Effectiveness of Teaching Metamorphic Testing, Part II

Kunal Swaroop Mishra, Gail E. Kaiser and Swapneel K. Sheth
Department of Computer Science
Columbia University
New York, NY, 10027

July 29, 2013

Abstract

We study the ability of students in a senior/graduate software engineering course to understand and apply metamorphic testing, a relatively recently invented advance in software testing research that complements conventional approaches such as equivalence partitioning and boundary analysis. We previously reported our investigation of the fall 2011 offering of the Columbia University course COMS W4156 Advanced Software Engineering [6], and here report on the fall 2012 offering and contrast it to the previous year. Our main findings are: 1) Although the students in the second offering did not do very well on the newly added individual assignment specifically focused on metamorphic testing, thereafter they were better able to find metamorphic properties for their team projects than the students from the previous year who did not have that preliminary homework and, perhaps most significantly, did not have the solution set for that homework. 2) Students in the second offering did reasonably well using the relatively novel metamorphic testing technique vs. traditional black box testing techniques in their projects (such comparison data is not available for the first offering). 3) Finally, in both semesters, the majority of the student teams were able to apply metamorphic testing to their team projects after only minimal instruction, which would imply that metamorphic testing is a viable strategy for student testers.

Introduction

Although testing can never assure the absence of errors, it still remains the most proven technique to improve the credibility of a program. Eventually, the vast majority of the test cases will "pass". These test cases are then kept for regression testing, but otherwise might not be re-used, thus meaningful information contained in them remains untapped [1]. This is particularly a concern since the test suite is often the only form of formal specification, as most specifications are written as prose use cases or usage scenarios. Considering the fact that testing is a costly and labor intensive procedure, it is wise to make the most use of each test case. *Metamorphic Testing* is an approach to software testing that aims to extract useful information from previously developed and executed test cases. Metamorphic properties of the whole program and/or isolated functions can be determined and then used to derive follow up test cases, enlarging the test suite [1]. Execution of these metamorphic test cases may reveal bugs that the original test suite did not and provide additional confidence in the program, upon successful bug fixes, beyond conventional testing techniques like equivalence partitioning and boundary analysis. Further, metamorphic testing has been shown to be useful for finding defects in software

applications where it is difficult to apply conventional testing techniques [4][5], because the correct output cannot practically be known *a priori* (or checked *a posteori*) for every valid input, in other words there is no test oracle. Weyuker described this situation as *"Programs which were written in order to determine the answer in the first place. There would be no need to write such programs, if the correct answer were known."* [8] This issue arises frequently in implementations of machine learning, simulation and optimization algorithms, which increasingly form the basis of mainstream software applications.

However, no testing approach will be very helpful if testers do not understand how to use it. In fact, a common concern about metamorphic testing is "Where do the metamorphic properties come from?", while the analogous question is rarely posed for equivalence partitions and boundary analysis -- perhaps since these techniques have been known and used for decades. Thus we decided to teach metamorphic testing in our regular software engineering course, targeted to seniors and first year MS students, and see how well the students did after minimal training (only a small portion of the semester could be devoted to this topic since we still had to cover all the conventional software engineering material). Could they determine the metamorphic properties of their team project systems? Could they effectively use these metamorphic properties to devise and execute test cases? MS students who have successfully completed a similar course at their undergraduate institution, or who have equivalent industry experience, are typically able to "waive" this core course. Thus the enrollment is primarily students with relatively little software development experience outside the classroom. We first provide some background on metamorphic testing, and then introduce our classroom approach.

Metamorphic Testing Concepts

Metamorphic Testing (MT) is a technique for generating follow up test cases from existing test cases. Consider a program p that implements a function f. Let the test suite for testing the program p be T={t1,t2,t3,…,tn}. Each of these test cases is executed so both the inputs and outputs are known. MT produces follow up test cases T'= {t1', t2',…, tn'} based on the metamorphic property(ies) of function f intended to be implemented by program p. These properties define a relationship between *pairs* of inputs and outputs, such that given an original input i and its actual output o, we can easily derive a new input i' and is expected output o'. Then when we execute the new test case with this input i' we can compare its actual output o'' to see if it matches o'.

A simple example of a function to which metamorphic testing could be applied would be one that calculates the standard deviation of a set of numbers. Certain transformations of the set would be expected to produce the same result. For instance, permuting the order of the elements should not affect the calculation; nor would multiplying each value by -1, since the deviation from the mean would still be the same (think about the numbers being "flipped" around the zero on the number line) [5].

Furthermore, other transformations will alter the output, but in a predictable way. For instance, if each value in the set were multiplied by 2, then the standard deviation should be twice as much as that of the original set, since the values on the number line are just "stretched out" and their deviation from the mean becomes twice as great. Thus, given one set of numbers, we can create three more sets (one with

the elements permuted, one with each multiplied by -1, and another with each multiplied by 2), and get a total of four test cases; moreover, given the output of only the first test, we can predict what the other three should be [5].

Another simple example is if the program searches the corpus "Romeo and Juliet" for the word Romeo as the first test case, it will designate certain lines. If it then searches for “Romeo OR foo” where “foo” is known to not be present in the corpus, the program should find exactly the same lines as for the original case.

Teaching Metamorphic Testing

Metamorphic Testing has been part of the curriculum of Columbia University’s COMS W4156 Advanced Software Engineering (ASE) for the past two years. The course was initially taught by the second author in fall 2011 and by the third author in fall 2012. The course, previously without this advanced material, has been offered every year since 2002, always taught by the second author except during her sabbaticals; the course is normally offered only during the fall semester. The enrollment is primarily incoming master of science graduate students, with a few senior undergraduate students, nearly all of them Computer Science majors. Students in fall 2011 were asked to find metamorphic properties in their team projects and then prepare test cases based on those properties, with no other assignments concerned with metamorphic testing. In order to evaluate how successful the students were in finding and applying metamorphic properties, a study was conducted by the first author under the guidance of the second author [6]. The current study is a ***follow up*** to the previous study.

It was found in the previous study that fewer students than hoped seemed to understand the notion of metamorphic testing sufficiently well to use it in their team projects. Thus, a few changes were made in fall 2012 with the aim of improving the students’ understanding of metamorphic testing. One change was to give a separate individual homework assignment focused solely on metamorphic testing, so that the students granted importance to the concept and try to fully understand it before applying it to their projects. Prepared solutions to this assignment were also released as part of the feedback to the students’ submissions. Many students did not do very well on this assignment, which we had thought might be the case, but expected the solutions provided afterwards by the teaching staff to help them understand what they did wrong and improve their understanding for the following application of metamorphic testing working with their teams. We shall evaluate the verity of this hypothesis in the Observations section. We should note that each team consisted of four students, so it was only necessary for one student per team to understand metamorphic testing in order for this aspect of the team project to succeed.

In the next section, we discuss the rationale for this experiment, and describe the homework and its solution set. After that we discuss the setup for the student projects and take a look into each of them. We then discuss our findings and try to justify our optimism that the question “Where do the metamorphic properties come from?” has the same answer as conventional equivalence partitioning and boundary analysis techniques - from the designers, developers and testers after minimal training – and thus should not be a significant impediment to adoption of the metamorphic testing approach.

Experiment Rationale

Our goal is to understand the effectiveness of teaching Metamorphic Testing Concepts in a senior/graduate software engineering classroom setting. MT can be used both at the unit (subroutine) level and at the full system level. In this experiment, students were asked to apply MT at the unit level. The following research questions are the central aspects of the case study:

1) ***To what extent were the students successful in finding metamorphic properties for testing in the individual assignment and in their team projects?*** The measure of students' success depended on whether or not they could find (at least) the same properties that the first author could find. The first author has not conducted research in metamorphic testing and had no prior knowledge of the topic prior to embarking on the original study. He had more knowledge of the individual homework assignment, since he helped the teaching staff design it, but of course less knowledge of the team projects since those were invented by each team.
2) ***How did the students fare in using metamorphic testing compared to traditional unit testing techniques?*** This research question is of particular importance with regards to acceptance of metamorphic testing by the Software Industry, where the learning curve must not be too steep. We should note that students are unlikely to be familiar with metamorphic testing (or metamorphic properties) from any of their previous classes, while in contrast they should already have some intuition regarding Boundary value analysis, Equivalence partitioning, etc. since they have heard about the difficulties of "edge" or "corner" cases from their earliest programming courses even if software testing was never formally taught [7]. We expected that the students would do better developing/applying test cases with traditional unit testing techniques than with metamorphic testing.
3) ***How did the students fare in finding metamorphic properties in fall 2012 as compared to their performance in finding metamorphic properties in fall 2011 [6]?*** The comparison is specifically between the team projects, where the 2012 students had the benefit of a prior individual assignment (with solutions) whereas the 2011 students did not.

It is worth mentioning that there were no projects where students were able to define metamorphic properties but were not able to devise the corresponding test cases, unsurprising since application of that part of the technique is straightforward, so the challenge is in "where do the metamorphic properties come from?"

Experiment Setup

The experiment was conducted within the premises of the COMS W4156 Advanced Software Engineering course offered by the Department of Computer Science, Columbia University to both Undergraduate and Graduate students in Fall 2012. This course is designed to impart knowledge about the software lifecycle from the viewpoint of designing and implementing N-tier applications with special emphasis on quality assurance (http://ase.cs.columbia.edu). The experiment utilized the students' assignments and projects to answer the various research questions listed in the section above.

Materials Provided to the Class

We only had time in the fall 2012 schedule to present one lecture specifically about Metamorphic Testing. The lecture introduced the concepts of metamorphic testing and metamorphic properties, and presented a few examples that concentrated on the use of Metamorphic Testing for applications with no test oracles. The students were also assigned to read a research paper [5] co-authored by the second author, which contained additional examples of metamorphic properties, even though we did not expect any student projects to include "non-testable" functions.

Testing Familiarity Survey

A survey (Appendix I) was conducted early in the semester in order to gauge the familiarity of students with traditional unit testing techniques vs. relatively new testing techniques like metamorphic testing. Such a survey was not conducted in the previous year. The survey was completed by 84 students (out of a total of 85 students taking the class). Prior to the survey, we believed that many of the students would already have some (self-reported) familiarity with conventional unit testing techniques but little or no familiarity with the more advanced formal specification approach to quality assurance, which indeed turned out to be the case. The results are summarized in Table I.

Familiarity with	**Low Familiarity %**	**Moderate Familiarity %**	**High Familiarity %**
Software Testing (in general)	25	44.05	30.95
Unit Testing (in general)	34.52	26.19	39.29
Boundary Value Analysis	51.19	28.57	20.24
Equivalence Partitioning	72.62	17.86	9.52
Code Coverage	55.95	25.00	19.05
Unit Testing Tools	59.52	25.00	15.48
Integration Testing	52.38	23.81	23.81
In-Vivo Testing	89.29	4.76	5.95
Metamorphic Testing	89.29	5.95	4.76
Formal Specification	80.95	9.52	9.52

Table I: Testing Familiarity Survey results

However, our expectation was that *none* of the students would have any familiarity at all with either In-Vivo Testing or Metamorphic Testing. Although metamorphic testing has enjoyed a niche in the software testing research community since the late 1990s, it is not covered by many university curricula or popular testing blogs. Further, In-Vivo Testing is a term invented by the second author for the name of one of her research projects; although some of her publications have appeared in conference proceedings [2][3], we were surprised that a few students claimed to already be familiar with it (none of these students had ever participated in her lab's research). Astute readers may have already inferred that some students' responses to the survey questions may not be entirely accurate.

Individual Assignment on Testing

The goal of the individual assignment (Appendix II) was to provide reasonably simple exercises where students were tasked to find the metamorphic properties of conventional software with a trivial test oracle. Even if the students could not identify all the properties on their own, we felt that discussion of the solutions to the assignment (Appendix III) would inculcate a better understanding of metamorphic testing than the lecture alone.

The beginning of this assignment is shown below:

> **For this assignment, you will do Unit Testing for the 2 systems described below.**
>
> **For each system, describe the following:**
>
> - **Black Box Testing using Equivalence Partitions and Boundary Value Analysis (10 points)**
> - **Black Box Testing using Metamorphic Testing (5 points)**
> - **White Box Testing (5 points)**
>
> **For both the Black Box Testing parts, describe specific equivalence partitions, boundary cases, and metamorphic properties.**
>
> **For the White Box Testing, it's sufficient to describe the general approach you would take (You don't have specific source code to work with, for this assignment.)**
>
> 1. **Hotel Booking System**
>   **Consider a Hotel Booking system with the following features**
>
>   **a. You can search for a particular hotel and the search feature displays N results per page. If search results for a particular hotel are M and M > N such that M/N = X, search feature shall display X pages upon searching for this hotel. For instance, let's assume the search feature displays 10 results per page. Now, if for a particular keyword, there are 100 search results, then the total number of pages that would be displayed upon searching for that keyword would be 100/10 = 10. You are only required to test the search result display function. You do not need to write test cases for the search functionality.**

The part of the solution corresponding to "Black Box Testing using Metamorphic Testing (5 points)" for the above question is:

> **Solution for Metamorphic Testing:**
>
> **For metamorphic testing, there is a function (input deriving function) that derives a new input I' from old input I and there is another function that derives an expected output (output deriving function) O' from old output O. So when the software under test is run with input I' and gets an actual output O'' it is compared to the expected**

output O'. You have to describe how this comparison is done. This may not be necessarily equality.

