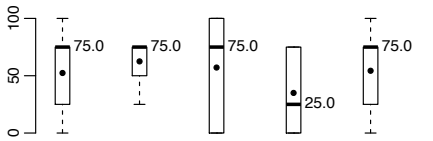
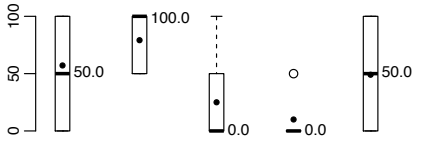
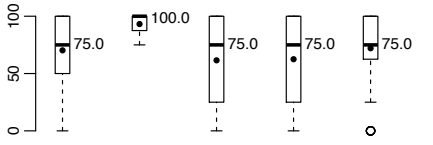
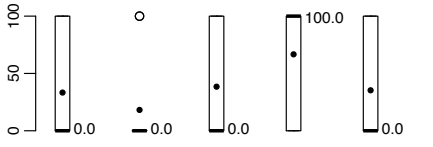
BYU = Brigham Young University
FUB = Freie Universität Berlin
UA = The University of Alabama
UPM = Universidad Politécnica de Madrid
sig. = significance (two-sided p-value)

*Aggregate metric. See Section 2.2 for additional scale information.

14. [GR-PAT] E_repl = 683 LOC, 13 C++ classes; E_joint = 578 LOC, 13 Java classes (including 2 Java interfaces).

15. [GR-ALT] E_repl = 667 LOC, 11 C++ classes; E_repl = 598 LOC, 11 Java classes (including 2 Java interfaces).

TABLE 12
Summary statistics for response variables. For a description of each variable, identified by field number, see Section 2.2 and Appendix G. The count of participants for each column is given in parentheses. For each box plot, the median value is labeled, the mean is shown as a black dot, and the max whisker range is 1.5 IQR. All p-values are two-sided; p-values less than or equal to 0.05 are bolded; p-values are based on the Kruskal-Wallis rank sum test comparing sample medians across sites (calculated with R 2.15.2).

	BYU (21)	FUB (12)	UA (14)	UPM (6)	All (53)	*sig.*
CO Task 1 Time (seconds) (field 66)						**0.002**
CO Task 2 Time (seconds) (field 69)						**0.050**
GR Task 1 Time (seconds) (field 75)						**<0.001**
GR Task 2 Time* (seconds) (field 78)						0.278
CO Task 1 Correctness (0–100%) (field 81)						0.608
CO Task 2 Correctness (0–100%) (field 87)						**0.001**
GR Task 1 Correctness (0–100%) (field 89)						0.147
GR Task 2 Correctness (0–100%) (field 91)						0.261

BYU = Brigham Young University
FUB = Freie Universität Berlin
UA = The University of Alabama
UPM = Universidad Politécnica de Madrid
sig. = significance (two-sided p-value)

*Two outliers not shown: UA=7239 and UA=5842.

company and their I had experience with them very little [*sic*]." The participant also left CO task 3 completely blank, got all tasks completely wrong, and listed almost no developer experience on the pre-questionnaire. The participant does not appear to meet the minimum participation guidelines.

- *91072*: The participant's CO task 1 timing is unreasonably low (76 seconds), whereas his or her CO task 2 timing is exceptionally high. Conversely, the participant achieved a correctness of 50% on task 1, but got task 2 completely wrong. The participant appears to have used the browser back button to confuse the web portal's timing mechanism.
- *94345*: The participant did not complete any of the program tasks.

We also partially exclude data for the following 3 participants. In these cases, we are missing data for either the CO or GR program:

- *15350*: The participant submitted GR source code for CO task 1, so we exclude all CO program data.
- *92863*: The participant submitted CO source code for GR task 1, so we exclude all GR program data.
- *95105*: The participant completed the questionnaires and the CO program tasks, but quit before completing the GR program tasks, so we exclude all GR program data.

APPENDIX G
DATASET SCHEMA DEFINITIONS

In this section, we describe the schema used for the E_joint dataset, which is provided in the lab package. The dataset includes 91 fields.

G.1 Experiment Metadata

These fields describe contextual or administrative elements of the experiment, such as research lab and treatment group assignments.

1) `id`: Unique participant id.
2) `group_idMOD4`: Treatment group (computed as $participant\ ID\ \%\ 4$); corresponds to the alternation of program *order* (CO/GR) and program *variant* (PAT/ALT), as described in Section 2.1.3.
3) `researchLab`: The research team providing the given participant's data, where: BYU=*Brigham Young University*, FUB=*Freie Universität Berlin*, UA=*The University of Alabama*, and UPM=*Universidad Politécnica de Madrid*.
4) `experimentLang`: The programming language in which the given participant completed the experiment, corresponding to one of three options: C++, C#, or Java. In our data, this value is always Java. BYU and UA required Java; the FUB and UPM participants all chose Java.
5) `experimentLangRequired`: Identifies whether the particular research team required their participants to use a specific language for the experiment (corresponding to "TRUE"), or whether the participants were allowed to choose their preferred language (corresponding to "FALSE").

G.2 Pre-Questionnaire 1—Developer Experience

Each field (except for 7, 9, and 18) represents one question on the developer experience survey. Exact questions are provided in the lab package. All scores are self-assessments. For summary statistics, see Appendix C.

6) `langsUsedLifetime`: A comma-separated list of programming languages used at least once in the participant's lifetime (listed in the order given by the participant). Spelling, capitalization, and punctuation have been standardized.
7) `langsUsedLifetime_count`: A count of the number of languages listed in field 6.
8) `langsUsedOften`: A comma-separated list of programming languages known well by the participant and worked with several times (listed in the order given by the participant). Spelling, capitalization, and punctuation have been standardized.
9) `langsUsedOften_count`: A count of the number of languages listed in field 8.
10) `locLifetime`: The total number of lines of code that the participant has ever written in any programming language. The value in this field should be $\geq$ that of field 11.
11) `locJava`: The total number of lines of code that the participant has ever written in Java. The value in this field should be $\leq$ that of field 10.
12) `progHoursPerWeek`: Hours per week in which the participant reads, writes, or modifies code.
13) `progSkill`: The participant's self-assessed programming skill, where: 1=*top 10%*, 2=*top 25%*, 3=*top 40%*, 4=*average*, 5=*bottom 40%*, 6=*bottom 25%*, 7=*bottom 10%*.
14) `studentStatus`: The participant's current student status, where: 1=*undergraduate*, 2=*graduate*, 3=*postgraduate*, 4=*non-student*.
15) `workHoursPerWeek`: Hours per week spent by the participant working as a "professional software developer." See also field 16.
16) `yearsProfExp`: Years spent by the participant working as a "professional software developer." See also field 15.
17) `major`: Main course of study; this question was stated such that it was to be answered only if the participant is currently a student. Thus a non-null answer should indicate that the participant is currently a student and should correspond with a value of 1, 2, or 3 for field 14.
18) `major_translated`: Translation of field 17 from German/Spanish into English.

G.3 Pre-Questionnaire 2—Pattern Knowledge

Each field represents one question on the pattern knowledge survey. Exact questions are provided in the lab package. All scores are self-assessments. Fields 20–37 represent individual patterns. All but two of the individual patterns were originally defined by Gamma *et al.* [5]. Field 31, the Multistructor pattern, does not actually exist and was included as a sanity check. Field 34, the Reactor pattern, was originally defined by Schmidt *et al.* [30, pp. 179–214]. Scores for the individual patterns are based on a 7-point ordinal scale: 1=*never heard of it*, 2=*have only heard of it*, 3=*understand it roughly*, 4=*understand it well*, 5=*understand it well and have worked with it once*, 6=*understand it well and have worked with it two or three times*, 7=*understand it well and have worked with it many times*. For summary statistics, see Appendix C.

19) `patternsUsedLifetime`: The number of software design patterns with which the participant has ever worked.
20) `abstractFactory`: Abstract Factory pattern.
21) `adapter`: Adapter pattern.
22) `bridge`: Bridge pattern.
23) `chainOfResponsibility`: Chain of Responsibility pattern.
24) `command`: Command pattern.
25) `composite`: Composite pattern.
26) `decorator`: Decorator pattern.
27) `factoryMethod`: Factory Method pattern.
28) `flyweight`: Flyweight pattern.
29) `mediator`: Mediator pattern.
30) `memento`: Memento pattern.
31) `multistructor`: Multistructor is *not* an actual pattern. This field was included as a sanity check. Most participants answered 1 (*never heard of it*). A few answered as high as 3 (*understand it roughly*), which is not too surprising out of 61 participants, since the term "multistructor" could reasonably be confused with other patterns. Most importantly, no one answered 4 (*understand it well*) or above. Also, of those participants who answered 2 or 3, all appear (based on their other responses) to have conscientiously completed the experiment.
32) `observer`: Observer pattern.
33) `proxy`: Proxy pattern.
34) `reactor`: Reactor pattern.
35) `strategy`: Strategy pattern.
36) `templateMethod`: Template Method pattern.
37) `visitor`: Visitor pattern.

G.4 Task Responses

These fields include responses submitted for short-answer tasks, as well as responses to post-task questionnaires. Note that the coding tasks (CO/GR task 1s) involved submitting source code; thus they do not appear here. Instead, the participants' source code solutions are included in the lab package.

Concerning fields 47–48 and 56–57, the participants appear to have interpreted these questions in two different ways. Most participants responded with values less than 100%—e.g., 50%, presumably meaning 50% less time, or half the time it would have taken. A few participants responded with values exceeding 100%—e.g., 300%, presumably meaning 300% faster.

38) `CO_task2`: The participant's short-answer response to CO task 2.
39) `CO_task2_translated`: Translation of field 38 from German/Spanish into English.
40) `CO_task3`: The participant's short-answer response to CO task 3.
41) `CO_task3_translated`: Translation of field 40 from German/Spanish into English.
42) `CO_patternsNoticed`: A comma-separated list of design patterns (listed in the order given by the participant), which the participant reports having noticed in the CO program. Spelling, capitalization, and punctuation have been standardized.
43) `CO_difficulty`: The participant's assessment of combined difficulty for all three CO program tasks, where: 1=*quite easy*, 2=*reasonably easy*, 3=*neither easy nor difficult*, 4=*reasonably difficult*, 5=*quite difficult*.

44) `CO_confidence`: The participant's self-reported confidence (as a percentage) that s/he has correctly solved the three tasks for the CO program.
45) `CO_difficultAspects`: Aspects of the CO program tasks which the participant found most difficult.
46) `CO_difficultAspects_translated`: Translation of field 45 from German/Spanish into English.
47) `CO_patKnowHelp`: The participant's self-assessment of how much faster (as a percentage of time) s/he solved the CO program tasks due to his/her personal "knowledge of the design patterns used in the program."
48) `CO_docHelp`: The participant's self-assessment of how much faster (as a percentage of time) s/he solved the CO program tasks due to "the explicit documentation of the design patterns used in the program."
49) `GR_task2`: The participant's short-answer response to GR task 2.
50) `GR_task2_translated`: Translation of field 49 from German/Spanish into English.
51) `GR_patternsNoticed`: A comma-separated list of design patterns (listed in the order given by the participant), which the participant reports having noticed in the GR program. Spelling, capitalization, and punctuation have been standardized.
52) `GR_difficulty`: The participant's assessment of combined difficulty for the two GR program tasks, where: 1 =*quite easy*, 2=*reasonably easy*, 3=*neither easy nor difficult*, 4=*reasonably difficult*, 5=*quite difficult*.
53) `GR_confidence`: The participant's self-reported confidence (as a percentage) that s/he has correctly solved the two tasks for the GR program.
54) `GR_difficultAspects`: Aspects of the GR program tasks which the participant found most difficult.
55) `GR_difficultAspects_translated`: Translation of field 54 from German/Spanish into English.
56) `GR_patKnowHelp`: The participant's self-assessment of how much faster (as a percentage of time) s/he solved the GR program tasks due to his/her personal "knowledge of the design patterns used in the program."
57) `GR_docHelp`: The participant's self-assessment of how much faster (as a percentage of time) s/he solved the GR program tasks due to "the explicit documentation of the design patterns used in the program."

G.5 Final Comments

These fields document all comments provided by participants at the end of the experiment. Note that participants also provided useful comments in fields 45 and 54.

58) `finalComments`: Any final comments the participant submitted after completing the experiment.
59) `finalComments_translated`: Translation of field 58 from German/Spanish into English.

G.6 Survey and Task Times

These fields record web page timings. All timings represent the time spent (in seconds) on the associated web portal page, which is *not* necessarily equivalent to the time spent working—e.g., the participant may have taken a break or been interrupted. Note in this regard that some participants mention in their final comments having taken breaks during the experiment. In most cases, we apply time adjustments to correct for interruptions (see fields 64, 68, 73, 77). In one case, due to insufficient information, we can only record the problem (see fields 70, 79) and exclude the data during analysis.

60) `devExpSurveyTime`: The time spent by the participant on the development experience pre-questionnaire page.
61) `patKnowledgeSurveyTime`: The time spent by the participant on the pattern knowledge pre-questionnaire page.
62) `CO_task1DownloadTime`: The time spent by the participant on the source code download page for CO task 1 (note that this time may include working time—e.g., the participant may have begun reading code before moving on to the task description page).
63) `CO_task1WorkTime`: The time spent by the participant on the description page for CO task 1.
64) `CO_task1WorkTimeCorr`: Time that is to be added to CO task 1 to correct for breaks reported by the participant in his/her final comments.
65) `CO_task1UploadTime`: The time spent by the participant on the solution upload page for CO task 1 (note that this time may include working time—e.g., the participant may have proceeded to this page after reading the task description, but before having worked on the task).
66) `CO_task1TotalTime`: The total time (with corrections) spent by the participant on the download, description, and upload pages for CO task 1 (i.e., the sum of fields 62–65). This is the time recommended for analysis.
67) `CO_tasks2-3WorkTime`: The time spent on the task page for CO tasks 2 and 3. These tasks were presented on the same page and timed together. Tasks 2 and 3 were short-answer questions that did not require the download, modification, or upload of source code.
68) `CO_tasks2-3WorkTimeCorr`: Time that is to be added to CO tasks 2 and 3 to correct for breaks reported by the participant in his/her final comments.
69) `CO_tasks2-3TotalTime`: The total time (with corrections) spent by the participant on the task page for CO tasks 2 and 3 (i.e., the sum of fields 67 and 68). This is the time recommended for analysis.
70) `CO_dataValid`: Specifies whether the data for the CO program tasks should be considered valid for the given participant. "FALSE" indicates that the participant's data are either known to be invalid or are so anomalous (e.g., impossibly small timings) that they cannot reasonably be considered as valid. "TRUE" indicates that the data appear to be valid. A parenthetical note indicates that the data are suspect—i.e., the researcher should consider them with caution. Parenthetical notes include a list of anomalous data fields.
71) `GR_task1DownloadTime`: The time spent by the participant on the source code download page for GR task 1 (note that this time may include working time—e.g., the participant may have begun reading code before moving on to the task description page).
72) `GR_task1WorkTime`: The time spent by the participant on the description page for GR task 1.
73) `GR_task1WorkTimeCorr`: Time that is to be added to GR task 1 to correct for breaks reported by the

participant in his/her final comments.

74) GR_task1UploadTime: The time spent by the participant on the solution upload page for GR task 1 (note that this time may include working time—e.g., the participant may have proceeded to this page after reading the task description, but before having worked on the task).
75) GR_task1TotalTime: The total time (with corrections) spent by the participant on the download, description, and upload pages for GR task 1 (i.e., the sum of fields 71–74). This is the time recommended for analysis.
76) GR_task2WorkTime: The time spent on the task page for GR task 2. task 2 was a short-answer question that did not require the download, modification, or upload of source code.
77) GR_task2WorkTimeCorr: Time that is to be added to GR task 2 to correct for breaks reported by the participant in his/her final comments.
78) GR_task2TotalTime: The total time (with corrections) spent by the participant on the task page for GR task 2 (i.e., the sum of fields 76 and 77). This is the time recommended for analysis.
79) GR_dataValid: Specifies whether the data for the GR program tasks should be considered valid for the given participant. "FALSE" indicates that the participant's data are either known to be invalid or are so anomalous (e.g., impossibly small timings) that they cannot reasonably be considered as valid. "TRUE" indicates that the data appear to be valid. A parenthetical note indicates that the data are suspect—i.e., the researcher should consider them with caution. Parenthetical notes include a list of anomalous data fields.

G.7 Task Correctness Scores

These fields list the correctness scores assigned to participant solutions and short-answer responses. All scores are percentages (0–100%). A suffix of "LabGrade" indicates that the scores were assigned by the research team specified in field 3—i.e., BYU, FUB, UA, or UPM. A suffix of "BYUGrade" indicates that the scores were assigned by the BYU research team, which regraded all solutions to ensure consistency across sites.

80) CO_task1LabGrade: CO task 1 correctness, assessed by the individual research team.
81) CO_task1BYUGrade: CO task 1 correctness, assessed by the BYU research team.
82) CO_task2LabGrade: CO task 2 correctness, assessed by the individual research team.
83) CO_task2BYUGrade: CO task 2 correctness, assessed by the BYU research team.
84) CO_task3LabGrade: CO task 3 correctness, assessed by the individual research team.
85) CO_task3BYUGrade: CO task 3 correctness, assessed by the BYU research team.
86) CO_tasks2-3LabGrade: Average of fields 82 and 84. This field corresponds with field 69.
87) CO_tasks2-3BYUGrade: Average of fields 83 and 85. This field corresponds with field 69.
88) GR_task1LabGrade: GR task 1 correctness, assessed by the individual research team.
89) GR_task1BYUGrade: GR task 1 correctness, assessed by the BYU research team.
90) GR_task2LabGrade: GR task 2 correctness, assessed by the individual research team.
91) GR_task2BYUGrade: GR task 2 correctness, assessed by the BYU research team.

APPENDIX H
DATA PREPARATION PROCESS

Producing the final dataset involved three steps:

1) *Merging individual datasets from the four research teams:* Given that the web portal provides all data in a fixed schema, this step was trivial.
2) *Unifying terminology and format:* This step involved correcting spelling errors, as well as matching capitalization and punctuation across columns to facilitate readability. We modified only the four columns representing lists of either programming languages or design patterns (e.g., changing "Decoratr" and "decorator pattern" both to read as "Decorator"). These four fields were provided by the participants as free-form text. We did not alter the list orderings, nor modify any other columns. To ensure consistency, we made these changes using spreadsheet tools (i.e., spell checking and search/replace tools).
3) *Annotating the data:* This step involved three subprocesses: 1) a column search for data errors (none were found); 2) a column search for outliers; and 3) a row search for participants who deviated from the instructions. All anomalies are recorded in the data file as annotations. The data file is provided in the lab package. We have added two columns to the data file to describe data validity: CO_dataValid and GR_dataValid. For an explanation of how to read these columns, see fields 70 and 79 in Appendix G. We have also added columns for recording time corrections (fields 64, 68, 73, and 77 in Appendix G), which we inferred from the participants' final comments.

The data preparation process involved only minor syntactic corrections for readability, as described in Step 2 above. All other anomalies were merely annotated. In cases where correct values are known, those values are recorded in the annotations. It is left up to the analyst to deal with anomalous data as s/he sees fit, based on the annotations provided.

APPENDIX I
VARIABLES EXCLUDED FROM ANALYSIS

In this section, we list the fields we exclude from statistical analysis, with an explanation for each. Future work may find some of these fields useful. Field numbers correspond to those shown in Appendix G.

- group_idMOD4 (field 2): Group is a surrogate for the combination of program *order* and *variant*.
- experimentLang (field 4): All participants used the same programming language (Java), so this field does not differentiate participants.
- experimentLangRequired (field 5): This field conveys little information beyond that already provided by field 3, researchLab.

- `langsUsedLifetime` (field 6): This field is replaced by field 7, `langsUsedLifetime_count`.
- `langsUsedOften` (field 8): This field is replaced by field 9, `langsUsedOften_count`.
- `studentStatus` (field 14): This field can be viewed as a high-level indicator of developer experience. However, it does not necessarily represent developer experience. For instance, in our data, student status does not correlate with `progSkill` or `progHoursPerWeek` (0.02 and 0.06, respectively)—both of which seem directly related to developer experience.[16] Although student status could represent some other important concept, we exclude it from analysis because, within the context of the PatMain study, it only seems relevant as a measure of developer experience.
- `workHoursPerWeek` and `yearsProfExp` (fields 15 and 16): We ignore working-hours-per-week and years-professional-experience for two reasons. First, both questions ask about "professional" work. Second, in hindsight, the term *professional* seems ambiguous for students. Should students count part-time work, or do these questions refer only to full-time work? Many participants (18) do, in fact, report part-time hours. However, more than half of the participants (29 of 53) report zero hours and zero years experience, at least some of whom may be working in professional software companies, but do not consider their jobs to be "professional" because the work is not full time.
- `major` (field 17): All participants are essentially the same major, so this field conveys little information.
- `patternsUsedLifetime` (field 19): This field includes several unreasonable outliers. We could deal with these outliers by bucketing or otherwise transforming the data. However, estimating the number of patterns one has ever used seems more difficult than estimating one's knowledge of specific patterns. Thus we prefer to use the individual pattern knowledge assessments instead (fields 20–37).
- `multistructor` (field 31): This field is a sanity check. No such design pattern actually exists.
- `CO_task2`, `CO_task3`, and `GR_task2` (fields 38, 40, and 49): These fields are replaced by correctness scores (fields 87 and 91).
- `devExpSurveyTime` (field 60): Survey times are unrelated to the experiment hypotheses.
- `patKnowledgeSurveyTime` (field 61): Survey times are unrelated to the experiment hypotheses.
- Component task times (including download, work, upload, and corrections; fields 62–65, 67–68, 71–74, and 76–77): We use the total times instead (sum of the component times) because of uncertainty in how the participants navigated the task download, description, and upload pages.
- `CO_dataValid` and `GR_dataValid` (fields 70 and 79): These fields apply only to pre-analysis decisions concerning which data entries are valid.
- "`LabGrade`" correctness scores (fields 80, 82, 84, 86, 88, and 90): For consistency across sites, we use the centrally-graded BYU scores instead.
- `CO_task2BYUGrade` and `CO_task3BYUGrade` (fields 83 and 85): Like E_repl, we combine CO tasks 2 and 3. Thus these fields are replaced by field 87, `CO_tasks2-3BYUGrade`.
- Post-questionnaire data and final comments (fields 42–48, 51–57, and 58): Where applicable, we apply these responses qualitatively to help interpret the statistical results.

16. Pearson product-moment correlation coefficients (calculated with R 2.15.2); two-sided p-values = 0.89 and 0.65, respectively.

APPENDIX J
DEVELOPER EXPERIENCE DERIVED METRIC

To compose the developer experience metric, we transform, scale, and average 6 component metrics (field numbers correspond to those shown in Appendix G):

1) `langsUsedLifetime` (field 6)
2) `langsUsedOften` (field 8)
3) `locLifetime` (field 10)
4) `locJava` (field 11)
5) `progHoursPerWeek` (field 12)
6) `progSkill` (field 13)

For a description of each component, see Appendix G. For summary statistics, see Appendix C. The aggregation process is accomplished in 6 steps:

1) Replace each response for `langsUsedLifetime` and `langsUsedOften` with its cardinality.
2) Apply a natural log transformation to each response for `langsUsedOften`, `locLifetime`, and `locJava` to correct for skew and outliers.
3) Scale all component variables (`langsUsedLifetime`, `langsUsedOften`, `locLifetime`, `locJava`, `progHoursPerWeek`, and `progSkill`) to a range of 1–7, such that zero maps to 1 and the variable's maximum value maps to 7. Note that `progSkill` is already on a 1–7 scale. However, for that variable 7 initially represents low skill; thus the scale must be reversed.
4) Average the components `langsUsedLifetime` and `langsUsedOften` to create a single variable representing the general concept, `langsUsed`.
5) Average the components `locLifetime` and `locJava` to create a single variable representing the general concept, `locWritten`.
6) Finally, average the four variables `langsUsed`, `locWritten`, `progHoursPerWeek`, and `progSkill` to produce the final aggregate metric.

The final metric is a continuous variable ranging from 1 to 7 (scaled to match the range of the pattern knowledge metric), where 7 represents high experience. The metric is an average of four core components—languages used, LOC written, programming hours per week, and self-assessed programming skill—with each component receiving a weight of 25% in the average. The four components (after pre-averaging) only moderately correlate (from 0.2 to 0.4), which is ideal for creating an effective aggregate metric. The components measure related concepts, yet each component incorporates unique information.

