Less is More? Investigating the Role of Examples in Security Studies using Analogical Transfer

Ashwini Rao[1], Hanan Hibshi[1,2], Travis Breaux[1]
Institute for Software Research,
Carnegie Mellon University[1]
College of Computing,
King Abdul-Aziz University[2]
{arao,hhibshi,breaux}@cmu.edu

Jean-Michel Lehker, Jianwei Niu
Department of Computer Science
The University of Texas at San Antonio
j.lehker@gmail.com, niu@cs.utsa.edu

ABSTRACT

Information system developers and administrators often overlook critical security requirements and best practices. This may be due to lack of tools and techniques that allow practitioners to tailor security knowledge to their particular context. In order to explore the impact of new security methods, we must improve our ability to study the impact of security tools and methods on software and system development. In this paper, we present early findings of an experiment to assess the extent to which the number and type of examples used in security training stimuli can impact security problem solving. To motivate this research, we formulate hypotheses from analogical transfer theory in psychology. The independent variables include number of problem surfaces and schemas, and the dependent variable is the answer accuracy. Our study results do not show a statistically significant difference in performance when the number and types of examples are varied. We discuss the limitations, threats to validity and opportunities for future studies in this area.

Categories and Subject Descriptors

H.1.2 [**Information Systems**]: User/Machine Systems—*human factors*; D.2 [**Software Engineering**]: Miscellaneous

General Terms

Security

Keywords

Security; Human Factors; Psychology; Analogical Transfer

1. INTRODUCTION

Security research often aims to evaluate whether a particular stimulus improves security. Examples of stimuli include context-appropriate messages such as pop-ups and warnings, displays, and user training to identify phishing and site spoofing. Researchers use experiments to evaluate the effectiveness of stimuli and they may use a training phase during the experiment to prepare participants for the study before testing participant performance in a later phase of the experiment. During the training phase, participants are shown examples that describe the stimulus, e.g., participants may be shown examples of e-mails from prior phishing attacks to later test the effectiveness of training on reducing their vulnerability to phishing [37].

Selecting examples is challenging as the choice of examples can influence participants in ways that skew the study results. While choosing examples, investigators have to consider the number of examples to use for training. Increasing the number of examples used in training provides more learning opportunities. However, using more examples may increase cognitive load and participants may abandon the study due to fatigue. Further, increasing the number of examples may increase time required for training, which reduces the time available for the subsequent study.

In addition, investigators must decide how to choose examples to show participants. When selecting fake URLs for a phishing study, an investigator may consider URLs containing the domain names "www.mycitibank.com" and "www.chase.c.com." Both the URLs use fake subdomain names, that is, "mycitibank" and "c". Citibank is a well-known bank and prefixing "my" to the domain name fits an existing user expectation for personalization in software. However, the domain suffix "c" may be difficult for users to visually recognize. The training implication of choosing one of these examples may limit a participants' ability to detect phishing e-mails.

To investigate the effect that number and types of examples play in security studies, we designed an experiment using the theory of analogical transfer from cognitive psychology. Analogical transfer is a problem solving strategy where a person applies knowledge gained from past examples to solve a problem [32]. In our experiment, we consider examples from the domain of secure coding to test several hypotheses based on analogical transfer.

Our approach and results inform the science behind choosing examples for stimulus training in security studies. In addition to the number and types of examples, factors such as individual performance (subject-to-subject variability) [22], experience [16, 26], motivation [7] and cognitive load [40]

HotSoS '14, April 08 - 09 2014, Raleigh, NC, USA

http://dx.doi.org/10.1145/2600176.2600182.

may affect problem solving. It is difficult to account for the impact of all these factors. As we explain in Section 3 on methodology, we consider some of these factors, for example experience, in our experimental design.

The rest of the paper is organized as follows: in Section 2, we review the theory of learning by analogy, and formulate hypotheses to test the effect of training examples; in Section 3, we present our research method and experimental design; in Section 4, we present our study results with threats to validity and limitations in Section 5; related work in Section 6; and we conclude in Section 7.

2. PSYCHOLOGY AND HYPOTHESES

When training users in experimental studies, the objective is to facilitate learning and problem solving through the application of previously acquired knowledge. In an experimental context, the user is shown examples to help him or her learn a given concept, for example, by showing examples of secure and insecure coding patterns. After learning the concept, the investigator tests the user's ability to apply the concept to solve similar and different problems. In Section 2.1, we briefly review theories from psychology that aim to explain learning and problem solving using examples. In Section 2.2, we present hypotheses based on analogical transfer theory to test the effect of examples in stimulus training.

2.1 Analogy, Learning and Problem Solving

Problem solving using analogies involves applying knowledge gained from previously encountered problems to solve a new problem of the same kind; this problem solving strategy is called analogical transfer [32]. The previously encountered or familiar problem is called the base analogue and the new problem is called the target analogue. Mathematician Polya has argued in his seminal work, "How to solve it," that problem solving by analogy is a dominant method for solving new problems [28].

Problems can be decomposed into the surface elements and the abstract or structural details [32]. Surface elements are the superficial features of how we describe the problem. The structural details, also called the schema, refer to the solution principle. For example, given a computer program in a programming language, the surface elements include the syntax of the language and structural details include the algorithm that the program implements. Problems from two different domains may have different surface elements, but the same structure for solving a problem. For example, the divide and conquer principle can be used to solve problems in programming or in politics. Research in domains such as physics has shown that experts more often than novices use structural details to represent problems [8]. The theory of analogical transfer includes multiple stages in cognition, including retrieval of the base analogue and mapping the base analogue's schema and/or surface elements to the target analogue. Theories of analogical transfer differ in how they consider the relative importance of surface and schema. For example, theories may state that mapping involves only the schema, only the surface, or a combination of both [32, 15, 20, 19, 24]. In other words, analogical transfer may be applied to two problems that overlap in schema, surface or both. During the mapping stage, with increase in domain experience, the use of schema information increases and use of surface information decreases [16]. Further, research in domains such as mathematics shows that experience improves the effectiveness of problem solving by analogy [26]. Empirical study results on analogical transfer agree that schema induction, which is the process of abstracting structural details from a set of examples, does occur [32]. Largely, the study findings agree that schema induction is unlikely with a single example, however, the findings disagree on the minimum number of examples required for schema induction. Schema induction with small number of examples is not effective unless we employ special strategies, such as providing sufficient explanation of the schema [1]. Research in cognitive load theory shows that using worked-out examples, which are problems presented with complete solutions, improves schema induction especially in novices [27]. Individual differences, such as use of self-explanations in learning from worked examples can affect schema induction [33]. The type of problem representation employed may increase cognitive load thereby reducing schema induction [40]. Finally, if people exert effort to learn the abstract principles present in the examples, for example by using self-explanations, then schema induction improves [7].