Note: Some of the solutions (here) for metamorphic properties are derived from the notion of invariants and can be considered a special, trivial case of metamorphic properties.

1.a

Explanation:

If N results are added to an existing search result count, then one more page than the current number of pages is needed to display those results.

If search result count is x and takes p pages to be displayed, then x + N should take p+1 pages for display.

Transformation:

Assume

Display function f(Y) = Y/N = K(output) pages, where Y(input) is the number of results and N is the number of results per page.

Input Deriving function, h(Y) = Y + N (add N to the existing search result count)

Output Deriving Function, g(K) = K + 1 (number of pages incremented by 1)

Initial test case: X results are displayed on P pages.

f(X) = X/N = P

Metamorphic Test Case:

Derived Input = h(X) = X + N

Derived Output = g(P) = P + 1

Display Function = f(X + N) = X+N/N = P + 1

Project Setup

The projects were carried out by groups of four students. There were a total of 21 projects, all of which were considered for this case study. The following section consists of a short description of each the projects and lists the metamorphic properties identified by the students, with commentary on their findings.

Feedback Survey on Studying Metamorphic Testing

A survey was conducted towards the end of the semester after the students were done with all their projects and homework, where the students were asked whether they thought they now understood metamorphic testing, and what materials helped them the most. The survey also asked the students to suggest changes that could be incorporated into the course in the future to improve the approach to the teaching of metamorphic testing.

Table II below shows the three questions asked in the survey.

Question	**Answer Type**
1. What do you think about your understanding of Metamorphic Testing now?	Radio List(choose one option out of many)
2. What do you think helped you the most in understanding Metamorphic Testing?	Radio List(choose one option out of many)
3. What change(s) do you think could improve student experience as far as teaching Metamorphic Testing is concerned?	Text Box

Table II: Description of questions in Feedback Survey

Table III shows the distribution of answers for question 1 and Table IV does the same for question 2. The survey was completed by 51 out of 85 students taking the class.

Question1 : **What do you think about your understanding of Metamorphic Testing now?**		
Option	**Count**	**Percentage**
I understand Metamorphic Testing Completely.	5	10.87
I still don't understand Metamorphic Testing at all.	3	6.52
I have some (not complete) understanding of Metamorphic Testing.	32	69.57
I am not sure if I understand it or I don't.	6	13.04

Table III: Distribution of answers for question 1

Question2: **What do you think helped you the most in understanding Metamorphic Testing?**		
Option	**Count**	**Percentage**
Professor's Lecture on Metamorphic Testing	6	13.04
Recommended Paper Reading	4	8.70
Metamorphic Testing (Individual) Assignment	2	4.35
Metamorphic Testing (Individual) Assignment's	18	39.13

Solutions		
Other (Internet, Google search, reading other papers about Metamorphic Testing. etc.)	12	26.09
None	4	8.70

Table IV: Distribution of answers for question 2

Table III shows that most of the students think that they have some understanding of metamorphic testing. About 70% of the students chose the option "I have some (not complete) understanding of Metamorphic Testing" and another 11% chose "I understand Metamorphic Testing Completely." Almost 90% had claimed no knowledge prior to instruction (Table I), so we think this experiment can be counted as a success.

Project Descriptions & Metamorphic Properties

We discuss each of the 21 student projects submitted as part of the course, first outlining the key ideas and then listing the metamorphic properties reported by the students.

Project 1

In order to raise the public awareness of the HIV/AIDS prevention and find out effective approaches to alleviate its spreading, the students of this team decided to implement an N-tier web application to visualize related data. By mapping groups of data to a graph, the application should be able to tell a story about recent trends of HIV/AIDS and potential correlated factors. By digging into open data, such as The World Bank data and The Open US data, the application was able to promote government transparency and facilitate the policy makers.

Metamorphic Properties as found in the students' Testing Report:

1) Input Deriving Function = h(x) = x' = x + 8 (incrementing the number of password characters by threshold value 8)
Output Deriving Function = g(acceptance) = 1 (since the length of password character >= 8)
In initial output, f(x) = acceptance = 0
Thus, g(0) = 1
Finding f(x') = f(x+8) = 1(here x + 8 >= 8)
If incrementing the number of password character by the threshold value 8, then output should be password length accepted.

2) Input Deriving Function h(x) = x' = x +/- 52 (incrementing/decrementing the year by 52)
Output Deriving Function = g(response) = 0 (since the year > 2011 or year < 1960)
In initial output, f(x) = response = 1
Thus, g(1) = 0
Finding f(x') = f(x +/- 52) = 0 (here x + 52 > 2011 or x - 52 < 1960)

If incrementing or decrementing year that initially between 1960 and 2011 by 52, then output should be invalid JSON data files

Comments from the first author:

The students have found two valid metamorphic properties. However, these two properties appear very similar to some properties presented in the solution for the individual assignment. It is good that the students applied the notions presented in the assignment's solution to their project. But there were more properties that the students could have found out in their high priority functionalities. One such property was around a functionality that showed comparison of HIV trends between various nations. There were several options (in form of checkboxes from which a user had to choose) that helped design the logic for the trend chart display. One metamorphic property could be permuting the order in which the options were selected or check boxes were checked should not change the displayed chart, if the same set of options were selected. There was another functionality that dealt with comparing HIV trends over the years between two countries. A property here could be result from comparison of country A with country B should be the inverse (exact opposite) of result from comparing country B with country A.

Project 2

This application is designed to help people in New York City to find free water source and restroom. The students planned to build a mobile phone application to show user where is the nearest fountain or restroom based on their location when they travelling in NYC. All the related information will be showed on the map with different mark that represents for different features like size or rating. User could comment on the place they have visited and give ratings. The locations of restroom and fountain will be obtained from NYC Open Data.

Metamorphic Properties as found in the students' Testing Report:

a. Test the Search Location function with different descriptions for the same spot
b. Test the Login page with different user account
c. Test the Routing Function with different destinations
d. Test the Rating Calculation Function on different spots
e. Test the Detection of My Location Function at different location
f. Test the Write a Comment Function on different spots
g. Test the Write a Report Function on different spots

Comments from the first author:

From the above section, it comes across that the students did not get a proper understanding of metamorphic testing. Neither does the above description show any sort of transformation functions used nor do the properties appear to be valid metamorphic properties in their current form. Perhaps the students got the idea of transforming the input (this could be surmised from the way the properties are mentioned) but they do not present any clear idea about what the transformed output would be with the transformed input. Since their application dealt with suggesting water fountains and restrooms in a

given perimeter on the map, one property could be that for any location within that perimeter, the user should be getting the same suggestions. Further, if at location A, the suggestion made were S1 (where S1 is a set of the suggestions made) and at location B suggestions made were S2, then permuting the order and first finding information at location B should still suggest S2 and then searching at location A should still suggest S1. The students failed to identify this property as well.

Project 3

The students designed a house search web application. Users can search for houses and the houses get rated and assigned a score based on criteria like transportation, entertainment, and noise.

Metamorphic Properties as found in the students' Testing Report:

*For functionality **transportation rating** in requirement of Rating system:*
Metamorphic testing:
for K, x satisfying K>0.01 and K+x>0.01, if f(K)=R, we have f(K+x)=R-10000x.
for K, x satisfying 0<=K<=0.01 and 0<=K+x<=0.01, we have f(K+x)=f(K)=100.

*For functionality **entertainment rating** in requirement of Rating system:*
Metamorphic testing:
For K, x satisfying 0<=K<200 and 0<=K+x<200, if f(K)=R, we have f(K+x)=R+x/2.
For K, x satisfying K>=200 and K+x>=200, we have f(K+x)=f(K)=100.

*For functionality **noise rating** in requirement of Rating system:*
Metamorphic testing:
For K, x satisfying 0<=K<200 and 0<=K+x<200, if f(K)=R, we have f(K+x)=R-x/2.
For K, x satisfying K>=200 and K+x>=200, we have f(K+x)=f(K)=0.

*For functionality **sanitary rating** in requirement of Rating system:*
Metamorphic testing:
For K, x satisfying 0<=K<200 and 0<=K+x<200, if f(K)=R, we have f(K+x)=R-x/2.
For K, x satisfying K>=200 and K+x>=200, we have f(K+x)=f(K)=0.

*For all the metamorphic transformation above, K is the original input, and K+x is the input after transformation.

Comments from the first author:

The first author could not find any other properties than what the students found. The students have also shown correct transformation of the properties that they found.

Project 4

The students designed a weekend planning tool that would prompt users to visit different locations in the city and share their views on the same. The tool would give suggestion for travel and relaxation to the users for a city. Owners of restaurants, cinemas, etc. can post activities and promotions to the website.

Metamorphic Properties as found in the students' Testing Report:

Length check in input textbox: if the length limit of a textbox is N, if the input's length is M and is accepted, then change N to N+a and M to M+a, it should also be accepted. Transition function is
input: N->N+a, M->M+a
output: accept if it accepted, deny if it denied before modification, where a is an integer.

Password and confirm in "sign up" and "modify password": if password N and confirm M are matched, then modify N to NA, M to MA, where A is a valid string, they will also be matched. Vice verse. Transition function is input: N->NA, M->MA
output: matched if they were, not matched if they were not.

Search: Search a keyword, it may return N targets. If search a part of that keyword, it will return M targets that M>=N. If search a new keyword including the old one, it will return M targets that M<=N. Transition function is descripted above. i.e. search " ann' " it returns one target, search " ann " it returns eight targets, search " ann's " it returns one target.

Star user for individual: System has a threshold value of star user: V. The number of comments of a user is N. If the user is a star user, then if modify V to V+a, N to N+a, this user will also be a star user as before. Vice verse. Transition function is input: V->V+a, N->N+a
output: the user will keep his status, where a is an integer.

Star user for system: as descried above, N is the number of star users in the system. If increase V, N will keep the same value or decrease. If decrease, N will keep the same values or increase. Transition function is described above.

Like: the number of targets liked by a user is N, the number of results shown in "my like list" is M, where N=M. Then, if modify N to N+a, in "my like list" there will be M+a results, where N=M. Transition function is input: N->N+a
output: M->M+a and M=N.

My merchant: the number of targets owned by a user is N, the number targets shown in "my merchant" is M, where N=M. Then, if modify N to N+a, in "my merchant" there will be M+a targets, where N=M. Transition function is input: N->N+a
output: M->M+a and M=N.

Number of Users: the total number of users in system is N, the number of personal users is N1, the number of merchants is N2, the number of editors is N3, the number of admins is N4. If modify N1, N2, N3, N4 to N1+a, N2+b, N3+c, N4+d, then N=N1+a+N2+b+N3+c+N4+d.
Transition function is input: N1->N1+a, N2->N2+b, N3->N3+c, N4->N4+d
output: N-> N1+a+N2+b+N3+c+N4+d.

Comments from the first author:
The first author could not find any additional properties. The properties listed by the students are valid and show that they have a good grasp of the concept.

Project 5

This project utilized NYC OpenData, a widely adapted source for hundreds of sets of data produced by various organizations in the New York City. The project was based around the idea of booking a taxi on a mobile device. Extending the possibility of getting one's location on an android, the app would also provide a possible route to the passenger using Google Maps API. The information of the driver and the cab would be pulled using the NYC Open Data API. The application was an n-tier application, exploiting the EJB3 framework and android API.

Metamorphic Properties as found in the students' Testing Report:

1. List of candidate drivers: The passenger endpoint can list 5 nearby drivers. The server updates driver location from time to time. The server at the same time stores drivers' location.
When the passenger asks the server for the list of 5 nearby drivers, the server searches the database and returns the 5 drivers that are closest to the passenger. They return the driver information to the passenger. In this way, which 5 passenger returned by the server is determined by their location not by their sign in order. Changing sign in order of the passenger will not change the return value from the server.
Transformation: Assume function f(Y)=S returns the 5 nearest driver to the passenger. Y (input) is the sign in order of the passenger. S (output) is the set of 5 nearby drivers.
Input Deriving Function:
If Y=(first sign with driver a1,a2,..an, then sign in with driver b1,b2,..bm)
Then h(Y)=Y'=(first sign with driver b1,b2,..bm, then sign in with driver a1,a2,..an).
Output Deriving Function:
g(S)=S (same driver set should be generated).
Initial Test Case:
Y=(first sign with driver a1,a2,..an, then sign in with driver b1,b2,..bm)
f(Y)=S

Metamorphic Test Case:
Derive input: h(Y)=Y'=(first sign with driver b1,b2,..bm, then sign in with driver a1,a2,..an)
Derive output: g(S)=S
New Output: f(Y')=S
If driver sign in order lead to nearby driver set S, change drive sign in order also lead to nearby driver set S.
2. Post Comment: The passengers can post their comment to the driver. Each comment will use x lines on the post board. Then post comments in different order will use the same number of lines on the post board.

For example, a passenger posts a comment about driver1 with n line, and posts a comment about driver2 with m lines. If he/she posts comment about driver1 and then posts that about driver2, he/she will use totally n+m lines. If he/she posts comment about driver2 and then posts that about driver1, he/she will also use totally m+n lines.

Transformation: Assume f(Y)=K is the function calculate how many lines being used by a passenger. Y (input) is the comments the passenger has posted. K (output) is the number of lines the passenger has used.