Additional notes:

- The log transformation impacts the Pearson product-moment correlation, but not the rank correlation. Scaling impacts neither.

- We pre-average some variables (steps 4 and 5) in order to reduce their collective impact on the final average. In each case, the variables are highly correlated—0.87 in the case of the LOC metrics and 0.63 in the case of the languages used metrics.[17] These correlations are significantly higher than those for any other pair of variables, with the next highest being only 0.43. Thus we pre-average in cases where multiple variables measure a similar concept to prevent that concept from having excessive influence on the final metric.
- A few entries in the dataset are clearly erroneous. In most cases, no correction is possible because we have no idea what the true responses should have been. However, in two cases we do adjust the data prior to computing the aggregate metric. For participants 38048 and 92689, we append a copy of their `langsUsedOften` response to the end of their `langsUsedLifetime` response. In both cases, the participants report disjoint sets for these variables. Conversely, all other participants report `langsUsedOften` as a subset of `langsUsedLifetime`.

APPENDIX K
JAVA FAMILIARITY DERIVED METRIC

The java familiarity metric was the only metric we tested in the statistical models that turned out to be completely unhelpful. The metric is derived from the languages-used-often variable, based on the following ordinal scale: 1=*the participant does not list any object oriented languages*, 2=*the participant lists only non-Java object oriented languages*, 3=*the participant lists Java*. Most participants (47/53) list Java, so this variable has little statistical impact. Thus, we ignore it in the discussion, other than to mention that we explored it.

APPENDIX L
STATISTICAL MODEL ASSUMPTIONS

The frequentist models depend on several assumptions, including: 1) response variables should be normally distributed; 2) explanatory variables should *not* be significantly correlated; and 3) explanatory variables should be homoscedastic (i.e., of constant variance, as opposed to heteroscedastic). Given our setup, assumptions 2 and 3 also apply to the Bayesian models.

L.1 Normality

The *time* data are skewed with several significant outliers. For the frequentist analysis, we normalize *time* by log-transformation (see Figs. 4–5 and Table 13). However, for the Bayesian analysis, we model *time* using a gamma distribution, so normality is not an issue in that case. Concerning *correctness*, the range is discretized into only five buckets, thus precluding the possibility of gross outliers (a primary threat to model validity and a principle reason for concern with normality). Also, with such a limited range, *correctness* cannot take on much of a skew. Thus, we do not apply a log transformation to *correctness*. That said, with only five buckets, the *correctness* variable is unlikely to fit a smooth normal distribution. Thus, although we assume normality for the frequentist analysis, we avoid that assumption entirely in the Bayesian analysis by modeling *correctness* with a beta distribution.

17. All correlations in this section are Pearson product-moment correlation coefficients (calculated with R 2.15.2). Parametric tests are appropriate for these data because they have been previously normalized via log transformation.

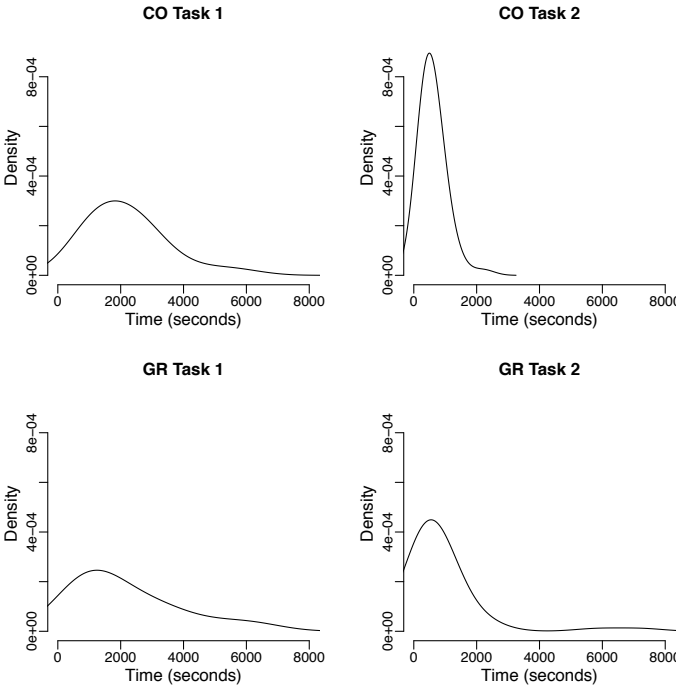


Fig. 4. Density plots for the *time* response variable *before* log transformation. Compare to Fig. 5 and Table 13.

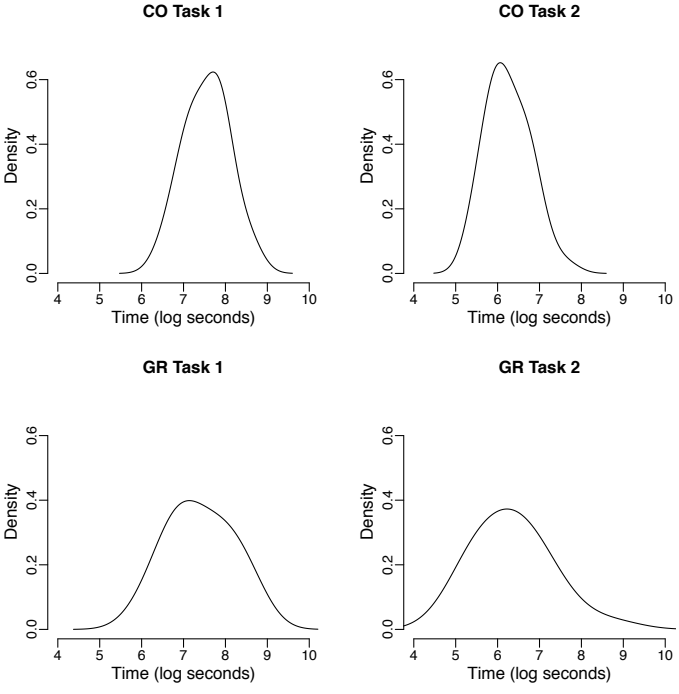


Fig. 5. Density plots for the *time* response variable *after* log transformation. Compare to Fig. 4 and Table 13.

TABLE 13
Results for the Shapiro-Wilk test of normality for the *time* response variable before/after log transformation (calculated with R 2.15.2). p-values represent the probability of obtaining a given sample, assuming a normally distributed population—i.e., low p-values indicate risk of non-normality. Compare to Figs. 4 and 5.

	Shapiro-Wilk p-value *Before* Log Transformation	Shapiro-Wilk p-value *After* Log Transformation
CO Task 1	2.16×10^{-04}	0.755
CO Task 2	9.17×10^{-07}	0.275
GR Task 1	1.86×10^{-05}	0.286
GR Task 2	1.52×10^{-11}	0.044*

*This p-value indicates possible non-normality. However, that conclusion depends on a single extreme outlier, participant 90620, without which the p-value becomes 0.211. Also, the log-time density plot for GR task 2 appears roughly normal even with the outlier (see Fig. 5), and our final results ultimately exclude participant 90620 (as described in Section 3.1). Thus, we are not concerned about normality in this case.

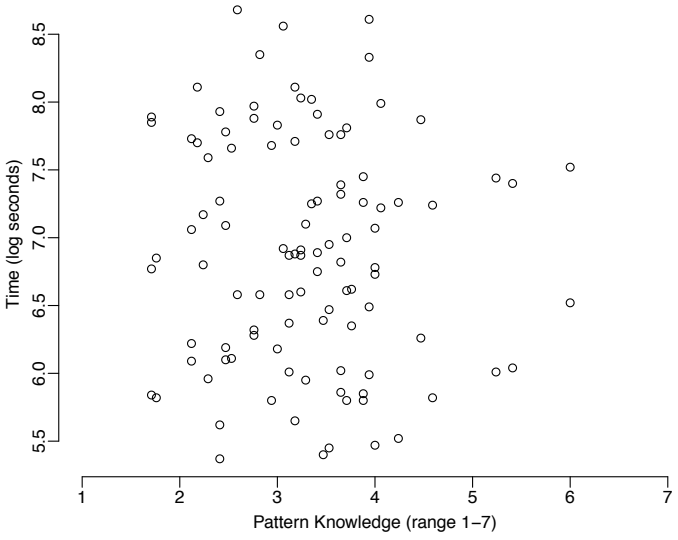


Fig. 6. Scedasticity plot for *patKnow* (CO_time model). Data sparseness in the upper range (>4.0) indicates the possibility of heteroscedasticity.

L.2 Multicollinearity

Multicollinearity occurs when two or more explanatory variables in a statistical model are significantly correlated. In the presence of multicollinearity a model's predictive power and overall reliability are not impacted. However, parameter estimates for the collinear variables may be inaccurate and can change erratically with only small changes to the data. For instance, two highly significant, but collinear variables can both appear insignificant when included in the model together.

All correlations between explanatory variables for the models considered in this paper are low. Generally, multicollinearity is not a problem for pairwise correlation magnitudes below 0.7 [39],[18] and the highest magnitude among our explanatory variables is only 0.4 (between *devExp* and *time_ln* on GR task 2).[19] Thus multicollinearity is not a concern. A complete list of all correlations is included in the lab package.

L.3 Heteroscedasticity

Heteroscedasticity occurs when the variance in a dataset differs across sub-populations (e.g., treatment groups). This condition can be seen by plotting each explanatory variable against the response variable. If any expanding, shrinking, or multi-modal patterns are visible across the range of the explanatory variable, then heteroscedasticity is a concern. We provide scedasticity plots for all explanatory variables in the lab package. For our dataset, scedasticity plots indicate concern in only one case—*patKnow*. For *patKnow*, only 13% of participants score above 4.0, such that the variance appears lower in the upper range (see Fig. 6). Our statistical models assume constant variance, which may be true, but the data are simply too sparse in the upper range to know for sure.

If *patKnow* is heteroscedastic, the statistical results would only be minimally impacted. First, *heteroscedasticity does not bias least squares coefficient estimates, so none of our parameter estimates in the frequentist analysis would be affected*. Second, heteroscedasticity does impact variance estimates, which in turn can bias p-values. However, such a bias would primarily affect only the upper range of *patKnow*, and it would most likely mean inflated p-values. Specifically, since most of the data reside in the lower range, variance for the upper range (if it is inaccurate) is likely overestimated by the common variance term. Consequently, if we were to allow our models to estimate separate variances for high and low *patKnow*, and if the data sparseness problem were resolved, the resulting p-values would more likely decrease or remain unchanged than to increase. *In other words, if our analysis is biased with respect to pattern knowledge, it is most likely biased toward type 2 errors—failure to reject the null hypothesis; furthermore, any such bias would apply only to the upper range.*

APPENDIX M
TUNING FREQUENTIST MODELS

We tune all frequentist models using a standard covariate pruning technique [31, p. 345]. The technique is essentially a modified form of backward stepwise regression. We avoid basic stepwise regression because it effectively constitutes data dredging (i.e., fishing for significance), which can seriously bias the results [31, pp. 353–354]. The primary difference between the modified form (which makes it appropriate) and the basic form is that we drop all main effects from the model before performing the elimination procedure. This way, tuning does not manipulate the final results. The main effects are only added back to the model once tuning is complete. Discarded covariates are still adjusted for in the final models, since they had a chance to be included prior to adding the main effects back in [31, pp. 345–347].

18. Most texts cite correlation thresholds in the range 0.5 to 0.9 as indicating potential multicollinearity [39], [40]. Also, note that correlation magnitudes are not a direct measure of collinearity, and they can fail to detect the condition in some cases. However, *all* collinearity detection methods are subject to some error, and rule-of-thumb correlation thresholds have been shown to perform at least as well as the more complicated methods [39].

19. We use Pearson product-moment correlation coefficients (calculated with R 2.15.2). Parametric tests are appropriate in this case because the data have been normalized via log transformation.

TABLE 14
Prior distributions for all Bayesian model parameters.

Response Variable	Models	CO Task 1 Variance	CO Task 2 Variance	GR Task 1 Variance	GR Task 2 Variance	Base Offset	All Other Parameters
time	T1–T6	$\Gamma\,(3, 480000)$	$\Gamma\,(3, 83333)$	$\Gamma\,(3, 403333)$	$\Gamma\,(3, 163333)$	$\mathrm{N}\,(1900, 500^2)$ $3\sigma = 25$ min.	$\mathrm{N}\,(0, 1000^2)$ $3\sigma = 50$ min.
correctness	C1–C6	$\Gamma\,(2, 0.07)$	$\Gamma\,(2, 0.08)$	$\Gamma\,(2, 0.055)$	$\Gamma\,(2, 0.12)$	$\mathrm{N}\,(0.5, (0.4/3)^2)$ $3\sigma = 40$ pts.	$\mathrm{N}\,(0, 0.3^2)$ $3\sigma = 90$ pts.

$\Gamma(k, \theta)$ = gamma distribution, where k and θ represent shape and scale.
$\mathrm{N}(\mu, \sigma^2)$ = normal distribution, where μ and σ^2 represent mean and variance.
3σ = the approximate practical range of a normal distribution on either side of the mean.

Conceptually, the tuning process acts as a high-level filter that removes any covariates unrelated to the response variable.

We use p-values for the elimination criterion. The process requires iteratively removing the least significant covariate until all remaining covariates are at least moderately significant (p-values $\lesssim 0.1$). For all models, we treat program *variant* and all interactions as main effects. After returning the main effects to the model, the final step is to drop any non-significant interactions. In general, we never include an interaction without including its lower order terms. Thus we always drop high-order interactions first. Pruning interactions does reintroduce the concern of data dredging. However, given the nearly complete lack of significance that the interactions have in all models, removing them is appropriate and the threat of data dredging is minimal.

Appendix N
Discretization of Bayesian Variables

As explained in Section 2.3, for the Bayesian models, we discretize all continuous explanatory variables into *low* and *high* categories. We divide variables based on a visual inspection of clustering and/or by conceptually interpreting the variable's scale. The resulting partitions for each variable are follows:

- *devExp*: 2 buckets, representing high and low developer experience (high = scores of 4.5–7.0 inclusive, matching 22 of 53 participants).
- *patKnow*: 2 buckets, representing high and low pattern knowledge (high = scores of 3.5–7.0 inclusive, matching 21 of 53 participants).
- *time* (when used as a covariate for *correctness*): 2 buckets, representing high and low work times. High times are determined on a per-task basis:
  - CO task 1: $\geq$2000 sec. (26 of 52 observations)
  - CO task 2: $\geq$500 sec. (25 of 52 observations)
  - GR task 1: $\geq$1900 sec. (22 of 51 observations)
  - GR task 2: $\geq$700 sec. (18 of 51 observations)
- *correctness* (when used as a covariate for *time*): 2 buckets, representing high and low solution correctness (high = scores of 75–100 inclusive, matching 103 of 206 observations).

Appendix O
Bayesian Priors

Bayesian priors are shown in Table 14. To avoid biasing the Bayesian analysis, we enlisted an external researcher to provide estimates for all priors. Our helper—who had 5 years of experience managing professional developers, as well as 10 years of experience teaching undergraduate and graduate computer science students—is an expert in the area of Bayesian statistics. To inform our helper, we gave him mean and variance data for all CO and GR tasks from E_orig. We did not give him any data from E_joint.

We gave our helper only two constraints in selecting the priors (both suggestions of Felt [34]): First, we instructed him to center all priors—with the exception of the variances and base offset—at zero, thus assuming no effect by default (i.e., the null hypothesis). Second, we instructed him to select broad priors. Doing so allows the posterior distributions to move more easily in response to the data, thus minimizing the weight of our biases in the analysis. In general, choosing broad priors leads to broader posteriors, but for a *post-hoc* analysis, sacrificing some precision is an acceptable tradeoff in order to focus the analysis more on the data. After all, the purpose of a *post-hoc* analysis is to formulate data-driven conjectures.

In selecting the priors, our helper assumed that the student participants would take longer and score lower (on average) than the professionals from E_orig. He also assumed that they would display greater variance than the professionals. Our helper chose normal distributions for most parameters primarily because we have no reason to believe anything other than symmetric noise. He chose gamma distributions for the variances because the support for gamma is limited to values greater than zero.[20]

Appendix P
Notes on Observation Filtering

E_repl used observation filtering when modeling the *time* response variable. As Vokáč *et al.* explain, "Since completion times have little meaning for solutions with low correctness, only those solutions achieving correctness score 4 ('almost correct') or 5 ('correct') were used in [the *time* analysis]" [20, p. 158]. Scores of 4 and 5 in E_repl's data correspond to scores of 75% and 100%, respectively, in our data. Although the authors do not explain why low-scoring solutions are problematic, one concern is that the associated timings may be *censored* (i.e., artificially capped). For example, some participants may have submitted an incomplete solution simply because they were tired of the task.

20. Inverse gamma is a common choice for variance, but given our use of Gibbs sampling, conjugacy was not necessary.

TABLE 15
Unfiltered Bayesian results showing the *correctness* × *variant* interaction and the marginalized effect of *variant* for each of the four tasks (*u*T4:46–54,387–406). Probabilities exceeding a significance (*sig.*) of 0.75 are bolded. Insignificant probabilities are those near 0.5.

program	*task*	*correctness*	ALT−PAT	*p*(ALT>PAT)	*sig.*
CO	1	low	120	0.61	0.61
CO	1	high	−373	**0.17**	**0.83**
CO	1	*marg.*	−126	0.39	0.61
CO	2	low	−69	0.40	0.60
CO	2	high	135	0.67	0.67
CO	2	*marg.*	−46	0.51	0.51
GR	1	low	−409	**0.23**	**0.77**
GR	1	high	317	**0.78**	**0.78**
GR	1	*marg.*	33	0.53	0.53
GR	2	low	−28	0.47	0.53
GR	2	high	−408	**0.17**	**0.83**
GR	2	*marg.*	−218	0.32	0.68

ALT−PAT = the difference between variants (in seconds)—i.e., the difference between the posterior distribution means.
p(ALT>PAT) = posterior probability that the ALT variant takes longer than the PAT variant.
sig. = significance of *variant* (i.e., max of p and $1-p$, where p is the posterior probability).
marg. = marginal posterior probability for *variant*, factoring out its interaction with *correctness*.

TABLE 16
Unfiltered frequentist and Bayesian results showing the significance of *correctness* as a covariate in the *time* models (and vice versa). p-values less than or equal to 0.05 and posterior probabilities exceeding 0.75 are bolded. All p-values are two-sided. All posterior probabilities describe the probability of a positive correlation between *correctness* and *time*.

Model	Covariate	p-value	Posterior Probability*
CO_time	correctness	0.071	-
CO_correctness	time_ln	0.110	-
GR_time	correctness	<**0.001**	-
GR_correctness	time_ln	**0.003**	-
T1	correctness	-	**0.99**
T2	correctness	-	**0.97**
T3	correctness	-	**0.91**
T4	correctness	-	0.68†
T5	correctness	-	**0.95**
T6	correctness	-	**0.90**
C1	time	-	**0.85**
C2	time	-	**0.84**
C3	time	-	**0.82**
C4	time	-	0.63†
C5	time	-	**0.81**
C6	time	-	**0.88**

*Source: *u*T1–T6,C1–C6:61–63.
†For these models only, the covariate is interacted with other variables. In both cases, the interaction requires estimating 16 parameters, rather than 2. The increase in parameters dampens statistical significance [31, p. 347].

Observation filtering can mitigate the problem of censored data, but it also limits the generality of the results. Further, from a statistical standpoint, observation filtering actually addresses two separate issues: 1) it accounts for variance due to an interaction between *correctness* and *variant*; and 2) it accounts for variance due to a relationship between *correctness* and *time* (i.e., participants may score higher simply by working longer). Thus, to improve on the use of observation filtering, we address these issues separately.

P.1 *correctness* × *variant* Interaction

Many conditions could induce an interaction between *correctness* and *variant*. For instance, added noise from data censoring could mask the effect of *variant* at low levels of *correctness*. We test for a *correctness* × *variant* interaction in the Bayesian models. For those models, we divide *correctness* into low and high buckets, based on the same threshold used by E_repl (as shown in Appendix N).

Table 15 shows the *correctness* × *variant* interaction and the marginalized effect of *variant* for each task. The table indicates a relatively strong interaction between *correctness* and *variant*. First, *variant* is more significant within the interaction than as a standalone variable. Second, for all tasks, the effect of *variant* (denoted ALT−PAT in the table) varies considerably across *correctness* levels (low, high). For instance, on GR task 1, PAT is estimated to take 409 seconds *longer* than ALT when correctness is low. However, for high correctness, PAT requires 317 seconds *less* than ALT—a difference of 12.1 minutes. For tasks requiring 20–30 minutes to complete, even 5 minute differences can be significant.

Thus, we do find an interaction between *correctness* and *variant*, but that interaction is inconsistent across tasks. Therefore, 1) observation filtering is inadvisable, and 2) excluding low-scoring data is too blunt a method to account for the *correctness* × *variant* relationship we observe across tasks.

We confirm this conclusion by applying observation filtering to the frequentist models. After filtering, the effect of *variant* is completely lost. Thus, the significance of *variant* depends on both low and high correctness scores, and consequently, we cannot filter individual observations based solely on correctness. For additional discussion of the *correctness* × *variant* interaction, see Section 3.2.

P.2 *correctness-time* Relationship

We test for a relationship between *correctness* and *time* by including *correctness* as a covariate in the *time* models (and vice versa). The Bayesian models (see Table 16) indicate that *correctness* and *time* are likely related in E_joint's data. The frequentist models further reveal that the relationship varies by program, being highly significant for the GR program, but less so for CO.

The *correctness-time* correlation is positive in all models—i.e., the participants are likely achieving higher scores at the expense of time. The magnitude of the effect is modest. For example, in the GR_time model, a 10-point increase in correctness corresponds with a 5.9% increase in time. The other three frequentist models show similar results (see Tables 25, 29, and 37 in Appendix Y). According to the Bayesian models, achieving a high score (75% or 100%) is associated with a 2–4 minute increase in time (which could be significant relative to 20–30 minute tasks).

E_repl also included *correctness* as a covariate in the *time* models, but found it to be insignificant. However, E_repl

only tested *correctness* after having filtered the data. If a relationship did exist, it was likely lost due to the filtering. As a test, we applied observation filtering to our own data, exactly as described by Vokáč *et al.*, and reran the frequentist models. Similar to E_repl, *time* and *correctness* both became insignificant in all models after the filtering (p-values ≥ 0.23). Thus we do find a significant, though modest, relationship between *correctness* and *time*, and observation filtering does account for that relationship.

P.3 Summary

We find significant evidence for an interaction between *correctness* and *variant*, but that interaction is inconsistent across tasks. Thus we cannot exclude low-scoring observations without eliminating the main effect. Further, we find evidence for a small positive correlation between *correctness* and *time*, which is fully accounted for by observation filtering. However, since observation filtering is not acceptable, we must account for the *correctness-time* relationship by including *correctness* as a covariate in the *time* models (and vice versa), rather than by filtering.

We conclude that observation filtering, as implemented by E_repl, does not reduce cross-site variance for E_joint. As a method for reducing irrelevant variance, it is simply too inefficient. We also propose that the analysis of E_repl *may* be improved by addressing the *correctness* $\times$ *variant* interaction and the *correctness-time* relationship separately, via statistical modeling, rather than through observation filtering.

Appendix Q
Notes on Participant Filtering

In this section, we provide additional information on the participant filtering discussed in Section 3.1.

Q.1 Filtering by Correctness

To help ensure objectivity, we enlisted four *independent* reviewers. The four reviewers were all software engineering researchers, including two not affiliated with this study. Based on a plot of the averages (see Fig. 7), the reviewers unanimously selected a threshold of 25 percentage points. All of the excluded participants received a zero on all, or nearly all tasks. A zero score is only given when *the solution is completely wrong, and it appears that the participant did not understand the requirements.* Thus the excluded participants were likely underqualified and/or insufficiently motivated.

Figs. 7a, 7b, and 7c present different views of the E_joint participants, plotted by average task time and correctness. Fig. 7a was the specific plot used by the four independent reviewers to decide the filtering threshold. Fig. 7b shows the participants categorized by site. Fig. 7c adds ID labels and the filter threshold. Notice that the American universities (BYU and UA) account for most of the participants filtered.

