Examining the Co-Evolution Relationship Between Simulink Models and their Test Cases

Eric J. Rapos and James R. Cordy
School of Computing
Queen's University
Kingston, Ontario, Canada
{eric,cordy}@cs.queensu.ca

ABSTRACT

This paper presents an industrial case study that explores the co-evolution relationship between Matlab Simulink Models and their associated test suites. Through an analysis of differences between releases of both the models and their tests, we are able to determine what the relation between the model evolution and test evolution is, or if one exists at all. Using this comparison methodology, we present empirical results from a production system of 64 Matlab Simulink Models evolving over 9 releases. In our work we show that in this system there is a strong co-evolution relationship (a correlation value of $r = 0.9, p < 0.01$) between the models and tests, and we examine the cases where the relationship does not exist. We also pose, and answer, three specific research questions about the practices of development and testing over time for the system under study.

CCS Concepts

•Software and its engineering → Software testing and debugging; Software evolution; Maintaining software;

Keywords

model-based testing; Simulink; co-evolution; automotive; case-study

1. INTRODUCTION

The link between development and testing is a strong one, and one that strengthens even more over the life-cycle of a software system. When this relationship is coupled with the fact that software (be it source code, models, etc.) is not a static object, and will change over time [5], it becomes increasingly important to ensure the corresponding tests remain up to date.

A change in a software system is very likely to require a corresponding change in its tests. This overall relationship is known as test co-evolution, and when applied to model-based tests is it is known as model-based test co-evolution. Our work aims to improve the ability of our industrial partner to easily and effectively maintain the model-based tests that correspond to source models as they evolve over time, and to infer associated practices to aid in this process. Our industrial partners currently maintain this relationship manually, which can be unnecessarily time consuming. Our long-term goal is to provide semi-automated guidance on the required updates to test cases, and ultimately to automate much of the test co-evolution process, in order to ensure the continued production of high quality software with reduced effort.

In order to properly understand the direct effects of model changes on the tests, we must first understand how the source models evolve over time, along with the same understanding of the associated tests, when manually maintained. The best way to obtain this information is through examining an existing system of models and their tests to determine the relationship between modeling and testing. By first completing the case study presented in this paper, we are able to identify the frequency of changes in models that require updates in tests, and other interesting basic relationships. Based on this, we have performed an analysis of a system of 64 production Matlab Simulink Models and their corresponding test cases, provided by an industrial partner, to answer the following research questions:

RQ1: Does test co-evolution happen synchronously or is there a delay?

RQ2: Is there a noticeable increase in development and test activity surrounding major releases or significant events? Is there a noticeable stabilization nearing the final release?

RQ3: Are there instances of changes to models that do not necessitate changes in the tests? If so, how common are they?

RQ1 and **RQ2** are very similar to the first two research questions investigated in the work of Zaidman et. al. [15] on the co-evolution of production and test code. Our study aims at answering these questions in relation to models, rather than source code.

These questions are being asked in order to better understand the relationship between Matlab Simulink Models and the tests for these models. This study is part of an overall project related to the automation of co-evolution of these tests based on the updates. Obtaining an understanding of the relationship between them will help us to determine how a change at the model level can propagate through to the tests. In addition, an understanding of the relationship between modeling and testing is of interest to our industrial partner; being able to determine where greater testing effort is required can help to increase software quality.

MiSE'16, May 16-17, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4164-6/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2896982.2896983

2. BACKGROUND AND RELATED WORK

In this section we begin by briefly presenting some background to introduce key topics. We present a number of related works related to the study of co-evolution of models and tests, as well as some applications to assist in the automation.

The work of Pinto et. al. is aimed at understanding the myths and realities of test-suite evolution [12]. In particular, they investigate why tests change over time. The authors state that test repair (fixing tests that no longer work after a change) is only one of many ways tests can evolve; in fact most changes occur as refactorings or additions and deletions of test cases. These types of changes are likely to appear in our study. One example of the automation of test evolution is the work of Mirzaaghaei et. al., in which they are able to automatically repair test cases for evolving method declarations [10].