Input Deriving Function:

If Y=(first post comment with n lines, and comment with m lines).

Then h(Y)=Y'=(first post comment with m lines, and comment with n lines).

Output Deriving Function:

g(K)=K (same number of lines should be calculated).

Initial Test Case:

Y=(first post comment with n lines, and comment with m lines).

f(Y)=(n+m) lines

Metamorphic Test Case:

Derive input: h(Y)=Y'=(first post comment with m lines, and comment with n lines)

Derive output: g(K)=K=(n+m) lines

New Output: f(Y')=m+n=(n+m) lines

Permuting the order of the comments posted ends up with the same number of lines used on the post board.

Comments from the first author:

The properties identified by the students are valid but they are not very strong metamorphic properties. One functionality in their program is that the system gathers a particular driver's location in every n seconds (n is a hardcoded value decided upon by the designers). There is a strong metamorphic property here; if after x seconds the system has gathered k locations for a particular driver then after x + n seconds the system should have k + 1 location points for that driver. The students did not identify this property.

Project 6

The idea for the project originated from the need that a typical user faces when searching for holiday destinations. Currently, there are number of web applications that recommend trip locations to users but most of them derive their search results based on user ratings and specific, place oriented, search strings. This system will be more generic that will take data feeds from open data sets and mine topics out of it. These topics will be ranked based on some scoring functions. The scored topics will be indexed and will be made available to the search and recommendation engine. Further we will be using the power of knowledge graph to suggest activities, places, restaurants, etc for each of the ranked places.

Metamorphic Properties as found in the students' Testing Report:

1. For indexing and scoring module it is a major requirement that the order in which the cities are indexed and scored must not change their ranking score. Following metamorphic properties were used to test the indexer

a. Permutation: In test1 we indexed cities C1, C2 and C3 and recorded their scores. In test 2 we permutated the order to C2, C1 and C3 and tested as to whether we are getting the same lucene score for various terms in indexed documents.

b. Inclusion & Exclusion: In test 1 we added cities C1 and C2 and noted the score of a term T. In tests 2 we added a city C3 to the index such that the C3 does not contain term 3. As per lucene's scoring we verified the expected output which was that the score for term T should not change. We tested inverse of this test to verify the exclusion property.

c. Noise Based: In test 1, we recorded the score of the index for cities C1, C2 and C3. In test 2 we added (in other test removed) a stop-word and verified that the score of the indexed terms should be same as that of test1.

2. For registration view (login page and edit password view was tested using similar approach) following were the metamorphic properties we used:-

a. Inclusion and Exclusion: In test 1, for a text box (for e.g.) username we used a set of valid characters (c1, c1,.. , cn) to sign-up. In test 2 and 3, we added and deleted a character from the valid character list (while maintain other constraints like using unique email). We verified that the expected output should be that the user should be able to sign up.

b. Permutation: We added three users U1, U2 and U3 in test 1. In test2, we permuted the order to U2, U3, U1 (after truncating the user table) and verified that the user accounts are created properly.

c. Semantic Equivalence property was also test where in 2 consecutive tests we tried to signup with two semantically equivalent usernames and verified that they should be able to signup or not based on whether the usernames are valid.

d. We repeated this activity for all text boxes.

3. For votes and Search (and block user) we used following metamorphic properties :-

a. Permutation: In test 1, for a search query Q we voted the results in order C1, C2, C3. In test 2, for the same query Q we voted the results in order C1, C3, and C2 and checked that the votes are getting registered. (In Bock we did the same with users)

b. For search we tested with semantically equivalence property. Since, lucene uses stemming, the search results for query Q in test 1 should match the search results for the stem Q' generated from query.

Comments from the first author:

Through the properties above, the students show that they have a good grasp of the concept of metamorphic testing. The first author could not find any additional properties apart from what the students did.

Project 7

Similar in concept to http://trendsmap.com, the students in this project proposed an interactive map that aggregates tweets by geographic region. Common words found in tweets coming from a specific location will appear over those geographic locations. To improve on the trends map concept they also proposed to clean up the interface by limiting the number of tweets shown based on a heuristic that they developed. To increase user interaction, they also implemented a search function to allow users to see where tweets matching their input keywords are being created globally.

Metamorphic Properties as found in the students' Testing Report:

We conducted extensive metamorphic testing of the tweet aggregation and display service. We first ran the aggregator on some test data and found the trending tweets per region. We then permuted the order of the tweets as they were run through aggregation algorithm. With this permutation of input (or x') we are expecting f(x') which in this case should equal f(x).

Our next test concerned the sizing of the tweet boxes in proportion to the number of tweets per region. There were three sizes of tweet boxes, small, medium and large. The decision to assign a particular size box to a trend was determined by finding the range of occurrences of all trends and finding the current trend's position within said distribution. Because the labels (small, medium, large) are relative to the total number of trends, we can do metamorphic testing on this property. Multiplying each trend occurrence by X should not change the final outcome of the binning process. The same can be said for subtracting X or adding X to all elements. We tested various combinations and for each X' we expect f(X') to equal f(X).

In addition to trend display size, we checked trend display position and order. By permuting the input to the display map we were able to check to ensure all trends were displayed properly over their location. An interesting caveat to this is that the order does matter because trends that are displayed last can cover trends that are placed on the map earlier. This helped us develop our layout mechanism to allow the most trends on the map without cluttering the display.

The highest level metamorphic test we conducted was we had two (different) users access the site in differing orders. Ideally the display for each user should be the same regardless of who accessed the site first. This tests the entire stack of the application including the client side stack (javascript/web browser). We used some automated javascript tests to ensure that trend objects each user was receiving were the same. We then swapped access order and tested again.

Comments from the first author:

The properties identified by the students are valid; however, there were more that could have been identified. The students had a functionality to display "Word Cloud". The word cloud was created from the tags with maximum counts across all the tweets. Permutation in the order of fetching these maximum count tags should still create the same word cloud. Similarly, the order in which tags are clicked in word cloud should not affect the display of those tagged tweets.

Project 8

This is a web-based application which helps users to choose a restaurant in New York City and make takeout orders. With the increasing numbers of restaurants and variety of types in NYC, it might be hard for users to make a good decision with any suggestion. This application provides users with abundant information about restaurants in NYC and they can search with various constrains, such as cuisine style, location, availability of WIFI and so on. After a user submits the query, a list of candidates that fulfill all constrains will be returned and users can browse the details of each candidate. Each restaurant can also register an account. With this account, a restaurant can post menus and accept users' takeout orders online. The application also has a review system, in which users and restaurants can write reviews of each other after a transaction. All reviews can be referred by others before they decide to start a transaction.

Metamorphic Properties as found in the students' Testing Report:

Client users management display on administrator page:

There are 30 client users shown on one page. Display function f(Y) = Y / 30 = K(output) pages, where Y(input) is the number of all client users. Input deriving function, h(Y) = Y + 30. Output deriving function, g(P) = P + 1.

Display function f(X + 30) = X + 30 / 30 = P + 1

Star users:

If a client user has 5 or more than 5 reviews, he/she will be a star user. The function of verifying the star users f(Y) = decision = {1 for star user and 0 for not}, where Y(input) is the number of comments the user makes. Input deriving function, h(Y) = Y + 5. Output deriving function, g(decision) = 1, since now the number of reviews the user makes is no less than 5.

Thus, f(Y + 5) = g(Y >= 5) = 1

Comments from the first author:

The students have identified valid metamorphic properties. However, there were more properties that could have been identified. Since they had an online order placing system for carry out, permuting the order of placing orders should not change the total number of orders placed finally; neither should it change the total amount that the user who placed the order would have to pay. For instance, if a user places order for item a worth x$ first followed by item b worth y$, then she should pay x+y$ for the two items. Permute the orders and place order for item b worth y$ and then for item a worth x$, the user still pays x+y$ for two items. This property was missed by the students.

Project 9

Cities like Baltimore can be dangerous, but some areas are much more dangerous than others. With this application, however, tourists never have to worry about getting lost again. When a tourist accesses this

application, he inputs his location and the application will immediately notify him regarding the safety of the area. If the area is safe, the application will give a green “all clear” alert. Otherwise, if the area is unsafe, the application will flash red and immediately display the quickest routes to a safer area.

Metamorphic Properties as found in the students' Testing Report:

We constructed four metamorphic test functions to test the implementation of our application.
One of the tests is if we ask for a safe exit route from a current location and the application provides a particular shortest route, then if we move along the given route any amount and ask for a new shortest route, the application should provide the same safe destination as before.
Assume a function f(x) = R which generates a safe exit route R for point x.
Input deriving function, h(x) = x + n (moving n units towards the destination)
Output deriving function, g(R) = R
Initial test case:
Let a search be done for a safe exit route from point x to give route R i.e. f(x) = R
Metamorphic test case:
Derived Input = h(x) = x + n
Derived output = g(R) = R
New Route = f(h(x)) = f(x + n) = R

If we input a set of safety parameters for a particular location and the area is outputted as either safe, moderate or unsafe, then adding a positive integer to one of the element of the input set (such that the sum does not cross the boundary of next higher parameter), then the output will still remain the same for the same location. For example, if the area is safe if the minimum crimes are 10 around that area, then that area will still be marked as safe if we increase the minimum crimes to be marked as safe to 20.
Mathematically,
f(x,y,z) = p, where x = safe parameter value, y = moderate parameter value, z = unsafe parameter value, and p = [safe|moderate|unsafe].
Input deriving function : h(x,y,z) = {x+k, y, z}
Output deriving function : g(p) = p
Initial Test Case:
Let the safety parameters be x, y, z for safe, moderate, unsafe level respectively. f(x, y, z) = p
Metamorphic Test Case:
Derived Input : h(x,y,z) = {x+k, y, z}
Derived Output : g(p) = p
isSafe Function = f(x+k, y, z) = p
Similarly, we can show the same output for y, z parameter.

The other metamorphic properties were:
If we input a value for the address and it returns an error because the address is outside of Baltimore, or any of the cities for which we dont have data, then if we change the value of the address but not the city or zip code by adding or subtracting a given number, then we should still receive the same error message.

If we input an address of an apartment building and it returns one of {safe, moderate, unsafe}, then we should receive the same output value if we input the address of one of the apartments in the previously searched apartment building.

Comments from the first author:

All the properties mentioned by the students are valid metamorphic properties except the last one. The last one seems ambiguous in the form it is presented. However, there were more properties that could have been found. One such example is if initially a place has x as criminal activity level and the safety level is y such that x > y, the place with x crimes is unsafe; incrementing the crime level of the place by n would make the criminal activity level of the place as x + n which will also be classified as unsafe (here n is a positive integer). Similarly x + n will also be classified as unsafe if the safety level is incremented by n as well, i.e., y + n (initially x was the crime level of the place and y was the safety level and initially the place was unsafe).

Project 10

The idea behind this application is to help people discovering the beautiful city of New York (using open data on NYC landmarks), by making them do "quests" consisting of a few questions about a neighborhood. To answer the questions, people will have to walk a little and look carefully around them.

Metamorphic Properties as found in the students' Testing Report:

Test Case 1

Display function: f(N) = M = N, where M is the number of quests that appear on the quest list when N quests are created

Input Deriving Function: h(N) = N+1

Output Deriving Function: g(M) = M+1

Initial test case: X quests are shown when Y quests are created (f(Y) = X)

Metamorphic test case: f(X+1) = Y+1 = X+1

This test case is done to check that the quest create and the quest list functionalities work properly. Basically, when a new quest is created, the quest list should be update to include the new quest.

Test Case 2

The functions for case 2 are very similar to test case 1, but M is the number of questions displayed when N questions are added. (Same for X and Y in the test cases) Similar to the first test case, this test case is done to check that the question list of a quest shows the correct number of questions for that quest.

Test Case 3

Display function: f(R) = L(x+R, y+R), where L is the collection of landmarks within the bounds of x+R, y+R, where x is the center longitude and y is the center latitude.

Input Deriving Function: h(R) = R + 1

Output Deriving Function: g(L(x+R, y+R)) = L(x+R+1, y+R+1)
Initial test case: The landmarks L(x+N, y+N) are within a distance N from the center
f(N) = L(x+N, y+N)
Metamorphic test case: f(N+1) = L(x+N+1, y+N+1)
This test case was performed to test that the quest create screen properly shows only the returned Google place landmarks that are within the boundaries defined by the quest creation.

Comments from the first author:

The have specified valid metamorphic properties but they have missed a few as well. One of the properties that their project has and which is very similar to a property that we have earlier listed for Project 2 is the set of quests suggested for any location within a specified perimeter (the perimeter is already decided by application) should be the same. Further, each quest had a number of objectives that a player could complete in any order. A property here is that the number of objectives completed should be same irrespective of the order in which the objectives were completed, i.e., the total count of objectives completed should be same if same number of objectives are completed irrespective of the fact whether the objectives completed are different or whether the same objectives are completed in different orders.

Project 11

The students designed a web based application that would provide "Reviews of places to checkout in the neighborhood, where your friends might be currently present". Today there are a large number of options that a person can select from; when they want to visit a particular place, and market research shows that when given large numbers of choices to select from humans are not good at making decisions. The aim of this application is to help them make these decisions easily, and not end up with a one that causes disappointed at the end. Users can choose from various categories of places such as night life, movie theaters, restaurants, etc. The application would then get the users approximate location from their IP address. Once the location is known, the application would query data from social networks using their api's and present the user with a visual representation of the data, which he/she can use for their benefit. It would use Yelp and Foursquare APIs to see if any users have checked-in to the place recently. They can also read reviews to decide which place to go to.