Fig. 7d is a *post-hoc* validation of the participant filtering—i.e., we generated it only *after* choosing the filter threshold. Fig. 7d shows the distribution of participants from E_joint whose data were identified during the annotation process as questionable. For example, participant 15350 reported having previously written zero lines of code in any language. For a complete list of such concerns, see the data file in the lab package. Fig. 7d also shows the distribution of the professionals from E_orig. E_orig's participants completed tasks on paper, rather than on a computer, so comparisons to E_joint are only tentative. That said, notice that the questionable data mostly fall to the left of the filter, whereas all of E_orig's participants (the professionals) fall to the right.

Figs. 8 and 9 show *time* and *correctness* displayed according to the *site* $\times$ *variant* interaction. The plots depict the data before and after participant filtering. We include the figures for two reasons: 1) to show that the main effect, program *variant*, significantly varies across the four sites; and 2) to show that participant filtering only marginally reduces that variance. See Sections 3.1–3.2 for further discussion.

Q.2 Filtering by Time

In addition to filtering by *correctness*, we also asked the reviewers to select a threshold for filtering by *time* (again, based on Fig. 7a). Possibly, high times indicate underqualified participants, similar to low correctness. Or more likely, high times indicate technical difficulties or the taking of unrecorded breaks. For example, at the outset of the analysis we discarded all data for participant 57033 because s/he reported having spent more than an hour setting up an IDE.

Each reviewer selected a different threshold (approximately 2250, 2550, 2650, and 3200 seconds). We tested each threshold and found that none substantially affect the results when applied in addition to the *correctness* filter. If anything, filtering by *time* slightly reduces the significance of *variant*—likely due to the loss of good data. Therefore, like E_repl, we do not exclude participants based on work times.

Appendix R
Visualization of Moderator Variables

In this section, we provide an extended version of Table 5 from Section 3.2. Table 17 extends Table 5 in three ways: 1) it shows probabilities for additional interactions; 2) it visualizes the probabilities via box plots; and 3) it provides back references to the Bayesian results tables, mapping the box plots to the posterior probabilities on which they are based. Below we explain how to read Table 17. The explanation assumes familiarity with Table 5.

In Table 17, we replace Table 5's column labels, $\neg c$ and c, with the labels *vpt* and *vptc* (where v=*variant*, p=*program*, t=*task*, and c=*candidate*). We then add six additional interaction columns: *v*, *vp*, *vt*, *vc*, *vpc*, and *vtc*. We include the additional columns to show that, in most cases, the significance of *variant* depends not only on the given candidate moderator, but also on *program* and *task*—i.e., *variant* is most significant in the four-way interactions (labeled *vptc*). Thus, we focus in the main paper on the interactions *vpt* and *vptc*.

In Table 17, instead of listing only the max significance for each interaction, we use box plots to show the full range of significances. Viewing the full range is helpful in order to see how the spread of probabilities changes across the interactions. For the *vpt* and *vptc* interactions, we label the max significance values and (in parentheses) the corresponding effect estimates. The labeled values match the numbers shown in Table 5.

Table 17 also includes back references to the Bayesian results tables. The back references are important as an audit

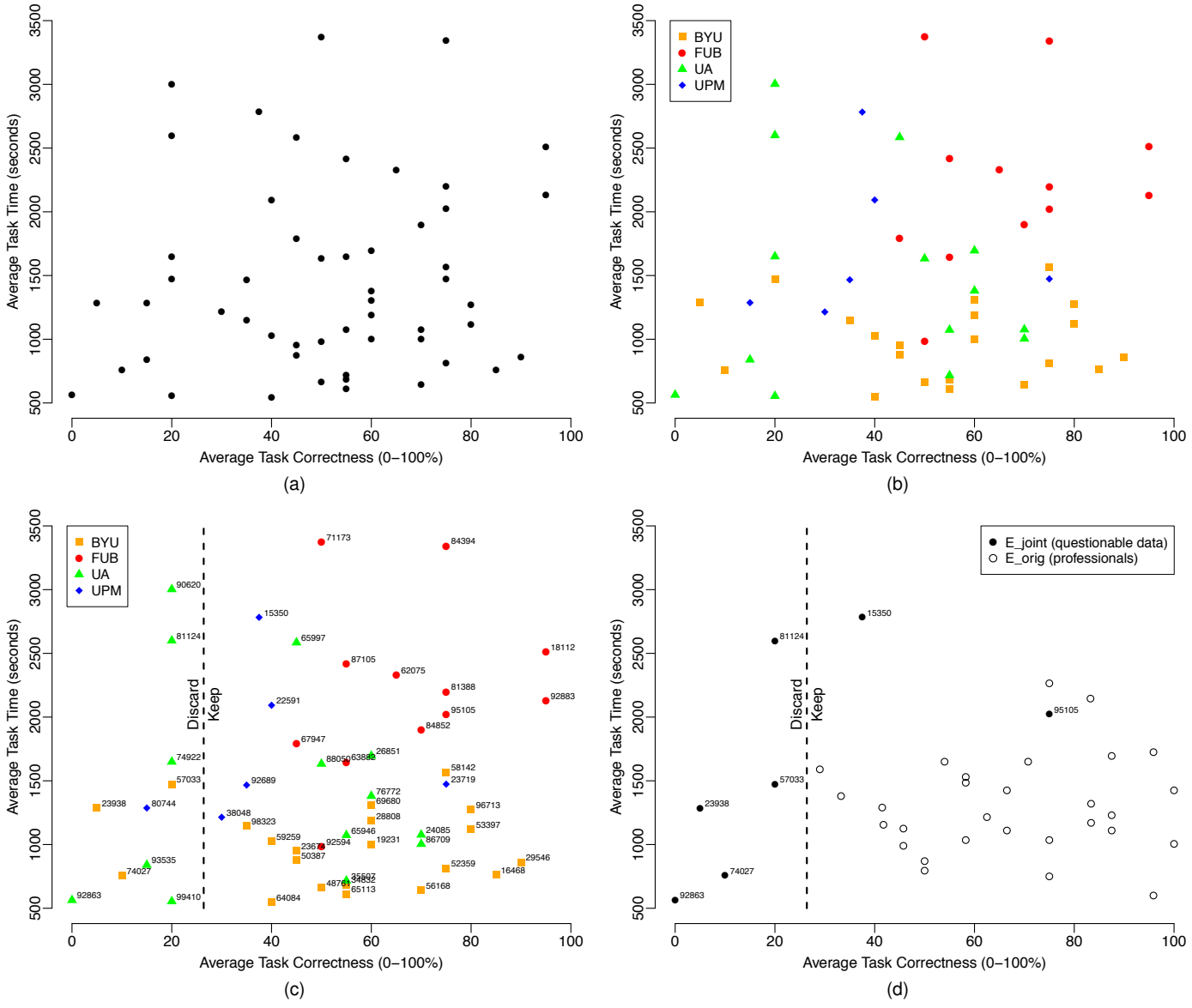


Fig. 7. Participants plotted by average task time and correctness. (a) Plot of E_joint participants used by the four independent reviewers to select the filter threshold. (b) Same as prior plot, but with participants categorized by site. (c) Same as prior plot, but including participant IDs and filter threshold. (d) *Post-hoc* validation of the filter threshold, showing questionable data from E_joint, as well as professionals from E_orig.

trail for the analysis. They also allow the reader to compare interactions in more detail, if desired. To locate the source data for a given box plot, note the table identified in the header (i.e., Table 59 or 60 from Appendix Z), the Bayesian model listed on the left (i.e., one of T1–T4 or C1–C4), and the row numbers provided directly below the box plot. For example, source data for the top left box plot can be found in Table 59, column T1, lines 33–35 (*u*T1:33–35).

APPENDIX S
MODERATOR VARIABLES CONTINUED

The material in this section is a continuation of Section 3.2.

S.1 Task Difficulty

Nearly twice as many E_joint participants complained in their post-questionnaire comments about the difficulty of the CO tasks, as compared to GR (13 versus 7). The participants also assessed the CO tasks as being slightly more difficult, and they reported feeling slightly less confident in their CO solutions. Correspondingly, *variant* is less significant in the CO_time model than in the GR_time model (before filtering, CO_time p-value = 0.925, GR_time p-value = 0.016). Also, filtering low-scoring participants strongly increases the significance of *variant* in the CO_time model, but has little impact on the GR_time model (CO_time p-value shifted from 0.925 to 0.019, GR_time p-value shifted from 0.016 to 0.025).

If the CO tasks were more difficult than the GR tasks, then presumably the low-scoring (i.e., underqualified/undermotivated) participants failed so badly that they masked the main effect in the unfiltered CO_time model. In the case of GR_time, however, those same participants did not struggle as much, and so the main effect is detectable without filtering. Thus, *the data suggest that a threshold of experience/motivation exists, which is dependent on task difficulty,*

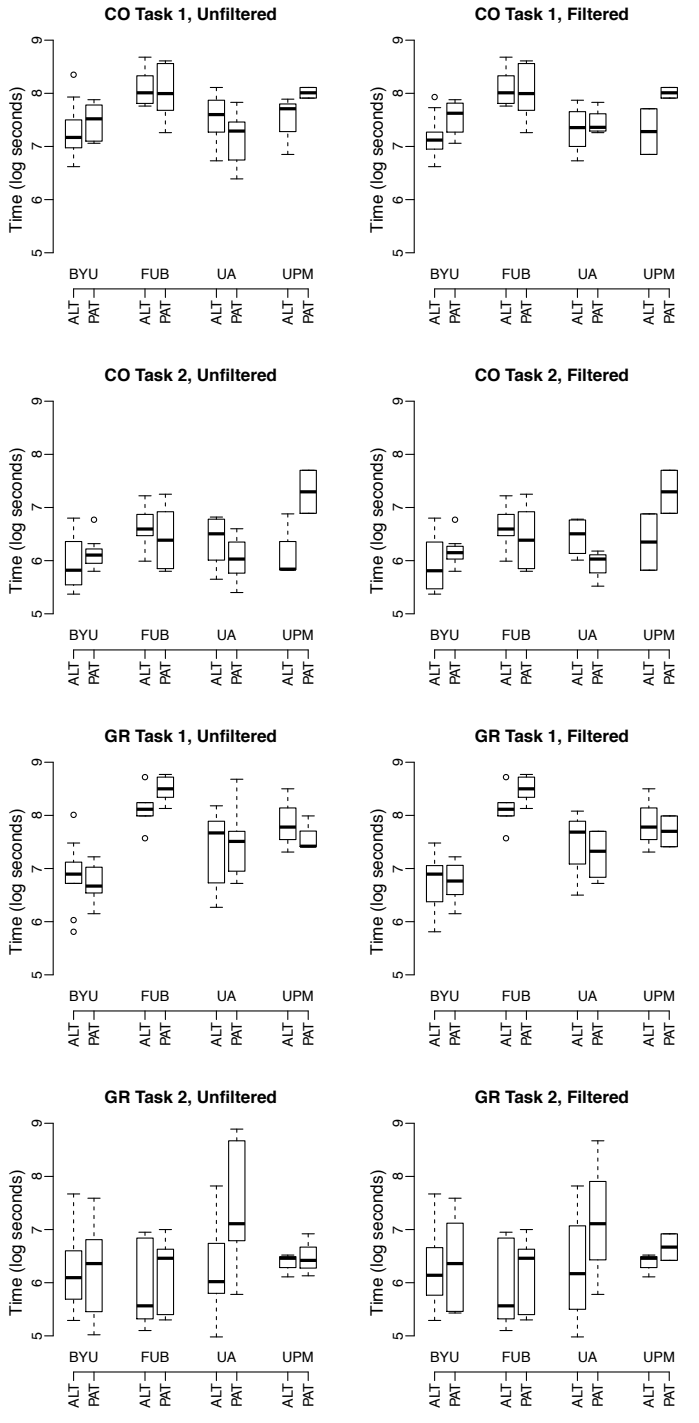


Fig. 8. Time data, showing ALT versus PAT displayed by site. Max whisker range is 1.5 IQR.

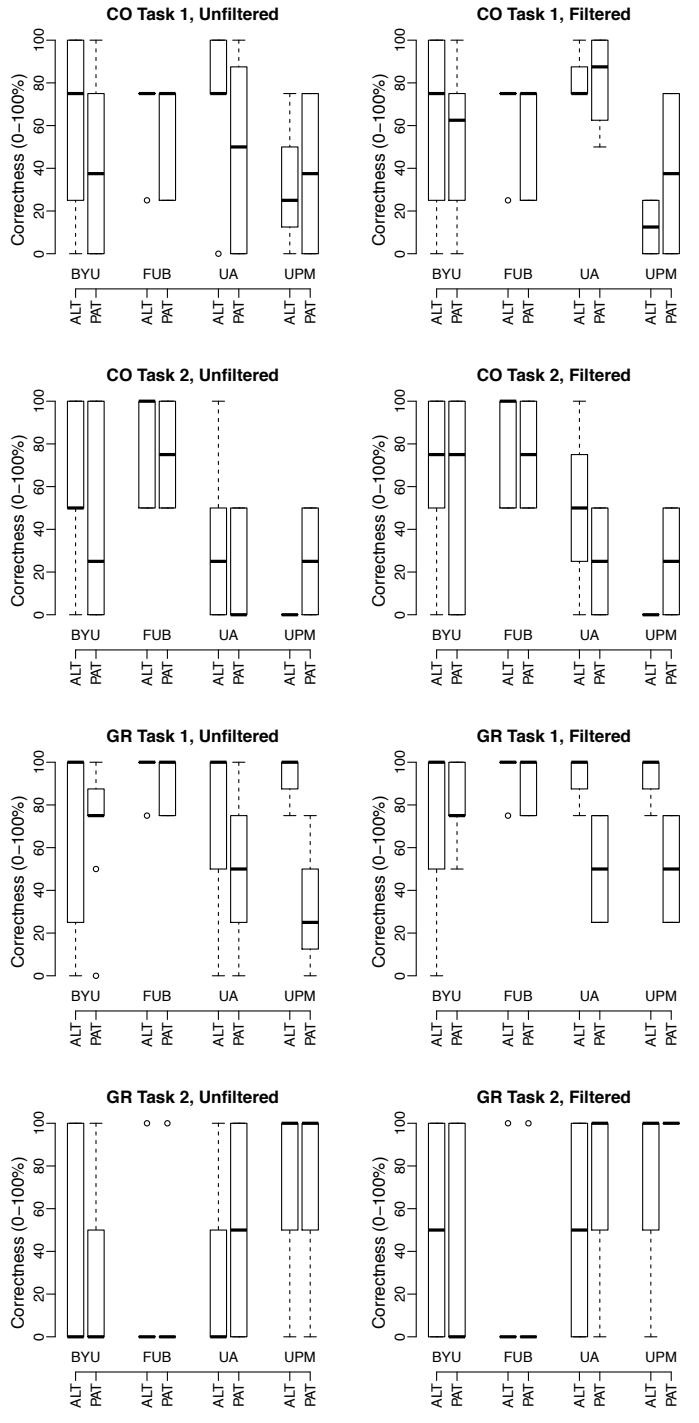


Fig. 9. Correctness data, showing ALT versus PAT displayed by site. Max whisker range is 1.5 IQR.

and below which a developer will fail so badly that design patterns have no measurable impact. We depict this threshold in Fig. 10, incorporated into the moderating effect of developer experience.

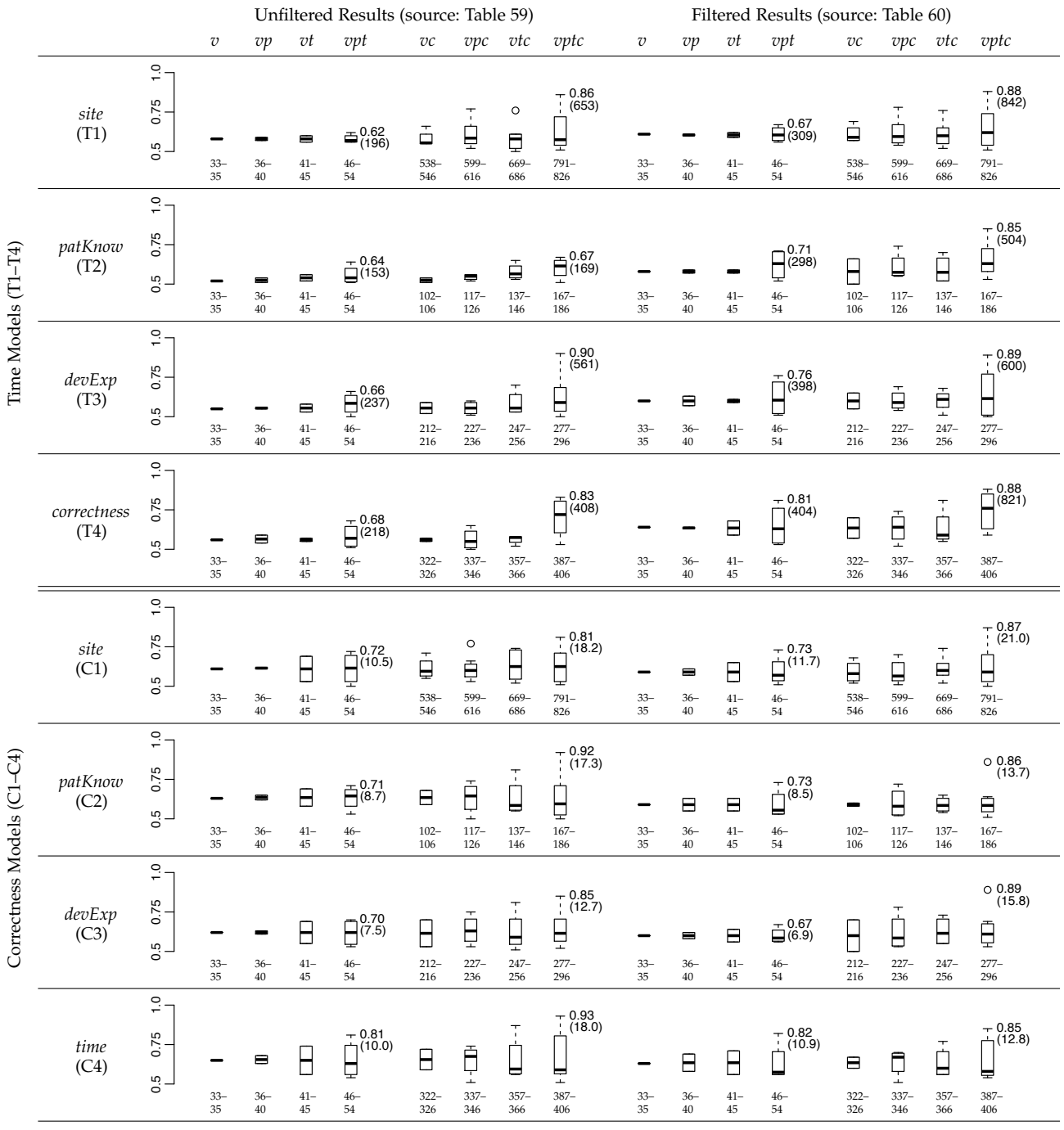
Interestingly, E_orig and E_repl both found *variant* to be more significant for the CO program than for GR. For example, E_orig obtained p-values < 0.001 for CO, but all p-values for GR were in the range 0.02–0.17. Thus, the filtering—which is explained in part by task difficulty—brings E_joint's results into greater alignment with the prior two PatMain studies.

S.2 *correctness* and *time*

Table 18 shows the frequentst results for *correctness* and *time*.[21] As covariates for one another, both *correctness* and *time* are clearly meaningful irrespective of filtering, especially for the GR program. The *correctness-time* correlation is estimated to be positive in all models—i.e., the participants are achieving higher scores at the expense of time. For example, the unfiltered GR_time model indicates that a 10-point increase

21. The relationship between *correctness* and *time*, as well as the *correctness* × *variant* interaction, are also discussed in Appendix P.

TABLE 17
Bayesian interaction results—moderator assessment. See Appendix R for a description of how to read and interpret this table.



in correctness corresponds with a 5.9% increase in time (80% CI: 4–8). The other frequentist models show similar results (see Appendix Y).

The most likely variable to account for a *positive* correlation between *correctness* and *time* is *motivation*. As previously discussed, some participants were significantly more motivated than others. Particularly at FUB, the participants appear to have been willing to take longer in order to do a better job (see also the discussion of "perceived time limits" below). Thus, the frequentist results for *correctness* and *time* reinforce the importance of analyzing motivation as a potential moderator in future design pattern studies.

Table 19 shows the Bayesian interaction results for *correctness* and *time*. Prior to filtering, the *correctness* and *time* interactions appear significant, especially in the *time* models. After filtering, however, the interactions are largely eliminated. Since filtering mitigates the interactions, we anticipate that the interactions were primarily due to the fact that *variant* has little impact on underqualified and/or undermotivated participants (as discussed in the prior subsection on "task difficulty"). Thus, upon removing those participants, the interaction disappears and general significance for the main effect increases. Overall, these results support the need to filter underqualified and undermotivated participants, as

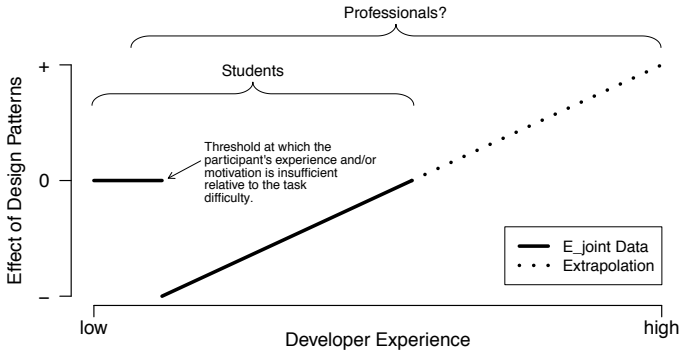


Fig. 10. Relative impact that developer experience has on the effect of design patterns, taking into account the co-moderating influence of task difficulty. Since the graph depends on a specific level of task difficulty, the axes are depicted as relative scales. A positive effect for design patterns (+) means that the patterns lead to lower work times and higher quality solutions.

TABLE 18
Frequentist model p-values for *correctness* and *time*. p-values less than or equal to 0.05 are bolded.

Model	Covariate	Unfiltered	Filtered
CO_time	correctness	0.071	0.104
CO_correctness	time	0.110	NS
GR_time	correctness	**<0.001**	**<0.001**
GR_correctness	time	**0.003**	**<0.001**

NS = not significant—i.e., the exact value is not available since the variable was removed during model tuning due to lack of significance. For a description of model tuning, see Appendix M.

TABLE 19
Bayesian interaction results—moderator assessment (excerpt from Table 5 in Section 3.2; included here for convenience).

time Models	Unfiltered ¬*c*	Unfiltered *c*	Filtered ¬*c*	Filtered *c*	*correctness* Models	Unfiltered ¬*c*	Unfiltered *c*	Filtered ¬*c*	Filtered *c*
correctness	0.68	**0.83**	**0.81**	**0.88**	*time*	**0.81**	**0.93**	**0.82**	**0.85**
(T4)	218	408	404	821	(C4)	10.0	18.0	10.9	12.8

was done in E_repl. They also represent indirect evidence that motivation could moderate the effect of *variant*, since the filtering targeted (in part) undermotivated participants.

S.3 Program Order

Table 20 shows the effect of program *order* in the frequentist and Bayesian models. For the *time* models, *order* is statistically significant, with effect estimates in the range 2.4–7.5 minutes. An effect size of 5 minutes could be meaningful given that each task required only 20–30 minutes to complete. For the *correctness* models, *order* is less significant, and its effect is fairly small (2.3–4.5 percentage points). Based on these results, we conclude that *the participants spent significantly less time, and possibly scored slightly higher, on whichever program was second*. Since performance tended to improve on the second program, the most likely explanation is a learning effect.

Interestingly, both E_orig and E_repl found *order* to be *in*significant. Since E_joint used the same basic design, the experiment proper likely did not cause the learning effect. Instead, we believe the effect is due to the initial setting up

TABLE 20
Frequentist and Bayesian results showing the significance and effect of program *order*. p-values less than or equal to 0.05 and posterior probabilities exceeding 0.75 are bolded. All p-values are two-sided.

