Derivation of Impacted Areas and Generation of Test User Stories for Agile Context

S. Reine De Reanzi
Karunya Institute of Technology and Sciences, Coimbatore

P. Ranjith Jeba Thangiah
Department of Computer Applications, Coimbatore
Karunya Institute of Technology and Sciences

ABSTRACT
Real time systems are not stand alone. Some are under development, some are built from the scratch, some use and share third party APIs, sdks, services, signals, infrastructure, platform and resources. Hence until a software reaches its end of life, it consistently continues to evolve, adapting to the changes in terms of requirements, technologies, new concepts that are outcomes of various research. At times a part of the system is adapted or morphed or new one is built on the existing codebase.

The test methodologies are extended to support such systems. The study aims to bring out (i) impacted areas based on changes in the entity and (ii) feasibility of leveraging user stories to generate testcase stories that can be used for testing of systems that are constantly evolving.

Keywords
Modeling, User stories, Impact areas, Formal Methods, Test User Stories, Heuristics, SaaS, Test Modeling

1. INTRODUCTION
Internet on smart phones have enabled the user to use both business and personal activities by accessing the features, apps on the handheld devices. The applications are built using varied technologies, integration, communication and sharing of APIs, handlers, SDKs, plugins, webservices, signal processing services, OTA (Over the Air), protocols, infrastructure, resources etc. When solutions for this type of software is delivered on cloud, it is complex. For example, the production environment is used by number of users, on top of which the new code has to be elevated. This brings in a quite a lot of challenges for testing community.

i. The existing and new users should be able to seamlessly use the existing part of software
ii. The existing and new users should be able to use the new features.
iii. Data Correctness, transient data

Testing such systems are complex in terms of backward compatibility.

i. There are changes to the existing feature
ii. Existing requirements are merged with the new ones
iii. New features, adapting a new solution, new technology, integration – migration of users, services, time zones and timer events
iv. Revised vendor contracts
v. Change in business rules

The challenge here would be to ascertain the test adequacy conditions of the tests that are generated [12] and which implicitly takes care of the impacted workflows in terms of coverage. It is important to ensure that the test model applies the risk-based end to end scenario computation that results in a high yield with high coverage test design strategy [20].

The user stories change as the requirements evolve with every iteration. The codebase undergoes a change and the paper attempts to leverage the change in user stories to identify the impacted areas and derive test user stories based on the above to narrow down the tests.

The observations on release iteration for this study was carried out on MaaS360[1], hosted on private cloud(SaaS[2]). Its software delivery followed agile methodology. Agile Modeling is an extension to more code-centric agile methods, such as XP [16].

2. BACKGROUND
When a system under test is considered, there is enough research to identify the model and apply the appropriate relevant formal methods, to generate the test-suite. Model driven testing of such systems bring in a lot of systematic and organized way of testing [12].

Model based testing[MBT] uses models to derive concrete tests. The test adequacy criteria are described in relation to models. They are used to evaluate reliability of derived tests and as predictor for determining coverage. The challenge here is to create and maintain relationships between model and code elements [14], in a dynamically changing environment. The OMG proposed a specification (Query/View/Transformations –QVT) that standardized, the ways, source models are transformed into target models (model – to – model transformations) [19]. Model based testing requires the ability to map the abstract values of specification to the exact values of the implementation [2]. It has proven that it is cumbersome to re-use standalone model-based testing approaches and test models over different projects due to their domain specific characteristics [1]. It is a difficult task to derive one common model, which caters to all the systems.

The research on domain specific modelling helps to use model user stories to raise the level of abstraction beyond programming, by directly specifying the solution using domain concepts [7]. There are specific advantages of integrating domain specific modelling with MBT like, reusing the model, light weight changes to test generator and simultaneous use of various test generators [15].

Given a set of tools and methods, MBT has been shown to be useful and effective means of high-level testing in different domains [11]. Most of the Model based testing tools are general purpose tools, focusing on generic models of software behavior, such as finite state machines [18].

In XP(Extended Programming) for instance, simple descriptions of requirements are documented as user stories that are mainly used for release planning, which in turn can be used as source for deriving the test model [20]. This brings the domain context. The UML specifications for usecase diagrams and sequence diagrams elicit the user's usage and behaviour of the flow under test [5].

3. USER STORY MODEL BASED ON DOMAIN ATTRIBUTES
In plain model based formal specifications, the tests are covered for their basic flows and alternate flows. Here the most important factor of user context is missed out in this case.

User assigned access roles to workflows are important for SaaS based applications in the terms of security. Each user story has a user context comprising sets of access roles that determines the access and flow and behavior specific to those roles. For example, master admin, product user, end user with less privileges, guest, third party system etc. This domain context helps to bring in the relations and transformations to test-model at the user story level.