Metamorphic Properties as found in the students' Testing Report:

Property 1: Number of people checked into location X as per "trending venues" functionality is m.
If n people check in then total no. of checkins should be m+n.

Justification: Checkin function f(Y) = m = K number of checkins showed on page, where Y is the number of people present.Input Deriving function, h(Y) = Y + N (add N people to the existing count of people) Output Deriving Function, g(K) = K + N (number of checkins increased by N) Initial test case: X people are reflected through P checkins. f(X) = X= P. Metamorphic Test Case: Derived Input = h(X) = X + N Derived Output = g(P) = P + N checkin Function = f(X + N) = X+N= P + N

Thus adding more people to the location should not change the behavior of trending venues. The testing oracle in this case would test if the total number of people checked in at a location would be reflected.

However, with the help of the metamorphic property, we can test abnormal behavior of the system by adding more people to the location and seeing if the number of checkins increased.

Property 2: When the admin logs into the system, he can see m people in the dropdown on the home page. If n more people create their user accounts then the admin can see m+n people in the dropdown. The same should hold true if n people create their user accounts followed by m people.

Justification: The testing oracle in this case would test if the number of users displayed in the dropdown to the admin would be equal to the number of non-admin, non-banned users in the system. However, with the help of the above metamorphic property we would test the scenario where there would be an irregularity in the system behavior if the order of number of people creating an account with the system was permuted. Initial Test Case:If input x = {m people followed by n people} Then f(x) = decision = 1. Metamorphic Test Case: Input Deriving function = h(x) = permute the order of creation input = x' = {n people followed by m people} Output Deriving function = g(decision) = decision(should remain same as old decision). Thus permuting the order should not change the behavior of the system.

Property 3: In case of recommended venues functionality, an "AND" operation is done while searching based on 3 parameters. There should be no search results displayed if even one of the keyword doesn't fetch any results. Justification: The above metamorphic property helped us test scenarios, when searching would fetch zero results if one of the parameters was supposed to fetch zero results. Input Deriving Function for AND = h'(x) = x'' = x AND foo (where foo is not present in the system, i.e. f("foo") = 0 search results since it is not present in the system)Output Deriving function = g'(n) = 0 Calculating f(x'') = f(x AND foo) = f(x) Intersection f(foo) = n intersection 0 = 0 search results(no common search results).

Other properties defined were as follows: 1) The recommended venues functionality should not change irrespective of the order in which the search is done. For instance, a search is done for a particular set of 3 parameters followed by another search for 3 parameters; the functionality should not change irrespective of the order. 2) Similarly, the order should not change irrespective of the current latitude and longitude of the user. For instance, if the user logs in from one location and then goes to another location and logs in, the functionality should not change irrespective of the order. 3) Permuting the order of selecting locations should not change the behavior of the system with respect to displaying the reviews. 4) If a normal user logs into the application followed by an admin user, the functionality observed to both the users should remain the same even if the order in which they login is reversed. 5) Similarly, metamorphic properties were defined while testing the behavior of individual fields on the page. i.e. m characters followed by n characters and reversing the order should not change the behavior of the system. 6) When an admin logs in and bans n people, then the total number of people in the drop down should become m-n where m was the original number of people in the dropdown.

Comments from the first author:

The first author could not find any additional properties. The properties listed by the students are valid and show that they have a good grasp of the concept.

Project 12

The students designed a web-based application which allows users to search for nearby libraries as well as information about books they are keeping. A user can specify the title of the book they'd like to borrow and the app returns a list of libraries having this book, along with their locations shown on a map. Information on whether this book has been checked out will also be available for the user, by either displaying it on the app or showing a link of the library webpage to the user. Some basic books info such as author, publisher, abstract will be provided for helping the user find the right book.

Metamorphic Properties as found in the students' Testing Report:

Function One: When searching libraries or books, if we have lots of results, we should separate the results into several pages. If N results are added to an existing search result count, one more page than the current number of pages is needed to display those results.

Explanation: In Metamorphic Testing, MR is an expected relation among the inputs and outputs of multiple executions of the target program. In our project, if searching result count is M, and the capacity of a page is N, there will be X = M/N pages. When adding more searching results, the page count will be creased by 1.

Transformation: Assume display function f(M) = M/N = X (output) pages, where M is the number of results and N is the number of results per page. Input Deriving function, h(M) = M + P (P is the extra result count). The output function, k(X) = X + 1 (number of pages incremented by 1)

Initial test case: M results are displayed on X pages

f(M) = M/N = X

Metamorphic Test Case;

Derived Input = h(M) = M + P

Derived Output = k(X) = X + 1

Find Display Function = f(M+P) = M + P / N = X + 1

Function Two: When searching libraries or books, we should input a keyword or several keywords to get the number of result. The searching result shouldn't change for each group irrespective of the order in which search is done.

Explanation

In our project, if we change the order of keywords, it should not affect the display of the total number of books or libraries in each group. Permuting the order in which search was done should still display the same number of books or libraries for that area. For example, assume we search two keywords for a book name such as 'hunger' and 'game' First we search the keyword 'hunger' should have X results. Then we search the keyword 'game' should have Y results.

Then we permute the order: First we search the keyword 'game' should have Y results. Then we search the keyword 'hunger' should have X results.

Transformation: Assume f(X) is the list of books searched where X is the keyword we searched. X= {keyword 1, keyword 2} where keyword 1 was queried first in search followed by keyword2, notice Keyword 1 and Keyword 2 should be different keywords. f(X) = list of books in each searching keywords in that order. Notice that the book name contains both keywords will display in high priority but not affect the number of searching result. If Keyword 1 has m results and Keyword 2 has n results.

Then we have f(x) - {m,n}.
Initial Test case:
if X = {Keyword 1, Keyword 2}
f(X) = Y = {m,n}
Metamorphic Test Case:
Input Deriving Function = h(X') = permute the order of keyword for search input
X' = {Keyword 2, Keyword 1}
Output Deriving Function = k(Y) = permute the order of output set = {n,m}
Find f(X') = f({Keyword 2, Keyword 1})={n,m}
It means permuting the order after searching was done; it should still display the same number of books for the relevant books.

Function Three: When we use map to search for the distance and time to arrive the target library, in the dame travel mode, we should get a linear relationship while the travel time increases by the distance increases.
Explanation: In our project, we have three different travel modes in our website, driving, walking and cycling. The searching route is the optimal route from the location of the user to the target library. If the searching distance is L, then the duration time is T. We can get the speed S for different travel modes. S = L/T. When searching another further library, the distance L is increasing and the duration time will linearly increase while the speed S in the same travel mode keeps the same
Transformation: Assume Speed function f(L,T)=L/T=S (output) the speed, where L is the distance between the location of starting point and destination library in a certain travel mode (walking, driving, cycling), T is the time it takes between the distance in the same travel mode. Input Deriving function, h(L,T)= (L + A)/(T+A'), (A is the extra distance from the library to another, A' is the extra time from the library to another). The output function, k(S)=S (the speed in the same mode keep the same as before).
Initial Test Case:
The distance and time spent in a travel mode between the starting point and destination is L and T. The speed in this mode is T.
f(L,T)=L/T=S
Metamorphic Test Case:
Derived Input = h(L,T) = L+A/T+A'
Derived Output = k(S) = S
Find Calculate Route Function = f(L+A,T+A') = L+A/T+A' = S

Comments from the first author:
The students have identified many metamorphic properties correctly. However, there are a few properties that the students missed; one of the properties could be found in the functionality where a user can add books to a favorite list. The order in which the books are added should not affect the total number of books in the list, i.e., if book A is added by user U in his favorite list followed by book B, U's favorite list has two books. Reversing the order and adding book B and then adding book A to an empty favorite list would still have a total of two books in U's favorite list. The students failed to identify this property.

Project 13

Searching for an apartment in NYC is at best a trying enterprise. Existing applications make this easier by making listings easier to submit; search and view apartments, but none take into account the needs of the informed apartment seeker. Instead, the apartment seeker must refer to other external resources or even worse- find things out after having moved in. The application designed by the students empowers the apartment seeker with tools to make the most informed decision by leveraging open data such as transportation data, proximity to neighborhoods, and building complaints.

Metamorphic Properties as found in the students' Testing Report:

For ComplaintInfo.get_complaints we identified the following metamorphic property: A search for complaints of some properly cased address should return the same number of complaints as a search with an improperly cased address.

For this, we constructed a test that takes a sample address (a model-based oracle) and mutates the input by mixing the casing of the characters. We then assert that the results for the mutant are equal to those from the original.

For both the Neighborhoods.get_neighborhoods and TrainStations.stations_within, both of which look for things within a specified distance from a location, we identified the following metamorphic property assuming both methods are encapsulated by f and di is a distance:

f(x,d0) <= f(x, d1) where d1 > d0

That is, for the same location, if the distance threshold is increased, the number of results should be strictly non-decreasing. We implemented tests which verify that this property is maintained for increasing values of d, both for neighborhoods and train stations.

For our facade class for the WalkScore API, we identified the following metamorphic property:

The walk score for two adjacent buildings should be the same.

Transformation:

Assuming get_walk_score function f(Y)=K, where Y is an address and K is its walk score, we defined an input deriving function, h(Y)=X, where X is a building address adjacent to Y and an output deriving function, g(K)=K (walk score is the same).

Initial test: address X has a walk score of K

f(X) = K

Metamorphic test case:

derived input = h(X) = Y (where Y is an adjacent building address)

derived output = g(K) = K

get_walk_score function f(Y)=K

For getFacts method of FactFinder, we identified the following metamorphic property:

Upper or lower case address should return the same facts

Transformation:

Assume getFacts function f(Y,1)=(K,C,S), where Y is the address and K is the walk score, C are the complaints and S are the train stations. We define an input deriving function, h(Y,1)=(y,1) , where y is a lower case address of Y and an output deriving function, g(K,C,S)= (K,C,S) (all information is the same).
Initial test: address X with upper case letters has facts (K,C,S)
f(X) = (K,C,S)
Metamorphic test case:
derived input=h(X)=x (where x is a lower case address)
derived output =g(K,c,S)=(K,C,S)
get_walk_score function f(x)=(K,C,S)
For search_oodle_listing of the OodleSearch class, we identified the following metamorphic property:
Searches with narrower price range should return fewer apartment listing results
Transformation:
Assuming search_oodle_listing function f(X,Y)=a+b, where X is low rent range and Y is high rent range (e.g. 1500-2000 dollars), a are listings with rent > X and rent < Y and b are listings with rent=Y.
We define an input deriving function, h(X,Y)=(X, Y-1) and an output deriving function, g(a+b)=a.
Initial test: price range X to Y has listings results a+b
f(X,Y) = a+b
Metamorphic test case:
derived input = h(X,Y) = X,Y-1
derived output = g(a+b) = a
search_oodle_listing function f(X,Y-1) = a

Comments from the first author:

The students have identified many metamorphic properties correctly. However, there are a few properties that the students missed; the search function took certain criteria (like checkboxes) for searching. Initially, if searching with n constraints can lead to T results, then increasing the number of search constraints should lead to a result set which is a subset of search results obtained when searching with n constraints. Similarly, if x houses are found because of a price range of P, and if k houses exist between prices P + i (where 'i' is positive integer greater than zero) and P and then if P is increased by i, x + k houses should be found now. Further, permuting the order in which search constraints are changed and search is done should not affect the end output, as long as the same set of search constraint is used for initial search test case and metamorphic search test case. The students haven't identified these properties.

Project 14

There are many places where public services are not perfect and there is a desperate need of a platform for residents to report such needs. The project was about designing such a platform. The students designed an application that provides the public with a portal for offering and discussing suggestions

regarding public service to the government, as well as a channel for governments to give feedback for such suggestions.

Metamorphic Properties as found in the students' Testing Report:

1. On suggestion page, we provide an interface where user can choose a place as center and we'll return all points of interests of certain category around this center within a certain radius. If we choose center C, and point of interest P is suggested by our system, then for any center C' such that the distance between P and C' is less than that of P and C, P should also be suggested.
Assume suggestion function S(C) = P.
Input Deriving function, h(C) = C', where dist(P, C') <= dist(P, C)
Output Deriving function, g(C) = P, in other words, it doesn't change.
Initial test case: P is returned if C is chosen as center
S(C) = P
Metamorphic Test Case:
Derived Input = h(C) = C', where dist(P, C') <= dist(P, C)
Derived Output = g(C) = P
Suggestion Function = S(C') = P

2. On our search page, we provide an interface where user can type keywords and we'll search for the problems containing these keywords in their descriptions. For this functionality, assuming user search with a keywords set K, and problems set P is returned, then if we search again with keywords set K', which is a subset of K, then the returned problem set P' should be a superset of P.
Assume search function f(K) = P.
Input Deriving function, h(K) = K', where K' is a subset of K.
Output Deriving function, g(K) = P', where P' is a superset of P.
Initial test case: P is returned if K is the keywords set
f(K) = P
Metamorphic Test Case:
Derived Input = h(K) = K', where K' is a subset of K.
Derived Output = g(K) = P', where P' is a superset of P.
Search function = f(K') = P'.

