Req2Test - Graph Driven Test Case Generation for Domain Specific Requirement

Veera Prathap Reddy M[#*1], Prasad P.V.R.D[#2], Manjunath Chikkamath[*3], Karthikeyan Ponnalagu[*4], Sarathchandra Mandadi[*5], and Praveen C.V.R[*6]

**Robert Bosch Engineering and Business Solutions Pvt. Ltd, Bangalore, India*

 # Department of Computer Science and Engineering, K.L Education Foundation, Vaddeswaram, Andhra Pradesh, India

Abstract

Software testing is a critical phase in the software development life cycle, as it validates the software against its requirements. Auto generation of test cases for software testing from natural language requirements pose a formidable challenge as requirements often do not follow a defined structure. In this paper, we propose Req2Test pipeline to auto generate test cases from a set of requirement statements. Our process includes domain specific knowledge graphs for extracting information, domain ontologies for identifying hierarchy of domain components and action sequences for actions to be performed in achieving a task. Knowledge graphs, domain ontology and action sequences contributes in addressing complete test coverage for requirement statements. The test cases are generated against industrial requirement statements on Automatic Wiper Control System in Automotive Domain and achieved promising results. We provide experimental results on industrial requirement and discuss the advantages and shortcomings of our approach.

Keywords - *Named Entity Recognition, Domain Knowledge, Domain Ontology, Test Case Generation*

I. INTRODUCTION

Software Testing is evaluation of the software against requirements gathered from users and system specifications. It is a salient step of software development, which is crucial to ensuring the programmed unit works under normal circumstances and to weed out faults prior to deployment. It is the cornerstone of verifying and validating that the expectations and requirements of the users and stakeholders have been met in the system under development [2]. Every newly developed product of any kind has to be tested to ensure that it correctly performs the functions for which it was designed [1]. It has been estimated that software testing uses up to 50% of the overall development cost [6] and the testing activities consume approximately 40% of the overall development time and effort [7].

The software requirement specification is the best source for understanding stakeholders expectations and generating the test cases corresponding to the same [2]. Software Requirement Specification (SRS) document contains detailed system-level description of the requirements and use-cases. The information contained in SRS is used to create detailed Test Cases [3]. The requirements-based testing process addresses two major issues: first, validating that the requirements are correct, complete, unambiguous, and logically consistent; and second, designing a necessary and sufficient set of test cases from those requirements [9]. Majority of the requirements specification documents have requirements written in Natural Language. According to some of surveys, 79% [4] of all requirements are documented in natural language and 7% [5] in formal specifications. Therefore, there is a need to transform these requirements written in natural language into computer-readable format in order to automate the process of generating test cases. Natural Language Processing (NLP) techniques enable us to morph sentences expressed in natural language into statements that can be understood syntactically and semantically and process accordingly by a machine [2] to generate test cases from requirements.

II. MOTIVATION AND BACK GROUND

Test cases can be developed in one of three major ways. They can be developed algorithmically, they can be taken from data from an existing application that is being replaced or upgraded, or they can be developed from requirements [1]. From given requirements, test cases can be developed by manually or by automation. Manual development of test cases have some serious concerns [1][2]. To improve the effectiveness and efficiency of testing, testers need to create high-quality test cases. Writing test cases, however, is a tedious task and prone to human errors. Thus, it is crucial that we can find a way of automatically generating high-quality test cases which can be used in making testing activities more effective and efficient to assure software quality [1].

This paper describes the auto generation of test cases form SRS statements. The proposed approach consumes SRS statement as input and creates test cases as output. The paper is structured as follows. In section III, we discuss the review of related work. Section IV includes a sample requirement statement processed through out the paper. We describe Req2Test pipeline in detail in

section V. In section VI, we present Validation and Discussion and Conclusion in section VII.