	Unfiltered		Filtered	
Model	*sig.*	1st−2nd	*sig.*	1st−2nd
CO_time	**0.007**	296	**0.004**	306
CO_correctness	NS	-	NS	-
GR_time	**0.002**	452	**0.038**	325
GR_correctness	NS	-	NS	-
T1	**0.994**	189	**0.964**	141
T2	**0.998**	228	**0.978**	175
T3	**0.998**	223	**0.981**	180
T4	**0.996**	214	**0.970**	164
T5	**0.996**	210	**0.970**	157
T6	**0.995**	189	**0.967**	142
C1	**0.865**	−3.4	**0.874**	−3.8
C2	**0.827**	−3.1	**0.850**	−3.6
C3	**0.852**	−3.5	**0.883**	−4.5
C4	**0.781**	−2.3	**0.852**	−3.6
C5	**0.799**	−2.6	**0.898**	−4.3
C6	**0.803**	−2.7	**0.812**	−3.1

sig. = significance of *order*—i.e. p-values for the frequentist models (first four rows), posterior probabilities for the Bayesian models (source: *u*T1–T6,C1–C6:11–13 and *f*T1–T6,C1–C6:11–13).
1st−2nd = estimated difference between program orderings (in seconds or percentage points).
NS = not significant—i.e., the exact value is not available since the variable was removed during model tuning due to lack of significance. For a description of model tuning, see Appendix M.

of development environments. E_joint's participants were given only two things: task instructions and source code. They had to set up their own development environments, including importing the source code. The effort required to setup IDEs could certainly account for extra time on the first program—especially since E_joint's participants were students, who may not have known prior to the experiment how to import existing code into an IDE.

Conversely, E_orig's participants completed tasks on paper, and E_repl's participants were given a standardized environment with the programs already set up as development projects. Moreover, "[a]ll [E_repl] subjects performed an initial, familiarization task in order to try out the programming environment and the user interface" [20, p. 188]. Thus, environment setup was not a significant factor for either E_orig or E_repl.

In general, program *order* is not directly relevant to design patterns, nor to industrial software development. To eliminate it in future studies, we recommend administering a pre-task, as was done in E_repl. In the case of E_joint, we statistically correct for the effect by including *order* as a covariate in all models.

S.4 Perceived Time Limits

The BYU participants appear constrained on the time they were willing or able to spend on the experiment, relative to the participants at the other three sites (see Fig. 11). Also, the FUB participants took longer (on average) than the participants at any other site—in particular, more than twice

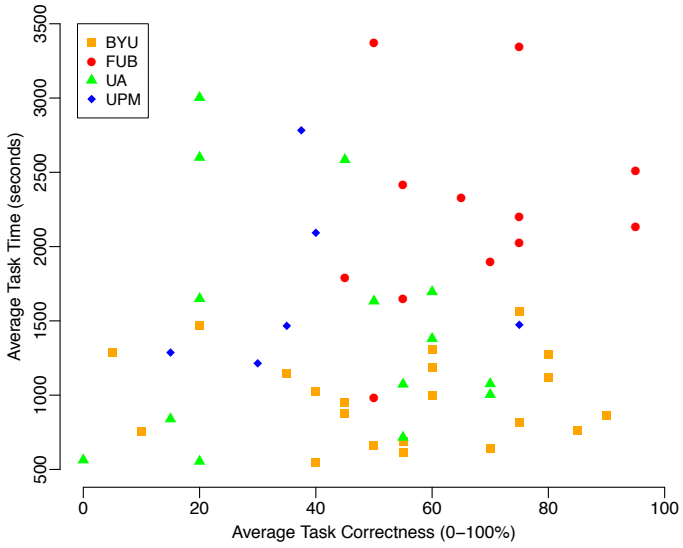


Fig. 11. Participants plotted by average task time and correctness, categorized by site (repeat of Fig. 7b from Appendix Q; included here for convenience).

TABLE 21
Average task time and correctness for each participant (same data as that shown in Fig. 11). The count of participants for each column is given in parentheses. For each box plot, the median value is labeled, the mean is shown as a black dot, and the max whisker range is 1.5 IQR. All p-values are two-sided; p-values less than or equal to 0.05 are bolded; p-values are based on the Kruskal-Wallis rank sum test comparing sample medians across sites (calculated with R 2.15.2).

	BYU (21)	FUB (12)	UA (14)	UPM (6)	All (53)	*sig.*
Average Task Time (seconds)	951	2164	1229	1470	1286	**<0.001**
Average Task Correctness (0–100%)	55.0	67.5	47.5	36.2	55.0	**0.018**

BYU = Brigham Young University
FUB = Freie Universität Berlin
UA = The University of Alabama
UPM = Universidad Politécnica de Madrid
sig. = significance (two-sided p-value)

as long as the BYU participants (see Table 21). These trends could be due to variations in the participants' perception of time requirements.

The BYU participants, though volunteers, were primarily motivated by course credit. As students, they were required to complete 7 hours per week of software engineering work. The students were permitted to count any time spent on the experiment toward their weekly 7 hours. However, a former teaching assistant indicated that students typically struggle to fill the 7 hours. In fact, the class from which the participants were solicited reported an average of only 6.6 hours per week. If BYU participants perceived a time constraint, it was not likely due to the course requirements.

Another possible explanation is that the BYU researchers unintentionally communicated a time limit to their participants. One BYU participant (31563) clearly mentioned a perceived time limit when describing the difficult aspects of the CO tasks: "Figuring out what all is going on in the alloted time left, since I spent all of my 2 hours download and installing Eclipse [*sic*]." Incidentally, we excluded this participant's data from analysis, as described in Appendix F.

At BYU, the participants were told that the experiment would require 2–3 hours of their time. This information was given as an estimate to help participants plan for the experiment. The estimate was provided verbally at the time of solicitation and in print on the BYU-specific instruction sheet for volunteers (a copy of which is included in the lab package). The verbal instructions were not recorded, but the instruction sheet stated, "Required 2–3 Hours." In hindsight, the term "Required" was a poor choice, since it can be ambiguously understood as implying a time limit. Also, a better estimate may have been 2–4 hours.

Concerning FUB's high times, Lutz Prechelt recalls:

> The FUB subjects were true volunteers. Martin Liesenberg produced a number of mishaps while he tried to get the portal to run and had to put off the subjects I think twice. Some of them disappeared. So the remaining ones were likely seriously interested in participating and not just doing minimal course duty. Also, I think they felt they were helping a fellow student (whose bachelor thesis involved needing participants for the experiment). Whether Martin ever uttered any specific expectations for the time required I cannot say. I found no email of his with such content, but he may not have copied me on that or may have said something orally. (email, Oct. 9, 2012)

Thus the FUB participants may have perceived a minimum time requirement, which may have led to their taking more time on average. However, based on the data, we can at least conclude that a minimum time limit was not prominently stressed. Thus it seems unlikely that all of the FUB participants would have perceived such a requirement.

Intrinsic motivation is the more likely explanation for FUB's high times, especially considering that the BYU and UA participants—who took the least time on the experiment (see Table 21)—were primarily extrinsically motivated (by course credit). Moreover, the UPM participants, whose times fall in the middle, did not receive course credit, but were solicited from a research lab. Having prior relationships with their experimenters (who were their graduate advisors), they likely felt greater social pressure to do a good job than did the participants at BYU and UA who did not previously know their experimenters. Also, we find no evidence that the UPM participants were particularly intrinsically motivated, thus explaining why their times were not as high as those at FUB.

In general, the perception of time limits is a relevant variable in industry settings. For instance, if a developer feels short on time, s/he may avoid examining a program's overall structure and, therefore, miss some of the benefits of a given design pattern. However, our data are insufficient to test time perception as a moderator. We recommend that future studies more carefully control and report on this variable.

S.5 Cultural (or Regional) Variation

During data analysis, we found several cases in which cultural (or regional) variation may have impacted the results. First, we discovered that the question for CO task 2 is slightly ambiguous, such that it resulted in two different interpretations. The question stated:

> For a given object `CommChannel channel`, the statement `channel.reset()` may produce the result `CommChannel.IMPOSSIBLE`. Under which condition does this result occur?

The intended meaning is that the first sentence provides a context in which the actual question (second sentence) is to be answered. The anticipated answer is thus: `CLOSED` channel or `FAILED` encrypted channel. However, the question can also be read such that the first sentence is merely an example, and "this result" refers to *all* cases of `IMPOSSIBLE` that can occur:

- `reset()` called on a `CLOSED` channel,
- `reset()` called on a `FAILED` encrypted channel,
- `close()` called on a `CLOSED` channel,
- `basicOpen()` called on a non-`CLOSED` channel.

Notice that the second interpretation is a superset of the first. Nine of the twelve FUB participants clearly provided the extra information, whereas none of the participants from any other site did so.[22] In Lutz Prechelt's words, "If it is indeed so that none of the non-FUB participants used this second interpretation, I would consider this a fascinating example of a subtle cultural dependency" (email, Aug. 25, 2011).[23]

A second example of a possible cultural dependency concerns self-reporting on the pre-questionnaires. As discussed in Appendix C, all of the extreme outliers for the lines of code questions came from an American school (BYU or UA), and most of those came from BYU. The BYU participants also reported substantially higher numbers of languages used. In this latter case, the participants listed the actual languages by name, so inflating the count was not likely due to disinterest with the question. However, BYU participants may not have actually used more languages. Their threshold for counting a language may simply have been lower.

We discuss additional cultural variables in the following subsections, some of which are clearly relevant to industrial contexts. However, they may or may not interact with design patterns. We recommend future studies to report such variables as much as reasonably possible. More work is needed to identify cultural variables, as well as to formulate methods for controlling them, both within and across studies.

S.6 IDE Preferences

Another cultural or regionally-influenced variable concerns programming environments.[24] On the source code download pages we provided the following instructions:

> Download the .zip file and import it into your workspace. In Eclipse this is done via: File - Import - General - Import existing project into workspace. Now select 'select archive file' and click 'Finish'.

The IDE instructions were meant to aid inexperienced participants. We did not require the use of Eclipse, nor was the code specific to any particular programming environment. However, in response to these instructions, 4 of the 21 BYU participants mentioned problems with and/or complained about having to use the Eclipse IDE. Several UPM participants also complained about Eclipse, as though it were a requirement, and half of the UPM participants complained about the difficulty of importing the CO and GR programs into the NetBeans IDE, as though the code was configured specifically for Eclipse (which it was not). Conversely, no participants at FUB or UA expressed these concerns or mentioned NetBeans.

Apparently, our advice impacted the sites unevenly. The imbalance could be due to a regional language effect, in which only participants at BYU and UPM mistakenly thought Eclipse was required. However, since BYU and UPM do not share native languages (English vs. Spanish), and neither do FUB and UA (German vs. English), the more likely explanation is regional preference for programming environments. Quite possibly, FUB and UA participants already preferred Eclipse, so any misperceived requirement to use Eclipse was simply not viewed as a problem.

In general, we believe participants should be encouraged to work in their native environments. First, standardizing the environment for a specific experiment does not solve the problem of generalizability, since the programming environment may still vary across studies. Second, when experiments involve small-scale tasks, the time spent learning an unfamiliar environment could mask the main effect. Third, allowing participants to use their preferred environments increases the realism of the experiment.

However, if we allow participants to work in their preferred environments, we must either make no mention of development tools, or we must clearly instruct the participants to use the tools to which they are most accustomed. Also, providing project import instructions is helpful, but if given these instructions must include all relevant IDEs and development frameworks. Otherwise, only some participants are benefited, and as we find in E_joint, that bias is not likely to be random with respect to sites or studies.

S.7 Language Barriers

This variable is related to the issue of cultural variation. In fact, the first example of cultural variation discussed above (concerning two interpretations for CO task 2) is directly related to language issues.

In E_joint, we administered all instruments in English.[25] However, two participants mentioned problems understanding the English text. UPM participant 38048 commented, "It was quite more easy for me to perform all the task if the instructions was in spanish [*sic*]." Similarly, FUB participant

22. One participant's response from BYU (64084) could, possibly, be construed as having followed the second interpretation, but the answer is so incomplete that it could just as well have been an incorrect response to the first interpretation.

23. The second interpretation was not penalized in the grading. The extra information was simply ignored. Additionally, it may have taken longer to answer the question according to the second interpretation, which would create additional variance. However, we find no evidence that the choice of interpretation interacted with *variant*. Thus, the final results should not be biased.

24. For a related discussion, see Section 4.4 of E_repl's published report [20, pp. 163–164].

25. E_orig administered German text to German participants, and concerning E_repl, Marek Vokáč comments, "The task descriptions the subjects got were in Norwegian to lessen the language friction" (email, Oct. 14, 2012).

71173 commented, "[T]he task descriptions in English created some problems for me." Quite possibly, other participants were also hindered by the English instructions, but they did not notice the effect or viewed it as not worth mentioning. Concerning FUB, Lutz Prechelt comments:

> I expect the English text will have slowed down the FUB participants somewhat (and some more than others, creating some additional variance), but should not have distorted the PAT/ALT effect. Further, the FUB participants all know the pattern terminology mostly in its English form, rather than German. (email, Oct. 9, 2012)

Another example of language barriers impacting experimental measurements involves the student status question. The choices provided for the student status question were (in order): undergraduate student, graduate student, postgraduate student, not a student. In hindsight, *postdoctoral* may have been a better term instead of *postgraduate* since, in some countries, the designation *postgraduate* is synonymous with and preferred over that of *graduate*. Interestingly, the UPM participants, who were all master's students, uniformly selected *postgraduate*. Conversely, all of the master's and PhD students at the other three sites—including those at FUB, a German university—selected *graduate*.

Thus, the participants' native languages impacted measurements in E_joint, and that impact was at least partially contingent on *site*. We do not anticipate that language would directly influence the effect of patterns, but it may at least add sufficient variance to mask the effect in an experiment or to muddle results across studies.

Specific to replication, language poses a particular challenge because we often need to maintain instruments across sites and studies (especially in the early stages of investigation). On the one hand, ensuring that various translations convey the same meaning is difficult. On the other hand, to administer experiments in a single language introduces the potential for misinterpretation by non-native readers. At this point, it is not clear to us which approach is best.

S.8 Clarity of Task Instructions

This variable relates to the issue of language barriers (discussed previously). In post-questionnaire comments, 17 of 53 participants (almost 1/3) expressed difficulty understanding the task instructions (see Table 22). The prevalence of these complaints is worth examining because the same tasks were used by E_orig and E_repl, but in neither of those studies did the participants complain to such a degree.

Perhaps the problem is due to our use of student participants, as opposed to the professionals of the prior two studies. In this case, it may be that students have more trouble due to lack of experience, and they manifest that trouble by complaining about the instructions. Alternatively, students may simply complain more than professionals when faced with similar frustrations. Or, perhaps the professionals also complained, but that information has since been lost.

According to Lutz Prechelt, principal investigator for E_orig, the third hypothesis is very unlikely, and indeed, we find no evidence to support that hypothesis in either the published reports or the datasets. More importantly, all three hypotheses are inconsistent with the fact that *all of E_joint's participants were students, and yet the complaints were imbalanced across sites*. As a percentage, the complaints were much more prevalent at BYU and UPM than at FUB and UA (see Table 22). Further, the complaints do not correspond with student status—BYU and FUB participants were mostly undergraduates, whereas UA and UPM participants were entirely graduate students (as shown in Table 8 in Appendix B). As discussed in Appendix C, the participants were also fairly homogeneous with respect to developer experience. Concerning pattern knowledge, the FUB and UA participants do report broader overall exposure than the participants at BYU and UPM. However, the UA participants also report greater pattern knowledge than those at FUB, which does not seem consistent with the results in Table 22.

TABLE 22
Prevalence of participants at each site who complained that the task instructions were difficult to understand.

Site	Complaints	Total Participants	Percentage
BYU	10	21	48%
FUB	0	12	0%
UA	3	14	21%
UPM	4	6	67%
Total	17	53	32%

BYU = Brigham Young University
FUB = Freie Universität Berlin
UA = The University of Alabama
UPM = Universidad Politécnica de Madrid

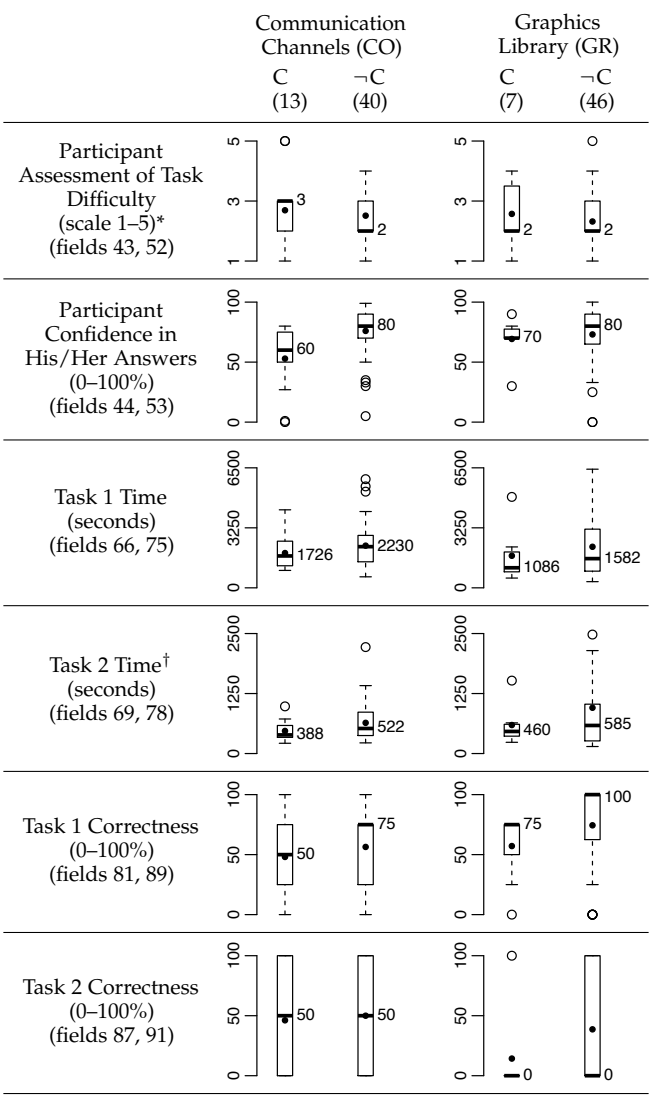
Another possibility is that the translation (from German to English) muddled the task instructions. We find some evidence for this hypothesis in that FUB provided the translation, and accordingly, none of the FUB participants complained. However, the complaints also do not appear to have been entirely contingent on native language, since the prevalence of complaints differed between the two English-speaking schools, BYU and UA. On the other hand, UA does report a significant portion of their participants as being international students (8/18; see Table 8 in Appendix B).

Table 23 indicates that the participants who complained performed marginally different from those that did not complain. On average, they worked a little more quickly and scored a bit lower. They also showed less confidence in their answers and viewed the tasks as slightly more difficult. Given the imbalance in complaints across sites, as well as the data in Table 23, the task descriptions likely account for at least a small portion of the cross-site variance—which means they likely inhibit isolation of the main effect. However, since the task descriptions were the same for ALT and PAT, they should not have biased the conclusions.

S.9 Compilation/Testing Expectations

E_orig was administered on paper, such that compilation was not a concern. Accordingly, some of the function bodies were replaced by the comment, "Body doesn't matter." When the code was translated to Java for E_joint, these stubs were maintained. Thus some of the Java methods contained only a comment, with no return statement, causing the code to not compile. On one hand, maintaining the stubs without modification minimizes differences between the replication

TABLE 23
Comparison of participants—those who complained that the task instructions were difficult to understand versus those who did not. The count of participants for each column is given in parentheses. For each box plot, the median value is labeled, the mean is shown as a black dot, and the max whisker range is 1.5 IQR. For a description of each variable, identified by field number, see Appendix G.



C = Complained
¬C = Did not complain
*1=low difficulty, 5=high.
†Two outliers for the GR program not shown: ¬C=7239 and ¬C=5842.

and the original study. However, the fact that the code did not compile appears to have caused problems for some participants—e.g., participant 74027 commented, "it didn't compile with my version of java, which caused problems because my ide did not work properly."[26]

Additionally, the code did not support full execution by default for either the CO or GR programs. The CO program simply contained no main method by which to execute the code. For GR, a "testrun" class was provided, but it worked for only one of the two initial output modes of the program. Several participants complained about not being able to execute the code. For example, in response to the statement, "I found these to be the most difficult aspects of the task," participant 26851 responded, "Not being able to run something to weed out stupid mistakes."

The problem is that many developers write software by testing as they go. When the code is not compilable by default, participants either have to spend time fixing it—which some participants did, although it was not the assigned task—or they have to interact with the task in an abnormal way. Further, as the graders explained, "Sometimes the participants can compile, sometimes they can't. Sometimes they can partially execute, sometimes they can't. Sends mixed messages, mixed expectations." Thus the problem was not just that the participants could not compile by default, but more particularly, we were unclear with them concerning our compilation and testing expectations.

Compilation and testing expectations can certainly be unclear in the real world. However, given that development practices can vary regionally (e.g., in response to where one was educated), compilation and testing issues likely affect sites unevenly, which means they induce extraneous variance unrelated to the question of interest. Therefore, in future PatMain studies we recommend eliminating compilation errors (as was done in E_repl[27]) even though doing so means modifying the original source code. It may also be worthwhile to clarify testing expectations and/or to systematically explore the use of testing to see if it influences the outcome of the PatMain experiment.

APPENDIX T
COMPARISON OF STATISTICAL METHODS

In this section, we compare statistical methods across the three PatMain studies. In particular, we show that the methods are sufficiently similar such that we can directly compare the numerical results. This discussion is particularly relevant to Table 7 in Section 3.3.

T.1 E_orig

To analyze the *time* response variable, E_orig used *analysis of variance* (ANOVA) and *nonparametric* (distribution-free) *bootstrap* methods. For *correctness*, E_orig simply compared the counts of solutions with errors. As Prechelt *et al.* explain, "For many tasks, all groups achieved near-perfect correctness" [10, p. 1136].

ANOVA was used for preliminary analysis to determine which variables best explain *time*. However, ANOVA is subject to normality assumptions, and the *time* data were skewed. Thus the authors avoided ANOVA for the final results. The authors also avoided Kruskal-Wallis and Wilcoxon (both of which are based on rank) because they wanted their results to summarize means, rather than medians. Bootstrap methods were a reasonable nonparametric alternative.

26. E_joint's grading rubric included compilation as one of the criteria. However, that particular criterion was applied only to the code the participants wrote and/or modified. This was accomplished by extracting the modified portions of code and placing them in a framework for testing. Thus the participants were not penalized for non-compiling code that they did not write.

27. Of E_repl, Marek Vokáč states, "The C++ code was as in Prechelt's original, the only changes were those needed to make it compile, plus introduction of a small library to give a console user interface. This library had only a few functions and played no significant role in the code" (email, Oct. 16, 2012).

Bootstrapping is a resampling technique [41]. In E_orig, the authors randomly resampled their data 10,000 times with replacement (making sure to preserve group cardinalities). To compare two groups, they calculated the mean difference between the groups within each resampling. This process resulted in a distribution of 10,000 difference estimates for each test statistic. It was from these distributions that the authors essentially "read off" their p-values.

T.2 E_repl

E_repl's analysis was more complex than E_orig's. The authors modeled both *time* and *correctness*, as well as addressed an additional statistical concern besides non-normality—non-independence of the data. This latter concern existed as well with E_orig, but was not accounted for in that analysis.

Non-independence occurs when subsets of the data correlate in response to some other variable. In the case of PatMain, the experiment protocol requires each participant to complete multiple tasks—thus observations cluster around individual participants. Ideally, participant effects should be factored out to reduce variance. Statistical procedures that account for these blocking (i.e., grouping) variables are more precise, in that they can accurately reduce error estimates. Lower error terms mean lower p-values and tighter confidence intervals. Consequently, E_orig's analysis was not incorrect, just less efficient. Where statistical significance was not obtained, accounting for non-independence may have yielded a significant result; otherwise, the results would not have changed much.

To account for non-normality and non-independence, E_repl used *generalized estimating equations* (GEE), which is a specialized form of *generalized linear models* (GLM). First, note that GLM is a generalization of linear regression, and by extension, of ANOVA. Consequently, GLM and GEE are both related to some of the methods used in E_orig. The benefit of GLM is that it allows the researcher to specify non-Gaussian distributional assumptions, thereby obviating the need for nonparametric methods. GEE further adds to GLM support for modeling blocking variables.