User assigned feature properties or product mix for customers based on their license tier is another dimension of user context. Different sets of users use different sets of services offered on the same platform as in figure 1

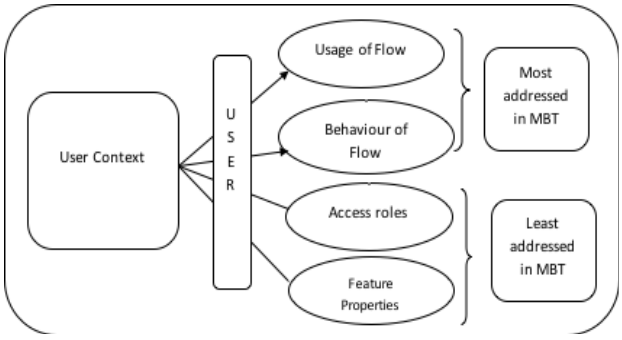


Fig: 1 User Context: User Assigned Roles and Properties

Each story is tagged with the user context involving all the above-mentioned criteria.

4. EXTENDING USER STORY FOR TEST-MODEL

Creation of meta models like data flow diagrams, UML diagrams, model based control graphs, hierarchy of test generation, traceability are the test related artifacts meta models [13]. Hence the attempt here was to try and dissect the user story as depicted in figure 1 to derive the following

- Entity meta model
- Impacted areas meta model, to identify the regression suite to include the least addressed.

4.1 Entity Meta Model

The user story cuts across the features to realize the end to end scenario. Here the control flow of the system follows multiple paths touching entities depending on the level of abstraction (how deep the user story has been layered). Entitiy can be a business object, user parameterized object, system object. Collection of such entities are connected by the edges of a flow form a user story. The edges are the action sequences that connect the entities. Creation of entities and identifying action sequences are the pre-requisite for story level abstraction. Hence when a user story is created it involves one or more entities. The entities are tagged to identify the impacted areas.

The sequence diagrams, control graph flows, finite state machines can be abstracted using the entities, to define a story. And entities can be fine grained by the use of the sequence diagrams, control graphs and finite state machines.

For example, '*As an IT admin, I want to locate the user device so that I can view the device's present location*'.

In the above user story, the entities are Device, Device Information, Locate User function, third party mediator, Global Positioning system, third party maps and GUI. The action sequences define the flow of the data from one entity to another.

The following figure 2 represents the entities involved in the action sequences

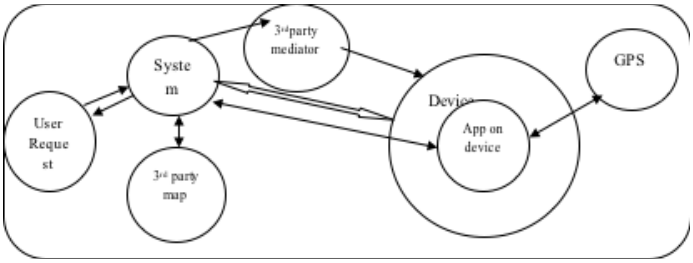


Fig: 2 Entities and the Action Sequences

The execution sequence for locate user is listed below.

1. Request.locateUser
    a. Trigger.push notification.device
    b. RouteRequest.device app
    c. DeviceApp.requestGPS
    d. GPSreturn.coordinates
2. systemResponse.coordinates
3. ViewInfo.Device
4. RequestAddress.latitudelongitude.maps
5. Display.address

In the manual testing context, the test executor is aware that the device has to be pre associated with the system and is accessible through a user context. This is the basis for tagging each user story to a pre-requisite. Now the high-level structure of each user story can be depicted as in figure 3.

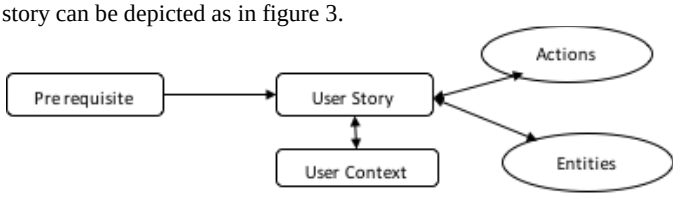


Fig: 3 High Level Structure of Each User Story

From the user story list all the entities and action sequences are collected to form the entity meta model. The formal modeling is discussed in section 6

4.2 Impacted Areas Meta Model

When the above user story is extended to provide new feature and is offered only as beta. Then it is controlled by user assigned property, where the user can access this feature only if the appropriate property is turned on.

For example,

Old feature: "User's location can be identified by performing 'Locate User'".

New feature: Device user can be dynamically assigned with policies based on location.

Modified user story '*As an IT admin, I want to set a rule to push a policy to a device based on location change, so that when the end user is outside of the corporate network range, the corporate resources are not accessible on his device*'.

The above mentioned are the explicit changes.