III. RELATED WORK

Automation of test cases is not a new idea, Weyuker et al. [8] introduced the approach of Automation of test cases for testing. In their paper, they discussed different strategies for automatically generating test cases. They developed algorithms to automatically generate test sets that would be substantially smaller than exhaustive test sets, but would nonetheless be highly effective at detecting faults. According to James Martin [10], the root causes of 56% of all defects identified in software projects are introduced in the requirements phase about 50% of requirement defects are result of poorly written, unclear, ambiguous, and incorrect requirements. The other 50% of requirement defects are due to incompleteness of specifications (i.e. omitted requirements). Therefore, in requirement based testing need for automation of test cases is even more serious. Multiple manual process are involved in requirement based testing, where software testers have to define test completion criterion, design test cases from requirements, build test cases, execute the test cases and verify whether requirements of the software are met or not. If the requirement based testing processes do not run correctly or consistently, the size of the test cases may be too enormous to complete testing in reasonable time or the test cases cannot provide the expected results [9]. Automation has proven to be an effective way to reduce cost and shorten the product release time, and will be a major factor for the success of software testing [11]. To minimize the effort and cost on testing, many companies have started investing in the automation of testing processes.

Anurag and Shubhashis [3] developed a Litmus tool to generate test cases from functional requirement document. The tool works on each requirement sentence and generates one or more test cases through a five-step process. Charles et al. [1] developed an Automated test case Generator (ATCG) that takes requirements statements as inputs and creates test cases as output.

However in all these approaches there is no completeness in test coverage generated by requirement statements. These approaches couldn't able to identify domain hierarchy of components and sequence of execution steps for test coverage to be accomplished.

In this Paper, we present Requirement to test case Generation (Req2Test). We have constructed a domain model by processing the domain specific corpus for building knowledge base specific to domain. We then process every requirement statement and validate the requirement against domain knowledge base and domain ontology to validate the requirement and extract domain hierarchy of entities participating in the requirement statement. We then construct a test template from the knowledge extracted and generate test cases from the template by using Natural Language Generation or Rule Based approaches.

IV. DATASET

Req2Test can be applied to any type of domain specific or domain agnostic data. In this paper, to explain the Req2Test pipeline we selected Automatic Wiper Control System from Automotive domain. The related corpus for analysis is collected from web source [12]. Please refer the appendix for the corpus.

Every step in the Architecture is explained with the following sample requirement statement: "when the humidity senor is switched to ON and drive voltage is in the specified range the wiper should set to start"

V. REQ2TEST

Req2Test identifies a requirement as a sentence and validate the requirement whether it is valid requirement or not with respect to domain we are interested in. Req2Test also identifies the testable intent in the requirement statement for appropriate actions/action sequences to be performed in generating test cases from the given requirement statement. A Test case includes the 'test condition', the 'test sequence' and the 'expected result'. Test condition includes pre-conditions and post-conditions for a test case to get executed and verified respectively. A pre-condition is defined as the entry criterion for the test case being tested. A post-condition is defined as the state changes to be verified after the test case is executed. Test sequence is the ordered sequence of steps a tester would have to execute to perform the test. The corresponding output from a correctly implemented system is denoted as the expected result. The test cases are generated using pipeline depicted in figure 1.

Every component in the pipeline is explained in the coming sections with the help of above mentioned sample requirement statement. The requirement sentence is tagged with named entities using Bi-LSTM-CRF model and parsed using Dependency parser developed by Spacy [13]. Spacy Dependency Parser provides information on the syntax of a grammatically correct English sentence by connecting pairs of words through dependency tokens. Information is extracted from the statement as Triplets (*Subject-PredicateObject*) for validating the requirement statement against domain ontology constructed. Once testable intent is identified from the requirement statement, required information (attributes, pre-conditions, post-conditions, action sequences) about entitiesparticipating in the test are extracted from Knowledge Graph(KG) to construct a

template. Template constructed will be used as input to generate Test cases.