With the expansion of the software field to include modeling, the field of software evolution has expanded to include the concept of model evolution as well. Paige et. al. present a comprehensive survey of model evolution work over the years in their recent paper [11]. Particularly, the authors discuss relevant work in the topics of metamodel and model co-evolution, and model versioning - two concepts that closely relate to our work.

While examining the evolution of software and models is important to our work, our focus lies more specifically in the application of maintenance of tests for systems developed using model-driven engineering (MDE). As such, it is important to fully understand testing in the model-based context. Dias Neto et. al. present survey on model-based testing (MBT) [3] in which they analyze 78 papers on MBT and come up with four succinct conclusions about the issues facing the field, the most prominent of which is that the testing and development processes in MBT are rarely well integrated.

The combination of model-based testing and the co-evolution of tests alongside their sources forms the area of model-based test co-evolution. Early work on this topic is presented by Zech et. al., in which they present a generic platform for model-based regression testing, dealing specifically with model versioning and evolution, known as MoVe [17].

Co-evolution of model-based tests is also the main focus of our current and future research. Details of our research plans have been presented in two doctoral symposium papers [13][14].

Another piece of relevant background is the specific application of our work, testing of Simulink Models. The recent work of Matinnejad et. al. is aimed at the automated testing of Simulink models [9], focusing on the creation of test suites for existing models. While this work deals with the creation of tests, it seemingly would require regeneration as models evolve, and doesn't explicitly provide a framework for updates to the models. Regeneration of tests can become expensive with many changes. Our future work aims to reduce or remove the need for the regeneration of test suites, aiming instead at automated adaptation of existing tests based on model changes. This technique could be combined with the work on automatic generation, or used to assist with manual test generation techniques.

A study similar to our work was conducted by Zaidman et. al. [15] in which they examine the relationship between source code and tests, as opposed to models and tests. The authors explore research questions regarding the synchronicity of co-evolution, testing efforts surrounding major events, detection of test strategies from repository data, and the relationship between test writing and test coverage. Following this initial work, a follow-up study was conducted on the use of association rules to study the co-evolution relation between tests and source code [6]. Their work was then expanded to include, in addition to open source, industrial software, again using repository mining [16]. In that work they show that the observations of their earlier work on open source systems also hold in the industrial case study. Most recently they have applied their technique at a more fine-grained level [8].

Another similar study, conducted on 6 open source software systems, was conducted by Marinescu et. al. [7] in which they present Covrig, their framework for the analysis of code, test, and coverage evolution. In their framework they pose 10 research questions, several of which relate to the co-evolution relationship between source and tests. This study however finds that testing does not increase as often as source changes, and they conclude that testing is a phased activity (with the exception of one of the 6 systems, Git).

Ens et. al. present a different approach to the analysis of test and source co-evolution: they propose the use of visualization to understand the relationship, using their tool ChronoTwigger [4]. ChronoTwigger uses co-change as it's basis, examining the correlations between source and tests, to visualize their relationship, based on the premise that visualization helps with comprehension. From a small user study (3 users) the authors conclude that ChronoTwigger can lead to inferences about the relation between source and testing that are not easily made using existing techniques.

While these co-evolution studies are focused on source code and associated tests, the motivation is similar to ours, and the insights from their findings led us to want to attempt a similar study on models and tests.

In addition to looking at other studies regarding the co-evolution relationship, it is also important to look at work on the applications of co-evolution. One of the first applications of co-evolution as a tool for test updating was presented by Arcuri and Yao [2] in which they present their methodology for using co-evolution to automate bug fixing. The authors demonstrate their application by automatically fixing bugs found in an implementation of a sorting algorithm.

3. DATA SET: MODELS AND TESTS

The set of models used for this experiment were provided to us by our industrial partner for analysis and experimentation. They are Matlab Simulink Models used in the automotive domain. Simulink [1] is a graphical modeling technology used to create components, and the relations between them; code is then generated from the models and used in electronic control units (ECUs) in a vehicle.

The models are structured into nine main components (referred to as *rings*), each of which is made up of a number of sub-components. For example, ring 1 is made up of 4 sub-components, so there is a model for each of the 4 sub-components, and one integration model for the ring, which links the sub-components together. In total there are 55 sub-component models, and 9 ring integration models, for a total of 64 different models in the system (not all of them exist in all versions).