Let us add more one change to the implementation of the above requirement.

Changed Feature: The vendor for third party map is changed and the system queries the maps only on demand.

This means (i) the old method of querying every time a location is reported is removed from the system (ii)The query is fired to the maps, only when the location information is viewed by the user and stores the location coordinates.

4.3 Business Reason for Change

This saves cost, as firing a query to the maps is a paid service from a 3rd party vendor. The device is configured for locations, based on the location at which the device is positioned, the policy should be applied. Post the change the user experience needs to be seamless. But the user story should still take care of these

changes and address them. The execution sequence for locate user has changed as given below to accommodate the dynamic policy assignment

1. Request.locateUser
    a. Trigger.push notification.device
    b. RouteRequest.device app
    c. DeviceApp.requestGPS
    d. GPSreturn.coordinates
2. systemResponse.coordinates
3. ViewInfo.Device
4. RequestAddress.latitudelongitude.maps
5. Display.address
6. Create.location
7. Create.policy
8. Monitor.locationOnDevice
9. Location(X).ApplyPolicy(Y)
10. Monitor.locationOnDevice

The figure 2 has changed to figure 3, displaying the changes incorporated. Here the entities affected are tagged to generate the impacted areas. The actions involving the entities are translated to user stories. The section 6 on formal specific language discusses more on this method. The action sequences on the entities with the various flows introduced due to the changes, indicate the impacted areas.

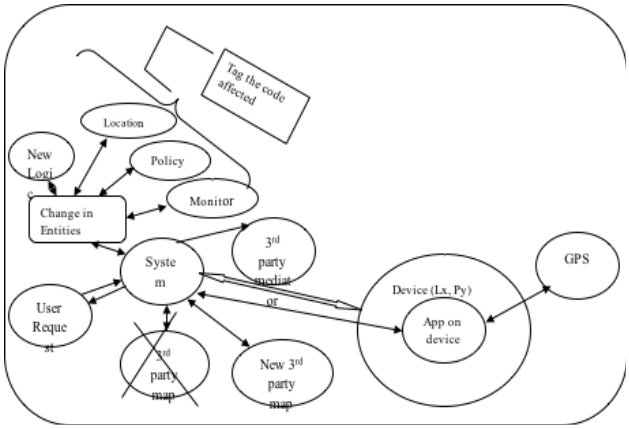


Fig: 4 Action Sequences on Entities and Tagging the Impacted areas

At the code level, entities are part of classes, objects, the application container that contains the collection of such entities and the relationships among them.

When a change is introduced in the requirement, the entities affected are tagged, implicitly tagging the above factors. So, when a user story involving such entities is generated with actions, the relationships are preserved. The tagging covers the code level and the association is reflected in the user stories. Figure 4 above can be abstracted as Figure 5.

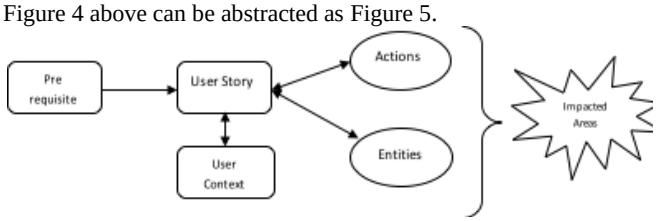


Fig 5 High Level User Story with Actions, Entities and Impacted Areas

Here the relationship between the model, code and the test are preserved [14]. Traceability Link Model [9] provides locators and relationships through which the requirements can be traced down to the tests. The *enhancement* to this model is to add apps or the components affected, and the degree of change should be determined by assigning weights to such changes. This ensures in deriving the intensity of the impacted areas and thereby resulting in appropriate selection of tests.

The formal method should preserve the correctness of traceability by facilitating Verification and validation methods [10].

5. QUALITY OF USER STORY SPECIFICATION

[21] Consistency of test is defined as the degree of uniformity, standardization and freedom from contradiction among the requirements, user stories design and test system. The output should be a meaningful and useful content. For this a set of assertions and validation rules are put in place.

They determine the validity of the user story with results namely: yes, no, error, none. The above method can be extended to generate data.

Rules (Functional, Design, System, and Business) could generate a lot of test conditions, which in turn could govern the course of coverage. There could be implicit and explicit rules. These rules provide a direction in modelling of the system which can generate validation functions.

For example: Consider the requirement, "User's location can be identified by performing 'Locate User'"

Rules pertaining to the above can be,

Business Rule:

i. Identify user location without compromising on user privacy
ii. Reducing cost by limiting the number of queries to the vendor

Functional Rule:

i. If the user allows access, collect, store and present the required information

Design Rule:

i. Communications from the system to the device are real-time
ii. 100,000 requests at any given instance

System Rule:

i. Respond based on the above rules.
ii. Maintain data privacy and tenancy

The assertions, pre-conditions and validation functions were derived based on the above rules. These assertion criteria determined the correctness of the generated user stories. They in turn implicitly validate the test adequacy from the generated user stories.