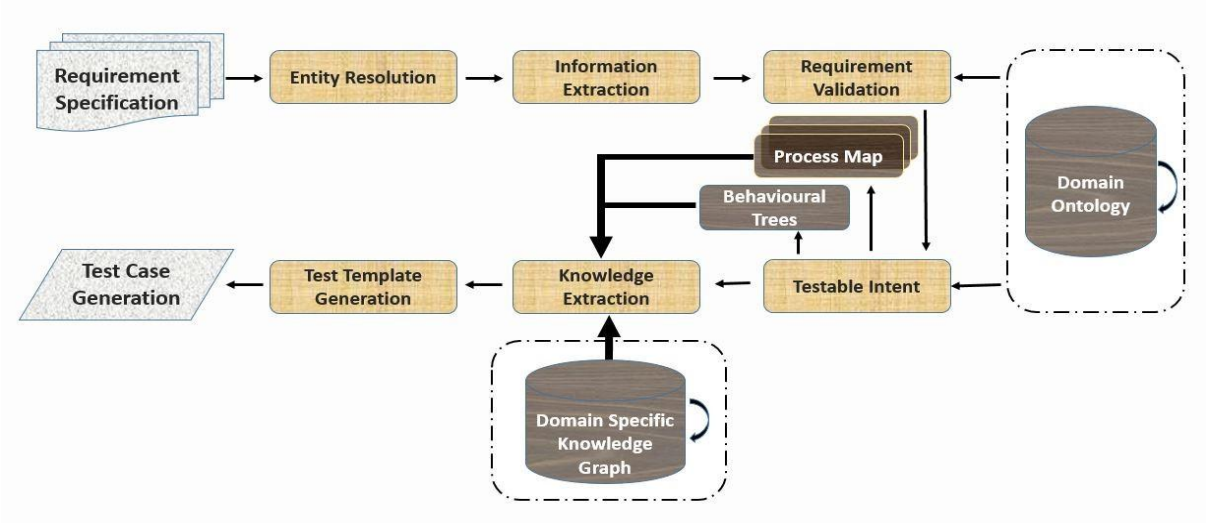

Fig. 1. Req2Test Architecture

A. Domain Based Knowledge Graph Generation

From the perspective of creating knowledge graphs (KG) based on ontology, there are two main approaches, one is top-down, and the other is bottom-up [20]. Top-down approachmeans that ontology and schema should be defined,and then knowledge instances are added into knowledge base. This approach emphasizes the well-defined domain ontologies to represent the actual instances of knowledge graphs. The bottom-up approach extracts knowledge instances from the Linked Open Data (LOD) or other knowledge resources. After knowledge fusing the populated instances, the top-level ontologism are built by means of knowledge instances to create the whole KG's. We followed bottom-up approach for construction of knowledge base. For construction of Knowledge Graph, knowledge is extracted from domain corpus available and is represented in machine readable format RDF (Resource Description Framework)[1].

The types of knowledge extraction are roughly divided into three types: entity extraction, relation extraction and attribute extraction [20]. In fact, the attribute extraction can be thought as a kind of special relation extraction. Entity extraction, including Named-Entity Recognition (NER), is to discover entities from a wide variety of knowledge resource and try to classify them into pre-defined categories. The quality of entity extraction usually greatly influences the efficiency and quality of subsequent knowledge acquisition, so it is one of the most fundamental and important part of knowledge extraction. After entity extraction, the relationships among the entities are analyzed to extract the conceptual relations. Relation extraction is to find the relations between entities and obtain semantic information in order to construct knowledge graphs. The attribute extraction is to define the intentional semantics of the entities while the relation extraction is to specify the denotational semantics of the entities. The attribute extraction is important to define the concept of an entity more clearly.

B. Processing Requirement Statement

The objective of processing requirement statement is to extract relationships between entities and to confirm action to be performed based on the requirement specification. The input is requirement statement where test cases need to be generated. The requirement statement is analyzed using standard NLP techniques and entities, verb relations, actions are recognized and extracted by the custom built algorithm. An overview of triplet extraction for a single functional requirement statement is shown in Figure 2.