Table 1: Number of models provided for each ring at each release

	Rings								
Rel.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	4	4	8	11	4	4	10	7	8
2				11			10		
3	4	4	8	10	4	4	10	7	8
4									
5									
6	5	5	8	10			10	7	8
7	5	4		10				7	
8	6	4	8	10	3	3	10	7	8
9									
10		4	4				10	7	8
11	6	4	8	10	3	3	10	7	8
12									
13	6				3	3	10		
14									
15	6	4	8	10	3	3	10	7	8

We were provided with 15 releases of the entire system from their version control system (VCS). Table 1 shows the breakdown of models per release per ring; each count includes the sub-component models and the ring integration model. There are actually several releases of the system during which there were no changes and no versions added to the VCS (releases 4,5,9,12,14). Since those involved no evolution, for this study they were removed entirely, and we were left with 10 releases to examine. In some of these remaining releases, there were no updates to some of the models, meaning there are not 10 versions of each model. For the purpose of this study, we created duplicates of the previous versions where a version does not exist for a release, for the purpose of comparing versions.

The term "release" in the context of this work does not actually refer to a release version of the system, as all of the versions examined are pre-release. A release in this context refers to a milestone in development; every incremental time block, the release number is increased. The releases where all models were originally provided (as seen in Table 1) can be considered major releases in the life-cycle of the system; thus releases 1, 3, 8, 11, and 15 are the major releases for this software.

For each of the models, there is an associated test suite which was provided along with the models. For the purposes of this application, a test suite is contained in an Excel workbook comprised of a number of Excel spreadsheets (using the tabs in Excel), each containing test inputs and expected outputs for the given model, over a number of time steps. Each row represents a time step, and each column is an input value or an expected output value. The tests are run by simulation within Simulink, and the results compared with the expected outputs.

4. EXPERIMENT

4.1 Preprocessing

We began by removing the ring integration models from the analysis, due to the fact that their contents is simply the merge of the sub-components along with connecting lines between them. Additionally, the test cases for the ring integration models were not created until the final release, thus there is no valuable information provided by including them in the study. Furthermore, there are two models which simply produce output, and do not reply on input. These were excluded from the analysis since there are no associated tests for these models, and thus no comparison results to be obtained. We also removed release 2 from the analysis, because there were some issues with one of the model files in that version. We dealt with this problem by performing the comparison between release 1 and release 3. We have no concern about the validity of removing this release since there were only two systems with changes, and only one model in each was changed. These changes are adequately captured in the comparison between releases 1 and 3.

After all of the pruning of extraneous models, we were left with the 53 component models over 9 releases, for a total of 477 models and their associated tests, to perform our experiment on.

4.2 Process

The experiment was conducted by performing a pairwise comparisons of two consecutive versions of the same model, and recording the results. This was repeated for each model, and then for every pair of releases, resulting in 8 release comparisons for the 53 models.

The result of a model comparison is one of the following: No Change, Model Does Not Exist Anymore (but existed previously), Model Does Not Exist Yet (but will exist in a later release), Model Newly Created This Release, Model Deleted This Release, or Model Was Modified (listed as # of additions, modifications, and deletions).

The same process was repeated for the test cases, again storing the results for analysis. The result of a test case comparison is one of the following: No Change, Test Does Not Exist Anymore (but existed previously), Test Does Not Exist Yet (but will exist in a later release), Test Newly Created This Release, Test Deleted This Release, or Test Was Modified (listed as # of test cases added or removed OR # of test which are the same and # of tests that have changed)

With results for both models and tests, matching types of results, we performed a comparison between each result for a given model and test evolution. This is to say, for subcomponent 1, between release 1 and release 3, we obtain a result of how the model changed (if at all) and how the test changed (if at all) and compare the two looking for a relationship. The result of this comparison is one of the following: (1) No Change In Both, (2) Model and Test Do Not Exist in Both Releases, (3) Model and Test Newly Added This Release, (4) Model and Test Deleted This Release, (5) Change in Model But Not In Test, (6) No Change in Model But Change In Test, or (7) Change in Model and Test.

The above outcomes can be summarized as matching (1, 2, 3, 4, 7) or not matching (5, 6). These results form the basis of our analysis.