Comments from the first author:

Students show a good grasp of the concept with the above properties identified by them in their project. However, there were other properties that could have been identified. One such property was in the search functionality: if search with keyword 'x' gets n results then searching with keyword 'x foo' should still get n results (here foo is does not have any search results, i.e, it is not present in the search database). Here the search function works as union of all the keywords, i.e., if search is done for ' x y', then search results for both x and y are displayed, i.e., search results for 'x' union search results for 'y' are displayed. The property mentioned earlier was not identified by the students.

Project 15

The students designed a web application that would provide information about the neighborhood/surrounding of a place such as restaurants, shopping malls, theatres, noise complaints, laundry facilities, subway stations, etc. Users can comment/review these aspects.

Metamorphic Properties as found in the students' Testing Report:

1 During registration, permute the order of password will not change the decision (sign up succeed or not).

Let x be password string contain {m chars, n chars}.
Display function: f(x) = y, where y is a boolean value representing registration succeed or not
Input Deriving function: h(x) = x' = {n chars, m chars} which is permutation of x
Output Deriving function: g(y) = y

2 Basic Search

Property 1:
Add nearby restaurant would change the number of restaurant displayed.
For specific location, let x = {group of restaurants}, n = number of restaurant in 0.5 miles
If we add m more restaurants which are located within 0.5 miles, the basic search should display (n + m) restaurant.
Display function f(x) = n
Input Deriving function, h(x) = {group of restaurants, m new nearby restaurants}
Output Deriving function, g (n) = n + m.
Property 2:
Add nearby laundry may affect the "nearest laundry distance" displayed.
For a specific geographic location, let x = {group of laundries}, D = distance to nearest laundry.
Nearest function f(x) = D
Input Deriving function h(x) = {group of laundries, 1 new laundry y}, y is D' miles away and D' < D
Output Deriving function, g(D) = min{ D, D' } = D'

3 Advanced Search

For a fixed set of preferences weights, the more number of facilities a location has, the higher its score would be. If two locations A and B have same number/distances of all facilities, they should have the same score. If we add one more restaurant to A, it will be expected to have higher score than B.
Let x = {x1, x2, x3, x4, x5}, where x1 represent number of nearby restaurant, etc.
Let w = the vector of corresponding weights.
Scoring function f(x) = x·w = score
Input Deriving function h(x) = x +Δx'
Output Deriving function, g(score) = score +Δx'·w > score

4 Comments

For a specific location, the comment number equals to (added - deleted). If we change the order of add/delete actions, the result should be the same.
Let x = set of actions = {x1 comment actions, x2 delete actions, x3 comment actions}
Comment number function, f(x) = x1 - x2 + x3 = total
Input deriving function, h(x) = {x1 comment actions, x3 comment actions, x2 delete actions}
Output deriving function, g (total) = x1 + x3 - x2 = total

5 History
When a user did another search, and then go to search log, his search amount should increase by 1.
Let x = {previous searching actions}, n = number of search log displayed.
Display function: f(x) = n
Input Deriving Function: h(x) = {previous searching actions, 1 more basic search action}
Output Deriving Function: g (n) = n + 1

6 Admin
Registering and cancelling the account should affect the number of users shown on admin's page.
If a user generated/cancel their account, the total number of user displayed should add/reduce by one.
Let x = {group of user}, n = number of user displayed.
Display Function: f(x) = n
Input deriving function: h(x) = {group of user, m more registered user}
Output deriving function: g(n) = n + m

Comments from the first author:
The first author could not find any additional properties. The properties listed by the students are valid and show that they have a good grasp of the concept.

Project 16

The students designed a web application that would help tourists/visitors of NYC to decide what events to attend and/or which places to visit when coming to cities like New York. Users can get information regarding upcoming events, restaurants and their ratings with regards to food as well as hygiene. They can also have a glimpse of the events that are going on during your period of travel. Further, these events can be personalized – the application will suggest things depending on age group, preferences, the timings of visit, etc. Users can also register their email address and the application will email things that should definitely be seen while in NYC!

Metamorphic Properties as found in the students' Testing Report:
1. Number of events in itinerary
Depending on the user provided preferences, our algorithm picks the event with the highest "score" for each day and adds it to the itinerary. So, if giving n days generates an itinerary with n events, n+1 days should create itinerary with n+1 events.
Transformation: Assume function f(x) picks the events for x(input) days, picking N(output) events.

Input deriving function, h(x) = x + 1, increases number of days by 1.
Output deriving function, g(N) = N + 1, increments number of events by 1.
Initial test case:
f(x) = N
Metamorphic Test case:
Derived input: h(x) = x+1
Derived output: g(N) = N+1
Number of events in itinerary = f(x+1) = x+1 = N+1

2. Event in the itinerary

If the user enters date X and itinerary is created with event A, and, if he enters date Y, an itinerary is created with event B, on entering date X & Y, an itinerary should be created with event A & B. And, so on.

Transformation: Assume function f(X) picks the events for date X, picking event A. And, assume that on applying f() to date Y, f(Y) = B.
Input deriving function, h(XY) = f(X) + f(Y), combines the dates X & Y.
Output deriving function, g(AB) = A + B, adds events A & B to itinerary.
Initial test case:
f(X) = A
f(Y) = B
Metamorphic Test case:
Derived input: h(XY) = f(X) + f(Y)
Derived output: g(AB) = A + B
Events in itinerary = f(X) + f(Y) = A + B

3. Weather reports in the itinerary

Depending on the dates entered for the itinerary, weather for those dates is displayed, if available. Weather is displayed as day and night weather forecast. Hence, 2 entries per day. So, if giving n days, generates displays 2n weather forecasts, n+1 days should display 2n+2 forecasts.

Transformation:
Assume function f(x) displays weather forecast for x(input) days. The output is N
forecasts.
Input deriving function, h(x) = x + 1, increases number of days by 1.
Output deriving function, g(N) = N + 2, increments number of forecasts initinerary by 1.
Initial test case:
f(x) = x * 2 = N
Metamorphic Test case:
Derived input: h(x) = x + 1
Derived output: g(N) = N + 2
Number of events in itinerary = f(x + 1) = (x + 1) * 2 = (x * 2) + 2 = N + 2

Comments from the first author:

The students have identified several properties correctly however they have missed some as well. There is a functionality in the project that suggests similar events for a user to visit/attend to. Thus, if an event is removed all events similar to the removed event should be removed too. The property here is that permuting the order of removing events, should not affect the total count of similar events suggested. For example, let's assume initially there are S similar events. If event A is removed and if event A has x similar events, the count of similar events should be S – x. If event B is removed after that and B has y similar events, total similar events suggested now should be S- x- y. If we change the order and first remove event B and then remove event A, the total remaining similar events should still be S – x – y. The students did not find this property.

Project 17

The project uses the NYC open data repository dataset about WiFi hotspots across the city.(https://nycopendata.socrata.com/Media/WiFi-map-remix/fi99-sd8n). Various businesses allow WiFi connectivity to increase customer services and popularity of their firms. This dataset has the address, contact information of the businesses and other public places where WiFi connectivity is available. Users can search for and get directions to WiFi hotspots, as well as give ratings and leave comments.

Metamorphic Properties as found in the students' Testing Report:

Property 1

If the average rating entered by a user who has never provided a rating for a hotspot is equal to A_old where A_old is same as the current average rating. Then the average rating should remain unchanged.

Property 2

If the average rating entered by a user who has never provided a rating for a hotspot is greater than A_old by a number say x, where A_old is the old rating average. Then the average rating should increase by a constant d which can be calculated (please see explanation for details).

Property 3

If the average rating entered by a user who has never provided a rating for a hotspot is less than A_old by a number say x, where A_old is the old rating average. Then the average rating should decrease by d. (please see explanation document for details).

Property 4

The search results in tabular form should be displayed in order sorted by increasing distance of the address provided. The topmost hotspot entry should be closest to the given address, the second hotspot entry be second closest and so on. This should apply for any set of valid inputs to search for WiFi hotspots.
(Assume all other parameters to be entered are constant and valid)

Property 5

Given, a valid user with email ‘e’, concatenate ‘c’ with ‘e’ to produce a new ‘u’. ‘u’ must never be able to log in.

Property 6
Given, a valid user with password p, concatenate ‘c’ with ‘p’ to produce a new ‘u’. ‘u’ must never be able to log in.

Comments from the first author:
The stated properties don’t show any notion of transformation and don’t appear to be metamorphic in their current form. One metamorphic property in their project was if the average rating provided by n people for a hotspot was ‘avg’ then if all the ratings are multiplied by 2 then the new average rating should be 2*avg. Similarly deleting a rating and adding a new rating of same value should not affect the avg. The students may have thought about similar properties, however the way the properties are mentioned shows that the students did not understand the notion of metamorphic testing correctly.

Project 18

The students implemented a web application version of amazing race. Users can generate locations of interest and compete with their friends to visit them all. Points are awarded for every location visited. Time can also be used to “race” against other teams.

Metamorphic Properties as found in the students’ Testing Report:

Changing the number of locations that belong to a race should change the amount of locations displayed in a race.

Changing the number of users that belong to a race should change the amount of users displayed in a race.

Adding a friend should increase the number of friends displayed in the selection drop down in the “race create” screen by one.

Adding a race should increase the number of races that are in your “resume a race” screen by one.

Adding a friend should increase the number of friends that a user has by one.

Sending a message to a user with N messages gives that user N+1 messages.

Comments from the first author:
The students identified several properties. However, there were more that could have been found. A few properties that the students could not identify, are similar to some of the properties that we have mentioned earlier. Another property that exists in their top score functionality has not been discussed yet for any of the projects. The property is : if a game is

played and the score obtained is smaller than the lowest score on the top scorers' list, then the score (and the player) should not appear on the list.

Project 19

The aim of this project is to create a system for designing complex cities using pre-made repositories and paint on a plane. The generated city will be supplemented by NYC OpenData to add traffic, noise, lights (electricity consumption), and other components based on attributes such as area wealth, industrial/commercial/residential zones, and others attributes. Many city planners start out by planning the types of buildings and the nature of the businesses or residences in the area they are planning. However, features like the traffic (both pedestrian and vehicular) and energy consumption or even crime projections are unavailable in initial designs. The system that the students developed utilizes information provided by the planner and then leverages NYC OpenData to layer on this additional information like traffic and energy consumption. This will be put together to create models of cities with buildings of different types that gives the designer in-depth information that would not ordinarily have been available at this early stage.

Metamorphic Properties as found in the students' Testing Report:

As detailed above, the nature of our program, being a plugin for blender necessitated we tackle testing by testing some of our major components of the project separately. This was true for metamorphic testing as well.

For the model manager explained above, the metamorphic testing was performed as follows:

i. Pass in the same building entry to the data manager and ensure that the same values are used to populate the remaining fields.

ii. Pass in the same set of buildings in reverse order and ensure that the order has nothing to do with what values the data manager returns when called by the model manager

For the DataManager class, test cases were created that checked to make sure that data files with the same distribution of data were processed equivalently. For instance, have one set of data files with a known data distribution when fed into DataManager. Then, for each data file, duplicate all of the rows. The result should be a file that is twice as large, but that should end up having the same distribution as the original set of data- as such, the output of DataManager should be the same for both sets of data files.

Comments from the first author:

The properties stated by the students don't come across as valid metamorphic properties. Further, there is no clear idea of transformation demonstrated in the above stated description. From their requirements, it is clear that the students were using a clustering algorithm to make connection between groups of data such as electricity consumption, noise levels, wealth, and

building height. One metamorphic property here could be permuting the order in which input is fed into the algorithm should not change the output, given that the same set of input is fed.

Project 20

The students created a web application that allows users to plan a day in Times Square. It uses NYC Opendata and other sources to show the list of restaurants, theaters, museums, movies, comedy club, stores, parking etc in the area. The application also shows upcoming events information at Times Square. The application allows user to plan his itinerary. The user of this system can recommend places and also post reviews about it. It also integrates with facebook which allows user to post his itinerary on his wall. Based upon the users reviews, the application recommends things to be done at Time Square.

Metamorphic Properties as found in the students' Testing Report:

1. The TodoList of a user resulted from adding n events should be same irrespective of the order in which the events are added, i.e., if we permute the order of adding events, the TodoList should not change.
For example : If adding n1 Events to a TodoList followed by adding n2 events results in a TodoList(T) for a user with n types of events. Then changing the order of adding events by adding n2 events followed by n1 events for the user should still result in the same TodoList(T) with n types of events .

2. If a user has N events on his todoList and number of events on a user's TodoList increases by x*N, then total number of events on a user's TodoList will be x*N +N=N(x+1)

For example: If a user has N events and he add N more events to his TodoList, then total number of events on his TodoList will be N*2=2N

3. If a user selects a date and adds X events to the TodoList of that day, and then if he tries to create a todolist of the same day and add M events, the same TodoList should have M+X events. It should not create 2 different TodoList of the same day with M and X number of events respectively.

For example: If a user has a TodoList for 21st Nov 2012 with M events and if he tries to create another TodoList for the same day and add X events to it, then the system should have only one TodoList for the day with M+X number of events in the TodoList.