User story patterns can be translated to transformational functions. Some of the metrics to assess the test model (conformance to standards) – degree of uniformity, freedom of contradiction of requirements, rules, within generated tests, consistency and correctness is ensured by the validation functions. At-least one test user story was arrived for each requirement which enables traceability [19]. There are a number of variants in terms of inputs and outputs that can affect the generation.

Understanding the anatomy of the user story was essential to identify patterns. This enables us to move forward in the direction of user story mark-up language [4]. Thus, it gives us a handle to derive the impacted areas and to generate the test user stories.

6. FORMAL MODELING – DERIVATION OF IMPACT AREAS & GENERATION OF TEST USER STORY

A formal specification language is introduced to derive the user stories based on the individual models represented by the various types of systems that work together to perform a task. Those specifications are then used to automatically derive a test model,

which comprise of user stories, with added test dimensions that can be test executed with tagged impact areas. This helps to identify the regression suite.

The test generation looks up the user story for events and action words and simply translated to entities and action sequences for creating high level test models. The action keywords describe the behavior that is implemented by lower-level keyword models from the test model library [6].

Test scenarios are generated using depth first traversal of generated control flow based state machines according to criteria [17]

6.1 Formal Specification Language

The entities and action sequences are processed further using a formal language to auto-generate test-user stories. This calls for tagging of user story with requirements [11] which can collectively determine the system model. The domain space mapping can be represented as follows. Each of the attributes from the space has many to many relationships.

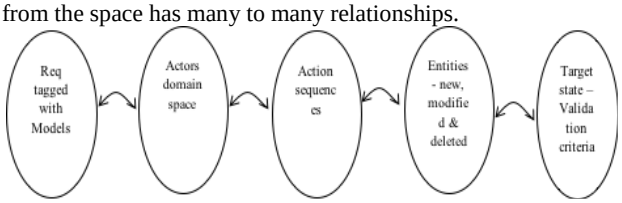


Fig: 6 Domain Space Mapping with Many-Many Relationships

To determine the impacted areas, a weight is added to the entities that are touched for code modification in this iteration. The weights are None, Minor, Major, Critical. Initially, all the weights are set to none. Then every time there is a change to an entity, it is tagged with one of the above mentioned weights. The state of the weights were collected throughout the iteration and impacted areas can be generated.

A counter is attached to each of the entities and each time the weights are assigned, it is incremented. The counter demonstrates the number of times the code has been modified. Once the iteration is completed, and a new branch is taken for the next iteration, these weights were reset to none and the counters were carried over. The figure 6 after incorporating these factors look like Figure 7

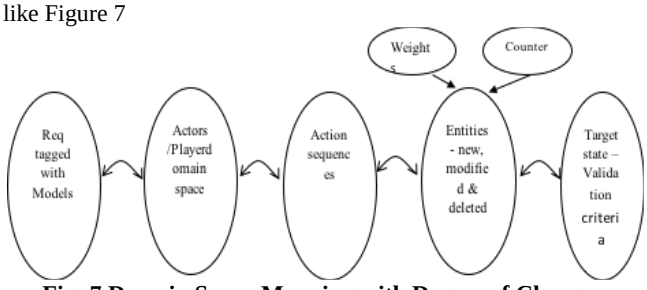


Fig: 7 Domain Space Mapping with Degree of Change Introduced

The general elements of user story comprising card, conversation and confirmation [3] are adapted for the user story modelling and test generation. The domain space provides the inputs to construct a user story. The syntax, operators and criteria are defined in the following figure 8.

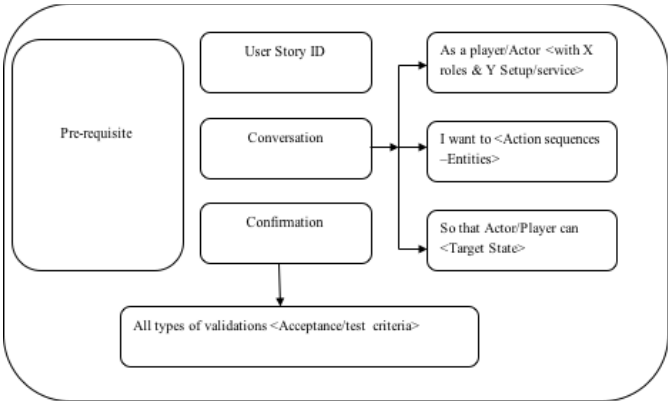


Fig: 8 Final Test User Story Structure

Entity Modeling for user stories are described with a finite number of factors as figure 9. User story is a 7 tuple defined as (P, A, C, E, S, T, I) where P is the set of pre-requisites, A is the set of actors or players, C is the set of action sequences, E is the set of entities, S is the set of Target states, T is the set of transition relations between the outcomes of the pre-requisites, actors, action sequences, entities and the target state and I is the set of impacted areas