2.2 Testing the Role of Examples

Based on the theory of learning from analogy, we formulate hypotheses to test the effect of the number and types of examples in stimulus training. Due to the timing limitations that constrain stimulus training in many studies, we consider training with 2-3 examples. For the type of examples, we distinguish between the surface elements and structural details. For example, URLs mychase.com and ez-trade.com do not have similar surface, but they share the same schema (fake, misleading subdomain). We hypothesize that the number of varieties of surfaces and schemas within a set of training examples affects user performance on different training goals as follows:

$H1_{null}$*:Users that train on more surfaces for the given schema exhibit no difference in performance than users that train on fewer surfaces for the same schema.*

$H1_{alt}$*: Users that train on more surfaces for the given schema perform differently than users that train on fewer surfaces for the same schema.*

In hypothesis $H1$, we consider the effect of number of different surface types for a single type of schema. During participant training, participants may be able to abstract out the common schema across these examples. Consider a scenario where the goal is to train the user to apply the acquired knowledge to a new example with the same schema but new type of surface. A sample scenario may be training the user to use an ATM interface; the user may encounter an ATM interface that looks different (new surface), but criteria for using it is the same (same schema). For this goal, participants who see more examples with different surfaces for a single schema may perform differently than participants who see fewer surfaces for the schema. One reason for the difference could be that participants who see more surfaces have a better chance of abstracting out the schema. If a statistically significant difference exists in performance, we would like to understand if performance is better or worse.

In our second hypothesis $H2$, we consider the effect of number of schema types. During participant training, participants may be able to perceive the difference between different schemas. For example, consider a training scenario where the goal is to train the user to identify phishing at-

tacks, which look different (i.e., each with a different new surface) as well as they vary in the actual strategy used to compromise the user's machine (new schema). Given this goal, participants who see multiple schemas, each with a different surface type may perform differently than participants who see fewer schemas each with a different surface type. One reason for the difference could be that users who see more schemas are prepared to encounter new schemas. If a statistically significant difference exists in performance, we would like to understand if performance is better or worse. We state our second null and alternative hypotheses as follows:

$H2_{null}$: Users that train on more schemas exhibit no difference in performance than users that train on fewer schemas.
$H2_{alt}$: Users that train on more schemas perform differently than users that train on fewer schemas.

3. METHODOLOGY

We conducted a human-subjects study to evaluate our hypotheses regarding the effect of number and type of examples in security studies. We conducted our study using an online survey, which we now discuss.

3.1 Targeted Subjects and their Recruitment

We recruited participants who enrolled in at least one course in computer security or had multiple years of industry experience in computer security. We recruited participants from three universities using in-class announcements and private mailing lists for security courses. Our participant pool consists of undergraduate and graduate students from the three universities.

Each participant that completed the study received an Amazon gift card. We informed the participants, before they started the study, that they would be compensated $10 if they scored 70% or higher, and $5 otherwise. We based the compensation on performance to motivate the participants to think carefully about their answers. However, for the actual compensation, we paid participants who provided thoughtful responses to at least two out of the three test problems, regardless of the correctness of their responses.

3.2 Experimental Design

We designed our experiment based on the theory of analogical transfer discussed in Section 2.1. Our experiment consisted of two phases: the training phase and the test phase. In the training phase, we used a training stimulus consisting of a set of problems along with their solutions. In the testing phase, we tested the effectiveness of our training, that is, whether participants were able to acquire the relevant knowledge. As we are interested in understanding the role of examples in security studies, we used problems from a security domain, specifically, secure coding. Recall that our hypotheses $H1$ and $H2$ from Section 2.2 test whether the schema and surface types used in the training stimulus cause a statistically significant difference in participant performance on test problems. We use a between-subjects design for the study. In our experiment, the independent variables were the number and types of schemas and surfaces used in the training stimulus. The dependent variable was participant performance. As we discuss in Section 3.8, we measured performance by counting the number of correct answers provided by participants for each test problem.

To test our hypothesis $H1$ and $H2$, we varied the number and types of schemas and surfaces used in the training stimulus. To test the effect of number of examples, we assigned participants to one of the two groups: 3-Example group or 2-Example group. To reduce the effect of confounding variables, we used a double-blind study design and randomly assigned participants to groups. In the 3-Example group, participants trained on three problems. In the 2-Example group, participants trained on two problems. In the test phase, all participants had to solve three test problems. Table 1 lists the order in which we presented training and test problems to the participant. Problems are labeled by a letter followed by a number; the letter denotes the schema and number denotes the surface, e.g., A1 and A2 use the same schema but with a different surface. During the training phase, participants in the 3-Example group saw three problems, A1, A2 and A3, of the same schema A. The 2-Example group saw two problems A1 and B6 with different schemas A and B. In the testing phase, participants in both groups saw three test problems, A4, C7 and A5.

Table 1: Organization of training and test problems

	3-Example Group	2-Example Group
Training	A1 (*C*)	A1 (*C*)
Problems	A2 (*C*)	B6 (*C*)
	A3 (*Java*)	
Test	A4 (*PHP*)	A4 (*PHP*)
Problems	C7 (*C*)	C7 (*C*)
	A5(*"text"*)	A5(*"text"*)

3.3 Schema and Surface

We chose schemas and surfaces from the secure coding domain. As listed in Table 1, we used three schemas A, B and C described below.

A. Error handling: describes a security practice in source code where the programmer needs to handle a prospective error or exceptional case to avoid failures

B. Checking input values: describes a security practice in source code where the programmer needs to check that values input to a function conform to certain assumptions, e.g., no null values, numbers are within a prescribed range, etc.

C. Integer security: describes a security practice in source code where the programmer needs to handle error conditions that can occur due to the finite representation of integers

In total, we used seven surfaces. Schema A was presented using five surfaces (A1, A2, A3, A4, A5), schemas B using one surface (B6), and C using one surface (C7). Surfaces 1-4, 6 and 7 consist of source code expressed in C, PHP, or Java programming language. Surface 5 consists of natural language text in English language.

3.4 Evaluating Hypothesis $H1$ and $H2$

Recall from Section 2.2 that hypothesis $H1$ tests the effect of training participants on multiple surfaces for a single

schema. $H1$ checks whether there is a statistically significant difference in performance when one group trains on examples with more surfaces as opposed to fewer surfaces. To evaluate $H1$, we trained participants in the 3-Example group with three surfaces (A1, A2, A3) and participants in the 2-Example group with one surface (A1) for the same schema A. We measured participant performance in both groups on test problems A4 and A5 that use the same schema A. Test problem A4 expresses the surface using source code, while test problem A5 expresses the surface using natural language text. By choosing code and text descriptions, we intended to examine the effect of different surface representations.