4.3 Implementation

In this section we discuss the implementation of our experiment. As we are working with Matlab Simulink Models, we chose to perform the comparisons in the Matlab environment. Matlab scripts were written to collect all of the appropriate model and test files, and to perform the pairwise comparisons.

4.3.1 Model Comparison

Our Matlab script took as input the paths to two versions of the same Matlab Simulink model, and performed a comparison between the two versions. To do this, we made use of the built-in Simulink differencing tool, which is an XML comparison method that returns a single object, containing all of the differences between any two Simulink models (or two versions in this case).

Given this result, our script then removes any non-semantic differences such as placement, colours, version number labels, etc. and then iterates over each difference one at a time. The script then determines whether the difference is the addition of a new element to the model, a modification to an existing element, or the deletion of an element from the previous version. The result is reported as a string of the form:

A: # M: # D: #

where the numbers of additions, modifications, and deletions are reported respectively.

This process is repeated for all of the models, performing a total of 424 pairwise comparisons of the models, and storing the results in a Matlab cell-array. Upon completion the results are written to an Excel spreadsheet for further investigation.

4.3.2 Test Suite Comparison

Similarly to the model comparison, we created a script which took as parameters the paths to the two Excel workbooks containing the test suites for two versions of the same test suite.

The first step in comparison was to examine the number of test cases in the test suite, which is done by counting the spreadsheets in the workbook, and determining if there are the same number of test cases. If there are an equal number of test cases investigation continues, otherwise the result is reported as the addition or removal of some number of test cases.

If there are the same number of test cases, we then begin a comparison of each test case from each test suite to the corresponding test case in the other test suite. This is done by performing an element by element comparison of the spreadsheet; if a single cell is found to be different, that individual test case is marked as changed, and exploration continues with the remaining test cases. Upon completion of the comparison, the result is reported as the number of test cases that have changed and the number that remained the same. If all of the tests change, or none of them change, this is the result reported, otherwise, the resulted is reported as a string of the form:

Same: # Diff: #

where the number of tests that are the same and the number that have changed are reported respectively.

This process is repeated for all of the test suites, performing a total of 424 pairwise comparisons of the test suites, and storing the results in a Matlab cell-array. Upon completion the results are written to an Excel spreadsheet for further investigation.

4.3.3 Co-Evolution Relationship

With the results of both comparisons available, a final Matlab script was created to perform a comparison between the model results and the test results, in order to come up with the relationship between the two comparisons.

This was done by simply iterating over the 424 cells in each table and comparing them to come up with a result; the result being a number from 1 to 7, corresponding to the seven outcomes presented in Section 4.2. The results were written to an Excel spreadsheet for further investigation and final analysis.

5. ANALYSIS

5.1 Results

The results of the co-evolution relationship analysis between the models and tests can be found in Table 2. Each column represents the evolution from one release to the next, while each row is a specific model. The values in the cells correspond to the comparison results described in Section 4.2. However, since several of the outcomes can be grouped together, the cells have been shaded to show the groupings of similar outcomes. The most common outcome (64.6%), was when there was no change in the model and no change in the test (result 1), which is shown by all of the white cells. The next most common outcome (21.7%) was the fact that a co-evolution relationship existed between the model and the test suite; this can mean that either both did not exist at that time (result 2), both were added (result 3), both were deleted (result 4), or both were changed (result 7); all of these are shown by the cells shaded light gray. The final two results, which occurred the least frequently (13.7%) were the results where there was a change in one artifact but not the other: a change in the test but not in the model (result 6) is shown in medium gray, and a change in the model but not the test (result 5) is shown in dark gray. It is these two categories that provide relationships outside of the expected, and thus the ones that require further investigation; this analysis can be found in the answers to **RQ1** and **RQ2**.

From this data, we wanted to determine how often throughout a particular model's life-cycle did it occur that there was a change in one artifact but not the other. This data is shown in Figure 1. What can be seen here is that 38 of the 53 models (71.1%) only have this occur 0 or 1 times throughout their life cycle, with a small portion of the models displaying this property more frequently. Of note, model 42 demonstrated the property of having a change in one artifact and not the other in 5 of the 8 evolutions. Upon further investigation, it was found that model 42 was a model that changed frequently, but the changes did not necessitate changes to the test cases as they were internal changes (more on this type of change is discussed in **RQ2**).