The authors modeled *time* with a gamma distribution and *correctness* with a normal distribution. The gamma distribution is ideal for skewed data, for which the range is strictly greater than zero. The authors also applied a log transformation to both *time* and *correctness*. Justifications for the transformations were not given in the paper. However, log transformations are typically used to normalize skewed data.

T.3 E_joint

For E_joint, we used both frequentist and Bayesian statistics. However, we do not discuss the Bayesian methods here because neither E_orig, nor E_repl performed a comparable analysis. Concerning the frequentist methods, we used *linear mixed models*, which are an extension of ANOVA and linear regression. These methods add support for modeling blocking variables and fitting non-Gaussian distributions. In these two respects, mixed models are similar to GEE.

Including a variable in a model does not achieve the same effect as blocking on that variable. For standard variables the data are assumed to be independent, whereas for blocking variables, the analysis specifically estimates and accounts for sub-correlations. In mixed models analysis, blocking variables are modeled as *random effects*. Accordingly, we model participant ID as a random effect in E_joint.

We also address the concern of non-normality for both *time* and *correctness*. For *time*, we apply a log transformation (like the analysis of E_repl), which effectively normalizes the observations. Accordingly, a normal distribution is an appropriate model for our data. Further, the range of *correctness* is discretized into only five buckets. As a result, it cannot take on much of a skew. Therefore, unlike E_repl, we do not apply a log transformation to *correctness*. That said, with only five buckets, the *correctness* variable is unlikely to fit a smooth normal distribution. Thus, although we assume normality for the frequentist analysis, we avoid that assumption entirely in the Bayesian analysis by modeling *correctness* with a beta distribution.

In summary, our methods are most similar to those of E_repl, particularly with respect to statistical efficiency. However, E_orig's methods are a bit more straightforward, which can be helpful in the early stages of investigation. Nevertheless, straightforward statistics come at a price. Although bootstrapping is unhampered by distributional assumptions, it sacrifices statistical power. Also, bootstrapping and ANOVA do not account for blocking variables. *Thus, we would expect little or no change in the results were E_repl and E_joint to swap statistical methods. However, using either GEE or mixed models in E_orig's analysis may increase statistical significance in some cases.*[28]

Appendix U
Additional Results Data

In Table 7 of Section 3.3, we provide a comparison of results across the three PatMain studies. For that comparison, we define concrete hypotheses based on the original hypothesis statements. Given the variables involved, twelve potential concrete hypotheses can be made for each task. For two of the tasks (CO and GR task 2), the original hypothesis statements do not address all of the possible combinations for concrete hypotheses. In this section, we provide results for the remaining combinations, which may be helpful for future meta-analysis (see Table 24).

Appendix V
Researcher Interactions

In this section, we describe interactions between replicating researchers and prior experimenters. We include this information per Carver's guidelines for reporting experimental replications [42]. This information is important for assessing shared bias between experiments.

V.1 E_repl

The published report from E_repl does not specifically mention interactions with E_orig's experimenters. However, the replication clearly reuses E_orig's artifacts, and the report states that Walter Tichy taught the patterns course for both

28. In fact, as part of their study, E_repl reanalyzed E_orig's data using GEE methods. In general, the p-values either did not change much or they decreased, as expected.

TABLE 24
Comparison of results across the three PatMain studies (supplement to Table 7 in Section 3.3). Shows results for the remaining combinations of concrete hypotheses not addressed by the original hypothesis statements.

Hypothesis Statement	Concrete Hypotheses	Baseline & Expectation	E_orig	Reanalysis of E_orig	E_repl	E_joint
CO Task 2, Comprehension	$t : H\ ?\ L$	L ?	+23%	-	-	−36% (.090)*
The PAT groups will take longer and commit more errors.	$t : PH\ ?\ PL$	PL ?	+6%	−5% (>.05)	−35% (>.05)	−61% (.007)
	$t : AH\ ?\ AL$	AL ?	+34%	+40% (>.05)	+28% (>.05)	−5% (.866)
	$c : H\ ?\ L$	L ?	-	-	-	+39 pp (.049)
	$c : PH\ ?\ PL$	PL ?	-	+14 pp (>.05)	+1 pp (>.05)	INS
	$c : AH\ ?\ AL$	AL ?	-	−9 pp (>.05)	+15 pp (<.05)	
GR Task 2, Comprehension	$c : H\ ?\ L$	L ?	-	-	-	NS
ALT and PAT will not significantly differ; the task will require less time at higher levels of pattern knowledge for both variants.	$c : PH\ ?\ PL$	PL ?	-	−10 pp (>.05)	−18 pp (>.05)	INS
	$c : AH\ ?\ AL$	AL ?	-	−5 pp (>.05)	+15 pp (>.05)	

*This value is taken from the CO_time model (filtered), as defined in Section 2.3.2, but with all interactions dropped.
? = hypothesis/expectation is undefined.

E_orig and E_repl. Further, we know from email archives that Vokáč, Sjøberg, Tichy, Unger, and Prechelt were actively discussing the replication in 2002. Their discussions included whether to let participants use personal laptops, whether to incorporate pair programming, and whether to conduct a qualitative analysis. Lutz Prechelt further recalls that those interactions began as early as 1999, and in his words, "There was quite a bit of interaction overall."

V.2 E_joint

E_joint involved four types of interaction:

1) *Interactions between E_joint organizers and prior experimenters during replication design.* During this phase we did not interact with E_repl's researchers. However, Lutz Prechelt—principal investigator for E_orig—designed E_joint's protocol.
2) *Interactions between E_joint organizers and the individual research teams.* To facilitate this interaction, we created a project website [23] and mailing list. Discussions on the mailing list were open to all research teams and concerned operation of the web portal, as well as clarifications about information posted on the project website.
3) *Interactions between the individual research teams and prior experimenters.* The research teams did not directly interact with prior experimenters, except for mailing list discussions involving Lutz Prechelt. Lutz answered many questions about the experiment and web portal.
4) *Interactions between E_joint organizers and prior experimenters during joint analysis.* Our interactions with E_orig were conducted entirely through Lutz Prechelt, who participated throughout the analysis process. During analysis, Jonathan Krein also communicated with Marek Vokáč, E_repl's principal investigator. Since this communication occurred late in the analysis, the only procedure impacted was data filtering (Section 3.1). We were already considering filtering, but Marek's feedback reinforced that intuition. Note that all other mentions of Marek in the paper reflect data we incorporated after the fact (i.e., during the drafting of the report).

V.3 Summary

We see strong indications of shared bias (which is not necessarily a bad thing) between E_orig and the two PatMain replications (E_repl and E_joint). However, the level of interaction appears to be more significant between E_orig and E_joint, since E_orig's principal investigator, Lutz Prechelt, was also a primary designer and collaborator for E_joint.

APPENDIX W
THREATS TO VALIDITY CONTINUED

The material in this section is a continuation of Section 4. In this section, we discuss minor threats to validity.

W.1 Construct Validity

The possibility exists that self-reporting on the pre-questionnaires was inaccurate, biased, or somehow inconsistent across sites—thus affecting operationalization of developer experience and/or pattern knowledge. For example, cultural trends may influence what a person considers to be "professional" experience. In fact, we found evidence that the BYU participants tended to exaggerate when reporting LOC written much more so than the FUB and UPM participants. For a discussion of this issue, see Appendix C. To mitigate inaccuracies, we collected several related metrics for developer experience and pattern knowledge. We then pruned away the least promising metrics and aggregated the rest. For an example with developer experience, see Appendix J.

W.2 Conclusion Validity

Concerning statistical assumptions, the one assumption we could not fully ensure for our models is that of homoscedasticity. In the case of the *patKnow* explanatory variable, our sample is thin in the upper range, such that we could not confirm constant variance, although the variance may indeed be constant. As a result, p-values for E_joint, which concern the high range of *patKnow*, may be overestimated—i.e., the p-values may be biased toward type 2 errors or failure

to reject the null hypothesis. For further details on model assumptions, see Appendix L.

The results of the Bayesian analysis are *not* likely to be biased by the selection of prior distributions. First, we enlisted a third party helper (who was not previously affiliated with the study) to estimate the priors using historical data from E_orig. Second, we intentionally chose broad priors so as to make them of little influence on the results.

Additionally, concerning the Bayesian analysis, in some cases only a few observations were available for estimating a particular parameter. In these cases, the minimal data do not mislead the models. The posterior distributions simply do not deviate much from the broad priors; thus the resulting probabilities are insignificant (i.e., near 0.5), as they should be. However, additional data could produce significance in these cases. Thus, the Bayesian analysis, by virtue of its broad priors, is biased toward type 2 errors, or failure to reject the null hypothesis. However, this bias is preferable because the Bayesian analysis is *post-hoc*, and *post-hoc* analyses are predisposed toward type 1 errors.

W.3 Internal Validity

A potential threat to internal validity is survivor bias. Survivor bias could occur in a software engineering experiment as follows: Condition A is in fact worse than B. It is even so much worse that most of the low-performers in the A group drop out of the experiment prematurely due to frustration. The rest of the A group thus appears much stronger than it should and the inferiority of A versus B disappears. Of the 61 E_joint participants to begin the experiment, 6 quit prematurely. For a list of the 6, see Appendix F. In general, we find no evidence that quitting correlates with any particular experiment group or site. Thus, quitting was most likely due to general disinterest with the experiment, or possibly due to frustration at some condition unrelated to design patterns.

In E_joint, we assessed pattern knowledge via a survey (i.e., an observational assessment) instead of a training course (i.e., a randomized, controlled assessment). Thus, causality inferences relating to pattern knowledge are tentative. However, since we synthesize our results with those of the prior two PatMain studies, this threat is not a significant concern for the final conclusions.

The experiment was translated from German into English. Unfortunately, several grammar errors occurred which were not corrected prior to the first site (FUB) administering the experiment. For the sake of consistency, we did not correct the text at the other three sites. Some participants noticed the errors, but were not bothered by them. We also pre-tested the framework at BYU on several students not affiliated with the study. The test participants had no problem understanding the correct meanings. Thus, we do not think the errors impacted the results of the study.

E_joint's programs were translated from C++ into Java. To minimize changes, we maintained the original structures as much as possible. However, this decision meant that the Java versions were written in a subtle C style. Possibly the style may have confused some participants. One participant (24085) did comment, "Those are not at all valid java codes. No experience programmer will write such code. That made understanding the code tough [*sic*]." In the worst case, the coding style could have slowed down some participants, creating additional variance, but we do not expect it to have systematically influenced the main effect.

All participants took the experiment in Java, but undoubtedly, not all participants had equal familiarity with Java (as would be the case for any language). If Java familiarity varied significantly within or across sites it may have added considerable noise to the results. To investigate this possibility, we developed a metric to measure Java familiarity. However, when added to the statistical models, we found the metric to have almost no impact on the results (as described in Appendix K). Additionally, 1) almost all of the participants (47 of 53) listed java as a language they use often; 2) at two of the sites (FUB and UPM) the participants all voluntarily chose Java (over C++ and C#); and 3) based on course curricula, we know that almost all of the BYU and UA participants had recently received formal training in Java. Thus, we do not believe that Java familiarity had a significant impact on the results of the experiment.

To ensure consistency, we had the same two people grade all tasks (as described in Section 2.2). For the coding tasks, the graders worked in a pair-programming style arrangement. However, for the short-answer tasks, the graders worked separately, each on half of the responses. Thus, the short-answer scores could be inconsistent across participants. In hindsight, we could have asked each grader to grade more than half of the participants and then used the overlap to compute an inter-rater reliability score. Having not done this, we note it here as a limitation and recommend it for future studies. That said, based on three factors, we believe the risk of inconsistency is low. First, unlike the coding tasks, the short-answer tasks involved fairly straightforward, unambiguous answers. Second, the graders initially graded several solutions together, from which they established both a grading rubric and an understanding of how that rubric was to be applied. Third, upon completing their work, the graders cross-checked each other's results specifically for the purpose of ensuring consistency.

Lastly, we observed a small learning (or maturation) effect in E_joint. However, we found the effect to be unrelated to the use of design patterns and were able to correct for it in the statistical analysis.

W.4 External Validity

Our use of a web portal may have impacted the participants' attitudes. Possibly the work-at-home nature of the portal led to reduced motivations or focus. In contrast, the participants in the prior two studies were required to work in dedicated rooms, which may have promoted a greater sense of seriousness about the experiment. Ultimately, both approaches have positives and negatives, inasmuch as they represent tradeoffs in experimental stress. Some developers work better under pressure, other do not. Thus, either setting could be viewed as more or less like industry. To an extent, having both represented in the final results strengthens external validity.

APPENDIX X
FUTURE WORK

Most of the ideas below focus on exploring variables that moderate the effect of design patterns. Ideally we can dis-

cover a handful of key variables sufficient to predict the outcome of pattern experiments in most contexts. Understanding the role of key moderators can also lead to the development of explanatory and predictive theories, which in turn may enable the formulation of transferable best practices.

X.1 PatMain Replications

The most obvious item for future work is to further replicate the PatMain experiment with additional controls and measurements for moderators. We recommend that the next round of investigation should focus on a few of the most promising variables. Controlling additional variables can help to reduce extraneous variance, but ideally we want to identify the smallest set of moderators possible, sufficient to predict experimental outcomes. We recommend the following moderators:

- *pattern knowledge*: Pattern knowledge has been shown in all three PatMain studies to moderate the effect of design patterns. However, the extent of its influence, particularly with respect to the Abstract Factory pattern in the GR program, is still not fully worked out.
- *developer experience*: Developer experience has also been considered in the prior PatMain studies, but its effect as a moderator has only been tested in E_joint; it needs to be further validated.
- *motivation*: Motivation is promising because it correlates with variance in E_joint, as well as with variance across the three PatMain studies. However, we were not able to directly test its interaction with design patterns in E_joint due to lack of the appropriate quantitative data.

Note that when investigating moderators, researchers should pay close attention to the level of heterogeneity in their samples. Insufficient heterogeneity can result in failure to detect a moderator, even in cases where the moderator is highly influential.

In addition to testing specific moderators, we also recommend that future replicators document other variables that appear to correlate with variance in their studies. Doing so will make those studies more useful in the future, especially if the variables identified above prove inadequate. Also, it may be worthwhile to explore relationships between moderators. For instance, can developer experience compensate for a lack of pattern knowledge (and vice versa)? Lastly, controlling variables within studies will not, by itself, solve the problem of generalizability. We also need to develop methods for mapping moderators across studies. This work could take the form of investigating best methods for assessing particular developer attributes and formulating standardized assessments.

X.2 PatMain Meta-analysis

Our analysis in this paper could be improved by statistically modeling all three PatMain studies together. A combined analysis is reasonable because the two replications both sought to closely duplicate the original. Thus, all three studies use nearly the same materials and comparable experimental designs. The resulting models would likely be similar in size to those we have already constructed, but the volume of data would nearly triple. For any attempt at such an analysis, note the following concerns:

1) E_joint used a survey to assess pattern knowledge instead of a training course. One approach to resolving this protocol difference would be to label all E_joint observations as "PRE" (meaning pre-training). Additionally, both of the prior PatMain studies collected demographic data on pattern knowledge. Those data could be used to create a unified pattern knowledge metric.
2) Three of E_joint's sites initially graded their own participants' solutions. However, for the joint analysis, we re-graded all solutions to ensure consistency. Correlating the centrally-graded scores with those of the individual sites revealed only marginal correspondence. For one task (at FUB) the correlation was perfect, but most correlations were in the range 0.25–0.75, and one was only 0.13.[29] Thus, care should be taken when comparing correctness scores between studies.
3) CO tasks 2 and 3 were combined in E_repl and E_joint, but not in E_orig. The combination involved adding times and averaging correctness scores [20, p. 179].
4) In E_repl, Vokáč *et al.* initially applied time corrections to adjust for participants who spent long periods resolving a technical nuance (e.g., finding a missing closing brace) [20, pp. 163–164]. However, they found the corrections to have little impact on the results and subsequently eliminated them from the analysis. When incorporating data from E_repl, time corrections should be ignored.
5) E_orig, E_repl, and one site from E_joint (UA) tested participants on the ST and BO programs. These programs could also be included in the combined analysis using the Bayesian methods shown in this paper, which allow for missing data.
6) Several variables, which we could not statistically model using E_joint data alone, could be investigated in the combined analysis. These variables include: student vs. professional status, paid vs. unpaid compensation, voluntary participation vs. participation by assignment, C++ vs. Java, and computer-based format vs. paper-based format.

X.3 Historical/Case Study Investigations of Moderators

In addition to further replicating the PatMain experiment, it would be interesting to review the design pattern literature for data on potential moderators. Many studies likely contain at least some traces of information on moderators, the synthesis of which may reveal significant insights. The results of such a literature review could be used to corroborate E_joint's findings, or even to generalize and extend them.

Another alternative would be industry case studies. It may be possible to find software projects involving maintenance tasks, wherein some parts of the software have been constructed with patterns and some parts without. We could then assess whether various moderators appear salient in practice. Such studies would only be observational, but in connection with experiments, they could help establish external validity. If enough is known about the identity of the developers, open source repositories may prove useful in this regard.

29. Pearson product-moment correlation coefficients.

X.4 Taxonomy of Interfering Variables

We have considered many different types of interfering variables in this paper. Some are inherent to the problem domain, whereas others are artifacts of the experimental setup. Some probably only influence overall variance, whereas others directly moderate the main effect. Some also overlap (e.g., site and culture), and many can be considered of one type or another, depending on one's frame of reference. As such, it may be beneficial to develop a structured understanding of interfering variables—e.g., to develop a taxonomy of variable types, along with methods for identifying and resolving each of the types. A literature review of interfering variables in software engineering experiments may be helpful in this regard. Also, it may be helpful to review how other disciplines deal with such variables.

X.5 Design Pattern Properties

According to Vokáč *et al.*, "each design pattern. . . has its own nature, so that it is not valid to characterize design patterns as useful or harmful in general" [20, p. 191]. If this conclusion is true, as our results suggest, then presumably some set of design pattern properties must exist (e.g., complexity), which if understood, could be used to better predict a pattern's impact on software maintenance. Via the PatMain studies, we are in the process of directly investigating several patterns. However, if we could understand the properties on which the usefulness of patterns depend, we could potentially predict outcomes for new and untested patterns.

For example, we find that the threshold of experience required for Abstract Factory to be beneficial during maintenance is greater than that required for Decorator. Similarly, the Visitor pattern (which was not tested in E_joint) was found by E_repl to be especially problematic, more so than both Decorator and Abstract Factory. Possibly pattern complexity explains these findings. If so, complexity is an example of a property that could be used to further generalize experimental findings.

X.6 Studies of Motivation

In our study, we found strong evidence to suggest that motivation affects the variance of developers. Based on our findings, we would expect intrinsically motivated developers to manifest less variance than extrinsically motivated developers. However, it is not clear whether these findings translate to industry. If they do, then a better understanding of developer motivations could enable greater control over the consistency (and therefore predictability) of software development outcomes.

Any study of motivation would need to develop (or borrow from other fields) a theoretical framework for differentiating types of motivation. Psychology or the social sciences may be a good place to start. Concerns include: whether an intrinsic/extrinsic distinction is the most effective characterization of motivation for the context of software engineering, as well as whether secondary motivations are as important as primary motivations in predicting developer performance.

APPENDIX Y
FREQUENTIST STATISTICAL RESULTS

For a description of the frequentist models, see Section 2.3. Tables 25–40 present results based on the full dataset (53 participants). Tables 41–58 present results for the same models, but after applying participant filtering (as described in Section 3.1). Statistical source code (SAS 9.3) and output are included in the lab package. All p-values are two-sided. Variables not appearing in the results tables have been removed via model tuning due to lack of significance. For a description of the tuning process, see Appendix M.

Y.1 Results Layout

Column headers are defined for all tables as follows:

- `Effect`: Explanatory variable.
- `Level`: Level for a categorical explanatory variable.
- `Level Diff`: Two categorical levels compared, the first minus the second.
- `F Value`: f-statistic.
- `Pr>F`: Two-sided p-value, computed using the f distribution.
- `Estimate`: Parameter estimate.
- `Orig Scale`: The parameter estimate converted back to the original scale (*time* models only).
- `t Value`: t-statistic.
- `Pr>|t|`: Two-sided p-value, computed using the t distribution.
- `Adj P`: Two-sided p-value, adjusted to account for multiple comparisons (Tukey-Kramer). Has no effect for binary categorical variables.
- `Ratio`: As explained below, back-transforming difference estimates yields ratios.

Y.2 Results Interpretation

Since we log-transformed the *time* variable, we must back-transform the results. When *time_ln* is the response variable, back-transformation requires computing e^x, where x is the log-scale estimate. In these cases, we must back-transform four types of estimates: slope estimates (e.g., Table 26), marginal means (e.g., Table 27), differences between marginal means (e.g., Table 28), and differences between interaction levels (e.g., Table 46).

When the log-scale estimate, x, represents a difference, i.e., $x = y - z$, back-transformation yields a ratio rather than an interval, as in $e^x = e^{y-z} = e^y/e^z$. Thus, the linear (or additive) effect on the log scale becomes a multiplicative effect on the original scale. In decimal form the back-transformed differences are essentially multiplicative factors that scale the response variable up or down by some percentage depending on whether the value is greater or less than one.

Note that interval differences can be computed on the original scale if y and z are known, as in $e^y - e^z$ (e.g., Table 28). However, if y and z are not marginal means, then their estimates depend on an arbitrary selection of values for all other variables in the model. In this case, shifting other variables also shifts y and z. On the log scale, such shifts are linear, so differences remain constant, but on the original scale shifts translate into multiplicative changes. Consequently, interval differences are only meaningful on the original scale when computed from marginal means.

Slope is a measure of change in one variable in response to change in another variable and can be represented as $\Delta response/\Delta covariate$. For the slope estimates shown in the tables, $\Delta covariate = 1$. Thus, the log-scale slope estimates represent differences (i.e., $\Delta response/\Delta covariate = \Delta response/1 = \Delta response$), such that back-transformation yields ratios:

$$e^{\Delta response} = e^{y-z} = e^y/e^z$$

Thus, on the *log* scale, a slope estimate represents the *linear* change in *time_ln* expected to occur in response to a 1-unit *linear* change in the associated covariate; but on the *non-log* scale, a slope estimate represents the *multiplicative* change in *time* expected to occur in response to a 1-unit *linear* change in the covariate.

Further, since $\Delta covariate = 1$ for the slopes shown in the tables, back-transformation via e^x yields estimates relative to 1-unit changes in the covariates. To obtain estimates relative to other values for $\Delta covariate$, simply back-transform the log-scale estimate using the more general formula e^{xd}, where d is the desired $\Delta covariate$. For example, in the first footnote (*) of Table 26, $d = 1$, such that back-transformation is computed as $e^{-0.1091*1}$; but in the second footnote (†), $d = 10$, such that back-transformation is computed as $e^{0.0022*10}$.

When *time_ln* is a covariate instead of the response variable (which occurs for slope estimates in the *correctness* models; e.g., Table 30), interpretation is handled differently. Since *time_ln* is the covariate, interpretation of slope estimates requires back-transforming $\Delta covariate$ by computing $e^{\Delta covariate}$. As mentioned above, $\Delta covariate = 1$ by default and back-transformation of differences yields ratios.

Thus, on the original scale, the slope estimates shown in the tables represent the *linear* change in *correctness* expected to occur in response to a *multiplicative* increase in non-log time of e^1 (≈ 2.7). To obtain a slope estimate relative to a multiplicative factor other than e^1, compute $ln(\theta)x$, where x is the slope estimate given in the table and θ is the desired multiplicative factor. For example, Table 30 indicates that a 1-unit increase in *time_ln* yields an average *correctness* increase of about 7.17 percentage points. Alternatively, on the original scale, an approximately 2.7-fold increase in *time* yields an average *correctness* increase of about 7.17 percentage points. Or stated more intuitively, a 2-fold increase in *time* yields an average correctness increase of about $ln(2)7.17 = 4.97$ percentage points.