Hypothesis $H2$ tests the effect of training participants on multiple schemas. $H2$ tests if there is a statistically significant difference in performance when one group trains on more schemas than the other group. To evaluate $H2$, we trained participants in the 3-Example group with the single schema A and participants in the 2-Example group with two schemas A and B. We measured participant performance using test problem C7 based on a schema C. We did not show schema C to participants during the training phase.

3.5 Training and Test Problems

For the training phase, we used worked-out examples. Worked-out examples can reduce cognitive load and improve schema induction, and hence improve effectiveness of the training stimulus [27]. Training problems were based on the security schemas discussed in Section 3.3. Schema A concerns error handling, schema B concerns checking input values, and schema C concerns integer security. Problems A1, A2 and B6 were in the C programming language, and A3 was in Java programming language. For each problem, we showed a code snippet that contained one or more lines of code that violated the security principle described by the schema. For example, consider the following code snippet from problem A1 based on the error handling schema.

```
1 /* function to copy network data to a file */
2 int copyData(struct connection *c, FILE *fp) {
3   char buf[200];
4   readFromNetwork(c, buf, 100);
5   writeToFile(fp, buf, 100);
6   /* ... */
7 }
```

Training Problem A1

In line 4, the code does not check if the `readFromNetwork` function returned an error. Similarly code snippet in A2 did not check whether memory allocation function returned an error, and A3 did not check whether reading from an object returned an error. Code snippet in B6 did not properly validate argument passed to a function. During training, we explained the schema to the participant using a question/answer format. The format consisted of the following questions: "What is the problem?", "What are the consequences of the problem?" and "What changes do you make to prevent the problem?" Details of the training examples are in Appendix A.

We showed the test problems to the participants using the same format employed for training, that is, code snippet followed by questions. However, we did not show the answers. Participants were required to provide open-ended responses. Test problems were also based on the security schemas described in Section 3.3. Problem A4 was in PHP web scripting language, C7 was in the C programming language and A5 was in English language. Test problem A4 contained a line of code that did not perform error handling after reading a record from a database. Problem C2 contained code that added the lengths of two integers and stored the sum in a short integer, which could produce a truncation error. Problem A5 described a server patching procedure where the administrator failed to check whether a new patch was successfully installed. Details of the test problems are in Appendix A.

3.6 Participant Background

We collected information about participants' background at the end of our study to avoid biasing participants and to reduce the survey non-response rate [34]. The collected background information includes industry experience in computer programming, experience using specific programming languages, level of education, and specific computer courses undertaken (see Appendix A for background questions). In our analysis, we counted summer jobs and internships as industry experience. We used background information to check for possible correlation between experience and participant performance on test problems. We did not ask participants about their age, gender or race.

3.7 Study Deployment

We implemented our study as an online survey using Survey Gizmo platform. We configured Survey Gizmo platform to randomly assign participants to either the 3-Example group or the 2-Example group. We provided a link to our survey in the recruitment email. After providing informed consent, participants completed the training and test phases of the study. During the training phase, to encourage participants to read the solution for a training problem, we provided a "show solution" link. Participants were required to click on the link before they could proceed to the next training problem. Although this feature did not guarantee that participants read the solution, it provides us information about at what point a participant exited each question. We used this information to analyze incomplete survey responses. We ran a pilot study on three computer science graduate students with background in computer security, and based on their feedback, we added explanatory comments to the training and test problems.

3.8 Grading of Participant Responses

The first and second authors coded and graded participant answers using an iterative approach, wherein they discussed disagreements until they reached consensus. Using a predefined answer key for each open-ended answer, the graders assigned one code [0: unintended answer, 1: intended answer]. Answers not part of the predefined answer key were considered unintended answers. If a participant identified the error in a test problem and provided a relevant solution, then the participant's answer was coded 1, otherwise it was coded 0. We used Cohen's Kappa [9] to compute inter-rater reliability (κ=0.85 after the first pass).

For unintended answers, the graders performed axial coding by assigning a second, exclusive code to distinguish types of unintended answers. This approach is based on grounded analysis, wherein the codes emerge from the dataset [11]. Because the graders devised new codes as they encounter new answer types, the second-level coding was performed in

two passes: the first pass was to discover the codes, and the second pass was to ensure that the complete code set was consistent across the entire dataset. The second-level codes assigned by the graders to the unintended answers were as follows:

- **Incorrect:** the answer was not a correct solution and did not address any errors present in the test problem. Answers were marked as incorrect, if participants identified errors not present or if they provided incorrect solutions. For example, a participant indicated that problem A5 dealing with patching was downloading patches from an insecure site. However, this was not correct as the problem indicated that patches were downloaded from a secure, verifiable server.

- **Extended-out-of-Context:** the answer was correct based on a participant's assumption that was outside the context of the problem description. In other words, the assumption was not directly supported by explicit information contained within the training examples or problem description. For example, in test problem A5, a participant identified the need to use different patches for different server operating systems. We considered this out-of-context as the problem described a single server scenario.

- **Extended-in-Context:** the answer was correct based on a best practice schema that we had not anticipated. The answer was applicable based on the amount of information contained within the context of problem description. For example, a participant identified the need to use strong access control while accessing database records in test problem A4.

Our grading resulted in multiple categorical assignments for a single response: a response could include an intended, correct answer, as well as an incorrect answer and an extended answer (in or out-of context). Only intended answers counted toward the correct answers and these excluded the extended-in-context and extended-out-of-context answers, which were technically correct but did not match our intended answers. Our intention was to see if participants had more extended answers for a certain test problem, and if the extended answers could distract participants from providing the intended answers for the given schema/ surface combination. Lastly, as part of grading, we computed compensation, $5 or $10, for participants.

3.9 Statistical Tests

Hypotheses $H1$ and $H2$ were evaluated using categorical data, that is, graded participant responses. Normally, we use contingency tables and non-parametric Pearson's Chi Square Test of Independence. However, this test requires at least five responses (frequencies) in each category in the contingency table[2, 22, 3]. For data sets with less than five responses in any category, Fisher's Exact test is recommended [2, 22, 3]. In our analysis, we applied Fisher's Exact test to our 2x2 contingency table: Group Assignment (2-Example, 3-Example) x Performance (Intended Answer, Unintended Answer). We also applied the Fisher's Exact test to participant experience responses, which is a 2x5 contingency table: Group Assignment (2-Example, 3-Example) x Experience Level (0, <1, 1, 1-3, >3 years).

To compare the time taken by participants to complete the testing phase, we used Wilcoxon-Mann-Whitney non-parametric test. As time is a continuous variable, we can use Wilcoxon-Mann-Whitney test. Analysis of our data using histogram and Shapiro-Wilk test [36] showed that the data was not normally distributed ($p - value = 2.505e - 12$), which is a supported assumption under the Wilcoxon-Mann-Whitney test [38].