4. The Number of Likes (N) of an event resulted from Liking N times should be same irrespective of the order in which the events have been liked, i.e., if we permute the order of liking the events the Number of Likes (N) should not change.

For example: For example, if liking an event n1 times followed by liking event for n2 times results in N (Number of Likes). Then changing the order and liking n2 times followed by liking n1 times should still result in N Number of Likes.

Comments from the first author:
The students have done very well and have identified most of the metamorphic properties that existed in their projects. However, they have missed one property that the first author could identify. According

to the requirements in their project, likes are used to calculate popularity. There can be a few metamorphic properties here. One property could be, if the number of likes for an event is increased, the popularity of the event in its genre should increase as well. For example, if number of likes for event X is increased by k', then its popularity should increase by ((k + k')/(T + k) – k/T) *100 percent. Here, k is the initial number of likes for event X and T is the total number of likes for all events in the same class/genre. So, if initially, if event X with k likes has a popularity of k/T *100 percent, then after increasing the number of likes for event X by k' should change the popularity of event X to (k/T *100) + ((k + k')/(T + k) – k/T) *100 percent.

Project 21

The students designed a system that allows users to keep track of and be alerted to movies and shows that are being filmed in the NYC area. Like GasBuddy, the system would rely on users posting sightings of the "purple posters" announcing filming in the area (For those who aren't familiar with purple posters, posters are generally placed around filming sites to inform passing pedestrians about the film that is being shot. Since these posters are often -- but not always -- purple, the students called their project "purple posters"). The system could then use the Open Data from RottenTomatoes.com to pull ancillary information such as actors, genre, etc., related to this film specified on the poster. This data would be used to alert other users who are interested in movies matching certain criteria that a movie they might be interested in is filming near them. Users can leave feedback and comments based on their experience.

Metamorphic Properties as found in the students' Testing Report:

First Metamorphic Property: Search Functionality

Purple Poster application has a search function that allows users to search purple posters, movie names, and actor names using a text box. Here, we will describe this function's behavior in mathematical format. From there, we will deduce some metamorphic properties and create test cases and execute them. We start with defining terms and entities.

A Purple Poster entry : P = {poster_name, movie_name, (actor_name1, actor_name2...) }

A Search Key S ={s},

If "s" matches any of the entries in element P, it returns element P.

Let's say Q is a set of P's and search function is f(s, Q) that returns a subset of Q of matching P's.

Test Case preparation

Added purple poster "Alias," movie "Man of Steel,", date with next day's date

(12/8/2012) and location as "central park."

We can now define the following metamorphic test cases:

● f(s,Q) returns 3 elements (start with creating a database of purple posters, movies, actors that have some common names)

Search for "man of steel." Returned 1 poster

Test Case (1)

● f(s, Q + R) should return at least 3 elements where R is a random set of P's.

Added this movie 3 more times with a different alias each time: Alias1, Alias2, Alias3.

Search for “steel,” and got back 4 movies as expected.
Test Case (2)
● f(s + t, Q) is expected to return less elements than f(s, Q).
Changed search key from "man of steel" to "man of steel superman" and no rows got returned as expected.

Second Metamorphic Property: Adding a new movie functionality
This functionality has less interesting metamorphic properties. We describe our method as follows:
The front page always shows the unexpired purple posters. When we run adding function f(p,t) where p is the purple poster details and t is the date of the purple poster, if t is >= current date, it appeared in the front end (n is the number of the purple posters in the front end). We tried the same functions with following--Created few entries with different dates:
Test Case (3)
● t = current date, i > 0
f(p', t+i) - As expected, this test resulted in more movies appearing in the front page
f(p', t-i) - As expected this test did not change the number of movies appearing in the front page.

Comments from the first author:
From the description above, it seems that the students did not understand the concept of metamorphic testing completely. The properties stated above may seem like metamorphic however they are not because the properties do not show precise transformation. For a property to be metamorphic, the transformation has to be precise. The properties mentioned by the students lack this understanding of precise transformation since they mention the transformation in terms of “less than” which is clearly not precise.

Analysis and Observations

Equivalence Partitioning and Boundary Value Analysis vs. Metamorphic Testing on the Individual Assignment:

Table V below shows some statistics on the students’ scores for the individual assignment.

	Equivalence Partitions/Boundary Value Analysis (max 20 points)	**Metamorphic Testing (max 10 points)**
Mean	11.88 (59.4%)	4.33 (excluding zeroes) (43.3%) 1.79 (including zeroes) (17.9%)
Median	12 (60%)	0 (0%)
Standard Deviation	4.45	2.59

Table V: Statistics on students’ scores for the Individual Assignment.

The statistics in Table V are derived from the scores received by the students for the individual assignment. It can clearly be seen that the students did much better on equivalence partitions and boundary value analysis as compared to what they did on metamorphic testing. The mean score for equivalence partitions and boundary value analysis was almost 60% whereas the mean score for metamorphic testing was 43.3% for those scores that were awarded. Some students received a zero for metamorphic testing, and the mean for metamorphic testing score including zero scores was 17.9%. 46 students (more than 50% of the class) got a zero for metamorphic testing!

There are two plausible inferences from this result: 1) Students did understand metamorphic testing but for some reason could not apply it to the homework assignment, or 2) Most of the students did not understand metamorphic testing at all. In order to drive speculation out of the equation, the first author (who was also one of the graders of the assignment) took a random sample from the students' solutions and carefully reviewed their answers. It was found out that most of the students could define something that might be called a "property" of the suggested software, but did not understand how to determine whether that property was metamorphic or not. In particular, the students described properties without showing any precise/correct transformation of input and output. Thus, those students got low scores for metamorphic testing. Hence, it is safe to conclude that the students did not understand the concept of metamorphic testing completely/correctly.

Equivalence Partitioning and Boundary Value Analysis vs. Metamorphic Testing on the Team Projects:

Table VI below shows the performance of students with respect to equivalence partitions/boundary value analysis and metamorphic testing on their projects. For ease of analysis, we did a binary classification such that if the students have used the concept correctly in testing, the project is classified as A and if they apparently lack understanding of the concept or have used the concept incorrectly in testing, the project is classified as B.

Class	**Number of Projects**	**% out of all the projects(21 projects)**	**Project Ids**
Class A: Equivalence Partition and Boundary Value Analysis concepts used correctly	21	100	1,2,3,4,5,6,7,8,9,10,11, 12,13,14,15,16,17,18,19, 20,21
Class B: Equivalence Partition and Boundary Value Analysis concept used incorrectly	0	0	NA
Class A: Metamorphic Testing concept used correctly	17	80.96	1,3,4,5,6,7,8,9,10,11, 12,13,14,15,16,18,20,21
Class B: Metamorphic Testing concept used incorrectly	4	19.04	2,17,19,21

Table VI: Classification of projects that used the concept of Equivalence Partitions/Boundary Value Analysis and Metamorphic Testing correctly/incorrectly.

100% of the students correctly applied the notions of equivalence partition and boundary value analysis, whereas almost 81% of the class could do the same with metamorphic testing, in contrast to the dismal performance on the individual assignments. Although for some projects the first author was able to find some additional properties apart from what the students found, the students could correctly show precise transformations of input and output and correctly apply these test cases to their applications for those properties they did find. Almost all the properties identified by the students in their test plan were demonstrated to be metamorphic by following the outline at the beginning of the solutions set.

Conclusions

In order to better appreciate these findings, we divide the projects into three categories: The projects that found all of the properties found by the first author were placed in Class A; the projects that could determine some of the metamorphic properties but not all were put in Class B; the projects that could not understand what metamorphic properties are and reported some false metamorphic properties that did not exist in the project or did not report any metamorphic property at all were placed in Class C. As previously noted, the first author is not an expert on metamorphic testing but received considerably more training (mostly while conducting the first study [6]) than the classroom students. Table VII shows the total number of projects in each category.

Class	**Number of Projects**	**Percentage of Projects**	**Project IDs**
Class A: found all Metamorphic properties	5	23.8	3, 4, 6, 11, 15
Class B: found some Metamorphic properties	12	57.14	1, 5, 7, 8, 9, 10, 11, 12, 13, 14, 16, 18, 20
Class C: didn't understand/reported false Metamorphic properties	4	19.06	2, 7, 19, 21

Table VII: analysis of fall 2012

Table VIII presents the corresponding classification data from the previous report (with the same first author) analyzing the fall 2011 offering of the course [6].

Class	**Number of Projects**	**Percentage of Projects**	**Project IDs**
Class A: found all Metamorphic properties	5	29.41	1, 2, 13, 15, 17
Class B: found some Metamorphic properties	7	41.18	5, 7, 8, 9, 10, 11, 16
Class C: didn't understand/reported false Metamorphic properties	5	29.41	3, 4, 6, 12, 14

Table VIII: analysis of fall 2011

As we can see, the students in fall 2012 did somewhat better than the students in fall 2011. However, even in the initial offering, a substantial portion of the class (12 out of 17 projects, 70.59%) was indeed able to apply metamorphic testing to some degree in their team projects.

This paper has presented the results of teaching the concept of Metamorphic Testing in a senior/graduate classroom environment. These results give us reason to believe that it is plausible for student testers to perform metamorphic testing with minimal training. The curriculum materials were somewhat different across the two course offerings. In the initial offering there were two lectures devoted instead of just one, but there was no individual assignment – and, perhaps most significantly, no solutions set hinting at a template for specifying metamorphic test cases. Further, the surveys added during the second offering might have helped signal the students as to the importance the teaching staff placed on testing techniques in general and metamorphic testing in particular. The metamorphic testing aspect of the team project assignment was the same in both cases. We believe that the fall 2011 offering represents a bit of a false start, when the teaching staff imagined that lectures alone would suffice, whereas less talking and more doing was shown to work better in fall 2012.

Acknowledgements

The Programming Systems Laboratory is funded in part by NSF CCF-1161079, NSF CNS-0905246, and NIH U54 CA121852.

We would like to thank Chris Murphy, a leading metamorphic testing researcher, for his comments and suggestions. We would also like to thank the TAs for the fall 2012 offering of the course, Priyank Singhal, Kunal Ghogale and Jonathan Bell, for their cooperation and support in conducting this study.

References

[1] T. Y. Chen, S. C. Cheung, and S. Yiu. *Metamorphic testing: a new approach for generating next test cases.* Technical Report HKUST-CS98-01, Hong Kong University of Science and Technology Department of Computer Science, January 1998.

[2] Matt Chu, Christian Murphy and Gail Kaiser. Distributed In Vivo Testing of Software Applications. Student paper track in *1st IEEE International Conference on Software Testing, Verification, and Validation*, April 2008.

[3] Christian Murphy, Gail Kaiser, Ian Vo and Matt Chu. Quality Assurance of Software Applications Using the In Vivo Testing Approach. *2nd IEEE International Conference on Software Testing, Verification and Validation*, April 2009.

[4] Christian Murphy, Kuang Shen and Gail Kaiser, Using JML Runtime Assertion Checking to Perform Metamorphic Testing in Applications without Test Oracles. *2nd IEEE International Conference on Software Testing, Verification and Validation*, April 2009.

[5] Christian Murphy, Kuang Shen and Gail Kaiser, Automatic System Testing of Programs without Test Oracles. *International Symposium on Software Testing and Analysis*, July 2009.

[6] Kunal Swaroop Mishra and Gail Kaiser, *Effectiveness of Teaching Metamorphic Testing.* Technical Report cucs-020-12, Columbia University Department of Computer Science, November 2012.

[7] Swapneel Sheth, Jonathan Bell and Gail Kaiser. A Competitive-Collaborative Approach for Introducing Software Engineering in a CS2 Class. *Conference on Software Engineering Education and Training*, May 2013.

[8] Elaine J. Weyuker. On testing non-testable programs. *Computer Journal*, 25(4):465-470, November 1982.

Appendix I (Testing Familiarity Survey Results)

Number of records in this query:		84
Total records in survey:		84
Percentage of total:	100.00%	

Field summary for 1(SQ001)
How familiar are you with the below mentioned concepts?

Please click on a radio button between 1-5 as per your familiarity (1 being the least and 5 being the highest).
[Software Testing]

Answer	Count	Percentage	Sum
1 (1)	7	8.33%	25.00%
2 (2)	14	16.67%	
3 (3)	37	44.05%	44.05%
4 (4)	25	29.76%	
5 (5)	1	1.19%	30.95%
Sum (Answers)	84	100.00%	100.00%
Number of cases	84	100.00%	
No answer	0	0.00%	
Not displayed	0	0.00%	
Arithmetic mean	2.99		
Standard deviation	0.92		

Field summary for 1(SQ002)
How familiar are you with the below mentioned concepts?

Please click on a radio button between 1-5 as per your familiarity (1 being the least and 5 being the highest).
[Unit Testing]

Answer	Count	Percentage	Sum
1 (1)	7	8.33%	34.52%
2 (2)	22	26.19%	
3 (3)	22	26.19%	26.19%
4 (4)	32	38.10%	
5 (5)	1	1.19%	39.29%
Sum (Answers)	84	100.00%	100.00%
Number of cases	84	100.00%	
No answer	0	0.00%	
Not displayed	0	0.00%	
Arithmetic mean	2.98		
Standard deviation	1.02		

Field summary for 1(SQ003)

How familiar are you with the below mentioned concepts?