TABLE 25
CO_time, unfiltered (52 participants)
Type 3 Tests of Fixed Effects

Effect	F Value	Pr>F
site	8.46	<**0.001**
order	8.03	**0.007**
task	270.93	<**0.001**
devExp	3.28	0.076
correctness	3.40	0.071
variant	0.01	0.925

TABLE 26
CO_time, unfiltered (52 participants)
Solution for Fixed Effects

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	6.8147	21.79	<**0.001**
site	BYU	−0.5129	−3.10	**0.003**
site	FUB	0.0029	0.02	0.988
site	UA	−0.5069	−2.83	**0.007**
site	UPM	0	-	-
order	1	0.2732	2.83	**0.007**
order	2	0	-	-
task	1	1.2887	16.46	<**0.001**
task	2	0	-	-
devExp	-	−0.1091*	−1.81	0.076
correctness	-	0.0022†	1.84	0.071
variant	ALT	−0.0094	−0.09	0.925
variant	PAT	0	-	-

*A 1-unit increase in developer experience yields an average time decrease of about 10.3%.
†A 10-point increase in correctness yields an average time increase of about 2.2%.

TABLE 27
CO_time, unfiltered (52 participants)
Marginal Means (Least Squares Estimates)

Effect	Level	Estimate	Orig Scale*
site	BYU	6.7265	834
site	FUB	7.2423	1397
site	UA	6.7324	839
site	UPM	7.2394	1393
order	1	7.1217	1239
order	2	6.8485	942
task	1	7.6295	2058
task	2	6.3408	567
variant	ALT	6.9805	1075
variant	PAT	6.9898	1086

*Computed as e^x, where x is the log-scale estimate.

TABLE 28
CO_time, unfiltered (52 participants)
Differences for Marginal Means

Effect	Level Diff	Estimate	t Value	Adj P*	Orig Scale†
site	BYU−FUB	−0.5158	−4.01	**0.001**	−563
site	BYU−UA	−0.0060	−0.05	1.000	−5
site	BYU−UPM	−0.5129	−3.10	**0.016**	−559
site	FUB−UA	0.5098	3.52	**0.005**	558
site	FUB−UPM	0.0029	0.02	1.000	4
site	UA−UPM	−0.5069	−2.83	**0.033**	−554
order	1−2	0.2732	2.83	**0.007**	296
task	1−2	1.2887	16.46	<**0.001**	1491
variant	ALT−PAT	−0.0094	−0.09	0.925	−10

*Adjusted for multiple comparisons (Tukey-Kramer).
†Computed as $e^x - e^y$, where x and y are log-scale marginal means (see Table 27 above).

TABLE 29
CO_correctness, unfiltered (52 participants)
Type 3 Tests of Fixed Effects

Effect	F Value	Pr>F
site	5.20	**0.003**
devExp	2.78	0.101
time_ln	2.64	0.110
variant	2.90	0.095

TABLE 30
CO_correctness, unfiltered (52 participants)
Solution for Fixed Effects

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	−70.60	−1.73	0.090
site	BYU	37.31	2.94	**0.005**
site	FUB	51.72	3.82	**<0.001**
site	UA	29.92	2.17	**0.035**
site	UPM	0	-	-
devExp	-	7.41*	1.67	0.101
time_ln	-	7.17†	1.63	0.110
variant	ALT	12.17	1.70	0.095
variant	PAT	0	-	-

*A 1-unit increase in developer experience yields an average correctness increase of about 7.4 percentage points.
†A 2x increase in work time yields an average correctness increase of about 5.0 percentage points.

TABLE 31
CO_correctness, unfiltered (52 participants)
Marginal Means (Least Squares Estimates)

Effect	Level	Estimate
site	BYU	53.93
site	FUB	68.35
site	UA	46.55
site	UPM	16.62
variant	ALT	52.45
variant	PAT	40.28

TABLE 32
CO_correctness, unfiltered (52 participants)
Differences for Marginal Means

Effect	Level Diff	Estimate	t Value	Adj P*
site	BYU−FUB	−14.42	−1.52	0.436
site	BYU−UA	7.39	0.80	0.855
site	BYU−UPM	37.31	2.94	**0.025**
site	FUB−UA	21.80	2.09	0.169
site	FUB−UPM	51.72	3.82	**0.002**
site	UA−UPM	29.92	2.17	0.147
variant	ALT−PAT	12.17	1.70	0.095

*Adjusted for multiple comparisons (Tukey-Kramer).

TABLE 33
GR_time, unfiltered (51 participants)
Type 3 Tests of Fixed Effects

Effect	F Value	Pr>F
site	3.62	**0.019**
order	10.91	**0.002**
task	34.40	<**0.001**
devExp	20.00	<**0.001**
correctness	12.94	<**0.001**
variant	6.18	**0.016**

TABLE 34
GR_time, unfiltered (51 participants)
Solution for Fixed Effects

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	7.8042	18.06	<**0.001**
site	BYU	−0.4337	−2.00	0.051
site	FUB	0.0846	0.36	0.722
site	UA	−0.0794	−0.34	0.736
site	UPM	0	-	-
order	1	0.4348	3.30	**0.002**
order	2	0	-	-
task	1	0.8358	5.87	<**0.001**
task	2	0	-	-
devExp	-	−0.3573*	−4.47	<**0.001**
correctness	-	0.0057†	3.60	<**0.001**
variant	ALT	−0.3343	−2.49	**0.016**
variant	PAT	0	-	-

*A 1-unit increase in developer experience yields an average time decrease of about 30.0%.
†A 10-point increase in correctness yields an average time increase of about 5.9%.

TABLE 35
GR_time, unfiltered (51 participants)
Marginal Means (Least Squares Estimates)

Effect	Level	Estimate	Orig Scale*
site	BYU	6.6131	745
site	FUB	7.1314	1251
site	UA	6.9674	1061
site	UPM	7.0468	1149
order	1	7.1571	1283
order	2	6.7223	831
task	1	7.3576	1568
task	2	6.5218	680
variant	ALT	6.7725	873
variant	PAT	7.1069	1220

*Computed as e^x, where x is the log-scale estimate.

TABLE 36
GR_time, unfiltered (51 participants)
Differences for Marginal Means

Effect	Level Diff	Estimate	t Value	Adj P*	Orig Scale†
site	BYU−FUB	−0.5183	−2.97	**0.023**	−506
site	BYU−UA	−0.3543	−2.05	0.184	−317
site	BYU−UPM	−0.4337	−2.00	0.201	−404
site	FUB−UA	0.1640	0.85	0.832	189
site	FUB−UPM	0.0846	0.36	0.984	101
site	UA−UPM	−0.0794	−0.34	0.986	−88
order	1−2	0.4348	3.30	**0.002**	452
task	1−2	0.8358	5.87	<**0.001**	888
variant	ALT−PAT	−0.3343	−2.49	**0.016**	−347

*Adjusted for multiple comparisons (Tukey-Kramer).
†Computed as $e^x - e^y$, where x and y are log-scale marginal means (see Table 35 above).

TABLE 37
GR_correctness, unfiltered (51 participants)
Type 3 Tests of Fixed Effects

Effect	F Value	Pr>F
task	4.89	**0.032**
time_ln	9.55	**0.003**
variant	1.00	0.322

TABLE 38
GR_correctness, unfiltered (51 participants)
Solution for Fixed Effects

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	−65.67	−2.02	**0.049**
task	1	20.80	2.21	**0.032**
task	2	0	-	-
time_ln	-	15.29*	3.09	**0.003**
variant	ALT	8.28	1.00	0.322
variant	PAT	0	-	-

*A 2x increase in work time yields an average correctness increase of about 10.6 percentage points.

TABLE 39
GR_correctness, unfiltered (51 participants)
Marginal Means (Least Squares Estimates)

Effect	Level	Estimate
task	1	64.29
task	2	43.49
variant	ALT	58.03
variant	PAT	49.75

TABLE 40
GR_correctness, unfiltered (51 participants)
Differences for Marginal Means

Effect	Level Diff	Estimate	t Value	Pr>|t|
task	1−2	20.80	2.21	**0.032**
variant	ALT−PAT	8.28	1.00	0.322

TABLE 41
CO_time, filtered (42 participants)
Type 3 Tests of Fixed Effects

Effect	F Value	Pr>F
site	13.53	<**0.001**
order	9.36	**0.004**
task	218.26	<**0.001**
patKnow	3.96	0.053*
correctness	2.76	0.104
variant	5.95	**0.019***
patKnow × variant	5.12	**0.029**

*Results for patKnow and variant are not meaningful outside the interaction. See Tables 43 and 46 instead.

TABLE 42
CO_time, filtered (42 participants)
Solution for Fixed Effects

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	7.1296	23.28	<**0.001**
site	BYU	−0.5887	−3.41	**0.002**
site	FUB	0.0833	0.45	0.657
site	UA	−0.3234	−1.57	0.125
site	UPM	0	-	-
order	1	0.2819	3.06	**0.004**
order	2	0	-	-
task	1	1.2974	14.77	<**0.001**
task	2	0	-	-
patKnow	-	−0.2527†	−2.86	**0.007**
correctness	-	0.0023*	1.66	0.104
variant	ALT	−0.8617†	−2.44	**0.019**
variant	PAT	0†	-	-
patKnow × variant	ALT	0.2388†	2.26	**0.029**
patKnow × variant	PAT	0†	-	-

*A 10-point increase in correctness yields an average time increase of about 2.3%.
†Results for patKnow and variant are not meaningful outside the interaction. See Tables 43 and 46 instead.

TABLE 43
CO_time, filtered (42 participants)
Slopes for *patKnow* × *variant* (from Table 42)

Effect	Variant Level	Estimate	t Value	Pr>|t|	Ratio
patKnow	ALT	−0.0139	−0.17	0.866	0.986*
patKnow	PAT	−0.2527	−2.86	**0.007**	0.777†

*For ALT tasks, a 1-unit increase in pattern knowledge yields an average time decrease of about 1.4%.
†For PAT tasks, a 1-unit increase in pattern knowledge yields an average time decrease of about 22.3%.

TABLE 44
CO_time, filtered (42 participants)
Marginal Means (Least Squares Estimates)

Effect	Level	Estimate	Orig Scale*
site	BYU	6.6045	738
site	FUB	7.2764	1446
site	UA	6.8698	963
site	UPM	7.1932	1330
order	1	7.1269	1245
order	2	6.8450	939
task	1	7.6347	2069
task	2	6.3373	565

*Computed as e^x, where x is the log-scale estimate.

TABLE 45
CO_time, filtered (42 participants)
Differences for Marginal Means

Effect	Level Diff	Estimate	t Value	Adj P*	Orig Scale†
site	BYU−FUB	−0.6720	−5.85	<**0.001**	−707
site	BYU−UA	−0.2653	−1.73	0.321	−224
site	BYU−UPM	−0.5887	−3.41	**0.008**	−592
site	FUB−UA	0.4067	2.80	**0.038**	483
site	FUB−UPM	0.0833	0.45	0.970	115
site	UA−UPM	−0.3234	−1.57	0.409	−368
order	1−2	0.2819	3.06	**0.004**	306
task	1−2	1.2974	14.77	<**0.001**	1503

*Adjusted for multiple comparisons (Tukey-Kramer).
†Computed as $e^x - e^y$, where x and y are log-scale marginal means (see Table 44 above).

TABLE 46
CO_time, filtered (42 participants)
Differences for *patKnow* × *variant*

PatKnow*	Variant Diff	Estimate	t Value	Pr>|t|	Ratio†
1.7 (min)	ALT−PAT	−0.4544	−2.45	**0.019**	0.635
3.3 (mean)	ALT−PAT	−0.0826	−0.89	0.381	0.921
5.4 (max)	ALT−PAT	0.4304	1.73	0.091	1.538

*Values shown are rounded to fit the table.
†Computed as e^x, where x is the log-scale estimate. Since each estimate (x) represents a difference ($y-z$), back-transformation yields a ratio ($e^x = e^{y-z} = e^y/e^z$). E.g., when *patKnow* is at its minimum value, the ALT/PAT ratio is 0.635, meaning ALT tasks require 36.5% less time than PAT tasks, on average.

TABLE 47
CO_correctness, filtered (42 participants)
Type 3 Tests of Fixed Effects

Effect	F Value	Pr>F
site	4.53	**0.008**
patKnow	4.11	**0.049**
variant	0.41	0.523

TABLE 48
CO_correctness, filtered (42 participants)
Solution for Fixed Effects

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	−11.46	−0.65	0.520
site	BYU	40.70	3.16	**0.003**
site	FUB	43.79	3.13	**0.003**
site	UA	24.30	1.49	0.144
site	UPM	0	-	-
patKnow	-	10.57*	2.03	**0.049**
variant	ALT	4.76	0.64	0.523
variant	PAT	0	-	-

*A 1-unit increase in pattern knowledge yields an average correctness increase of about 10.6 percentage points.

TABLE 49
CO_correctness, filtered (42 participants)
Marginal Means (Least Squares Estimates)

Effect	Level	Estimate
site	BYU	66.12
site	FUB	69.21
site	UA	49.72
site	UPM	25.42
variant	ALT	55.00
variant	PAT	50.24

TABLE 50
CO_correctness, filtered (42 participants)
Differences for Marginal Means

Effect	Level Diff	Estimate	t Value	Adj P*
site	BYU−FUB	−3.09	−0.34	0.986
site	BYU−UA	16.40	1.37	0.524
site	BYU−UPM	40.70	3.16	**0.015**
site	FUB−UA	19.49	1.72	0.324
site	FUB−UPM	43.79	3.13	**0.016**
site	UA−UPM	24.30	1.49	0.453
variant	ALT−PAT	4.76	0.64	0.523

*Adjusted for multiple comparisons (Tukey-Kramer).

TABLE 51
GR_time, filtered (42 participants)
Type 3 Tests of Fixed Effects

Effect	F Value	Pr>F
site	8.97	<**0.001**
order	4.60	**0.038**
task	22.63	<**0.001**
patKnow	12.30	**0.001**
correctness	23.30	<**0.001**
variant	5.44	**0.025**

TABLE 52
GR_time, filtered (42 participants)
Solution for Fixed Effects

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	7.0909	19.08	<**0.001**
site	BYU	−0.3862	−1.65	0.107
site	FUB	0.4631	1.75	0.087
site	UA	0.5660	1.84	0.073
site	UPM	0	-	-
order	1	0.3030	2.14	**0.038**
order	2	0	-	-
task	1	0.7334	4.76	<**0.001**
task	2	0	-	-
patKnow	-	−0.3569*	−3.51	**0.001**
correctness	-	0.0086†	4.83	<**0.001**
variant	ALT	−0.3409	−2.33	**0.025**
variant	PAT	0	-	-

*A 1-unit increase in pattern knowledge yields an average time decrease of about 30.0%.
†A 10-point increase in correctness yields an average time increase of about 9.0%.

TABLE 53
GR_time, filtered (42 participants)
Marginal Means (Least Squares Estimates)

Effect	Level	Estimate	Orig Scale*
site	BYU	6.4258	618
site	FUB	7.2751	1444
site	UA	7.3780	1600
site	UPM	6.8120	909
order	1	7.1242	1242
order	2	6.8212	917
task	1	7.3394	1540
task	2	6.6060	740
variant	ALT	6.8023	900
variant	PAT	7.1432	1265

*Computed as e^x, where x is the log-scale estimate.

TABLE 54
GR_time, filtered (42 participants)
Differences for Marginal Means

Effect	Level Diff	Estimate	t Value	Adj P*	Orig Scale†
site	BYU−FUB	−0.8493	−4.64	<**0.001**	−826
site	BYU−UA	−0.9522	−4.07	**0.001**	−983
site	BYU−UPM	−0.3862	−1.65	0.364	−291
site	FUB−UA	−0.1029	−0.46	0.968	−156
site	FUB−UPM	0.4631	1.75	0.311	535
site	UA−UPM	0.5660	1.84	0.270	692
order	1−2	0.3030	2.14	**0.038**	325
task	1−2	0.7334	4.76	<**0.001**	800
variant	ALT−PAT	−0.3409	−2.33	**0.025**	−366

*Adjusted for multiple comparisons (Tukey-Kramer).
†Computed as $e^x - e^y$, where x and y are log-scale marginal means (see Table 53 above).

TABLE 55
GR_correctness, filtered (42 participants)
Type 3 Tests of Fixed Effects

Effect	F Value	Pr>F
time_ln	34.87	<**0.001**
variant	1.39	0.245

TABLE 56
GR_correctness, filtered (42 participants)
Solution for Fixed Effects

Effect	Level	Estimate	t Value	Pr>|t|
intercept	-	−117.67	−3.88	<**0.001**
time_ln	-	25.40*	5.91	<**0.001**
variant	ALT	9.69	1.18	0.245
variant	PAT	0	-	-

*A 2x increase in work time yields an average correctness increase of about 17.6 percentage points.

TABLE 57
GR_correctness, filtered (42 participants)
Marginal Means (Least Squares Estimates)

Effect	Level	Estimate
variant	ALT	66.45
variant	PAT	56.76

TABLE 58
GR_correctness, filtered (42 participants)
Differences for Marginal Means

Effect	Level Diff	Estimate	t Value	Pr>|t|
variant	ALT−PAT	9.69	1.18	0.245

APPENDIX Z
BAYESIAN STATISTICAL RESULTS

For a description of the Bayesian models, see Section 2.3. Table 59 presents results based on the full dataset (53 participants). Table 60 presents results for the same models, but after applying participant filtering (as described in Section 3.1). Statistical source code (R 2.15.2) is included in the lab package.

Z.1 Results Layout

The Bayesian tables are abbreviated versions of a Microsoft Excel file, which is provided in the lab package (BayesianAnalysisResults.xlsx). The numbers at the right margin map the table rows to the Excel file. The Excel file adds additional data and visualizations. Coding and comprehension tasks are abbreviated in the tables as 't1' and 't2'. All other abbreviations are as previously defined. All probabilities are rounded—i.e., none are exactly 1 or 0.

Results for the *time* and *correctness* models are represented as columns in the tables (labeled T1–T6 and C1–C6, respectively). Rows are grouped by bold subheadings, which identify two types of information. On the far left, the subheadings identify the variable or interaction under consideration (e.g., the results on rows 37–40 were computed from the *program* × *variant* interaction). At center and on the right, the subheadings identify the specific effect being analyzed—which effect corresponds to the variable listed on the left, or if an interaction is listed on the left, then it corresponds to a variable within the interaction. For example, on row 36, "compare: variant" means that ALT and PAT are being compared, and since the interaction *program* × *variant* is listed on the left, ALT and PAT are being compared separately for the CO and GR programs.

For each comparison, we provide two types of values: probabilities and differences. Probabilities are listed in black and labeled $p(x > y)$, meaning the posterior probability that condition x takes longer (models T1–T6) or scores higher (models C1–C6) than condition y. Differences are listed in gray and labeled $x - y$, meaning the average difference in *time* or *correctness* between conditions x and y (computed as the difference between posterior distribution means).

Z.2 Results Interpretation

In addition to the "Results Interpretation" discussion in Section 2.3, note the following two concerns:

- Since the Bayesian analysis is based on binary variables, insignificant comparisons are those for which the probabilities are near 0.5. Thus, a probability of 0.25 is as significant as a probability of 0.75. Probabilities less than 0.5 simply indicate that the reverse comparison is more likely. The directionality of the comparisons shown in the tables (i.e., $x > y$ as opposed to $y > x$) is arbitrary. To reverse a comparison, compute $1 - p$ for probabilities and $-x$ for differences.
- Since *statistical power is influenced by both model size and by the distribution of observations over model parameters* [31, p. 347], probabilities should not be directly compared across models. Instead, cross-validation requires checking that two models support similar conclusions.

TABLE 59
Unfiltered Bayesian Results. See Appendix Z for a description of how to read and interpret this table.

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
order	**compare: order**						**compare: order**						11
p(1st > 2nd)	0.99	1.00	1.00	1.00	1.00	1.00	0.13	0.17	0.15	0.22	0.20	0.20	12
1st − 2nd	189	228	223	214	210	189	−3.4	−3.1	−3.5	−2.3	−2.6	−2.7	13
variant	**compare: variant**						**compare: variant**						33
p(PAT > ALT)	0.58	0.52	0.55	0.56	0.54	0.55	0.39	0.37	0.38	0.35	0.30	0.31	34
PAT − ALT	152	24	73	89	30	42	−5.1	−5.1	−4.3	−5.4	−5.7	−4.8	35
program × variant	**compare: variant**						**compare: variant**						36
p(CO PAT > CO ALT)	0.57	0.51	0.56	0.54	0.53	0.49	0.38	0.35	0.37	0.37	0.30	0.29	37
CO PAT − CO ALT	129	5	67	47	29	−13	−5.2	−5.6	−4.8	−5.2	−5.9	−5.4	38
p(GR PAT > GR ALT)	0.59	0.54	0.55	0.59	0.54	0.62	0.39	0.38	0.39	0.32	0.30	0.33	39
GR PAT − GR ALT	175	43	79	132	31	97	−5.0	−4.7	−3.7	−5.5	−5.4	−4.1	40
task × variant	**compare: variant**						**compare: variant**						41
p(t1 PAT > t1 ALT)	0.56	0.48	0.53	0.55	0.49	-	0.31	0.31	0.31	0.26	0.20	-	42
t1 PAT − t1 ALT	132	−17	26	86	−12	-	−9.2	−7.3	−7.1	−9.0	−8.6	-	43
p(t2 PAT > t2 ALT)	0.60	0.56	0.58	0.57	0.59	-	0.47	0.42	0.45	0.44	0.40	-	44
t2 PAT − t2 ALT	172	65	121	92	72	-	−1.0	−3.0	−1.5	−1.8	−2.7	-	45
program × task × variant	**compare: variant**						**compare: variant**						46
p(CO t1 PAT > CO t1 ALT)	0.56	0.52	0.61	0.61	0.57	-	0.33	0.34	0.30	0.32	0.23	-	47
CO t1 PAT − CO t1 ALT	111	33	129	126	62	-	−7.8	−5.9	−7.5	−8.0	−8.0	-	48
p(GR t1 PAT > GR t1 ALT)	0.56	0.44	0.44	0.49	0.41	-	0.28	0.29	0.32	0.19	0.17	-	49
GR t1 PAT − GR t1 ALT	154	−66	−78	46	−86	-	−10.5	−8.7	−6.7	−10.0	−9.3	-	50
p(CO t2 PAT > CO t2 ALT)	0.58	0.49	0.50	0.47	0.50	-	0.44	0.37	0.44	0.42	0.37	-	51
CO t2 PAT − CO t2 ALT	148	−23	5	−33	−4	-	−2.5	−5.2	−2.2	−2.5	−3.8	-	52
p(GR t2 PAT > GR t2 ALT)	0.62	0.64	0.66	0.68	0.67	-	0.50	0.47	0.47	0.46	0.43	-	53
GR t2 PAT − GR t2 ALT	196	153	237	218	149	-	0.5	−0.7	−0.8	−1.1	−1.5	-	54
time_or_correctness	**compare: correctness**						**compare: time**						61
p(Low > High)	0.01	0.03	0.09	0.32	0.05	0.10	0.15	0.16	0.18	0.37	0.19	0.12	62
Low − High	−232	−182	−135	−198	−155	−120	−4.4	−4.1	−4.0	−3.5	−3.6	−4.7	63
variant × patKnow	**compare: variant**						**compare: variant**						102
p(ALT Low > PAT Low)	-	0.46	-	-	-	-	-	0.68	-	-	-	-	103
ALT Low − PAT Low	-	−27	-	-	-	-	-	6.8	-	-	-	-	104
p(ALT High > PAT High)	-	0.49	-	-	-	-	-	0.59	-	-	-	-	105
ALT High − PAT High	-	−22	-	-	-	-	-	3.5	-	-	-	-	106
program × variant × patKnow	**compare: variant**						**compare: variant**						117
p(CO ALT Low > CO PAT Low)	-	0.44	-	-	-	-	-	0.62	-	-	-	-	118
CO ALT Low − CO PAT Low	-	−40	-	-	-	-	-	4.1	-	-	-	-	119
p(CO ALT High > CO PAT High)	-	0.54	-	-	-	-	-	0.67	-	-	-	-	120
CO ALT High − CO PAT High	-	31	-	-	-	-	-	7.0	-	-	-	-	121
													122
p(GR ALT Low > GR PAT Low)	-	0.48	-	-	-	-	-	0.74	-	-	-	-	123
GR ALT Low − GR PAT Low	-	−13	-	-	-	-	-	9.5	-	-	-	-	124
p(GR ALT High > GR PAT High)	-	0.44	-	-	-	-	-	0.50	-	-	-	-	125
GR ALT High − GR PAT High	-	−74	-	-	-	-	-	−0.1	-	-	-	-	126
task × variant × patKnow	**compare: variant**						**compare: variant**						137
p(t1 ALT Low > t1 PAT Low)	-	0.58	-	-	-	-	-	0.81	-	-	-	-	138
t1 ALT Low − t1 PAT Low	-	95	-	-	-	-	-	12.1	-	-	-	-	139
p(t1 ALT High > t1 PAT High)	-	0.45	-	-	-	-	-	0.56	-	-	-	-	140
t1 ALT High − t1 PAT High	-	−62	-	-	-	-	-	2.5	-	-	-	-	141
													142
p(t2 ALT Low > t2 PAT Low)	-	0.35	-	-	-	-	-	0.55	-	-	-	-	143
t2 ALT Low − t2 PAT Low	-	−148	-	-	-	-	-	1.5	-	-	-	-	144
p(t2 ALT High > t2 PAT High)	-	0.53	-	-	-	-	-	0.61	-	-	-	-	145
t2 ALT High − t2 PAT High	-	19	-	-	-	-	-	4.5	-	-	-	-	146
program × task × variant × patKnow	**compare: variant**						**compare: variant**						167
p(CO t1 ALT Low > CO t1 PAT Low)	-	0.53	-	-	-	-	-	0.70	-	-	-	-	168
CO t1 ALT Low − CO t1 PAT Low	-	41	-	-	-	-	-	6.8	-	-	-	-	169
p(CO t1 ALT High > CO t1 PAT High)	-	0.42	-	-	-	-	-	0.63	-	-	-	-	170
CO t1 ALT High − CO t1 PAT High	-	−107	-	-	-	-	-	5.0	-	-	-	-	171
													172
p(CO t2 ALT Low > CO t2 PAT Low)	-	0.36	-	-	-	-	-	0.54	-	-	-	-	173
CO t2 ALT Low − CO t2 PAT Low	-	−122	-	-	-	-	-	1.4	-	-	-	-	174
p(CO t2 ALT High > CO t2 PAT High)	-	0.67	-	-	-	-	-	0.72	-	-	-	-	175
CO t2 ALT High − CO t2 PAT High	-	169	-	-	-	-	-	9.1	-	-	-	-	176
													177
p(GR t1 ALT Low > GR t1 PAT Low)	-	0.62	-	-	-	-	-	0.92	-	-	-	-	178
GR t1 ALT Low − GR t1 PAT Low	-	149	-	-	-	-	-	17.3	-	-	-	-	179
p(GR t1 ALT High > GR t1 PAT High)	-	0.49	-	-	-	-	-	0.50	-	-	-	-	180
GR t1 ALT High − GR t1 PAT High	-	−17	-	-	-	-	-	0.1	-	-	-	-	181
													182