4. ANALYSIS AND RESULTS

We had 124 participants in our study, and 80 participants completed the study. The 3-Example group had 54 participants (33 complete, 21 incomplete), and the 2-Example group had 70 participants (47 complete, 23 incomplete). We had unequal number of participants in the two groups due to the random assignment algorithm used by Survey Gizmo platform. This difference would decrease with increase in sample size. All participants who completed the survey provided thoughtful responses to at least two test problems and were compensated $10. Hence, we did not have participants that were compensated $5. It appears that participants either tried to get the full $10 compensation or decided to drop-out without completing the study. In the remainder of this section, we discuss the results from our study based on the 80 complete responses. We discuss participant dropout based on the 44 incomplete responses in Section 4.6.

4.1 Participant Experience and Background

Out of the 80 participants who completed the study, 52 students had graduate or doctoral degrees (20 in 3-Example and 32 in 2-Example). The remaining 28 students in the sample were undergraduates: 16 had 3-4 years of college or higher (8 in both 3-Example and 2-Example), and 12 students had 1-2 years of college experience (5 in 3-Example and 7 in 2-Example).

In Table 2, we list how many of our participants had completed courses related to programming languages, data structures, computer security and secure programming. Note that the percentages in Table 2 do not add up to 100% as participants could have taken more than one course from the list.

Table 2: Participant course background

Course	3-Example		2-Example		Total	
	#	%	#	%	#	%
Programming languages	30	38%	45	56%	75	94%
Data structures	26	33%	42	53%	68	85%
Computer security	15	19%	25	31%	40	50%
Secure programming	5	6%	6	8%	11	14%

#: number %: percentage

Table 3 lists the participants' programming experience in the industry. We included summer jobs and internships as industry experience. Figure 1 shows participants' experience with different programming languages.

Analysis of participants' background and experience data from the 3-Example and 2-Example groups showed no statistically significant difference between the two groups (Fisher's Exact test, $p - value = 0.529$). Hence, both the groups had participants with similar course background and program-

Table 3: Participant programming experience in industry (years)

Experience	3-Example		2-Example		Total	
	#	%	#	%	#	%
No experience	3	4%	14	18%	17	21%
Less than 1	7	9%	13	16%	20	25%
One year	2	3%	13	16%	15	19%
Between 1 and 3	3	4%	10	13%	13	16%
Three or more	2	3%	13	16%	15	19%

#: number %: percentage

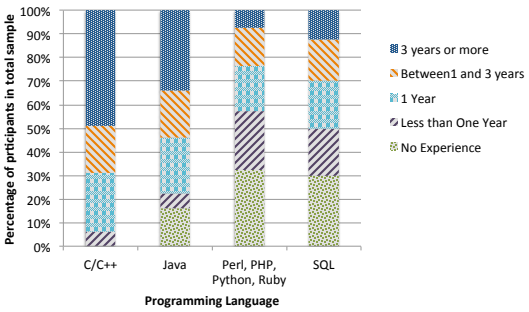

Figure 1: Participant programming language experience

ming experience. We will elaborate on the effect of background on performance in Section 4.4.

4.2 Completion Time

We recorded the overall completion time participants took to complete the survey. Completion time includes time spent on training and testing phases. In online surveys, participants may carry out additional tasks. Hence, we cannot accurately compute completion time. To estimate completion time, we start by excluding outliers. The cutoff for outliers was chosen by plotting a histogram of the total time spent by all participants. With a 100-minute cut-off, we had seven outliers out of 80 values. After removing the outliers, the median total time for participants was 23 minutes for participants in the 3-Example group, and 22 minutes for participants in the 2-Example group.

As discussed in Section 3.9, our data is not normally distributed. Hence, we used non-parametric Wilcoxon-Mann-Whitney test to compare the median completion time of the participants in the 3-Example and 2-Example groups. We did not observe statistically significant difference between the two groups (Wilcoxon-Mann-Whitney test $p - value = 0.56$).

4.3 Participant Performance

We analyzed participant performance on test problems A4, C7 and A5 described in Section 3.5. Recall that we use the test problems to evaluate hypotheses $H1$ and $H2$ regarding the effect of number and types of examples in security studies. We compared performance of participants in the 3-Example and 2-Example groups to identify any statistically significant difference that can be attributed to the difference in the number and types of examples in the two groups. As we discussed in Section 3.8, we measured performance by counting the number of intended and unintended answers provided by participants for each test problem.

4.3.1 *Effect of Using More Surfaces per Schema*

Hypothesis $H1$ from Section 2.2 aims to determine whether training using more surfaces per schema causes a change in participant performance. Recall from Section 3.4 that we use our first test problem A4, based on schema A, to evaluate $H1$. Participants in the 3-Example group trained on three surfaces for schema A, and participants in the 2-Example group trained on one surface for schema A. Hence, participants in the two groups may perform differently in identifying schema A represented using a new surface. Theories of analogical transfer discussed in Section 2.1 indicate that training with more surfaces may improve schema induction. Hence, we expected that participants in the 3-Example group may perform better than participants in the 2-Example group. Although participants in the 3-Example group gave more intended answers (30/33 or 90.9%) than participants in the 2-Example group (39/47 or 82.9%), the difference was not statistically significant (Fisher's Exact test, $p-value = 0.51$). As the p-value is large, we cannot reject the null hypothesis $H1_{null}$. Table 4 lists the numbers of participants who gave intended answers versus unintended answers for test problem A4.

Table 4: Participants responses for the three test problems

	3-Example		2-Example	
	YES	NO	YES	NO
A4 (test effect of more surfaces per schema)	30	3	39	8
C7 (test effect of more schemas)	18	15	21	26
A5 (test effect of different surface representations)	11	22	14	33

YES:Intended answer NO:Not-intended answer

4.3.2 *Effect of Using More Schemas*

Hypothesis $H2$ from Section 2.2 tests whether training with more schemas causes a change in participant performance. Recall from Section 3.4 that we use our second test problem C7, based on schema C, to evaluate $H2$. Participants in the 3-Example group trained on one schema A, and participants in the 2-Example group trained on two schemas, A and B. Hence, participants in the two groups may perform differently in identifying a new schema C. Further, we expected that participants in the 2-Example group may perform better than participants in the 3-Example group as they trained on more schemas. However, participants in the 3-Example group performed better (18/33 or 54.5%) than participants in the 2-Example group (21/47 or 44.6%), but the difference was not statistically significant (Fisher's Exact test, $p - value = 0.50$). Because of the large p-value, we cannot reject the null hypothesis $H2_{null}$. Table 4 shows the performance of participants in the two groups on test problem C7.