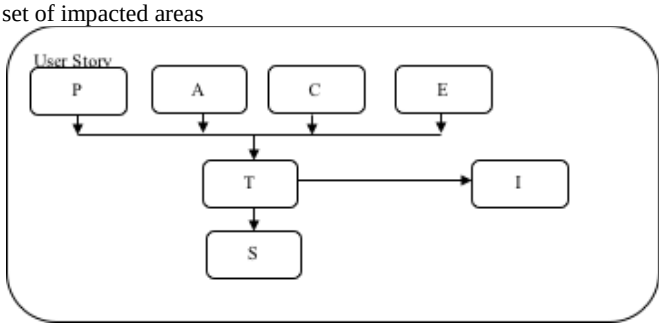


Fig: 9 User Story Tuple

The transition relation can be denoted as follows,

T: P X A X C X E ->h(S), where T returns a set of target states, assuming that some of the target states are non deterministic and therefore are used for observation.

For each action sequence C, there exists sets of entities E that are either old or new.

So, the impacted areas I can be derived by the following, Let E_{new} and E_{old} be the sets of new and old entities affected by an action sequence C. Then, impacted areas can be obtained by

$I = C \rightarrow (E_{new} \cap E_{old})$, E_{new} and E_{old} are quantified by the weights and counters

There exists a set of test attributes viz correctness of data, UI, Error handling, usability, timing/performance, security etc which are to be executed for each test user story. And every user story along with test attributes has to be executed against each dimension.

For example, a test user story should be executed for its *test attributes* like data, UI, memory etc. And a user story with test attributes should be executed for each *test dimension* like Browser stack, OS stack etc.

The user story transducer is a 9 tuple defined as (P, A, C, E, S, T, I, t, D) where t is the set of test attributes and D is the set of test dimensions and the corresponding functions can be defined as t: P X A X C X E ->h(S) and D(t): P X A X C X E ->h(S) as given in figure 10 below.

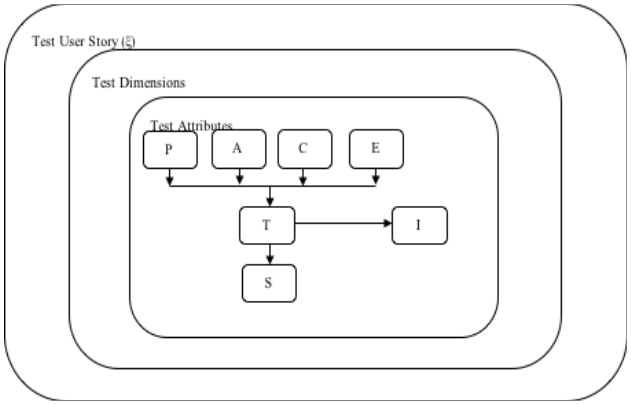


Fig: 10 Test User Story Transducer

The set of test attributes with variants are denoted by t. Here the test attributes are discrete. For example, end to end test user stories, covers a subset of total stories generated for this requirement. If the subset is executed individually, it has n variants.

Let f, g be a functions on the test attributes and dimensions for every user story U, where

f: t->h(U) such that, t=(t0, tn) where {x ∈ h(U)|t0< x < tn}, to and tn are test attributes.

g: D->h(U) such that, D=(D0, Dm) where {y∈ h(U)|D0<y<Dm), D0 and Dm are test dimensions.

Each user story U has N variations, so, if they have to be converted to testcases, the number of testcases are a Cartesian product of set of user stories and the N attributes. Impacted areas helps in prioritizing and hence reduces the number of testcases, but narrowed down coverage, every time a change is introduced.

To derive the execution effort, consider the set of user stories along with the set of test attributes and test dimensions. For each dimension D, and each test attribute or set of test attributes t, a user story has to be executed.

Assume the dimensions and attributes are discrete. So, the test execution effort (TEE) can be defined as,

$$TEE = \sum_{D=0}^{m} \sum_{t=0}^{n} U(D, t), \text{ for all } m, n$$

But in real time, the dimensions and attributes may sometimes be continuous. However, it is out of scope for this paper. Also, not all define dimensions and attributes are applicable for all stories all the time. Hence, the risk based prioritization [20] is very essential to reduce the testing effort. Here, the impacted areas was used to highlight the risk of not testing the change.

6.2 Formal Language Grammer

The Action sequences (A) can be derived based on the formal language grammar constructs. There has been some research on user story markup language (USML) [4]. The domain specific constructs can be formulated information derived from the user story tuple. This along with the action sequences result in test user stories.