(continued on next page)

(Table 59 continued – Unfiltered Bayesian Results)

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
p(GR t2 ALT Low > GR t2 PAT Low)	-	0.34	-	-	-	-	-	0.56	-	-	-	-	183
GR t2 ALT Low − GR t2 PAT Low	-	−175	-	-	-	-	-	1.6	-	-	-	-	184
p(GR t2 ALT High > GR t2 PAT High)	-	0.39	-	-	-	-	-	0.49	-	-	-	-	185
GR t2 ALT High − GR t2 PAT High	-	−131	-	-	-	-	-	−0.2	-	-	-	-	186
variant × devExp	**compare: variant**						**compare: variant**						212
p(ALT Low > PAT Low)	-	-	0.41	-	-	-	-	-	0.70	-	-	-	213
ALT Low − PAT Low	-	-	−117	-	-	-	-	-	7.3	-	-	-	214
p(ALT High > PAT High)	-	-	0.48	-	-	-	-	-	0.53	-	-	-	215
ALT High − PAT High	-	-	−30	-	-	-	-	-	1.3	-	-	-	216
program × variant × devExp	**compare: variant**						**compare: variant**						227
p(CO ALT Low > CO PAT Low)	-	-	0.42	-	-	-	-	-	0.66	-	-	-	228
CO ALT Low − CO PAT Low	-	-	−92	-	-	-	-	-	5.9	-	-	-	229
p(CO ALT High > CO PAT High)	-	-	0.47	-	-	-	-	-	0.60	-	-	-	230
CO ALT High − CO PAT High	-	-	−42	-	-	-	-	-	3.7	-	-	-	231
													232
p(GR ALT Low > GR PAT Low)	-	-	0.40	-	-	-	-	-	0.75	-	-	-	233
GR ALT Low − GR PAT Low	-	-	−141	-	-	-	-	-	8.7	-	-	-	234
p(GR ALT High > GR PAT High)	-	-	0.49	-	-	-	-	-	0.47	-	-	-	235
GR ALT High − GR PAT High	-	-	−18	-	-	-	-	-	−1.2	-	-	-	236
task × variant × devExp	**compare: variant**						**compare: variant**						247
p(t1 ALT Low > t1 PAT Low)	-	-	0.53	-	-	-	-	-	0.81	-	-	-	248
t1 ALT Low − t1 PAT Low	-	-	47	-	-	-	-	-	11.3	-	-	-	249
p(t1 ALT High > t1 PAT High)	-	-	0.42	-	-	-	-	-	0.58	-	-	-	250
t1 ALT High − t1 PAT High	-	-	−98	-	-	-	-	-	2.9	-	-	-	251
													252
p(t2 ALT Low > t2 PAT Low)	-	-	0.30	-	-	-	-	-	0.60	-	-	-	253
t2 ALT Low − t2 PAT Low	-	-	−280	-	-	-	-	-	3.3	-	-	-	254
p(t2 ALT High > t2 PAT High)	-	-	0.53	-	-	-	-	-	0.49	-	-	-	255
t2 ALT High − t2 PAT High	-	-	38	-	-	-	-	-	−0.3	-	-	-	256
program × task × variant × devExp	**compare: variant**						**compare: variant**						277
p(CO t1 ALT Low > CO t1 PAT Low)	-	-	0.34	-	-	-	-	-	0.76	-	-	-	278
CO t1 ALT Low − CO t1 PAT Low	-	-	−187	-	-	-	-	-	9.9	-	-	-	279
p(CO t1 ALT High > CO t1 PAT High)	-	-	0.44	-	-	-	-	-	0.64	-	-	-	280
CO t1 ALT High − CO t1 PAT High	-	-	−72	-	-	-	-	-	5.1	-	-	-	281
													282
p(CO t2 ALT Low > CO t2 PAT Low)	-	-	0.50	-	-	-	-	-	0.56	-	-	-	283
CO t2 ALT Low − CO t2 PAT Low	-	-	2	-	-	-	-	-	1.9	-	-	-	284
p(CO t2 ALT High > CO t2 PAT High)	-	-	0.49	-	-	-	-	-	0.57	-	-	-	285
CO t2 ALT High − CO t2 PAT High	-	-	−11	-	-	-	-	-	2.4	-	-	-	286
													287
p(GR t1 ALT Low > GR t1 PAT Low)	-	-	0.71	-	-	-	-	-	0.85	-	-	-	288
GR t1 ALT Low − GR t1 PAT Low	-	-	280	-	-	-	-	-	12.7	-	-	-	289
p(GR t1 ALT High > GR t1 PAT High)	-	-	0.40	-	-	-	-	-	0.52	-	-	-	290
GR t1 ALT High − GR t1 PAT High	-	-	−124	-	-	-	-	-	0.6	-	-	-	291
													292
p(GR t2 ALT Low > GR t2 PAT Low)	-	-	0.10	-	-	-	-	-	0.65	-	-	-	293
GR t2 ALT Low − GR t2 PAT Low	-	-	−561	-	-	-	-	-	4.6	-	-	-	294
p(GR t2 ALT High > GR t2 PAT High)	-	-	0.58	-	-	-	-	-	0.41	-	-	-	295
GR t2 ALT High − GR t2 PAT High	-	-	88	-	-	-	-	-	−3.0	-	-	-	296
variant × time_or_correctness	**compare: variant**						**compare: variant**						322
p(ALT Low > PAT Low)	-	-	-	0.43	-	-	-	-	-	0.72	-	-	323
ALT Low − PAT Low	-	-	-	−96	-	-	-	-	-	7.6	-	-	324
p(ALT High > PAT High)	-	-	-	0.45	-	-	-	-	-	0.59	-	-	325
ALT High − PAT High	-	-	-	−82	-	-	-	-	-	3.1	-	-	326
program × variant × time_or_correctness	**compare: variant**						**compare: variant**						337
p(CO ALT Low > CO PAT Low)	-	-	-	0.50	-	-	-	-	-	0.74	-	-	338
CO ALT Low − CO PAT Low	-	-	-	26	-	-	-	-	-	9.8	-	-	339
p(CO ALT High > CO PAT High)	-	-	-	0.42	-	-	-	-	-	0.51	-	-	340
CO ALT High − CO PAT High	-	-	-	−119	-	-	-	-	-	0.7	-	-	341
													342
p(GR ALT Low > GR PAT Low)	-	-	-	0.35	-	-	-	-	-	0.69	-	-	343
GR ALT Low − GR PAT Low	-	-	-	−218	-	-	-	-	-	5.4	-	-	344
p(GR ALT High > GR PAT High)	-	-	-	0.48	-	-	-	-	-	0.66	-	-	345
GR ALT High − GR PAT High	-	-	-	−46	-	-	-	-	-	5.6	-	-	346
task × variant × time_or_correctness	**compare: variant**						**compare: variant**						357
p(t1 ALT Low > t1 PAT Low)	-	-	-	0.42	-	-	-	-	-	0.87	-	-	358
t1 ALT Low − t1 PAT Low	-	-	-	−144	-	-	-	-	-	13.5	-	-	359
p(t1 ALT High > t1 PAT High)	-	-	-	0.48	-	-	-	-	-	0.62	-	-	360
t1 ALT High − t1 PAT High	-	-	-	−28	-	-	-	-	-	4.5	-	-	361
													362
p(t2 ALT Low > t2 PAT Low)	-	-	-	0.43	-	-	-	-	-	0.57	-	-	363
t2 ALT Low − t2 PAT Low	-	-	-	−48	-	-	-	-	-	1.7	-	-	364
p(t2 ALT High > t2 PAT High)	-	-	-	0.42	-	-	-	-	-	0.56	-	-	365
t2 ALT High − t2 PAT High	-	-	-	−137	-	-	-	-	-	1.8	-	-	366

(continued on next page)

(Table 59 continued – Unfiltered Bayesian Results)

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
program × task × variant × time_or_correctness	**compare: variant**						**compare: variant**						387
p(CO t1 ALT Low > CO t1 PAT Low)	-	-	-	0.61	-	-	-	-	-	0.93	-	-	388
CO t1 ALT Low − CO t1 PAT Low	-	-	-	120	-	-	-	-	-	18.0	-	-	389
p(CO t1 ALT High > CO t1 PAT High)	-	-	-	0.17	-	-	-	-	-	0.43	-	-	390
CO t1 ALT High − CO t1 PAT High	-	-	-	−373	-	-	-	-	-	−2.0	-	-	391
													392
p(CO t2 ALT Low > CO t2 PAT Low)	-	-	-	0.40	-	-	-	-	-	0.56	-	-	393
CO t2 ALT Low − CO t2 PAT Low	-	-	-	−69	-	-	-	-	-	1.6	-	-	394
p(CO t2 ALT High > CO t2 PAT High)	-	-	-	0.67	-	-	-	-	-	0.60	-	-	395
CO t2 ALT High − CO t2 PAT High	-	-	-	135	-	-	-	-	-	3.3	-	-	396
													397
p(GR t1 ALT Low > GR t1 PAT Low)	-	-	-	0.23	-	-	-	-	-	0.80	-	-	398
GR t1 ALT Low − GR t1 PAT Low	-	-	-	−409	-	-	-	-	-	9.1	-	-	399
p(GR t1 ALT High > GR t1 PAT High)	-	-	-	0.78	-	-	-	-	-	0.81	-	-	400
GR t1 ALT High − GR t1 PAT High	-	-	-	317	-	-	-	-	-	10.9	-	-	401
													402
p(GR t2 ALT Low > GR t2 PAT Low)	-	-	-	0.47	-	-	-	-	-	0.58	-	-	403
GR t2 ALT Low − GR t2 PAT Low	-	-	-	−28	-	-	-	-	-	1.8	-	-	404
p(GR t2 ALT High > GR t2 PAT High)	-	-	-	0.17	-	-	-	-	-	0.51	-	-	405
GR t2 ALT High − GR t2 PAT High	-	-	-	−408	-	-	-	-	-	0.3	-	-	406
variant × site	**compare: variant**						**compare: variant**						538
p(ALT BYU > PAT BYU)	0.45	-	-	-	-	-	0.71	-	-	-	-	-	539
ALT BYU − PAT BYU	−66	-	-	-	-	-	8.2	-	-	-	-	-	540
p(ALT FUB > PAT FUB)	0.44	-	-	-	-	-	0.55	-	-	-	-	-	541
ALT FUB − PAT FUB	−145	-	-	-	-	-	1.9	-	-	-	-	-	542
p(ALT UA > PAT UA)	0.45	-	-	-	-	-	0.61	-	-	-	-	-	543
ALT UA − PAT UA	−87	-	-	-	-	-	5.3	-	-	-	-	-	544
p(ALT UPM > PAT UPM)	0.34	-	-	-	-	-	0.58	-	-	-	-	-	545
ALT UPM − PAT UPM	−311	-	-	-	-	-	5.0	-	-	-	-	-	546
program × variant × site	**compare: variant**						**compare: variant**						599
p(CO ALT BYU > CO PAT BYU)	0.39	-	-	-	-	-	0.77	-	-	-	-	-	600
CO ALT BYU − CO PAT BYU	−152	-	-	-	-	-	10.9	-	-	-	-	-	601
p(CO ALT FUB > CO PAT FUB)	0.54	-	-	-	-	-	0.53	-	-	-	-	-	602
CO ALT FUB − CO PAT FUB	74	-	-	-	-	-	1.5	-	-	-	-	-	603
p(CO ALT UA > CO PAT UA)	0.56	-	-	-	-	-	0.60	-	-	-	-	-	604
CO ALT UA − CO PAT UA	80	-	-	-	-	-	4.5	-	-	-	-	-	605
p(CO ALT UPM > CO PAT UPM)	0.23	-	-	-	-	-	0.56	-	-	-	-	-	606
CO ALT UPM − CO PAT UPM	−521	-	-	-	-	-	3.8	-	-	-	-	-	607
													608
p(GR ALT BYU > GR PAT BYU)	0.52	-	-	-	-	-	0.66	-	-	-	-	-	609
GR ALT BYU − GR PAT BYU	20	-	-	-	-	-	5.5	-	-	-	-	-	610
p(GR ALT FUB > GR PAT FUB)	0.34	-	-	-	-	-	0.56	-	-	-	-	-	611
GR ALT FUB − GR PAT FUB	−363	-	-	-	-	-	2.3	-	-	-	-	-	612
p(GR ALT UA > GR PAT UA)	0.34	-	-	-	-	-	0.62	-	-	-	-	-	613
GR ALT UA − GR PAT UA	−254	-	-	-	-	-	6.0	-	-	-	-	-	614
p(GR ALT UPM > GR PAT UPM)	0.44	-	-	-	-	-	0.60	-	-	-	-	-	615
GR ALT UPM − GR PAT UPM	−102	-	-	-	-	-	6.1	-	-	-	-	-	616
task × variant × site	**compare: variant**						**compare: variant**						669
p(t1 ALT BYU > t1 PAT BYU)	0.41	-	-	-	-	-	0.74	-	-	-	-	-	670
t1 ALT BYU − t1 PAT BYU	−125	-	-	-	-	-	9.4	-	-	-	-	-	671
p(t1 ALT FUB > t1 PAT FUB)	0.39	-	-	-	-	-	0.56	-	-	-	-	-	672
t1 ALT FUB − t1 PAT FUB	−277	-	-	-	-	-	2.7	-	-	-	-	-	673
p(t1 ALT UA > t1 PAT UA)	0.52	-	-	-	-	-	0.74	-	-	-	-	-	674
t1 ALT UA − t1 PAT UA	26	-	-	-	-	-	11.2	-	-	-	-	-	675
p(t1 ALT UPM > t1 PAT UPM)	0.43	-	-	-	-	-	0.72	-	-	-	-	-	676
t1 ALT UPM − t1 PAT UPM	−153	-	-	-	-	-	13.5	-	-	-	-	-	677
													678
p(t2 ALT BYU > t2 PAT BYU)	0.50	-	-	-	-	-	0.68	-	-	-	-	-	679
t2 ALT BYU − t2 PAT BYU	−6	-	-	-	-	-	7.1	-	-	-	-	-	680
p(t2 ALT FUB > t2 PAT FUB)	0.48	-	-	-	-	-	0.53	-	-	-	-	-	681
t2 ALT FUB − t2 PAT FUB	−12	-	-	-	-	-	1.1	-	-	-	-	-	682
p(t2 ALT UA > t2 PAT UA)	0.39	-	-	-	-	-	0.48	-	-	-	-	-	683
t2 ALT UA − t2 PAT UA	−199	-	-	-	-	-	−0.6	-	-	-	-	-	684
p(t2 ALT UPM > t2 PAT UPM)	0.24	-	-	-	-	-	0.43	-	-	-	-	-	685
t2 ALT UPM − t2 PAT UPM	−470	-	-	-	-	-	−3.6	-	-	-	-	-	686
program × task × variant × site	**compare: variant**						**compare: variant**						791
p(CO t1 ALT BYU > CO t1 PAT BYU)	0.24	-	-	-	-	-	0.81	-	-	-	-	-	792
CO t1 ALT BYU − CO t1 PAT BYU	−336	-	-	-	-	-	12.5	-	-	-	-	-	793
p(CO t1 ALT FUB > CO t1 PAT FUB)	0.62	-	-	-	-	-	0.53	-	-	-	-	-	794
CO t1 ALT FUB − CO t1 PAT FUB	199	-	-	-	-	-	1.4	-	-	-	-	-	795
p(CO t1 ALT UA > CO t1 PAT UA)	0.56	-	-	-	-	-	0.69	-	-	-	-	-	796
CO t1 ALT UA − CO t1 PAT UA	81	-	-	-	-	-	8.6	-	-	-	-	-	797
p(CO t1 ALT UPM > CO t1 PAT UPM)	0.32	-	-	-	-	-	0.64	-	-	-	-	-	798
CO t1 ALT UPM − CO t1 PAT UPM	−388	-	-	-	-	-	8.8	-	-	-	-	-	799
													800
p(CO t2 ALT BYU > CO t2 PAT BYU)	0.53	-	-	-	-	-	0.73	-	-	-	-	-	801

(continued on next page)

(Table 59 continued – Unfiltered Bayesian Results)

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
CO t2 ALT BYU − CO t2 PAT BYU	33	-	-	-	-	-	9.3	-	-	-	-	-	802
p(CO t2 ALT FUB > CO t2 PAT FUB)	0.45	-	-	-	-	-	0.53	-	-	-	-	-	803
CO t2 ALT FUB − CO t2 PAT FUB	−51	-	-	-	-	-	1.6	-	-	-	-	-	804
p(CO t2 ALT UA > CO t2 PAT UA)	0.57	-	-	-	-	-	0.51	-	-	-	-	-	805
CO t2 ALT UA − CO t2 PAT UA	79	-	-	-	-	-	0.4	-	-	-	-	-	806
p(CO t2 ALT UPM > CO t2 PAT UPM)	0.14	-	-	-	-	-	0.48	-	-	-	-	-	807
CO t2 ALT UPM − CO t2 PAT UPM	−653	-	-	-	-	-	−1.2	-	-	-	-	-	808
													809
p(GR t1 ALT BYU > GR t1 PAT BYU)	0.58	-	-	-	-	-	0.68	-	-	-	-	-	810
GR t1 ALT BYU − GR t1 PAT BYU	86	-	-	-	-	-	6.2	-	-	-	-	-	811
p(GR t1 ALT FUB > GR t1 PAT FUB)	0.16	-	-	-	-	-	0.60	-	-	-	-	-	812
GR t1 ALT FUB − GR t1 PAT FUB	−754	-	-	-	-	-	4.0	-	-	-	-	-	813
p(GR t1 ALT UA > GR t1 PAT UA)	0.48	-	-	-	-	-	0.79	-	-	-	-	-	814
GR t1 ALT UA − GR t1 PAT UA	−30	-	-	-	-	-	13.8	-	-	-	-	-	815
p(GR t1 ALT UPM > GR t1 PAT UPM)	0.54	-	-	-	-	-	0.81	-	-	-	-	-	816
GR t1 ALT UPM − GR t1 PAT UPM	83	-	-	-	-	-	18.2	-	-	-	-	-	817
													818
p(GR t2 ALT BYU > GR t2 PAT BYU)	0.46	-	-	-	-	-	0.64	-	-	-	-	-	819
GR t2 ALT BYU − GR t2 PAT BYU	−45	-	-	-	-	-	4.9	-	-	-	-	-	820
p(GR t2 ALT FUB > GR t2 PAT FUB)	0.51	-	-	-	-	-	0.52	-	-	-	-	-	821
GR t2 ALT FUB − GR t2 PAT FUB	27	-	-	-	-	-	0.7	-	-	-	-	-	822
p(GR t2 ALT UA > GR t2 PAT UA)	0.21	-	-	-	-	-	0.46	-	-	-	-	-	823
GR t2 ALT UA − GR t2 PAT UA	−478	-	-	-	-	-	−1.7	-	-	-	-	-	824
p(GR t2 ALT UPM > GR t2 PAT UPM)	0.34	-	-	-	-	-	0.39	-	-	-	-	-	825
GR t2 ALT UPM − GR t2 PAT UPM	−287	-	-	-	-	-	−6.0	-	-	-	-	-	826

TABLE 60
Filtered Bayesian Results. See Appendix Z for a description of how to read and interpret this table.