Please click on a radio button between 1-5 as per your familiarity (1 being the least and 5 being the highest).
[Boundary Value Analysis]

Answer	Count	Percentage	Sum
1 (1)	23	27.38%	51.19%
2 (2)	20	23.81%	
3 (3)	24	28.57%	28.57%
4 (4)	15	17.86%	
5 (5)	2	2.38%	20.24%
Sum (Answers)	84	100.00%	100.00%
Number of cases	84	100.00%	
No answer	0	0.00%	
Not displayed	0	0.00%	
Arithmetic mean	2.44		
Standard deviation	1.14		

Field summary for 1(SQ004)
How familiar are you with the below mentioned concepts?

Please click on a radio button between 1-5 as per your familiarity (1 being the least and 5 being the highest).
[Equivalence Partitioning]

Answer	Count	Percentage	Sum
1 (1)	42	50.00%	72.62%
2 (2)	19	22.62%	
3 (3)	15	17.86%	17.86%
4 (4)	6	7.14%	
5 (5)	2	2.38%	9.52%
Sum (Answers)	84	100.00%	100.00%
Number of cases	84	100.00%	
No answer	0	0.00%	
Not displayed	0	0.00%	
Arithmetic mean	1.89		
Standard deviation	1.09		

Field summary for 1(SQ005)
How familiar are you with the below mentioned concepts?

Please click on a radio button between 1-5 as per your familiarity (1 being the least and 5 being the highest).
[Code Coverage]

Answer	Count	Percentage	Sum

1 (1)	21	25.00%	55.95%
2 (2)	26	30.95%	
3 (3)	21	25.00%	25.00%
4 (4)	14	16.67%	
5 (5)	2	2.38%	19.05%
Sum (Answers)	84	100.00%	100.00%
Number of cases	84	100.00%	
No answer	0	0.00%	
Not displayed	0	0.00%	
Arithmetic mean	2.4		
Standard deviation	1.11		

Field summary for 1(SQ006)
How familiar are you with the below mentioned concepts?

Please click on a radio button between 1-5 as per your familiarity (1 being the least and 5 being the highest).
[Unit Testing Tools]

Answer	Count		Percentage	Sum
1 (1)		18	21.43%	59.52%
2 (2)		32	38.10%	
3 (3)		21	25.00%	25.00%
4 (4)		11	13.10%	
5 (5)		2	2.38%	15.48%
Sum (Answers)		84	100.00%	100.00%
Number of cases		84	100.00%	
No answer		0	0.00%	
Not displayed		0	0.00%	
Arithmetic mean		2.37		
Standard deviation		1.04		

Field summary for 1(SQ007)
How familiar are you with the below mentioned concepts?

Please click on a radio button between 1-5 as per your familiarity (1 being the least and 5 being the highest).
[Integration Testing]

Answer	Count		Percentage	Sum
1 (1)		21	25.00%	52.38%
2 (2)		23	27.38%	
3 (3)		20	23.81%	23.81%
4 (4)		20	23.81%	
5 (5)		0	0.00%	23.81%
Sum (Answers)		84	100.00%	100.00%
Number of cases		84	100.00%	
No answer		0	0.00%	

Not displayed	0	0.00%
Arithmetic mean	2.46	
Standard deviation	1.11	

Field summary for 1(SQ008)
How familiar are you with the below mentioned concepts?

Please click on a radio button between 1-5 as per your familiarity (1 being the least and 5 being the highest).
[In-Vivo Testing]

Answer	Count	Percentage	Sum
1 (1)	55	65.48%	89.29%
2 (2)	20	23.81%	
3 (3)	4	4.76%	4.76%
4 (4)	5	5.95%	
5 (5)	0	0.00%	5.95%
Sum (Answers)	84	100.00%	100.00%
Number of cases	84	100.00%	
No answer	0	0.00%	
Not displayed	0	0.00%	
Arithmetic mean	1.51		
Standard deviation	0.84		

Field summary for 1(SQ009)
How familiar are you with the below mentioned concepts?

Please click on a radio button between 1-5 as per your familiarity (1 being the least and 5 being the highest).
[Metamorphic Testing]

Answer	Count	Percentage	Sum
1 (1)	53	63.10%	89.29%
2 (2)	22	26.19%	
3 (3)	5	5.95%	5.95%
4 (4)	3	3.57%	
5 (5)	1	1.19%	4.76%
Sum (Answers)	84	100.00%	100.00%
Number of cases	84	100.00%	
No answer	0	0.00%	
Not displayed	0	0.00%	
Arithmetic mean	1.54		
Standard deviation	0.86		

Field summary for 1(SQ010)
How familiar are you with the below mentioned concepts?

Please click on a radio button between 1-5 as per your familiarity (1 being the least and 5 being the highest).
[Formal Specifications/Model Checking]

Answer	Count	Percentage	Sum
1 (1)	36	42.86%	80.95%
2 (2)	32	38.10%	
3 (3)	8	9.52%	9.52%
4 (4)	8	9.52%	
5 (5)	0	0.00%	9.52%
Sum (Answers)	84	100.00%	100.00%
Number of cases	84	100.00%	
No answer	0	0.00%	
Not displayed	0	0.00%	
Arithmetic mean	1.86		
Standard deviation	0.95		

Appendix II (Individual Assignment Questions)

For this assignment, you will do Unit Testing for the 2 systems described below.

For each system, describe the following:

- Black Box Testing using Equivalence Partitions and Boundary Value Analysis (10 points)
- Black Box Testing using Metamorphic Testing (5 points)
- White Box Testing (5 points)

For both the Black Box Testing parts, describe specific equivalence partitions, boundary cases, and metamorphic properties.

For the White Box Testing, it's sufficient to describe the general approach you would take (You don't have specific source code to work with, for this assignment.)

1. Hotel Booking System
   Consider a Hotel Booking system with the following features

   a. You can search for a particular hotel and the search feature displays N results per page. If search results for a particular hotel are M and M > N such that M/N = X, search feature shall display X pages upon searching for this hotel. For instance, let's assume the search feature displays 10 results per page. Now, if for a particular keyword, there are 100 search results, then the total number of pages that would be displayed upon searching for that keyword would be 100/10 = 10. You are only required to test the search result display function. You do not need to write test cases for the search functionality.

   b. Upon successful search, you can book a hotel, reschedule your booking and cancel your booking. It is safe to assume that these operations manipulate tariff correctly. Also keep in mind that booking is only active for current year, i.e., you do not have to provide an year of booking to the system.

   Some examples of rescheduling the booking are:

   i. Book a room for duration 11th - 15th. Now cancel the booking for one day i.e. 15^{th} and the new booking should be from 11^{th}-14^{th}.

   ii. Book a room for duration 11th - 15th. Now extend the booking to 16th. The new booking should be from 11^{th}-16^{th}.

2. Social Networking System
   Consider a Social Networking site with the following features

   a. The application has something similar to a “WALL” where you can post comments. The default size of the wall is N characters (including special characters and spaces/tabs). When your friend’s wall space reaches N-100 characters, the wall size resizes, i.e, it becomes (current size of wall) + N. For instance, let’s assume the wall can hold 1000 characters. When there is a total of 900 characters on the wall, its new size would be 900 + 1000 = 1900. Now when the number of characters on the wall reaches 1900 - 100 = 1800 characters, the wall resizes again, i.e., its current capacity is 1800 + 1000 = 2800 characters. You can assume that the wall can grow infinitely. Your testing should only focus on the resizing feature of the wall.

   b. As any other social networking site, this one also has an upload picture feature. However, there is a page dedicated to your profile picture and you can put the profile picture anywhere on the page. There is no pre-defined slot (on that page) to put the picture in. However the picture is of only a predefined size M sq unit. The area of the page is A sq unit. Your friends can leave their comments on any portion of that page but they can’t write on the picture. The comments are horizontal (like in a text editor). For instance, let’s say the area given to the picture is 100 sq units. And the total area of the picture page is 1000 sq units. You can upload your picture and then drag and drop on any part of the picture page. After uploading the picture, you are only left with 1000 – 100 = 900 sq units where your friends can write their comments. You can assume that each comment is 100 sq units in area.

   The app has a search feature and the search feature searches for a person. The search displays N results and is very similar to the search feature in the Hotel Booking System mentioned above. As in Hotel Booking System, you are only required to test the search result display function. You do not need to write test cases for the search functionality for the normal credit.

Extra Credit (20 points)

Do the same Black Box (Equivalence Partitioning, Boundary Value Analysis, Metamorphic Testing) and White Box testing as required above for the extended system described.

Extending the Social Networking site

   a. The search feature is modified and the search can be for multiple keywords. You can either specify to search for keyword1 OR keyword2 or search can be done for keyword 1 AND keyword2 or you can search for an exact string. (OR and AND

have the same contextual meaning as in Logical Operations). Again, you are only required to test the search result display function.

b. You can divide your friends in groups and you can search your friends by groups. There is no limit on the number of groups that you can make. You can assume the groups have names like professional acquaintance, school friends, etc.

c. You get suggestions for friends. The friends' suggestion feature works by finding out how many of your current friends know a particular person. If the count of your friends that know a certain person (who is not your friend) is greater or equal to than a threshold value, lets' say N, the person appears in your friends' suggestion. For instance, let's say 10 of your friends know me. If the threshold value for friend's suggestion is 9 (greater than or equal to), then I should be suggested to you as a probable friend by the application.

d. There is a concept of the state of the wall. The number of messages/comments on your wall defines the state of the wall, i.e., state of the wall is a function of the number of messages on the wall. Obviously, you can delete unwanted messages from your wall.

Note: For the Extra Credit portion, you cannot have any help from the TAs or the instructor. Any clarification questions must be addressed to the instructor.

Appendix III (Individual Assignment Answers)

Only the portions relevant to metamorphic testing are shown.

Solution for Metamorphic Testing:

For metamorphic testing, there is a function (input deriving function) that derives a new input I' from old input I and there is another function that derives an expected output (output deriving function) O' from old output O. So when the software under test is run with input I' and gets an actual output O'' it is compared to the expected output O'. You have to describe how this comparison is done. This may not be necessarily equality.

Note: Some of the solutions (here) for metamorphic properties are derived from the notion of invariants and can be considered a special, trivial case of metamorphic properties.

1.a

Explanation:

If N results are added to an existing search result count, then one more page than the current number of pages is needed to display those results.

If search result count is x and takes p pages to be displayed, then x + N should take p+1 pages for display.

Transformation:

Assume

Display function f(Y) = Y/N = K(output) pages, where Y(input) is the number of results and N is the number of results per page.

Input Deriving function, h(Y) = Y + N (add N to the existing search result count)

Output Deriving Function, g(K) = K + 1 (number of pages incremented by 1)

Initial test case: X results are displayed on P pages.

f(X) = X/N = P

Metamorphic Test Case:

Derived Input = h(X) = X + N

Derived Output = g(P) = P + 1

Display Function = f(X + N) = X+N/N = P + 1

1.b

Property1

Explanation

Extending a booking of x days by n more days should result in an increase in tariff by n days' tariff.

i.e. if you book for x days then you pay x *T $ where T is the per day tariff

if you extend it to x + n days, then you should be paying (x+n)*T $

Transformation

Assume function f(y) generates tariff for a booking of a number of days.

f(y)= (y*T) = Ty $ where T is the per day Tariff and y is the booking period.

Input Deriving function, h(y) = y + n, where n is a positive integer (extend the booking by n days)

Output Deriving Function, g(Ty) = g(y*T)= y*T + n*T = (y+n) * T$ (tariff should increase by n*T$)

Initial Test Case: Let a booking be done for a period of x days.

Tariff for x days = f(x) = (x*T)$ = Tx$

Metamorphic Test Case:

Derived Input = h(x) = x + n

Derived Output = g(Tx) = g(x*T) = x* T + n*T = (x+n) *T$

New Tariff = f(h(x)) = f(x+n) = (x+n)*T$

If you extend the booking for n days on top of initial booking of x days, you should be paying (x+n) * T $ as compared to initially paying x*T$.

Property2

Explanation

The amount resulted from the booking of n rooms should be same irrespective of the order in which the rooms are booked, i.e., if we permute the order of room booking the final amount/tariff should not change.

For example, if booking Room1 for n days followed by booking Room2 for m days results in a total tariff of D $. Then changing the order and booking Room2 for m days followed by booking Room1 for n days should still result in a tariff of D$

Transformation

Assume that f(x) is a booking tariff calculator function where you input a valid period of dates and f(x) returns the tariff for the number of days booked. x can be multiple periods for multiple rooms , e.g., x={Room R1 for n days, Room R2 for m days} where Room R1 was booked first and Room R2 was booked after R1.

f(x) would calculate the total tariff sequentially, i.e., first for Room R1 and then for Room R2.

If we assume that each room has the same tariff then

if x={Room R1 for n days, Room R2 for m days},

f(x) = n*T + m*T = (n+m)*T$ = T'$

Input Deriving function = h(x) = Permute the order of input set of bookings = x'

If x = {Room R1 for n days, Room R2 for m days},

h(x) = x' = {Room R2 for m days, Room R1 for n days}

Output Deriving Function = g(T') = T'$(same tariff should be generated)

Initial Test case: Let y = {Room1 for z days, Room2 for s days}

If Room1 and Room2's tariff is T$

f(y) = z*T + s *T= (z+s) * T$ = Tm $

Metamorphic Test Case:

Derived Input = h(y) = y' = { Room2 for s days, Room1 for z days} (Permuted order)

Derived output = g(Tm) = Tm $

New Output = f(y') = s*T + z* T = (s+z)*T $ = (z+s)*T $ = Tm $.