6.3 Modular Functional Test User

The test user stories are modularized by feature modules. The functional test user stories (FTUS) are generated based on the combinatorial test design algorithms (CTD) with the functional attribute classes based on entities. Then the test parameters and the dimensions are tagged to the FTUS. Each individual FTUS with be in the form of a test user story tuple T, as derived in Fig 5. The atomic FTUS was grouped to form end to end FTUS.

6.4 Maintain the Latest Test at any given Instance

The functional attribute classes for CTD were checked in a version management system. Any changes to the requirements were again fed back into the class model that contains the transition relation Fig 9 and along with the functional attribute classes the test parameters and dimensions are revisited. FTUS are regenerated and this cycle.

The FTUS remains optimized and up to date in real-time and hence the tests are manageable in terms of both quantity and quality. Also impact of the new changes can be easily mapped as discussed in Fig 9

6.5 Maintain the Latest Automation Suite at Given Instance

The functional tests from section 5 are imported into the test management tools in use either manually or use the APIs provided by them. Most commercial tools have APIs. They are tagged with an attribute called Automation Status. This attribute holds a list of values like automated, to be automated, Manual and Impacted due to feature change. Based on the changes to the requirement the tests are regenerated. This might result in the following cases

1. Take the completely regenerated tests as the requirements have changed drastically
2. Retain old tests along with the changed, removed and new tests
3. Retain old tests along with the changed, removed, new and added tests

In all the above 3 cases one or more tests were changed and they are linked to automated suite, to indicate that the test has changed, and the automated test needs a fix. The changed tests are flagged and fixed to reflect the latest test.

6.6 Test User Story Generation Process

The ideal user story generation process is discussed in the below given figure 11

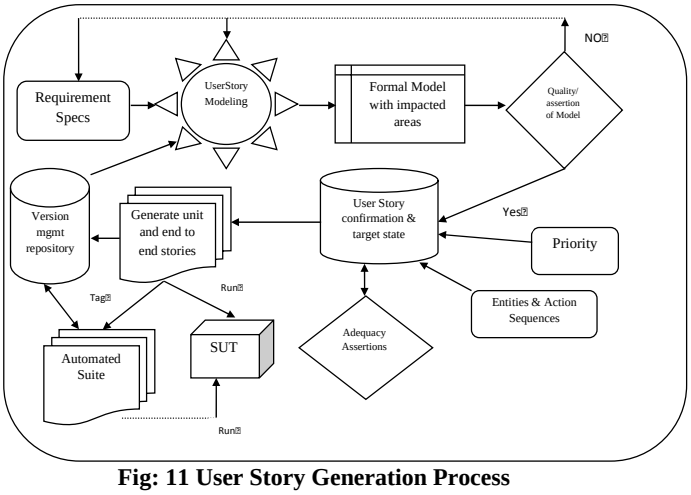


Fig: 11 User Story Generation Process

In the subsequent iteration, based on the changes in requirements, as discussed earlier, the entities and action sequences can change, at times leading to the change in domain space attributes. Hence, adjustments to the model was done. In order to be able to generate complete stories the test model has to be adjusted [8].

7. EXPERIMENT, MEASUREMENT AND RESULTS

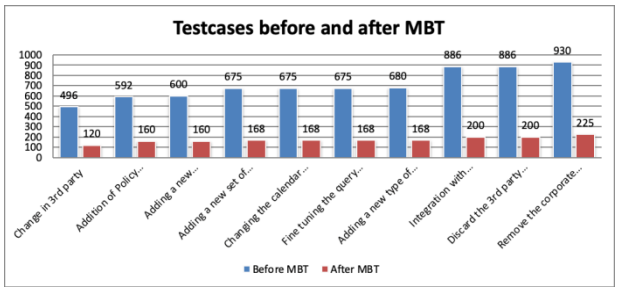
The feasibility of this test model was experimented and measured over 10 iterations of product development through the entire software development life.

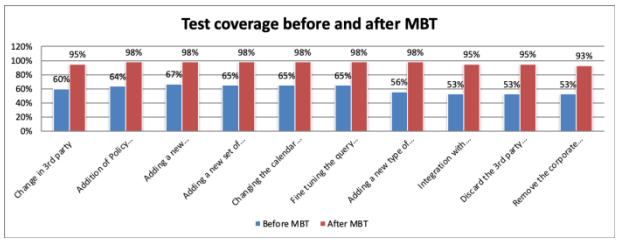
There are many factors that are involved in the real time product delivery cycle, there is too much of the noise. The factors involved are the stakeholders, multiple iterations of arriving at requirements, the entire rendition of project management: people, process and resources, and phases from development to customer enablement. All these are filtered out and only the parameters that matter for this research are considered.