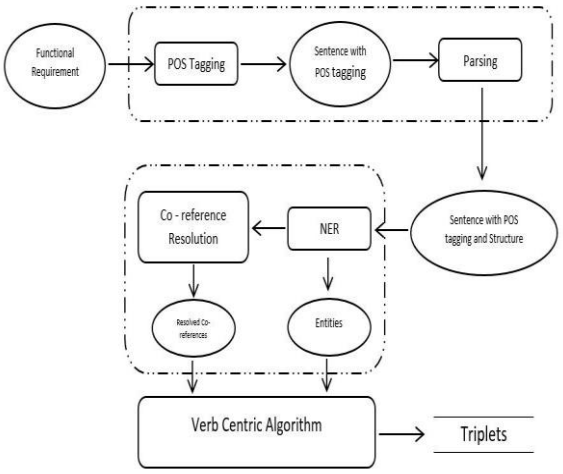

Fig. 2.Information Extraction Process

Every requirement statement is parsed with Spacy dependency parser. For information extraction, NER identifies relevant entities and parser identifies

POS tagging and dependency tokens for each word in the sentence.

Finally, the system outputs the relations by resolving the co-references into a list of binary relations in the form of Entity — Verb relationship — Entity.

1) Entity Recognition:The aim of the Named Entity Recognition(NER) and extraction is to identify all entities from the requirement sentence. Requirement sentence is passed through BI-LSTM-CRF network model to identify the named entities in the statement. It uses generic stochastic Gradient Descent (SGD) forward and backward training procedure. While training the model, in every epoch, the training data is divided in to multiple batches and one batch at a time is processed. Every batch contains a list of sentences which is determined by the parameter of batch size. For every batch, we first run bi-directional LSTM-CRF model forward pass which includes the forward pass for both forward state and backward state of LSTM. As a result, we get the the output score $f_\theta(|x|_1^T)$ for all tags at all positions. We then run CRF layer forward and backward pass to compute gradients for network output and state transition edges. After that, we can back propagate the errors from the output to the input, which includes the backward pass for both forward and backward states of LSTM. Finally we update the network parameters which include the state transition matrix $[A]_{i,j} \forall_{i,j}$ and the original bidirectional LSTM parameters Θ. The BI-LSTM-CRF architecture for the given requirement statement is shown in figure 3.

2) Information Extraction: The main objective of the information extraction is to extract relationships between entities from requirement sentences having complex structures. In order to extract single or multiple relations between entities a verb centric algorithm was built which can extract relations between complex compound sentences. The input to the algorithm is requirement statement which will extract single/multiple relations between entities as triplets in the form of (Subject-Verb-Object). The system process each sentence to find co-references and resolve them using Neural-Co-ref [16] if present. For the selected requirement statement the algorithm extracted the following triplets.

Extracted triplets for sample requirement statement:
(Humidity Sensor, is, on),
(drive voltage, is in, specified range),
(wiper, should set, to start)

3) Extraction of Domain Ontology: An ontology represents the fundamental knowledge pertinent to the application domain, namely the concepts constituting the domain and the relationships between them. Survey by Mich et al. [4] shows, majority of the requirements documents are written in natural language. As a consequence, most requirement documents are imprecise, incomplete, and become apparent when we compare the project-specific model with a generic model of the application domain [21]. We need to transform natural language representation of requirement information into a form that facilitates comparison with a domain model.

Naturally, we also need a domain model against which to compare and this presupposes a means to construct such models.

An ontology is generated from a generic domain description and a project specific model is generated from requirement documents. For ontology generation, natural language processing techniques (Pos tagging, NER, IE) are used to aid the construction. We compare both the models to validate. When inconsistencies are found, we generate feedback for the analyst. The generated feedback was validated on a case study and has proven useful to improve both requirement documents and models.

First, we extract domain-specific abstractions and relationships between them from a document or documents that are representative of the application domain. Based on the lexical form of the extracted concepts, built in custom classifier classifies them and provides a taxonomy. Finally, we look for non-taxonomic relationships between the concepts.

The domain abstraction for components involved in the requirement statement is shown in figure 4.