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
order	**compare: order**						**compare: order**						11
p(1st > 2nd)	0.96	0.98	0.98	0.97	0.97	0.97	0.13	0.15	0.12	0.15	0.10	0.19	12
1st − 2nd	141	175	180	164	157	142	−3.8	−3.6	−4.5	−3.6	−4.3	−3.1	13
variant	**compare: variant**						**compare: variant**						33
p(PAT > ALT)	0.61	0.58	0.60	0.64	0.63	0.58	0.41	0.41	0.40	0.37	0.33	0.32	34
PAT − ALT	206	104	152	217	133	69	−4.2	−3.5	−3.9	−4.6	−4.9	−4.8	35
program × variant	**compare: variant**						**compare: variant**						36
p(CO PAT > CO ALT)	0.61	0.59	0.63	0.64	0.67	0.50	0.43	0.45	0.42	0.42	0.39	0.40	37
CO PAT − CO ALT	213	129	196	188	195	−2	−3.2	−2.4	−3.2	−2.9	−3.2	−2.7	38
p(GR PAT > GR ALT)	0.60	0.57	0.57	0.63	0.58	0.66	0.39	0.37	0.38	0.31	0.27	0.24	39
GR PAT − GR ALT	200	79	108	245	71	140	−5.2	−4.7	−4.5	−6.3	−6.6	−7.0	40
task × variant	**compare: variant**						**compare: variant**						41
p(t1 PAT > t1 ALT)	0.62	0.57	0.61	0.68	0.64	-	0.35	0.37	0.36	0.29	0.27	-	42
t1 PAT − t1 ALT	242	105	176	328	168	-	−7.5	−5.0	−5.5	−7.3	−6.8	-	43
p(t2 PAT > t2 ALT)	0.59	0.59	0.59	0.59	0.61	-	0.47	0.45	0.44	0.44	0.39	-	44
t2 PAT − t2 ALT	171	103	128	106	98	-	−0.9	−2.1	−2.3	−2.0	−3.0	-	45
program × task × variant	**compare: variant**						**compare: variant**						46
p(CO t1 PAT > CO t1 ALT)	0.67	0.71	0.76	0.81	0.85	-	0.42	0.47	0.40	0.41	0.38	-	47
CO t1 PAT − CO t1 ALT	309	298	398	404	404	-	−3.3	−1.4	−4.1	−3.6	−3.4	-	48
p(GR t1 PAT > GR t1 ALT)	0.58	0.44	0.47	0.55	0.44	-	0.27	0.27	0.33	0.18	0.16	-	49
GR t1 PAT − GR t1 ALT	174	−88	−46	251	−68	-	−11.7	−8.5	−6.9	−10.9	−10.2	-	50
p(CO t2 PAT > CO t2 ALT)	0.56	0.48	0.49	0.47	0.48	-	0.44	0.42	0.44	0.44	0.41	-	51
CO t2 PAT − CO t2 ALT	117	−40	−6	−27	−14	-	−3.2	−3.5	−2.4	−2.2	−2.9	-	52
p(GR t2 PAT > GR t2 ALT)	0.63	0.70	0.68	0.71	0.73	-	0.51	0.47	0.43	0.44	0.38	-	53
GR t2 PAT − GR t2 ALT	225	246	263	239	210	-	1.4	−0.8	−2.2	−1.8	−3.1	-	54
time_or_correctness	**compare: correctness**						**compare: time**						61
p(Low > High)	0.01	0.06	0.11	0.34	0.08	0.13	0.20	0.25	0.26	0.42	0.30	0.23	62
Low − High	−216	−159	−123	−215	−130	−102	−4.0	−2.9	−3.0	−2.2	−2.2	−2.9	63
variant × patKnow	**compare: variant**						**compare: variant**						102
p(ALT Low > PAT Low)	-	0.34	-	-	-	-	-	0.60	-	-	-	-	103
ALT Low − PAT Low	-	−207	-	-	-	-	-	3.6	-	-	-	-	104
p(ALT High > PAT High)	-	0.50	-	-	-	-	-	0.58	-	-	-	-	105
ALT High − PAT High	-	−1	-	-	-	-	-	3.5	-	-	-	-	106
program × variant × patKnow	**compare: variant**						**compare: variant**						117
p(CO ALT Low > CO PAT Low)	-	0.26	-	-	-	-	-	0.48	-	-	-	-	118
CO ALT Low − CO PAT Low	-	−309	-	-	-	-	-	−0.9	-	-	-	-	119
p(CO ALT High > CO PAT High)	-	0.56	-	-	-	-	-	0.63	-	-	-	-	120
CO ALT High − CO PAT High	-	51	-	-	-	-	-	5.7	-	-	-	-	121
													122
p(GR ALT Low > GR PAT Low)	-	0.41	-	-	-	-	-	0.72	-	-	-	-	123
GR ALT Low − GR PAT Low	-	−105	-	-	-	-	-	8.1	-	-	-	-	124
p(GR ALT High > GR PAT High)	-	0.45	-	-	-	-	-	0.53	-	-	-	-	125
GR ALT High − GR PAT High	-	−54	-	-	-	-	-	1.2	-	-	-	-	126
task × variant × patKnow	**compare: variant**						**compare: variant**						137
p(t1 ALT Low > t1 PAT Low)	-	0.37	-	-	-	-	-	0.65	-	-	-	-	138
t1 ALT Low − t1 PAT Low	-	−193	-	-	-	-	-	5.8	-	-	-	-	139
p(t1 ALT High > t1 PAT High)	-	0.48	-	-	-	-	-	0.61	-	-	-	-	140
t1 ALT High − t1 PAT High	-	−17	-	-	-	-	-	4.1	-	-	-	-	141
													142
p(t2 ALT Low > t2 PAT Low)	-	0.30	-	-	-	-	-	0.54	-	-	-	-	143
t2 ALT Low − t2 PAT Low	-	−221	-	-	-	-	-	1.4	-	-	-	-	144
p(t2 ALT High > t2 PAT High)	-	0.52	-	-	-	-	-	0.56	-	-	-	-	145
t2 ALT High − t2 PAT High	-	15	-	-	-	-	-	2.9	-	-	-	-	146
program × task × variant × patKnow	**compare: variant**						**compare: variant**						167
p(CO t1 ALT Low > CO t1 PAT Low)	-	0.16	-	-	-	-	-	0.44	-	-	-	-	168
CO t1 ALT Low − CO t1 PAT Low	-	−504	-	-	-	-	-	−2.1	-	-	-	-	169
p(CO t1 ALT High > CO t1 PAT High)	-	0.43	-	-	-	-	-	0.62	-	-	-	-	170
CO t1 ALT High − CO t1 PAT High	-	−92	-	-	-	-	-	4.9	-	-	-	-	171
													172
p(CO t2 ALT Low > CO t2 PAT Low)	-	0.37	-	-	-	-	-	0.51	-	-	-	-	173
CO t2 ALT Low − CO t2 PAT Low	-	−115	-	-	-	-	-	0.3	-	-	-	-	174
p(CO t2 ALT High > CO t2 PAT High)	-	0.68	-	-	-	-	-	0.64	-	-	-	-	175
CO t2 ALT High − CO t2 PAT High	-	194	-	-	-	-	-	6.6	-	-	-	-	176
													177
p(GR t1 ALT Low > GR t1 PAT Low)	-	0.59	-	-	-	-	-	0.86	-	-	-	-	178
GR t1 ALT Low − GR t1 PAT Low	-	118	-	-	-	-	-	13.7	-	-	-	-	179
p(GR t1 ALT High > GR t1 PAT High)	-	0.53	-	-	-	-	-	0.59	-	-	-	-	180
GR t1 ALT High − GR t1 PAT High	-	57	-	-	-	-	-	3.4	-	-	-	-	181
													182

(continued on next page)

(Table 60 continued – Filtered Bayesian Results)

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
p(GR t2 ALT Low > GR t2 PAT Low)	-	0.23	-	-	-	-	-	0.58	-	-	-	-	183
GR t2 ALT Low − GR t2 PAT Low	-	−328	-	-	-	-	-	2.4	-	-	-	-	184
p(GR t2 ALT High > GR t2 PAT High)	-	0.37	-	-	-	-	-	0.47	-	-	-	-	185
GR t2 ALT High − GR t2 PAT High	-	−165	-	-	-	-	-	−0.9	-	-	-	-	186
variant × devExp	**compare: variant**						**compare: variant**						212
p(ALT Low > PAT Low)	-	-	0.35	-	-	-	-	-	0.70	-	-	-	213
ALT Low − PAT Low	-	-	−224	-	-	-	-	-	7.9	-	-	-	214
p(ALT High > PAT High)	-	-	0.45	-	-	-	-	-	0.50	-	-	-	215
ALT High − PAT High	-	-	−81	-	-	-	-	-	−0.2	-	-	-	216
program × variant × devExp	**compare: variant**						**compare: variant**						227
p(CO ALT Low > CO PAT Low)	-	-	0.31	-	-	-	-	-	0.63	-	-	-	228
CO ALT Low − CO PAT Low	-	-	−294	-	-	-	-	-	5.1	-	-	-	229
p(CO ALT High > CO PAT High)	-	-	0.43	-	-	-	-	-	0.53	-	-	-	230
CO ALT High − CO PAT High	-	-	−98	-	-	-	-	-	1.4	-	-	-	231
													232
p(GR ALT Low > GR PAT Low)	-	-	0.39	-	-	-	-	-	0.78	-	-	-	233
GR ALT Low − GR PAT Low	-	-	−153	-	-	-	-	-	10.8	-	-	-	234
p(GR ALT High > GR PAT High)	-	-	0.46	-	-	-	-	-	0.46	-	-	-	235
GR ALT High − GR PAT High	-	-	−63	-	-	-	-	-	−1.7	-	-	-	236
task × variant × devExp	**compare: variant**						**compare: variant**						247
p(t1 ALT Low > t1 PAT Low)	-	-	0.39	-	-	-	-	-	0.73	-	-	-	248
t1 ALT Low − t1 PAT Low	-	-	−183	-	-	-	-	-	9.2	-	-	-	249
p(t1 ALT High > t1 PAT High)	-	-	0.39	-	-	-	-	-	0.55	-	-	-	250
t1 ALT High − t1 PAT High	-	-	−169	-	-	-	-	-	1.8	-	-	-	251
													252
p(t2 ALT Low > t2 PAT Low)	-	-	0.32	-	-	-	-	-	0.68	-	-	-	253
t2 ALT Low − t2 PAT Low	-	-	−265	-	-	-	-	-	6.7	-	-	-	254
p(t2 ALT High > t2 PAT High)	-	-	0.51	-	-	-	-	-	0.45	-	-	-	255
t2 ALT High − t2 PAT High	-	-	8	-	-	-	-	-	−2.1	-	-	-	256
program × task × variant × devExp	**compare: variant**						**compare: variant**						277
p(CO t1 ALT Low > CO t1 PAT Low)	-	-	0.11	-	-	-	-	-	0.57	-	-	-	278
CO t1 ALT Low − CO t1 PAT Low	-	-	−600	-	-	-	-	-	2.5	-	-	-	279
p(CO t1 ALT High > CO t1 PAT High)	-	-	0.36	-	-	-	-	-	0.64	-	-	-	280
CO t1 ALT High − CO t1 PAT High	-	-	−196	-	-	-	-	-	5.6	-	-	-	281
													282
p(CO t2 ALT Low > CO t2 PAT Low)	-	-	0.51	-	-	-	-	-	0.69	-	-	-	283
CO t2 ALT Low − CO t2 PAT Low	-	-	12	-	-	-	-	-	7.7	-	-	-	284
p(CO t2 ALT High > CO t2 PAT High)	-	-	0.50	-	-	-	-	-	0.42	-	-	-	285
CO t2 ALT High − CO t2 PAT High	-	-	0	-	-	-	-	-	−2.9	-	-	-	286
													287
p(GR t1 ALT Low > GR t1 PAT Low)	-	-	0.66	-	-	-	-	-	0.89	-	-	-	288
GR t1 ALT Low − GR t1 PAT Low	-	-	235	-	-	-	-	-	15.8	-	-	-	289
p(GR t1 ALT High > GR t1 PAT High)	-	-	0.41	-	-	-	-	-	0.46	-	-	-	290
GR t1 ALT High − GR t1 PAT High	-	-	−143	-	-	-	-	-	−2.1	-	-	-	291
													292
p(GR t2 ALT Low > GR t2 PAT Low)	-	-	0.12	-	-	-	-	-	0.66	-	-	-	293
GR t2 ALT Low − GR t2 PAT Low	-	-	−542	-	-	-	-	-	5.8	-	-	-	294
p(GR t2 ALT High > GR t2 PAT High)	-	-	0.51	-	-	-	-	-	0.47	-	-	-	295
GR t2 ALT High − GR t2 PAT High	-	-	16	-	-	-	-	-	−1.3	-	-	-	296
variant × time_or_correctness	**compare: variant**						**compare: variant**						322
p(ALT Low > PAT Low)	-	-	-	0.30	-	-	-	-	-	0.67	-	-	323
ALT Low − PAT Low	-	-	-	−327	-	-	-	-	-	5.6	-	-	324
p(ALT High > PAT High)	-	-	-	0.43	-	-	-	-	-	0.60	-	-	325
ALT High − PAT High	-	-	-	−106	-	-	-	-	-	3.7	-	-	326
program × variant × time_or_correctness	**compare: variant**						**compare: variant**						337
p(CO ALT Low > CO PAT Low)	-	-	-	0.33	-	-	-	-	-	0.65	-	-	338
CO ALT Low − CO PAT Low	-	-	-	−204	-	-	-	-	-	5.4	-	-	339
p(CO ALT High > CO PAT High)	-	-	-	0.39	-	-	-	-	-	0.51	-	-	340
CO ALT High − CO PAT High	-	-	-	−173	-	-	-	-	-	0.4	-	-	341
													342
p(GR ALT Low > GR PAT Low)	-	-	-	0.26	-	-	-	-	-	0.69	-	-	343
GR ALT Low − GR PAT Low	-	-	-	−451	-	-	-	-	-	5.7	-	-	344
p(GR ALT High > GR PAT High)	-	-	-	0.48	-	-	-	-	-	0.70	-	-	345
GR ALT High − GR PAT High	-	-	-	−39	-	-	-	-	-	7.0	-	-	346
task × variant × time_or_correctness	**compare: variant**						**compare: variant**						357
p(t1 ALT Low > t1 PAT Low)	-	-	-	0.19	-	-	-	-	-	0.77	-	-	358
t1 ALT Low − t1 PAT Low	-	-	-	−576	-	-	-	-	-	9.2	-	-	359
p(t1 ALT High > t1 PAT High)	-	-	-	0.45	-	-	-	-	-	0.64	-	-	360
t1 ALT High − t1 PAT High	-	-	-	−79	-	-	-	-	-	5.3	-	-	361
													362
p(t2 ALT Low > t2 PAT Low)	-	-	-	0.40	-	-	-	-	-	0.56	-	-	363
t2 ALT Low − t2 PAT Low	-	-	-	−79	-	-	-	-	-	1.9	-	-	364
p(t2 ALT High > t2 PAT High)	-	-	-	0.42	-	-	-	-	-	0.56	-	-	365
t2 ALT High − t2 PAT High	-	-	-	−133	-	-	-	-	-	2.1	-	-	366

(continued on next page)

(Table 60 continued – Filtered Bayesian Results)

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
program × task × variant × time_or_correctness	**compare: variant**						**compare: variant**						387
p(CO t1 ALT Low > CO t1 PAT Low)	-	-	-	0.26	-	-	-	-	-	0.76	-	-	388
CO t1 ALT Low − CO t1 PAT Low	-	-	-	−331	-	-	-	-	-	9.4	-	-	389
p(CO t1 ALT High > CO t1 PAT High)	-	-	-	0.13	-	-	-	-	-	0.43	-	-	390
CO t1 ALT High − CO t1 PAT High	-	-	-	−477	-	-	-	-	-	−2.2	-	-	391
													392
p(CO t2 ALT Low > CO t2 PAT Low)	-	-	-	0.40	-	-	-	-	-	0.54	-	-	393
CO t2 ALT Low − CO t2 PAT Low	-	-	-	−77	-	-	-	-	-	1.4	-	-	394
p(CO t2 ALT High > CO t2 PAT High)	-	-	-	0.66	-	-	-	-	-	0.58	-	-	395
CO t2 ALT High − CO t2 PAT High	-	-	-	131	-	-	-	-	-	3.0	-	-	396
													397
p(GR t1 ALT Low > GR t1 PAT Low)	-	-	-	0.12	-	-	-	-	-	0.79	-	-	398
GR t1 ALT Low − GR t1 PAT Low	-	-	-	−821	-	-	-	-	-	8.9	-	-	399
p(GR t1 ALT High > GR t1 PAT High)	-	-	-	0.78	-	-	-	-	-	0.85	-	-	400
GR t1 ALT High − GR t1 PAT High	-	-	-	318	-	-	-	-	-	12.8	-	-	401
													402
p(GR t2 ALT Low > GR t2 PAT Low)	-	-	-	0.41	-	-	-	-	-	0.58	-	-	403
GR t2 ALT Low − GR t2 PAT Low	-	-	-	−82	-	-	-	-	-	2.4	-	-	404
p(GR t2 ALT High > GR t2 PAT High)	-	-	-	0.17	-	-	-	-	-	0.54	-	-	405
GR t2 ALT High − GR t2 PAT High	-	-	-	−397	-	-	-	-	-	1.1	-	-	406
variant × site	**compare: variant**						**compare: variant**						538
p(ALT BYU > PAT BYU)	0.39	-	-	-	-	-	0.68	-	-	-	-	-	539
ALT BYU − PAT BYU	−147	-	-	-	-	-	7.0	-	-	-	-	-	540
p(ALT FUB > PAT FUB)	0.43	-	-	-	-	-	0.55	-	-	-	-	-	541
ALT FUB − PAT FUB	−166	-	-	-	-	-	2.2	-	-	-	-	-	542
p(ALT UA > PAT UA)	0.43	-	-	-	-	-	0.61	-	-	-	-	-	543
ALT UA − PAT UA	−140	-	-	-	-	-	6.2	-	-	-	-	-	544
p(ALT UPM > PAT UPM)	0.31	-	-	-	-	-	0.52	-	-	-	-	-	545
ALT UPM − PAT UPM	−372	-	-	-	-	-	1.2	-	-	-	-	-	546
program × variant × site	**compare: variant**						**compare: variant**						599
p(CO ALT BYU > CO PAT BYU)	0.34	-	-	-	-	-	0.65	-	-	-	-	-	600
CO ALT BYU − CO PAT BYU	−228	-	-	-	-	-	6.5	-	-	-	-	-	601
p(CO ALT FUB > CO PAT FUB)	0.54	-	-	-	-	-	0.54	-	-	-	-	-	602
CO ALT FUB − CO PAT FUB	75	-	-	-	-	-	1.7	-	-	-	-	-	603
p(CO ALT UA > CO PAT UA)	0.45	-	-	-	-	-	0.57	-	-	-	-	-	604
CO ALT UA − CO PAT UA	−120	-	-	-	-	-	3.9	-	-	-	-	-	605
p(CO ALT UPM > CO PAT UPM)	0.22	-	-	-	-	-	0.51	-	-	-	-	-	606
CO ALT UPM − CO PAT UPM	−579	-	-	-	-	-	0.7	-	-	-	-	-	607
													608
p(GR ALT BYU > GR PAT BYU)	0.44	-	-	-	-	-	0.70	-	-	-	-	-	609
GR ALT BYU − GR PAT BYU	−66	-	-	-	-	-	7.6	-	-	-	-	-	610
p(GR ALT FUB > GR PAT FUB)	0.32	-	-	-	-	-	0.56	-	-	-	-	-	611
GR ALT FUB − GR PAT FUB	−407	-	-	-	-	-	2.7	-	-	-	-	-	612
p(GR ALT UA > GR PAT UA)	0.40	-	-	-	-	-	0.65	-	-	-	-	-	613
GR ALT UA − GR PAT UA	−160	-	-	-	-	-	8.6	-	-	-	-	-	614
p(GR ALT UPM > GR PAT UPM)	0.41	-	-	-	-	-	0.53	-	-	-	-	-	615
GR ALT UPM − GR PAT UPM	−165	-	-	-	-	-	1.8	-	-	-	-	-	616
task × variant × site	**compare: variant**						**compare: variant**						669
p(t1 ALT BYU > t1 PAT BYU)	0.33	-	-	-	-	-	0.74	-	-	-	-	-	670
t1 ALT BYU − t1 PAT BYU	−243	-	-	-	-	-	9.5	-	-	-	-	-	671
p(t1 ALT FUB > t1 PAT FUB)	0.37	-	-	-	-	-	0.58	-	-	-	-	-	672
t1 ALT FUB − t1 PAT FUB	−316	-	-	-	-	-	3.3	-	-	-	-	-	673
p(t1 ALT UA > t1 PAT UA)	0.41	-	-	-	-	-	0.66	-	-	-	-	-	674
t1 ALT UA − t1 PAT UA	−156	-	-	-	-	-	9.3	-	-	-	-	-	675
p(t1 ALT UPM > t1 PAT UPM)	0.39	-	-	-	-	-	0.63	-	-	-	-	-	676
t1 ALT UPM − t1 PAT UPM	−251	-	-	-	-	-	7.7	-	-	-	-	-	677
													678
p(t2 ALT BYU > t2 PAT BYU)	0.46	-	-	-	-	-	0.61	-	-	-	-	-	679
t2 ALT BYU − t2 PAT BYU	−51	-	-	-	-	-	4.5	-	-	-	-	-	680
p(t2 ALT FUB > t2 PAT FUB)	0.48	-	-	-	-	-	0.52	-	-	-	-	-	681
t2 ALT FUB − t2 PAT FUB	−16	-	-	-	-	-	1.1	-	-	-	-	-	682
p(t2 ALT UA > t2 PAT UA)	0.44	-	-	-	-	-	0.56	-	-	-	-	-	683
t2 ALT UA − t2 PAT UA	−124	-	-	-	-	-	3.1	-	-	-	-	-	684
p(t2 ALT UPM > t2 PAT UPM)	0.24	-	-	-	-	-	0.41	-	-	-	-	-	685
t2 ALT UPM − t2 PAT UPM	−493	-	-	-	-	-	−5.2	-	-	-	-	-	686
program × task × variant × site	**compare: variant**						**compare: variant**						791
p(CO t1 ALT BYU > CO t1 PAT BYU)	0.18	-	-	-	-	-	0.78	-	-	-	-	-	792
CO t1 ALT BYU − CO t1 PAT BYU	−469	-	-	-	-	-	11.7	-	-	-	-	-	793
p(CO t1 ALT FUB > CO t1 PAT FUB)	0.62	-	-	-	-	-	0.55	-	-	-	-	-	794
CO t1 ALT FUB − CO t1 PAT FUB	209	-	-	-	-	-	2.2	-	-	-	-	-	795
p(CO t1 ALT UA > CO t1 PAT UA)	0.28	-	-	-	-	-	0.45	-	-	-	-	-	796
CO t1 ALT UA − CO t1 PAT UA	−393	-	-	-	-	-	−2.3	-	-	-	-	-	797
p(CO t1 ALT UPM > CO t1 PAT UPM)	0.25	-	-	-	-	-	0.52	-	-	-	-	-	798
CO t1 ALT UPM − CO t1 PAT UPM	−584	-	-	-	-	-	1.4	-	-	-	-	-	799
													800
p(CO t2 ALT BYU > CO t2 PAT BYU)	0.51	-	-	-	-	-	0.53	-	-	-	-	-	801

(continued on next page)

(Table 60 continued – Filtered Bayesian Results)

	T1	T2	T3	T4	T5	T6	C1	C2	C3	C4	C5	C6	
CO t2 ALT BYU − CO t2 PAT BYU	13	-	-	-	-	-	1.3	-	-	-	-	-	802
p(CO t2 ALT FUB > CO t2 PAT FUB)	0.45	-	-	-	-	-	0.53	-	-	-	-	-	803
CO t2 ALT FUB − CO t2 PAT FUB	−60	-	-	-	-	-	1.2	-	-	-	-	-	804
p(CO t2 ALT UA > CO t2 PAT UA)	0.62	-	-	-	-	-	0.69	-	-	-	-	-	805
CO t2 ALT UA − CO t2 PAT UA	153	-	-	-	-	-	10.1	-	-	-	-	-	806
p(CO t2 ALT UPM > CO t2 PAT UPM)	0.18	-	-	-	-	-	0.50	-	-	-	-	-	807
CO t2 ALT UPM − CO t2 PAT UPM	−574	-	-	-	-	-	0.0	-	-	-	-	-	808
													809
p(GR t1 ALT BYU > GR t1 PAT BYU)	0.49	-	-	-	-	-	0.70	-	-	-	-	-	810
GR t1 ALT BYU − GR t1 PAT BYU	−18	-	-	-	-	-	7.3	-	-	-	-	-	811
p(GR t1 ALT FUB > GR t1 PAT FUB)	0.12	-	-	-	-	-	0.61	-	-	-	-	-	812
GR t1 ALT FUB − GR t1 PAT FUB	−842	-	-	-	-	-	4.5	-	-	-	-	-	813
p(GR t1 ALT UA > GR t1 PAT UA)	0.54	-	-	-	-	-	0.87	-	-	-	-	-	814
GR t1 ALT UA − GR t1 PAT UA	81	-	-	-	-	-	21.0	-	-	-	-	-	815
p(GR t1 ALT UPM > GR t1 PAT UPM)	0.54	-	-	-	-	-	0.74	-	-	-	-	-	816
GR t1 ALT UPM − GR t1 PAT UPM	82	-	-	-	-	-	14.0	-	-	-	-	-	817
													818
p(GR t2 ALT BYU > GR t2 PAT BYU)	0.40	-	-	-	-	-	0.70	-	-	-	-	-	819
GR t2 ALT BYU − GR t2 PAT BYU	−115	-	-	-	-	-	7.8	-	-	-	-	-	820
p(GR t2 ALT FUB > GR t2 PAT FUB)	0.52	-	-	-	-	-	0.52	-	-	-	-	-	821
GR t2 ALT FUB − GR t2 PAT FUB	29	-	-	-	-	-	0.9	-	-	-	-	-	822
p(GR t2 ALT UA > GR t2 PAT UA)	0.27	-	-	-	-	-	0.43	-	-	-	-	-	823
GR t2 ALT UA − GR t2 PAT UA	−401	-	-	-	-	-	−3.8	-	-	-	-	-	824
p(GR t2 ALT UPM > GR t2 PAT UPM)	0.29	-	-	-	-	-	0.32	-	-	-	-	-	825
GR t2 ALT UPM − GR t2 PAT UPM	−412	-	-	-	-	-	−10.4	-	-	-	-	-	826