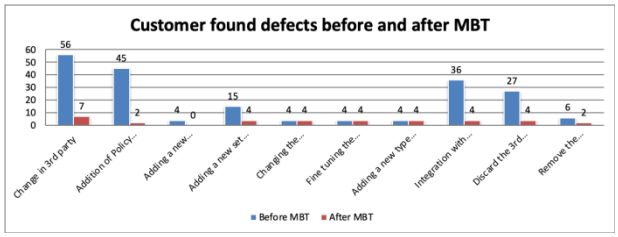
The parameters that matter for this research are requirements, whether the requirements are new or a change in the existing functionality, change in code and story points to measure the test adequacy, effectiveness, coverage optimization, test effort, customer found defects, performance tests and impacted areas. Also, there were tools that were used for project management, requirements, test management, bug tracking etc. The type of the tool used has no impact on the parameters that are experimented, monitored and measured for this research purpose.

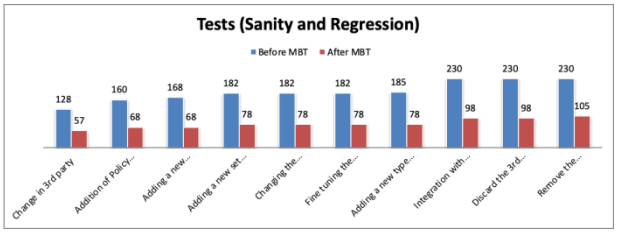
The experiment, observation and measurement were done over a period of 70 weeks. The parameters under observation are stated in the Appendix

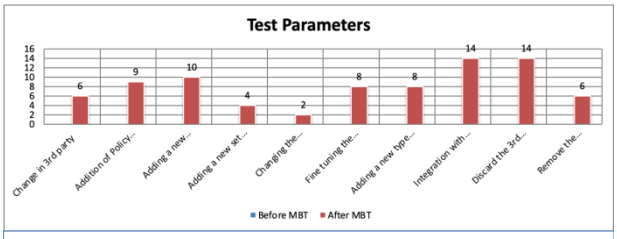
The results for the domain based MBT for test attributes and test dimensions with Impacted areas are presented in the figure 12 below.

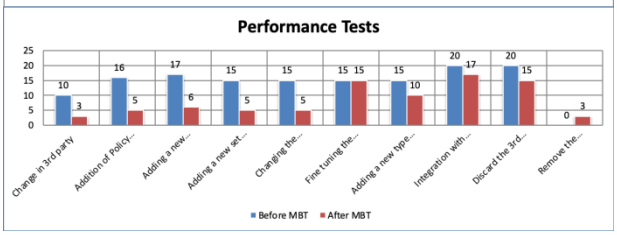












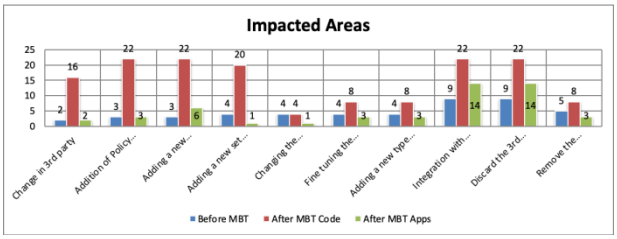


Fig: 12 Results

8. CONCLUSION

The software under test is a dynamically evolving system. The line between the new requirements and the changes introduced are almost negligible. The project span time for each iteration was 7weeks. In this scenario, existing traditional methods of generating tests and identifying the regression areas are extremely time consuming and cumbersome.

Hence, this attempt to adapt the existing models to extend to user stories and auto identification of impacted areas help in narrowing down the regression. The future work can comprise of a study, the performance and security paths for the feature are identified and are marked up with appropriate volume and relevant test data.

9. ACKNOWLEDGMENT

We are thankful to MaaS360, SaaS product and its management for allowing us to experiment and present the method in this paper.

10. REFERENCES

[1] Dias-Neto, A. C. & Travassos, G. H., 2009. *Model-based testing approaches selection for software projects*. Information and Software Technology, Nov, Vol 51(Issue 11), pp. 1487-1504

[2] Dick, J. & Faivre, A., 1993. *Automating the generation and sequencing of test cases from model-based specifications.* Springer-Verlag London, UK, FME '93 Proceedings of the

First International Symposium of Formal Methods Europe on Industrial-Strength Formal Methods , pp. 268-284.

[3] Jeffries, R., 2001. Agile Glossary. [Online] Available at: *https://www.agilealliance.org/glossary/three-cs/*

[4] Kamthan, P., 2011. Representation of User Stories in Descriptive Markup, Montreal, Quebec, Canada : Spectrum Research Repository, https://spectrum.library.concordia.ca/36094/

[5] Katara, M. & Kervinen, A., 2006. Making Model-Based Testing More Agile: A Use Case Driven Approach. Haifa, Israel , HVC'06 Proceedings of the 2nd international Haifa verification conference on Hardware and software, verification and testing, pp. 219-234 .