4) Domain Knowledge Graph Vs Domain Ontology*:* Knowledge Base (KB) is fact-oriented and an ontology is much more schema-oriented. In Knowledge Graph, you have certainly a schema (like the one described by DBPedia, Freebase, Yago, etc.) and a set of facts,“New Delhi isA City”, “New Delhi hasInhabitants 2M”, ”New Delhi isCapitalOf India”, etc. So, you can (easily) answer to questions like ”What is the capital of India?”, ”How many inhabitants does New Delhi have?” or you can provide a short description of New Delhi (or any given entity in general).

Domain ontology explain main concepts of a domain, how are these concepts related and which attributes do they have. Here the focus is on the description, with the highest possible expressiveness (disjointness, values restrictions, cardinality restrictions) and the useful annotations (synonyms, definition of terms, examples, comments on design choices, etc.), of the entities for a given domain. Data (or facts) are not the main concern when designing a domain ontology.

5) Validating Requirement Statement:Requirement statement is fed into the model for entity extraction. After identifying entities every sentence in the requirement is parsed for incorporating the information about entities. A successfully linked sentence is tagged as either testable or non-testable. We define a testable sentence as one which adheres to the definition of a

requirement [15]. A requirement is a contract that specifies what a user/agent does to a system (impetus) and how the system should respond. We, thus define a testable sentence as one which has a subject, an action and optionally an object. The action should contain a modal Verb (like should, would, etc.). Information about domain entities and actions an user performs were validated against the domain ontology. For any inconsistency in the requirement statement is checked with analyst.

An example of a non-testable sentence is ”The DHT11 is a basic, ultra low-cost digital temperature and humiditysensor”. Notice such sentences are assumptions or definitions and by themselves are not testable. Quick identification of non-testable requirements helps a business analyst to review and modify the requirements if needed. The aspect of testability is also critical in identifying parts of a sentence that should be tested.. For requirement sentence humidity senor is switched to ON and drive voltage is in the specified range then wiper should set to start, the underlined fragment should be inconsequential to a tester as it talks about the underlying logic.The test case should be based on the action of the system which is *“to set the wiper ON”*. This aspect is automatically caught by the testability module and the Test cases generated are around the action and pre-conditions related to the entity as specified in the requirement.

6) ***Identifying Testable Intent****:* We define a test intent as the smallest segment of a requirement sentence that conveys enough information about the purpose of the test [3]. A test intent has a subject, an action and (optionally) an object. For requirement statement processed we have three test intents which were

(humidity sensor, is, ON),
(drive voltage, is in, specified range),
(wiper, should set, to start).

The first two intents talk about pre-conditions or assumptions to be checked. The third intent is a testable intent where test cases needs to be generated.

C. Extracting Knowledge from Domain Knowledge Base

After identifying the testable intent and action to be performed on the entities, a procedure to extract information regarding how such an action can be performed on entities have to be extracted from the knowledge base. The extracted information should set the context on when and how such action can be performed on the entities. When specifies the pre-conditions to be met to perform the action. How specifies the sequence of actions to be performed in achieving this task. Sequence of actions include the sequence of steps to be followed in achieving such action.We accept that graph search is a NP-hard (nondeterministic polynomial-time hardness) problem. Therefore, we specify a computationally bounded search architecture with some parameters, and automatically optimize its performance based on the needs of the application using training queries drawn from a target distribution [22].