4.3.3 *Effect of Different Surface Representations*

Recall from Section 3.4 that we use natural language text to represent the surface for our third and last test problem A5, which is based on schema A. Our intention is to examine

hypothesis $H1$, but using different surface representations. The first test problem A4 also tested hypothesis $H1$, but used a source code representation instead of a text representation. As we discussed in Section 4.3.1, we expected that participants in the 3-Example group may perform better than participants in the 2-Example group as they trained on more surfaces for schema A. From Table 4, we can see that participants in the 3-Example group performed only slightly better (11/33 or 33.3%) than participants in the 2-Example group (14/47 or 29.7%), but the difference was not statistically significant (Fisher's Exact test, $p - value = 0.80$).

Further, comparing the performance of participants on problems A4 and A5, we see that participants gave more unintended answers for problem A5. Compare the number of unintended answers on the two problems from Table 4. Participants in the 3-Example group gave three unintended answer or A4 and 22 unintended answers for A5. Similarly, participants in the 2-Example group gave 8 unintended answers for A4 and 33 unintended answers for A5. This difference in performance may be due to the difficulty of reading natural language text description or the opportunity for inferring additional information from ambiguities present in text.

Since we did not randomize the order of the test problems, we cannot rule out fatigue effect. Fatigue may explain the increase in the number of unintended answers on the last test problem A5. However, analysis of the unintended answers shows that participants provided more answers that are extended in context for test problem A5 than test problem A4. Recall from Section 3.8 that answers extended in context are unintended, but not incorrect. For test problem A5 (text representation), 19 participants (6 in 3-Example group and 13 in 2-Example group) spotted possible security problems that are extended in context. For test problem A4 (code representation) 17 participants (7 in 3-Example group and 10 in 2-Example group) spotted possible security problems that are extended in context. There is no statistically significant difference between the performance of the two groups regarding answers extended in context. We need to further investigate the reasons for the increase in answers extended in context, which could be considered "out-of-the-box" thinking from participants.

4.4 Programming Experience and Participant Performance

We analyzed the correlation between participant performance and participant programming experience in the industry (Table 3). As discussed in Section 2.1, problem solving by analogy may improve with experience. Table 5 shows the performance of participants in the 3-Example and 2-Example groups broken down by their programming experience. From the table, performance on the three test problems appears similar. We did not find statistically significant difference between the groups for performance on the three test problems broken down by experience (Fisher's Exact test, p-values 0.48, 0.95, and 0.40 for A4, C7, and A5 respectively). Recall from Section 4.1, participants in both the groups had similar background in computer security. This may explain similar performance by participants in both groups on test problems from a security domain. We may expect to see differences in performance when one group has more experts or novices, however.

4.5 Participant Overall Performance

Table 6 shows the overall performance of the participants on the three test problems. Overall performance implies how many test problems, in total, the participants were able to solve correctly. By correct, we mean that participants provided intended answers. For example, Table 6, "none" implies that they did not provide intended answers for any test problems. There was no statistically significant difference between the overall performance of participants in the 3-Example and 2-Example groups (Fisher's Exact test, $p - value = 0.13$). In future, we intend to investigate the breakdown of overall performance by experience.

Table 6: Participant overall performance on test problems

	3-Example	2-Example
No correct problem	0	5
One correct problem	11	17
Two correct problems	19	18
Three correct problems	3	7

4.6 Participant Dropout

In an online survey, it is difficult to prevent participants from dropping out of the study. Our study setup collected incomplete or partial responses of participants who dropped out after consenting to take the study. We analyzed the incomplete responses and determined at what point during the study a participant dropped out. Total of 49 participants dropped out from the study: 21 from the 3-Example group, 23 from the 2-Example group, and 5 who dropped out before they were assigned to a group. In Table 7, we show the number of participants in groups 3-Example and 2-Example that dropped out. We also show at what point in the study they dropped out. We found no statistically significant difference in drop out between groups 3-Example and 2-Example ($p - value = 0.84$, Fisher test). However, we do note from the table that participants appear to drop out more during the training phase than after they start the test phase (19 vs. 2 for group 3-Example and 20 vs. 3 for group 2-Example).

Table 7: Participant drop out

Dropout position	3-Example	2-Example	Total
Before treatment assignment	unknown	unknown	5
Before seeing examples	7	6	13
After 1st example	4	3	7
After 2nd example and before 3rd	2	N/A	2
After all examples and before questions	6	11	17
After 1st test problem	1	2	3
After 2nd test problem	1	1	2

5. THREATS TO VALIDITY

We now discuss the threats to validity [35] and the limitations of this study and how they could be addressed in future studies.

Table 5: Participant performance on test problems and programming experience

	Test Problem A4		Test Problem C7		Test Problem A5	
	Intended	Not Intended	Intended	Not Intended	Intended	Not Intended
3-Example Group						
No Experience	4	0	2	2	0	4
Less than one year	9	0	5	4	3	6
One year	6	1	3	4	2	5
Between 1 and 3 years	6	1	4	3	4	3
Three or more years	4	2	4	2	2	4
2-Example Group						
No Experience	12	1	5	8	3	10
Less than one year	9	2	6	5	5	6
One year	6	2	4	4	4	4
Between 1 and 3 years	6	0	3	3	0	6
Three or more years	6	3	3	6	2	7

5.1 Construct Validity

Construct validity is the extent to which what we measured accurately reflects the construct that we proposed to study [35]. Surface representations and schemas are difficult constructs to produce as the space of variation is quite large. Measuring performance differences based on how humans perceive these differences also varies. To improve construct validity, we selected programming languages (C, Java and PHP) with wide adoption to represent surface structures. We chose the schemas error handling, checking input values and integer security from secure coding. These schemas may be familiar to security experts, but not to novices. We selected participants who had completed at least one course in security or had multiple years of industry security experience. Finally, we measured participant performance using open-ended answers to three test questions. Open-ended answers are usually more difficult to grade, but are more predictive of actual performance than other performance measures such as completion time. During grading, to reduce performance measurement bias, we used two graders. To increase confidence in our measurements, we can repeat the study using different training and test problems, and measure variance across results from multiple studies. Further, we can randomize the order of training and test problems to avoid learning and fatigue effects.