[6] Katara, M. & Kervinen, A., 2006. Making Model-Based Testing More Agile: A Use Case Driven Approach. Finland, https://link.springer.com/chapter/10.1007/978-3-540-70889-6_17, pp. 219-23

[7] Kelly, S. & Tolvanen, J.-P., 2008. Domain-Specific Modeling: Enabling Full Code Generation. s.l.:Wiley-IEEE Computer Society Press

[8] Löffler, R., Meyer, M. & Gottschalk, M., 2010. Formal scenario-based requirements specification and test case generation in healthcare applications. Cape Town, South Africa, SEHC '10 Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care , pp. 57-67

[9] Maletic, J. I. & Collard, M. L., 2009 . TQL: A query language to support traceability. Vancouver, BC, Canada, Traceability in Emerging Forms of Software Engineering, 2009. TEFSE '09. ICSE

[10] Merilinna, J. & Pärssinen, J., 2010. Verification and validation in the context of domain-specific modelling. Reno, Nevada, DSM '10 Proceedings of the 10th Workshop on Domain-Specific Modeling, p. Article 9

[11] Miller, T. a. S. P., 2012. A Case Study in Model-Based Testing of Specifications and Implementations.. Software Testing Verification and Reliability, Volume 22, pp. 22(1):33-63

[12] Naslavsky, L., Ziv, H. & Richardson, D. J., 2007. Towards leveraging model transformation to support model-based testing. Atlanta, Georgia, USA, ASE '07 Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, pp. 509-512

[13] Naslavsky, L., Ziv, H. & Richardson, D. J., 2007. Towards traceability of Model based Testing Artifacts. London, UK, AMOST'07, ACM 978-1-59593-850-3/07/0007

[14] Naslavsky, L., Ziv, H. & Richardson, D. J., 2008. Using Model Transformation to Support Model based Test Coverage MEasurement. Leipzig, Germany, AST'08, ACM 978-1-60558-030-2/08/05

[15] Puolitailval, O.-P. & Kansten, T., 2010. Towards flexible and Efficient Modle Based Testing Utilizing Domain Specific Modelling. NewYork, USA, DSM '10, ACM.

[16] S.W, A., 2006. Agile Modeling Home Page. [Online] Available at: www.agilemodeling.com

[17] Somé, S. S. & Cheng, X., 2008. An approach for supporting system-level test scenarios generation from textual use cases. Fortaleza, Ceara, Brazil, SAC '08 Proceedings of the 2008 ACM symposium on Applied computing, pp. 724-729

[18] Utting, M. & Legeard, B., 2007. Practical Model-Based Testing: A Tools Approach. s.l.:Morgan Kaufmann Publishers Inc

[19] www.omg.org, 2005. Object Management Group. [Online] Available at: http://www.omg.org/cgi- bin/doc?ptc/2005-11-01

[20] Yanping.C, RobertL.P & Sims, D. P., 2002. Specification based Regression Test Selection with Risk Analysis. Toronto, Ontario, Canada, CASCON '02, IBM Press ©2002

[21] Zander-Nowicka, J., Mosterman, P. J. & Schieferdecker, I., 2008. Quality of test specification by application of patterns. Nashville, Tennessee, USA, PLoP '08 Proceedings of the 15th Conference on Pattern Languages of Programs , p. Article 3

11. APPENDIX

Data for Release Cycles with New & Change in Requirements for Features

Release Cycle	Requirements	Change / New	Change/New	Story Points
1	Email Calendar Contacts Attachments Authentication Location based Policy	Change in 3rd party	Change	160
2	Email Calendar Contacts Attachments Authentication Location based Policy	Addition of Policy Parameter based on user authentication	New	178
3	Email Calendar Contacts Attachments Authentication Location based Policy	Adding a new authentication type	Change	180
4	Email Calendar Contacts Attachments Authentication Location based Policy	Adding a new set of default rules to the email	New	197
5	Email Calendar Contacts Attachments Authentication Location based Policy	Changing the calendar to match the new version of OS for the device	Change	197
6	Email Calendar Contacts Attachments Authentication Location based Policy	Fine tuning the query to get location details faster and minimize the cost	Change	197
7	Email Calendar Contacts Attachments Authentication Location based Policy	Adding a new type of file for attachment	New	200
8	Email Calendar Contacts Attachments Authentication Location based Policy	Integration with corporate gmail a/c	Change	210
9	Email Calendar Contacts Attachments Authentication Location based Policy	Discard the 3rd party gateway and write our own gateway	New	210
10	Email Calendar Contacts Attachments Authentication Location based Policy	Remove the corporate details if the device is jail-broken or rooted	New	250