While we had an initial belief there should be a relationship between changes in the models and changes in the tests, we wanted to examine this relationship in detail. Figure 2 shows us the relationship between the percentage of models that change between each release and the percentage of tests that change in each release. From the data we are able to conclude there is a strong positive correlation between models changed and tests changed, $r(6) = 0.9, p < 0.01$.

Our next area of analysis was an examination of the type of change (or lack thereof) over time. For this we returned to our groupings from Table 2, but group the two types where changes occur in only one artifact into one group. This left us with the fact that in a given evolution, models and tests could either both not change, both change, or only one change. The results of this can be found in Figure 3.

Table 2: Co-Evolution Relationships Between Models and Tests

	Model Release Pairs							
mdl	1|3	3|6	6|7	7|8	8|10	10|11	11|13	13|15
1	1	7	5	5	1	5	1	1
2	7	7	1	5	1	6	1	1
3	2	2	2	3	1	7	1	1
4	2	3	7	5	1	7	7	1
5	7	1	4	2	2	2	2	2
6	2	2	3	7	1	5	1	1
7	1	7	1	5	1	5	1	5
8	7	7	1	1	1	1	1	1
9	7	1	4	2	2	2	2	2
10	2	3	1	7	5	7	1	1
11	1	7	1	1	1	1	1	1
12	1	5	1	1	6	5	1	1
13	1	1	1	1	1	1	1	1
14	1	1	1	5	1	1	1	1
15	5	1	1	1	5	1	1	1
16	1	7	1	1	1	5	1	1
17	7	1	1	5	1	1	1	1
18	5	1	1	1	1	1	1	7
19	7	7	1	1	1	5	1	1
20	7	1	1	5	1	1	1	1
21	7	1	1	1	1	1	1	1
22	7	1	1	1	1	1	1	1
23	7	5	7	7	1	1	1	7
24	4	2	2	2	2	2	2	2
25	7	7	1	1	1	5	1	1
26	1	1	1	7	1	1	1	1
27	1	1	1	7	1	1	7	1
28	7	1	1	4	2	2	2	2
29	1	1	1	7	1	1	6	1
30	1	1	1	7	1	1	5	1
31	7	1	1	4	2	2	2	2
32	5	1	1	1	1	1	1	1
33	7	1	1	7	1	1	1	1
34	5	5	1	5	1	1	1	1
35	5	7	1	1	1	1	1	1
36	7	1	1	7	1	1	7	1
37	5	1	1	1	1	1	1	1
38	5	1	1	1	1	1	1	1
39	5	1	1	5	1	1	1	1
40	5	1	1	5	7	1	1	1
41	1	7	1	1	1	1	1	1
42	5	5	1	5	6	5	1	1
43	1	1	1	1	1	1	1	1
44	1	1	1	5	1	1	1	1
45	1	1	7	1	1	1	1	1
46	5	1	1	1	5	1	1	1
47	7	1	1	5	1	1	1	1
48	7	5	1	5	1	1	1	1
49	6	1	1	5	1	1	1	1
50	6	1	1	5	1	1	1	1
51	7	1	1	5	5	1	1	1
52	6	7	1	5	5	1	1	1
53	6	1	1	7	1	1	1	1

Numbers refer to definitions in Section 4.2

White – No Change in Both (1,2)
Light Gray – Co-Evolution Occurrence (3,4,7)
Medium Gray – Change in Test, None in Model (6)
Dark Gray – Change in Model, None in Test (5)

5.2 Discussion

RQ1: *Does co-evolution happen synchronously or is there a delay?*

Since it is the case that at each release the tests are run on the models prior to checking into the VCS, and we only have the information of the version check-ins for the releases, we cannot provide any further granularity beyond releases to know exactly when changes in tests occur in relation to changes in the models. Given the information available, we