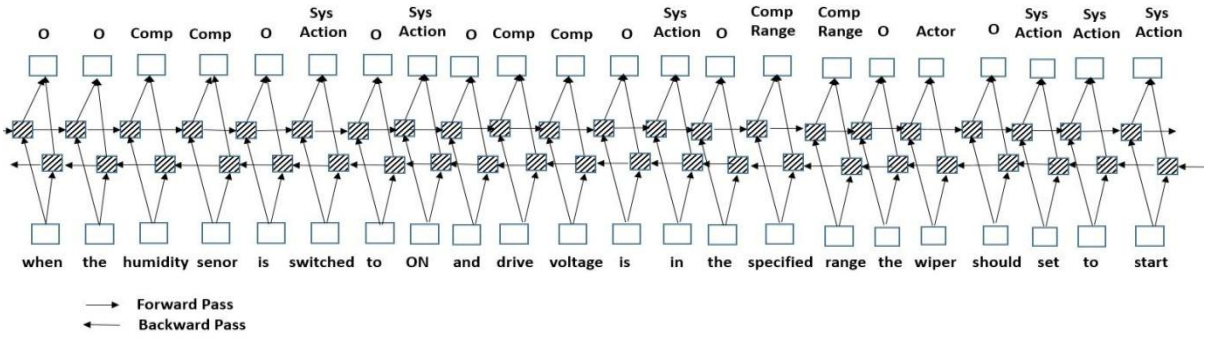

Fig. 3.BI-LSTM-CRF Named Entity Recognition Model

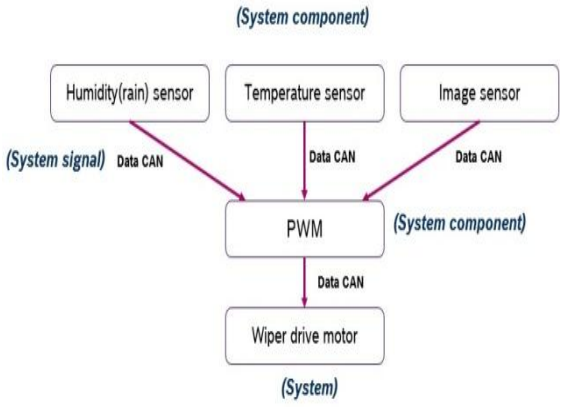

Fig. 4.Ontology extracted for Requirement Statement

The architecture for extracting knowledge from knowledge base for the given requirement statement consists of two parts 1. *Selection* and 2. *Evaluation*. The architecture for subgraph extraction from the knowledge base is shown in figure 5.

1) ***Selection:***Will generate a candidate subgraph of size (parameter) to answer the input query with entities and actions as input to query.

2) ***Evaluation****:* will find the best answer by evaluating the input graph query on the generated

subgraph. The scoring function F(.) is used to score the candidate answers. Given a graph query Q, data graph G, and search bound τ, the selection component generates a subgraph $G_{Q,T} \subset G$ whose size is τ and the evaluation component finds the best answer Â by searching $G_{Q,T}$.

Subgraph extracted will consist of information for the entities with its attributes, pre-conditions, post-conditions and attribute values range for all the participating entities with the action to be performed. The best answer Â isassigned a score based on the no. of levels in the graph to be traversed for

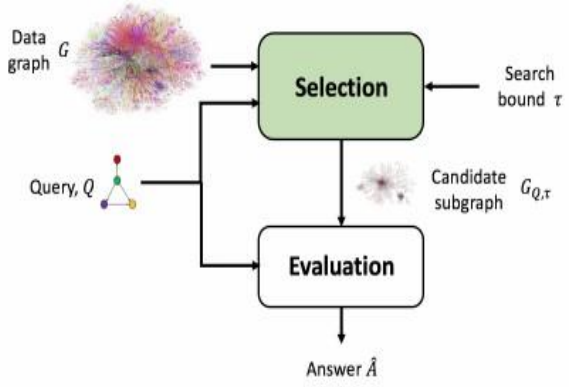

Fig. 5. Process of Knowledge Extraction from Knowledge Base [22]

accurate information to be extracted. The informationextracted from the knowledge graph is identified by scoring function based on the depth. The required information for requirement statement is searched for entity classes Component and System is shown in table I.

D. Pre-condition and Post-condition Identification

A Test case is characterized by a known input and by an expected output. The known input should test a pre-condition and the expected output should test a post-condition that is caused by a scenario using the pre-condition. A state can be a pre-condition or a post-condition based on its relation to the event. If the state is a result of the event then it is a postcondition for this event, if the event needs the system to be in this particular state to perform then it is a pre-condition for this event. A State is the state the system in before or after performing an action. An event is the action to be performed.

Functional requirements are expressed as Behavior Trees (BT) in terms of components realizing states. Like graphical form that represents behavior of individual or networks of entities which realize or change states, make decisions, respond to cause events, and interact by exchanging information or passing [16]. The BT technique is based on the assumptions that systems exhibit components that have attributes and realize and change states, so they make other components realize and change state as well [17].