5.2 Internal Validity

Internal validity is the extent to which the findings follow from the data [35], which is similar to statistical conclusion validity or the extent to which our statistical inferences are valid. To reduce the effect of confounding effects, we randomized the assignment of participants to conditions and used double blinding. As discussed in Section 4.1 there was no skew in distribution of participants by programming experience, programming languages and education level. However, we did not collect information about gender and age, and must assume that the random assignment ensures even distribution. Factors such as motivation level of participants, past exposure to similar training or test problems, and cognitive load induced by problem representation format may confound our inferences form the data. In this study, we evaluated two hypotheses using a single experiment. To isolate the interaction of schema and surface, we can use a factorial design with two levels for the independent variables, schema and surface, and select extreme levels (e.g. 1schema-1surface, 3schema-3surface, 1schema-3surface, and 3schema-1surface). We can also introduce a control group that will receive the testing phase, but no training. To control the effect of experience, we can screen participants using a security related questionnaire.

5.3 External Validity

We recruited graduate and undergraduate students with a computer security background. Hence, we cannot generalize our results to the general population. We can improve generalizability by recruiting security professionals. It is possible that students with high motivation, drive or similar characteristics participated in the survey and, if so, this may bias our results. Students who dropped out due to lack of such characteristics may also bias our results. Lastly, we used an online survey tool, and results from an online survey may differ from a laboratory study.

5.4 Type 1 and Type 2 Errors

Type 1 error occurs when the investigator incorrectly rejects the null hypothesis. To decrease the chance of Type 1 error, we chose $\alpha \leq 0.05$. As discussed in Section 3 and Section 4, we chose appropriate statistical tests for the data based on the assumptions of those tests and our sample data.

Type 2 error occurs when the investigator fails to reject the null hypothesis when it is false. Low power of an experiment increases the chance of a Type 2 error. Two main factors that influence power are sample size and effect size, which, in our study, is the effect of the number of schema and surfaces on participant performance. Since we did not find statistically significant results, which implies we cannot reject the null hypothesis, we did a post-hoc power analysis [14] to compute achieved power. For Fisher's Exact test, the achieved power (1-β) for the three test problems was 0.12, 0.11 and 0.04 respectively. For the three test problems, to achieve a power of 0.8 with the observed effect sizes, we would need a sample size of 598, 842 and 6342 participants, respectively.

We can improve power by reducing variability. Because we are measuring higher cognitive abilities such as problem solving, individual performances (subject-to-subject variability) can be high [22]. This variability affects our results because we used a between-subjects experimental design.

Decreasing measurement errors, as discussed in Section 5.1 on construct validity, reduces this variability and increases power. As we administer the tests using an online survey, it is not possible to control the environment and hence reduce variability due to environmental factors. For example, participants may carry out multiple tasks or take the survey on different sized screens (e.g. tablet or desktop). Although environmental factors may cause higher variations in individual performances, they may be a positive factor toward the generalizability of our results: in the real world, users may indeed multi-task or use different form-factors. Lastly, we can improve power by increasing the strength of the treatment. As discussed in Section 5.2 on internal validity, we can increase the strength of treatment by using a factorial design and selecting the extremes.

6. RELATED WORK

We will now discuss related work from human-computer interaction (HCI) and usable security. Related work appears in the proceedings of SOUPS, CHI, and security conferences, such as IEEE Security and Privacy and USENIX Security, among others. Researchers in the field of HCI have applied theory of learning by analogy to several problems. Maclean et al. study design rationale of user interfaces [23], Douglas and Moran explain how novices learn text editor semantics [13], Redmiles studies reducing variability in programmer performance [31], and Siegle implements an interactive graphical prototyping environment using structure mapping [39]. Halasz and Moran argue that analogical models are useful for learning about computer systems, but they argue they are harmful for reasoning about computer systems [18].

In general, theories from cognitive psychology have driven research in the field of human-computer interaction especially in the formative years [4, 6]. These theories have helped in the design and evaluation of HCI systems, for example, Card et al. study how users interact with text editors [5], Kitajima et al. study how users navigate websites [21], Green et al. analyze usability of visual programming environments [17], and Nielson proposes heuristics to find usability problems in user interfaces [25].

Research in usable security has applied theories of human cognition to study problems such as user authentication and social engineering attacks. For example, Yan et al. study tradeoffs between usability and memorability of passwords [43], Witten and Tyger study usability of PGP encryption [42], Dhamija et al. analyze why phishing is effective [12], and Conti et al. propose a framework for analyzing attacks on information visualization systems [10]. Researchers have also studied user metal models of security. A mental model are thought to resemble how users think about a problem, and they loosely resemble the definition of a schema. For example, Wash identified security mental models of home computer users [41] and Rader et al. studied how users generate security mental models using informal stories [29].

For improving user training, researchers in HCI and usable security have employed different types of examples. Sheng et al. use a conceptual-procedural approach from learning science to differentiate between phishing examples [37]. Raja et al. use examples of firewall warnings that contain visual metaphors in place of semantically equivalent text [30].

7. DISCUSSION AND CONCLUSIONS

As we discussed in Section 4, our study results do not show statistically significant differences in performance when we vary the number and types of surfaces and schemas in the training stimulus. However, as we discussed in Section 5 on threats to validity, the power of our experiment was low. Increasing the sample size, using a factorial design, and increasing the number of training examples may allow us to observe statistically significant differences.

We propose to further investigate several observations from our study. First, using text representations for describing surface structure appears to increase the number of unintended answers extended in context. Answers extended in context are not incorrect answers, and seem to indicate "out-of-the-box" thinking by the participants. Second, we see from Table 4 that there is a decrease in participant performance from the first to the last test problem. Participants in general were able to answer the first test problem correctly (90% of the 3-Example group and 82% of the 2-Example group). For the second test problem, the percentage of correct responses in each group fell almost to half (54% of the 3-Example and 44% of the 2-Example). For the third test problem that used natural language text representation, both groups performed poorly (33% of the 3-Example and 29% of the 2-Example). Future studies should rule out the effect of fatigue and randomize the order of test problems to understand the effect, if any, of text representation on the gradual decrease in performance. Lastly, from the overall performance details listed in Table 6, we observe that all participants in the 3-Example group answered at least one test problem correctly, whereas 10% of participants in the 2-Example group answered all test problems incorrectly.

To conclude, we presented details of a study to investigate the effect of the number and types of examples in computer security studies. We discussed preliminary results from our study involving 80 complete, participant responses. In our research, we sought to study the science of choosing examples for training stimulus in security studies. We used the theory of analogical transfer to explain the impact of number and types of examples used in a training stimulus. Based on this theory, we selected training examples with varying schema and surface elements. While our results are not statistically significant, we believe our experimental design and approach can be used to inform future study design and investigation in security training.

Acknowledgment

The authors wish to thank the anonymous reviewers of the paper, and members of the Usable Privacy and Security Lab at Carnegie Mellon University for their valuable feedback. This research was funded by Army Research Office - (W911NF-09-1-0273) and King Abdul-Aziz University.