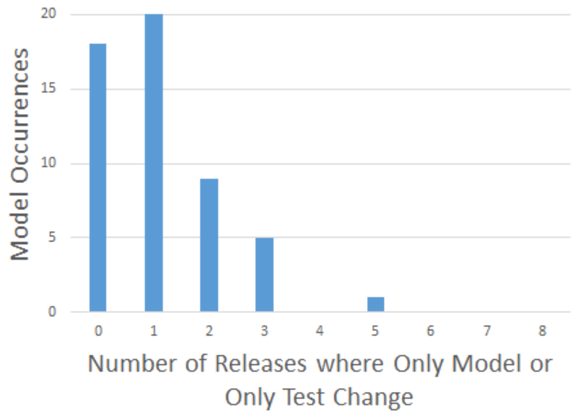

Figure 1: Frequency Graph of Evolution Step Disparity

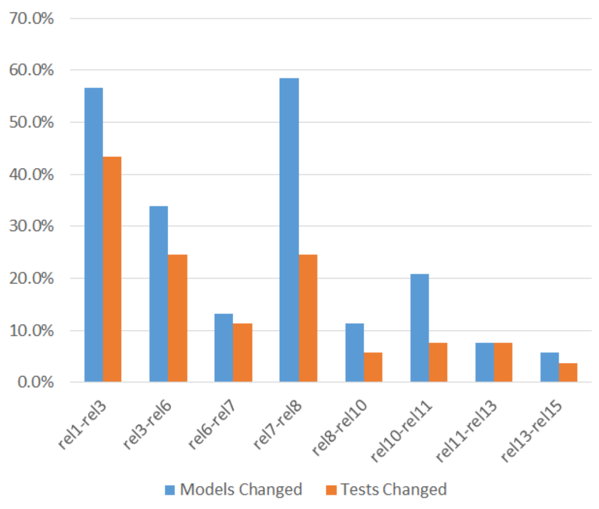

Figure 2: Percentage of Models & Tests Changed at Each Release

can conclude that it does happen consistently when required, with only a few minor exceptions.

The exceptions are those instances where there is a change in one artifact and not the other: a change in model and not test (result 5), and a change in test but not model (result 6). **RQ2** will address those instances of result 5, however when a change is made to a test but not the model, this can be seen as an instance of a delayed update to the test, and warrants further investigation.

There are 8 instances where the tests changed and not the model (result 6), and these occurred only once in the life time of models 2, 12, 29, 42, 49, 50, 52, and 53. Each of these will be discussed here in order to understand what the lone occurrence of test changes means for the system.

To begin, we can address Models 49-53; the changes made to these tests was that they were newly added tests. These 5 models are all sub-components of the same ring, and the absence entirely of tests in the first release of the system means that there was in fact, not a delay of changes to the models, they were just not included with the check-in of this

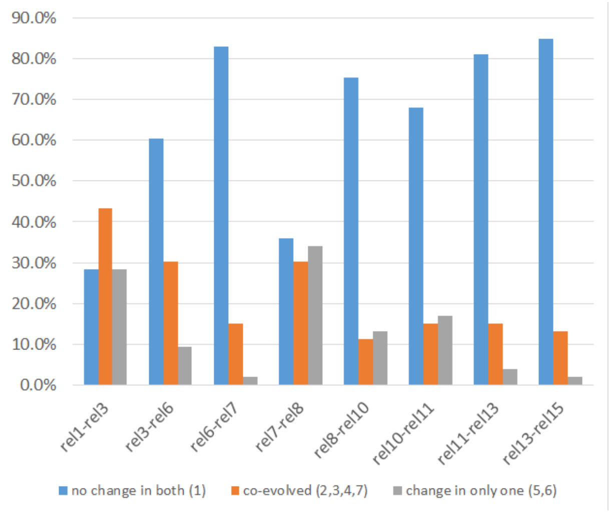

Figure 3: Types of Change over Time

version for some reason. Thus, this negates the concept of a delay in test updates to these tests.

The occurrences of changes in tests but not models to models 2, 29, and 42 all share similar circumstances; there were no changes made to the existing tests, and additional tests were created. This again negates the concept of a delay in testing, and simply represents an increased effort in testing when changes were not required to the models during this release period.

The final occurrence of changes to a test suite but not the source model is within model 12, where all of the test cases were modified. This marks the first result requiring investigation into the actual test cases to determine the changes; however the findings were that the changes to each test case were changes to the frequency of changes in values (the time step of the values), and not the inputs or expected outputs themselves.