The representation of BT for the requirement statement is shown in State Transition Table or Behavior Trees in Figure 6.

In the represented Behavior Tree for Wiper, Path ”a” gives information on when Humidity Sensor is On and Drive Voltage in Range then state of Wiper is set to ON. In path ”b” and path ”c” the state of Wiper didn’t change when Humidity Sensor is Off and so on.

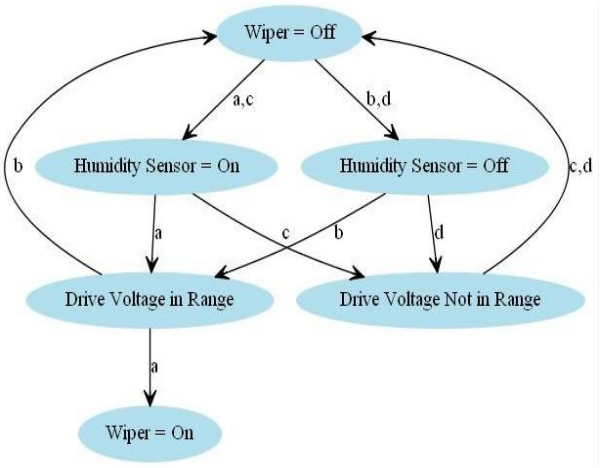

Fig. 6. Transition states of components as Behavior Trees

E. Action Sequence

According to Anderson [18], there are two essential components of knowledge: declarative knowledge and procedural knowledge. Declarative knowledge collects useful knowledge/information. It is the factual or conceptual knowledgethat a person has. In designing a generic architecture to represent procedural knowledge, the actions defined by domain experts and the control of action flow are two important tasks. Unlike declarative knowledge, the meaning of procedural knowledge cannot be figured out until the whole process is finished. Processes are hardto describebut important in relation to problem solving. Focusing on the process by which knowledge is constructed is more important than focusing on target knowledge. For our work Req2Test, we followed an ontological representation scheme called Process Map (PM) [19] to represent procedural knowledge.

The main idea is to identify (1) the activity structure from given behavioral models of components; and (2) the connection relations of these components.

Table I. Required Information Extraction For Requirement Statement

Entity1 Class	Entity1	Relation	Entity2 Class	Entity2
Component	Sensors like temperature	are used to detect	other	environment conditions
Component	Sensors like humidity	are used to detect	other	environment conditions
Component	Sensors like precipitation	are used to detect	other	environment conditions
Component	Sensors like humidity	are used to switch on	System	Wiper
Component	Sensors like temperature	are used to switch on	System	Wiper
Component	Sensors like precipitation	are used to switch on	System	Wiper
System	Wiper System	Is Activated By	Component	Engine Control Unit (ECU)
system	wiper system	is activated in tandem with	Component	control unit
Component	Drive motor	depends on	Component	Pulse Width Modulator (PWM)
Component	Drive motor	depends on the Pulse width modulator (PWM) for	Component	drive voltage
Component	Drive motor	depends on the Pulse width modulator (PWM) for	Component	control of speed
system	Wiper	switch	attribute	ON
system	Wiper	switch	attribute	OFF
Component	Drive Motor Voltage	is set between	attribute	9v to 14v
Component	Drive Motor Voltage	checked during	other	Running condition
system	Wipers	is checked for OFF condition of	Component	driver motor
system	Wipers	is checked for ON condition of	Component	driver motor

Table II. Process Map For Requirement Statement

Component	Pre-Condition	Input	Output	Effect	Action Sequence
Wiper	Wiper is OFF	Humidity sensor=ON and Drive Motor in 9v to 14v	Wiper=On	Wiper=On	Humidity sensor=ON and Drive Motor in 9v to 14v