8. REFERENCES

[1] W.-k. Ahn, W. F. Brewer, and R. J. Mooney. Schema acquisition from a single example. *J. of Experimental Psychology: Learning, Memory, and Cognition*, 18(2):391, 1992.

[2] M. G. Bulmer. *Principles of statistics*. DoverPublications. com, 1979.

[3] G. Camilli and K. D. Hopkins. Applicability of chi-square to 2× 2 contingency tables with small

expected cell frequencies. *Psychological Bulletin*, 85(1):163, 1978.

[4] S. K. Card, T. P. Moran, and A. Newell. *The psychology of human computer interaction*. Routledge, 1983.

[5] S. K. Card, T. P. Moran, and A. Newell. *The psychology of human computer interaction*. Routledge, 1983.

[6] J. M. Carroll. Human-computer interaction: psychology as a science of design. *Annu. review of psychology*, 48(1):61–83, 1997.

[7] M. T. Chi, M. Bassok, M. W. Lewis, P. Reimann, and R. Glaser. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Sci.*, 13(2):145–182, 1989.

[8] M. T. Chi, P. J. Feltovich, and R. Glaser. Categorization and representation of physics problems by experts and novices. *Cognitive Sci.*, 5(2):121–152, 1981.

[9] J. Cohen. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psych. bulletin*, 70(4):213, 1968.

[10] G. Conti, M. Ahamad, and J. Stasko. Attacking information visualization system usability overloading and deceiving the human. In *Proc. of the 2005 Symp. on Usable privacy and security*, pages 89–100. ACM, 2005.

[11] J. Corbin and A. Strauss. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage, 2007.

[12] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *Proc. of the SIGCHI Conf. on Human Factors in computing Syst.*, pages 581–590. ACM, 2006.

[13] S. A. Douglas and T. P. Moran. Learning text editor semantics by analogy. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Syst.*, pages 207–211. ACM, 1983.

[14] F. Faul, E. Erdfelder, A.-G. Lang, and A. Buchner. G* power 3: A flexible statistical power analysis program for the social, behavioral, and biomedical sciences. *Behavior research methods*, 39(2):175–191, 2007.

[15] D. Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Sci.*, 7(2):155–170, 1983.

[16] D. Gentner and C. Toupin. Systematicity and surface similarity in the development of analogy. *Cognitive Sci.*, 10(3):277–300, 1986.

[17] T. R. G. Green and M. Petre. Usability analysis of visual programming environments: a cognitive dimensions framework. *J. of Visual Languages & Computing*, 7(2):131–174, 1996.

[18] F. Halasz and T. P. Moran. Analogy considered harmful. In *Proc. of the 1982 Conf. on Human factors in computing Syst.*, pages 383–386. ACM, 1982.

[19] D. L. Hintzman. "Schema Abstraction" in a multiple-trace memory model. *Psychological review*, 93(4):411–428, 1986.

[20] K. J. Holyoak. The pragmatics of analogical transfer. *The psychology of learning and motivation*, 19:59–87, 1985.

[21] M. Kitajima, M. H. Blackmon, and P. G. Polson. Cognitive architecture for website design and usability evaluation: Comprehension and information scent in performing by exploration. In *HCI Int. 2005*, volume 4. L. Erlbaum Associates Mahwah, NJ, 2005.

[22] J. Lazar, J. H. Feng, and H. Hochheiser. *Research methods in human-computer interaction*. Wiley. com, 2010.

[23] A. MacLean, V. Bellotti, R. Young, and T. Moran. Reaching through analogy: a design rationale perspective on roles of analogy. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Syst.*, pages 167–172. ACM, 1991.

[24] D. L. Medin and B. H. Ross. The specific character of abstract thought: Categorization, problem solving, and induction. 1989.

[25] J. Nielsen. Usability inspection methods. In *Conf. companion on Human factors in computing Syst.*, pages 413–414. ACM, 1994.

[26] L. R. Novick and J. Campbell. The role of expertise in solving arithmetic and algebra word problems by analogy. *The nature and origins of Math.al skills*, pages 155–188, 1992.

[27] F. Paas, A. Renkl, and J. Sweller. Cognitive load theory and instructional design: Recent developments. *Educ.al psychologist*, 38(1):1–4, 2003.

[28] G. Polya. *How to solve it: A new aspect of mathematical method*. Princeton University Press, 2004.

[29] E. Rader, R. Wash, and B. Brooks. Stories as informal lessons about security. In *Proc. of the 8th Symp. on Usable Privacy and Security*, page 6. ACM, 2012.

[30] F. Raja, K. Hawkey, S. Hsu, K.-L. C. Wang, and K. Beznosov. A brick wall, a locked door, and a bandit: a physical security metaphor for firewall warnings. In *Proc. of the 7th Symp. on Usable Privacy and Security*, page 1. ACM, 2011.

[31] D. F. Redmiles. Reducing the variability of programmers' performance through explained examples. In *Proc. of the INTERACT'93 and CHI'93 Conf. on Human Factors in Computing Syst.*, pages 67–73. ACM, 1993.

[32] L. Reeves and R. W. Weisberg. The role of content and abstract information in analogical transfer. *Psychological Bulletin*, 115(3):381, 1994.

[33] A. Renkl. Learning from worked-out examples: A study on individual differences. *Cognitive Sci.*, 21(1):1–29, 1997.

[34] M. T. Roberson and E. Sundstrom. Questionnaire design, return rates, and response favorableness in an employee attitude questionnaire. *J. of Appl. Psychology*, 75(3):354, 1990.

[35] W. R. Shadish, T. D. Cook, and D. T. Campbell. *Experimental and quasi-experimental designs for generalized causal inference*. Houghton, Mifflin and Company, 2002.

[36] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[37] S. Sheng, B. Magnien, P. Kumaraguru, A. Acquisti, L. F. Cranor, J. Hong, and E. Nunge. Anti-phishing

phil: the design and evaluation of a game that teaches people not to fall for phish. In *Proc. of the 3rd Symp. on Usable privacy and security*, pages 88–99. ACM, 2007.

[38] S. Siegel. Nonparametric statistics for the behavioral sciences. 1956.

[39] G. Siegle. The clim prototyping environment (cpe). In *INTERACT'93 and CHI'93 Conf. Companion on Human Factors in Computing Syst.*, pages 39–40. ACM, 1993.

[40] J. Sweller, J. J. Van Merrienboer, and F. G. Paas. Cognitive architecture and instructional design. *Educ. psychology review*, 10(3):251–296, 1998.

[41] R. Wash. Folk models of home computer security. In *Proc. of the 6th Symp. on Usable Privacy and Security*, page 11. ACM, 2010.

[42] A. Whitten and J. D. Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Proc. of the 8th USENIX Security Symp.*, volume 99. McGraw-Hill, 1999.