From this analysis, we can conclude that in this model set, there is no concept of delayed updates to test suites after model updates, and that when a change to test suites is required, it is completed prior to the end of the current release cycle.

RQ2: *Is there a noticeable increase in development and test activity surrounding major releases or significant events? Is there a noticeable stabilization nearing the final release?*

The answer to this question can be found by examining the percentage of models and/or test that change surrounding these events, as compared to the releases immediately prior. A summary of this information can be seen in Figure 2.

Recall that releases 1, 3, 8, 11, and 15 are the major releases for this software. By eliminating the first release, since there is no prior information, which can also be said for the change from release 1 to release 3, we can highlight the changes made between releases 7 and 8, 10 and 11, and 13 and 15 as the points of interest for this research question, as they are the updates made prior to the major events in the product life cycle.

Upon a visual inspection of Figure 2, it becomes clear that there is indeed an increase in development and testing prior to release 8 and 11, but not release 15.

Between releases 6 and 7 13.2% of the models changed, and between releases 7 and 8 58.5% of the models changed, an increase of 343.2%; similarly changes in tests at these two evolution steps increased from 11.3% to 24.5%, an increase of 116.8%. Examining these same differences for the major event surrounding release 11, we note that changes in models increased from 11.3% to 20.8%, an increase of 84.1%, and changes in tests increased, only slightly, from 5.7% to 7.5%, an increase of 31.6%.

While it is shown that changes in models and changes in tests actually decrease prior to the final release (decreasing 24% and 49.3% respectively), we still feel confident in concluding that there are substantial increases in development and testing efforts surrounding major events of a software system prior to release, with the exclusion of the final release. Our understanding of this exception is that the final release changes would only be small minor 'polishing' changes.

Regarding the question of stabilization, referring to the reduction of changes made at each evolution step, as the system approaches the final release, examination of Figure 3, showing the amount of change over time, does indeed indicate a stabilization. The obvious conclusion here is that we can indeed notice a point of stability as the system approaches the final release, with the peak of no change being the final evolution step, at 94.3% of the models and tests not changing at all. There are however 3.8% of the models and tests that have a consistent co-evolution at this co-evolution step; we can again attribute this the same types of 'polishing' changes discussed above.

RQ3: *Are there instances of changes to models that do not necessitate changes in the tests? If so, how common are they?*

The answer to this question is of importance for our continued work, in that identifying the types and frequency of changes in models that do not have a direct impact on the test suites can help reduce the testing efforts by identifying when a change needs (or need not) to be made to the test suite.

Since the test suite for each model is run and must pass at every release prior to being checked into the VCS, any time where there is a change in the model but not the test (a result of 5 in Table 2) provides us with an instance of changes that do not necessitate changes in the test suites. Based on this, we can claim that there are indeed changes that do not require direct updates to the test suite.

To the question of frequency, there are 50 occurrences of this over the life of the system, whereas there are 110 instances of model changes occurring over the life of the system; this equates to 45.5% of the model changes not requiring a change in the associated test suite.

From this we can conclude that there are a substantial number of changes that occur to source models that do not impact the overall behaviour of the models, nor the results of running the existing functional tests.

Identifying the exact types of these changes will help us better understand the impact of Simulink model changes on their test suites, which will be explored further in our future work.

6. CONCLUSIONS & FUTURE WORK

In this study, we were able to examine in detail the relationship between evolution of models and the evolution of their associated tests in a production model-driven automotive industrial system, with the goal of determining whether or not a co-evolution relationship exists. This was done by comparing the differences between versions of both the models and the tests, at every version, then creating a link between these comparisons. The results of this can be found in Table 2. Of note, we were able to conclude that there is a strong positive correlation between models changed and tests changed, $r(6) = 0.9, p < 0.01$, thus confirming that there is indeed a co-evolution relationship between the source models and their tests.

In addition to simply examining the existence of the relationship, we also looked into 3 specific research questions regarding the existence of a delay in updates, increases in efforts surrounding major events in a life-cycle and approaching final release, and the existence of changes to models that do not require changes to tests. Each of these questions was discussed in detail, with a definitive answer provided.