If booking in certain order lead to a total amount of X $, then permuting that order would also lead to X $ as total tariff.

2.a

Property 1

Explanation

If the number of characters on the wall is increased by N-100, then the number of resizes should be increased by 1, i.e., if N -100 characters lead to resize of the wall once, x*(N-100) should lead to resize of the wall x times.

Transformation

Let f(x) decides about the number of resizes where x is the number of characters on the wall.

If x > N-100 but <2*(N-100), then the number of resizes should be 1.

f(x) = 1 where N-100 <= x <2*(N-100)

Generalizing the above formula

f(x) = k where k*(N-100) <= x < (k+1)*(N-100)

Input Deriving Function = h(x) = x + N – 100 (adding N-100 to the present number of characters)

Output Deriving Function = g(k) = k + 1 (incrementing current output by 1)

Initial Test Case: If x is the number of characters on the wall, then the number of resizes for those x characters should be

f(x) = k where k*(N-100) <= x < (k+1)*(N-100)

Metamorphic Test Case:

Derived Input = h(x) = x + N – 100

Derived Output = g(k) = k + 1

Number of resizes = f(x + N - 100) = k + 1(since now (k+1)*(N-100)<= x+N-100 <(k+2)*(N-100))

Property2

Explanation

The order in which characters are input on the wall should not affect the resizing feature. If first inputting n characters followed by m chars leads to resize of the wall then inputting m chars followed by n chars should lead to resize of wall as well.

Transformation

Let f(x) be the function that decides whether a resize should happen or not. Let x = {n chars followed by m chars}, i.e., sequence of input characters.

f(x) calculates the total number of characters input and then decides if total chars>=k*N-100.

Output of f(x) = decision = {1 for resize, 0 for no resize}

Initial Test Case:

If input x = {n chars followed by m chars} such that n + m >=N-100

Then f(x) = decision = 1 (Since n+m>=N-100)

Metamorphic Test Case:

Input Deriving function = h(x) = permute the order of characters input = x' = {m chars followed by n chars}

Output Deriving function = g(decision) = decision(should remain same as old decision)

In the initial test case, decision = 1, thus g(1) = 1

Resize function with derived input = f(x') = 1 (since m + n >=N-100)

Permuting the order of input should not affect the resizing decision.

2.b

Explanation

If uploading a picture first and then writing n comments on the page results in covering X sq unit of area on the page, then changing the order and writing comments first followed by uploading picture will not affect the total area covered on the page, i.e., the total area covered on the page will still be same.

For example

Assuming each comment as well as uploaded picture takes "x" sq units.

first the picture is uploaded => x square units covered.

After picture upload n comments are written => x+n*x = (n+1)*x area covered

Permuting the order:

n comments written before picture upload = >n*x area covered

picture is uploaded after the comments are written=> x + n*x = (n+1)*x square units covered.

Permuting the order of activities does not change the total area covered.

Transformation

Let f(x) be the function that calculates the total amount of area covered by picture upload and/or comments written on the page.

x is the input set such that it stores the activities chronologically, i.e., if

x = { picture upload, n comments written} then first a picture was uploaded and then n comments were written in that order.

Initial Test case:

Let x, the input be { picture upload, n comments written}. If both picture and the comment need x sq unit, then

f(x) = x + n*x = (n+1) * x sq unit. = S sq unit

Metamorphic Test Case:

Input Deriving function = h(x) = permute the order of activities on the page

=> x' = { n comments written, picture upload} i.e. n comments were written and then a picture was uploaded.

Output Deriving function = g(S) = S(same as the initial output)

In initial test case, S = (n+1) * x sq unit.

So g(S) = (n+1) * x sq unit.

Now calculating the total area covered for x'

f(x') = n*x + x sq unit = (n+1) * sq unit

Permuting the order of the activities should still end up resulting in coverage of the same sq unit of are on the page.

Extra Credit

1.a

Explanation

The number of search results found for a keyword should not change if the keyword is ORed with another keyword that is not present in the database. (union of the results generated by both the keywords)

Similarly there should be no search results generated for a keyword ANDed with another keyword where the second keyword is not present in the database as well.(intersection of the results generated by both the keywords)

Transformation

Let f(x) = Number of search results found for keyword "x".

Initial Test Case:

keyword "x" has n search results.

f(x) = n

Metamorphic Test Case:

Input Deriving Function for OR = h(x) = x' = x OR foo (where foo is not present in the system, i.e. f("foo") = 0 search results since it is not present in the system)

Output Deriving function = g(n) = n

Calculating f(x') = f(x OR foo) = f(x) Union f(foo) = n union 0 = n search results

Input Deriving Function for AND = h'(x) = x'' = x AND foo (where foo is not present in the system, i.e. f("foo") = 0 search results since it is not present in the system)

Output Deriving function = g'(n) = 0

Calculating f(x'') = f(x AND foo) = f(x) Intersection f(foo) = n intersection 0 = 0 search results(no common search results)

For the display function, the metamorphic property and description is similar to 1.a regular credit.

1.b

Explanation

The search result should not change for each group irrespective of the order in which search is done.

If you change the order, it should not affect the display of the total number of friends in each group. i.e., permuting the order in which search was done should still display the same number of friends for that group.

For example:

Assume group A and group B have x and y friends respectively.

First search for groupA=>x friends

Then search for groupB=>y friends

Permute the order:

First search for gorupB=>y friends

Then search for groupA=>x friends

Transformation

Let f(x) list the friends in each group where x is a set of input containing group names

x = {group1, group2} where group1 was queried first in search followed by group2.

f(x) = list the friends in each group in that order . If group1 has m and group2 has n friends,

then

f(x) = {m, n}

Initial Test case:

If x = {group1, group2}

f(x) = O = {m,n}

Metamorphic Test Case:

Input deriving Function h(x) = permute the order of group names for search input

=> x' = {group2, group1}

Output Deriving function = g(O) = permute the order of output set = {n, m}

Find f(x') = f({group2, group1}) = { n, m}

Permuting the order in which search was done should still display the same number of friends for the pertinent group.

1.c

Explanation

If the number of common friends is less than the threshold value then incrementing number of common friends by the threshold value should result in a friends' suggestion.

Alternatively, if the number of common friends is greater than the threshold value then incrementing number of common friends by the threshold value should still result in a friends' suggestion (Decrementing will not result in a friend's suggestion).

Transformation

Let f(x) be the function that decides if a person can be suggested as friend or not. Here x is the total number of common friends (recommendations).

Let t be the threshold such that if x>= t, the person is suggested as a friend.

Output of f(x) = decision = {1 for friend suggestion(>=t), 0 for no friend suggestion(<t)}

Initial Test Case:

Let initially x < t where x is a positive integer.

f(x) = decision = 0

Metamorphic Test Case:

Input Deriving Function = h(x) = x' = x + t(incrementing the number of common friends by threshold value t)

Output Deriving function g(decision) = 1(since now number of common friends > = t)

In initial output, f(x) = decision = 0

Thus, g(0) = 1

Finding f(x') = f(x + t) = 1(here x +t >= t)

If you increment the number of common friends by threshold value, then output should be a friend suggestion.

1.d

Explanation

The order in which characters are input on the wall should not change the state of the wall.

If you first input n chars followed by m chars and this results in a state "s" then permuting the order and inputting m chars followed by n chars should still result in a state "s".

Transformation

Let f(x) be the function that finds the state of the wall corresponding to an input set of characters.

Thus x = {m chars, n chars} , i.e., m chars are input followed by n chars.

f(x) calculates the state of the wall by first adding the total characters and then equating the state of the wall based on the total number of characters on it.

Initial Test Case:

Let x = {m chars, n chars} , i.e., m chars are input followed by n chars.

f(x) = f(m+n) = f(total) = s (Assuming m+n = total which corresponds to a state "s")

Metamorphic Test Case:

Input Deriving Function= h(x) = permute the order of character input

=>x’ = permuted order of character input = {n chars, m chars} , i.e., n chars are input followed by m chars.

Output Deriving Function = g(s) = s (should be same as initial output or state in this case)

Finding state for x’

f(x’) = f({n chars, m chars}) = f(n + m) = f(total) = s

Permuting the number of characters input would still end up in same state of the wall in this case.

Appendix IV (Feedback Survey Results)

Number of records in this query:		51
Total records in survey:		51
Percentage of total:	100.00%	

Field summary for 1

What do you think about your understanding of Metamorphic Testing now?

Answer	Count	Percentage
I understand Metamorphic Testing Completely. (A1)	5	9.80%
I still don't understand Metamorphic Testing at all. (A2)	4	7.84%
I have some (not complete) understanding of Metamorphic Testing. (A3)	35	68.63%
I am not sure if I understand it or I don't. (A4)	7	13.73%
No answer	0	0.00%
Not displayed	0	0.00%

Field summary for 2:

What do you think helped you the most in understanding Metamorphic Testing?

Answer	Count	Percentage
Swapneel's Lecture on Metamorphic Testing (A1)	6	11.76%
Recommended Paper Reading (A2)	5	9.80%
Metamorphic Testing (Individual) Assignment (A3)	2	3.92%
Metamorphic Testing (Individual) Assignment's Solutions (A4)	21	41.18%
Other (Internet, Google search, reading other papers about Metamorphic Testing. etc.) (A5)	13	25.49%
None (A6)	4	7.84%
No answer	0	0.00%
Not displayed	0	0.00%

Field summary for 3

What change(s) do you think could improve student experience as far as teaching Metamorphic Testing is concerned?

Answer	50	98.04%
No answer	1	1.96%
Not displayed	0	0.00%

Answers for question 3:

1. NA
2. NA
3. give more examples
4. More examples
5. Give the defintiion, but also go through more examples of metamorphic testing being applied to various functions across many domains. The essence of metamorphic testing has to be communicated better and in a more organized fashion.

6. na
7. May be more practice. Jon did help us with MT before submission. Another go on MT would help us better understand it.
8. there was a lack of concrete examples
9. i believe the lecture on metamorphic testing falls more under the umbrella of theory of computation. It is difficult to integrate one's mindset to looking at software as a mathematical function later towards end of one's degree in CS. i think it would be very beneficial to introduce the concept at an earlier stage than a later stage -previous question should be "chose all applicable" not just one of the options
10. Devoting more time in the class to a concept which is not there in the textbook and more examples should be given in the class.
11. Teach well in class
12. Testing assignment with more detailed function description.
13. The homework solutions actually confused me more. When I looked at my own homework, I felt like I had done pretty much the same exact thing as the solutions -- identified metamorphic relations and proposed tests based on that etc. -- but I got zero points on each metamorphic question. So it seems like I don't understand this concept at all. Maybe more examples in class, with detailed descriptions and explanations on how to derive the equations (e.g., rather than just homework solutions for us to read) would have helped?
14. introducing more concrete examples
15. Make good examples to lecture
16. Giving more examples for different cases. The solution is great. I kind of know what the test is. Thanks
17. Add real-world case study so that students can have a good understanding as to how metamorphic testing is used in real-world projects.
18. I think metamorphic testing was not given enough time during the lecture
19. Please show more examples in lecture.
20. still unclear about multiple inputs/multiple output functions. Should include one such example in teaching
21. I was not able to attend the lecture on Metamorphic testing, so I am not sure about this, but more examples in the class would definitely help.
22. Maybe providing a lot more examples of it, where all of them are of different scenarios.
23. Better examples, model solutions to problems would be ideal.
24. Give more examples
25. Talk about it in more detail and give some concrete examples.
26. Do more examples
27. I don't know I was not in class during that explanation, because the day after we had to do a demo of our application (for the same class). I think it s a really bad timing
28. More examples could be helpful
29. I think providing more examples and post it, gives us time to read and think, would make things better. I mean before homework of course.
30. Give more examples about it, that will be great helpful

31. Give some detailed examples of some metamophic testing, which are often used.
32. give examples in class.
33. Provide a different set of examples maybe
34. Use concrete examples when explaining it in class
35. Some real world case studies as I believe in learning by examples.
36. More examples and walking us through scenarios
37. A hands-on example in class about how the input is, how it is transformed etc, so that students are able to observe a real world application, instead of the concept only . Since the professor will be there in class, their doubts about the application of metamorphic testing(I'm sure there will be quite a few) can be resolved quickly and clearly by a concrete source.
38. it is good.48show some examples
39. It would have really helped to show more examples and have them posted on the class website
40. more substantiate examples perhaps, y=sqrt(x) wont help cracking the assignment.
41. None
42. I was unable to attend the metamorphic testing lecture, so I can't really comment on what should be changed.
43. More examples of metamorphic testing and how it applies to our group project.
44. IDK
45. Assignment was good enough.
46. I missed the lecture where metamorphic testing was covered. But, as I understand most students had difficulty with it. Maybe more examples would help.
47. The idea is really straight which can be learned from Wikipedia. But for people who encounters it first time, we need some detailed example how it can be applied in testing.
48. Less handwaving
49. A series of slides on metamorphic testing, with more examples and definitions.
50. More practical examples