[43] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password memorability and security: Empirical results. *Security & Privacy, IEEE*, 2(5):25–31, 2004.

APPENDIX

A. SURVEY DETAILS

Introduction

Background: Security analysts are trained to defeat hackers by thinking like a hacker: how would a hacker try to defeat this system and what techniques can they use to succeed? Similar to other types of computer science expertise, security analysts require the ability to learn concepts from security problems and apply these concepts to new situations. We will train you with a set of security examples, and then test your ability to solve problems like a security analyst. Please study the following examples and take all the time that you need before proceeding.

Training Example 1 (full) for 3-Example Group

Carefully read the code snippet below (including all comments) before answering any questions. Please describe all issues you can identify.

```
1  /* function to copy network data to given file */
2  int copyData(struct connection *c, FILE *fp)
3  {
4  char buf[200];
5  readFromNetwork(c, buf, 100);
6  writeToFile(fp, buf, 100);
7  ...
8  }
9  /* read in the given number of bytes count from */
10 /* the given network connection */
11 int readFromNetwork(struct connection *c, char *
       buf, int count)
12 {
13 /* wait until count number of bytes become */
14 /*  available and return them in buf;  */
15 /* return -1 if there is an  error reading from
       the network; */...
16 }
```

Question 1: What is the problem?
Question 2: What are the consequences of the problem?
Question 3: What changes do you make to prevent the problem?

When you're ready, click the radio button below to expand the solutions.*
( ) Show Solutions

When you're ready, click the radio button below to expand the solutions.*
( ) Show Solutions

Question 1: What is the problem?
In lines 5 and 6 reproduced from above:
`readFromNetwork(c, buf, 100);`
`writeToFile(fp, buf, 100);`
The developer does not check the return value from the function readFromNetwork. They instead assume that the read function was successful and they proceed to write the data buffer out to the file.

Question 2: What are the consequences of the problem?
If the read was not successful, then the function writeToFile will write 100 bytes from buffer to the file that may be incorrect (e.g., garbage bytes). This action can corrupt the file.

Question 3: What changes do you make to prevent the problem?
The developer should check the return value from the function readFromNetwork to ensure that reading from the network succeeds. The function readFromNetwork returns -1 on failure.

```
if (readFromNetwork(c, buf, 100) != -1)
  {
  writeToFile(fp, buf, 100);
  }
 /*function to copy network data to given file */
```

Training Example 2 (code only) for 3-Example Group

```
1  /* allocate memory block & initialize it with 0 */
2  char* create_init_mem(int size)
3  {
4   if (size <=0) { return NULL; }
5  /* allocate memory block and initialize to 0 */
6  char *buf = (char*) malloc(size);
7  int i;
8  for (i=0; i<size; i++) { buf[i] = 0;}
9
10 return buf;
11 }
```

Training Example 3 (code only) for 3-Example Group

```
1  // get the OS version name
2  String osVersion = getItem("OS_VERSION");
3  if (osVersion.equals("Windows 7"))
4   { System.out.println("Version Supported");  }
5 else
6   { System.out.println("Version NOT Supported");}
7
8  public String getItem(String itemName)
9  {
10  //sysInfo is a global object that contains system
        info.
11  if (sysInfo != null) {
12    String itemValue;
13     // sysInfo.get(String s) will return a:
14     // either String value for item s
15     // or null , if no such description exists
16     itemValue = sysInfo.get(itemName);
17     return itemValue;
18   }
19   return null;
20 }
```

Training Example A (code only) for 2-Example Group

```
/* function to copy network data to given file */
int copyData(struct connection *c, FILE *fp)
{
char buf[200];
readFromNetwork(c, buf, 100);
writeToFile(fp, buf, 100);
...
}
/* read in the given number of bytes count from */
/* the given network connection */
int readFromNetwork(struct connection *c, char *
    buf, int count)
{
 /* wait until count number of bytes become */
/*  available and return them in buf;  */
/* return -1 if there is an  error reading from
    the network; */...
}
```

Training Example B (code only) for 2-Example Group

```
int getValueFromArray(int *array, int len, int
    index)
{
int getValueFromArray(int *array, int len, int
    index)
{
int value;
/* check that the array index is less than the */
/* maximum length of the array */
if (index < len) {

/* get the value at the specified array index */
value = array[index];
}
/* if array index is invalid then output error */
/*  message and return value indicating error */
else {
printf("error: index %d invalid.\n", index);
value = -1;
}

return value;
}
```

Test Problem 1 (full)

Carefully read the code snippet below (including all comments) before answering any questions. Please describe all issues you can identify.

```
/* connect to an existing records_db database and
    get user records */
$connection = mysql_connect("localhost", "user1",
    "pass465");
mysql_select_db("records_db", $connection);
$result = mysql_query("select * from user_table");
while ($row = mysql_fetch_assoc($result))
{  /* print results */... }
```

Question 1: What is the problem, if any?
Question 2: What are the consequences of the problem, if any?
Question 3: What changes do you make to prevent the problem, if any?

Test Problem 2 (code only)

```
/* concatenate 2 strings */
char* concatenate(char *str1, char *str2)
{
short int total;
/* compute length of resulting string */
/* strlen does not count terminating '\0' */
total = strlen(str1) + strlen(str2) + 1;

/* allocate buffer enough to hold str1 & str2 */
char *buf = (char *) malloc(total);

if (buf != NULL) {
/* copy the str1 including terminating '\0' */
strcpy(buf, str1);

/* concatenate str2 to the end of str1 */
/* strcat starts copying at '\0' in buf */
/* strcat copies terminating '\0' of str2 */
strcat(buf, str2);
}

return buf;
}
```

Test Problem 3 (description only)

Bob is a system administrator at a datacenter and is responsible for completing a list of tasks each month. One of these tasks is to run patches when they become available. Bob has developed a patching script to install patches on each server. He has scheduled the script to run periodically. The patching script downloads available patches from a secure and verified location, executes the patches not already installed, and reboots the server (if necessary). Bob is confident that the script has covered all the steps in the patching process, thus Bob marks the task as completed.

Background Experience

Please answer the following four questions to complete the survey.

What is your current education level?*
( ) High school
( ) 1-2 years of college
( ) 3-4 years of college
( ) Graduate student or higher

How many years experience do you have in the following programming languages?
(Please use decimals to one tenth, e.g., 1.5 years)*
C/C++: .....
Java:.....
Perl, PHP, Python, Ruby: .....
SQL: .....

Which of the following courses have you completed?
( ) A programming language course
( ) Data Structures
( ) Computer Security
( ) Secure Programming

How many years of industry experience, including summer jobs and internships, do you have in computer programming?
(Please use decimals to one tenth, e.g., 1.5 years)*
.....