With an understanding of the relationship between the models and tests, and how they evolve alongside one another, the next step in our work is to investigate at a lower level and determine the relationship between individual changes and the required test changes. This study focused on whether or not models changed between two versions, and the same for the corresponding tests for those versions, and an analysis of these relationships was conducted; however we now would like to expand beyond the fact that a model has X additions, Y modifications, and Z deletions into an analysis of what each of these model changes are and which exact changes occurred in the test as a result of each of these changes. These results will be useful in determining what updates are required to a previous test version, given the differences between a current model and its previous version. Eventually, these updates will be automated, removing the need for manual updates.

The important results from this study that will carry forward into this future work stem from the any of the occurrences of a result of 5 or 7 in Table 2. The occurrences of 7 mark a successful co-evolution, and provide us with a direct link between changes in the models to changes in the tests. Additionally, results of 5 provide us with clear examples of changes in the models that do not require changes in the tests. These two types of relations provide us with an excellent starting point for continued analysis of co-evolution of Simulink model-based tests.

Acknowledgments

This work is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), as part of the NECSIS Automotive Research Partnership with General Motors, IBM Canada and Malina Software Corp. We particularly thank our partners at General Motors Canada for their continuing collaboration and support.

7. REFERENCES

[1] MathWorks Simulink product page. http://www.mathworks.com/products/simulink/. Accessed: 2015-10-27.

[2] Arcuri, A., and Yao, X. A novel co-evolutionary approach to automatic software bug fixing. In *Congress on Evolutionary Computation, 2008* (2008), IEEE, pp. 162–168.

[3] Dias Neto, A., Subramanyan, R., Vieira, M., and Travassos, G. A survey on model-based testing approaches: a systematic review. In *ASE '07* (2007), ACM, pp. 31–36.

[4] Ens, B., Rea, D., Shpaner, R., Hemmati, H., Young, J., and Irani, P. Chronotwigger: A visual analytics tool for understanding source and test co-evolution. In *VISSOFT '14* (2014), IEEE, pp. 117–126.

[5] Lehman, M. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE 68*, 9 (1980), 1060–1076.

[6] Lubsen, Z., Zaidman, A., and Pinzger, M. Using association rules to study the co-evolution of production & test code. In *MSR '09* (2009), IEEE, pp. 151–154.

[7] Marinescu, P., Hosek, P., and Cadar, C. Covrig: A framework for the analysis of code, test, and coverage evolution in real software. In *International Symposium on Software Testing and Analysis, 2014* (2014), ACM, pp. 93–104.

[8] Marsavina, C., Romano, D., and Zaidman, A. Studying fine-grained co-evolution patterns of production and test code. In *SCAM '14* (2014), IEEE, pp. 195–204.

[9] Matinnejad, R., Nejati, S., and Briand, L. Automated test suite generation for time-continuous simulink models. Tech. Rep. TR-SnT-2015-7, 2015.

[10] Mirzaaghaei, M., Pastore, F., and Pezze, M. Automatically repairing test cases for evolving method declarations. In *ICSM '10* (2010), IEEE, pp. 1–5.

[11] Paige, R., Matragkas, N., and Rose, L. Evolving models in model-driven engineering: State-of-the-art and future challenges. *J. Syst. Software* (2015).

[12] Pinto, L., Sinha, S., and Orso, A. Understanding myths and realities of test-suite evolution. In *FSE '12* (2012), ACM, p. 33.

[13] Rapos, E. Co-evolution of model-based tests for industrial automotive software. In *ICSME '14* (September 2014), IEEE, pp. 663–663.

[14] Rapos, E. Co-evolution of model-based tests for industrial automotive software. In *ICST '15* (April 2015), pp. 1–2.

[15] Zaidman, A., Van Rompaey, B., Demeyer, S., and Van Deursen, A. Mining software repositories to study co-evolution of production & test code. In *ICST '08* (2008), IEEE, pp. 220–229.

[16] Zaidman, A., Van Rompaey, B., van Deursen, A., and Demeyer, S. Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining. *Empirical Software Engineering 16*, 3 (2011), 325–364.

[17] Zech, P., Felderer, M., Kalb, P., and Breu, R. A generic platform for model-based regression testing. In *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*. Springer, 2012, pp. 112–126.