Process Map:We call a series of processes that indicate a fact as a Process Map (PM). A Process Map is composed from a collection of processes that accomplish a specific goal. Each process can be regarded as a series of actions. Each action can be further decomposed into a series of frames. This idea is very similar to the decomposition of a large program into components. A process can be categorized into atomic and composite ones with different parameters in PM. An atomic process means a single process that has three properties: ID, process name and type, and five attributes: precondition, input, output, effect and action. A composite process is one composed from atomic processes. The relations of process, actions and frames is as shown in figure 7

The Process Map includes two parts: process and action. The function of process is to control processes, which includes junction control and direction control. An action is defined as the interpreted behavior of a component. An action engine is a mapping from actions to components. We need domain expert to analyze all required actions, and to identify exactly which actions can help learning.

Action sequence to be executed is captured in process maps if it already exists, else will be updated if such a sequence exists. For the requirement statement the action sequence is shown in Table II.

F. Structured data creation for requirement

In the process of generating test cases from requirement statement, using Req2Test we have identified the entities using named entity recognition model. The sentence is then parsed to extract information, which result in Triplets in the form of *(subject - predicate/action - object)*.

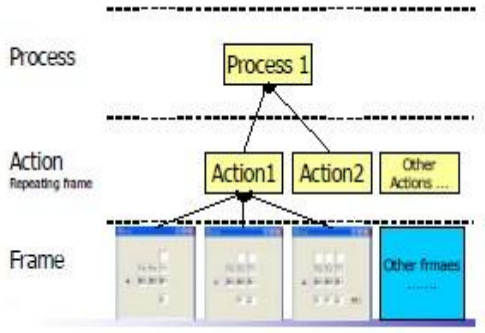

Fig. 7. Process Maps [19]

These entities with their actions to be performed were cross checked with domain ontology for testability. Information regarding the entities that needs to be extracted and how depth the graph to be traversed to extract knowledge of entities along with their attributes were identified. Pre-conditions to be met and Post-conditions to satisfy when action performed is extracted from Behavior Trees and action sequence for performing tasks were extracted from process maps.

The representation of extracted information regarding the tasks or actions to be performed is stored in a structured representation in tabular format. Information stored in tabular representation consists of entity class, entity, relation class, relation, attribute, pre-condition, post-condition and action sequence. All these information is captured in three different tables.

For the requirement statement, the data extracted from Knowledge base is shown in Table I and data extracted from State Transition Tables or Behavior Trees is shown in figure 6. Data extracted from the action sequence is shown in Table II.

By consolidating all the three tables, a structured representation of data is created which is a source for template creation.

G. Test Template creation

Test case template is used as an input for auto generation of test cases using Natural Language Generation/Rule based Algorithm. It acts as an intermediary step between knowledge extracted for a given requirement from knowledge base and actual test cases user will look for testing.

The essence of creating a test template for auto generation of test cases of given requirement statement is to ensure the coverage of the test cases. Task that needs to be performed as extracted and validated against testable intent from domain ontology is captured in the test template in structured representation. Task that requires sequence of actions to be performed in achieving a task is extracted from the process map. For Req2Test, test template is generated by custom built algorithm by aggregating the necessary information from the files that were populated with the necessary data extracted from the knowledge base.

The template representation of test cases to be generated is shown in figure 8.

Entity1	Entity2	Attr	Cond	Attr_Val
Wiper	Wiper. Humidity sensor	Yes	prec	0, 1
	Wiper. Driver motor	Yes	prec	On, Off
Driver Motor	Driver Motor. Voltage range	Yes	prec	9v to 14v
Wiper	Wiper.Swithced on	Yes	postc	1

Fig. 8.Test Template for Requirement Statement

H. Test Case Generation using Template

A test case (denoted as TC) comprises of a Test Condition (C), the Test Sequence (TS) and the Expected Results (ER). The Test Condition defines the logical condition for which the test is to be performed. The Test Sequence is a set of execution steps. The corresponding output from a correctly implemented system is captured in the expected results. The test condition, the test sequence and the expected results are identified and extracted from requirement sentence through the Req2Test pipeline.

Test template is sourced as input for generation of Test cases. Template provides information about the hierarchy of the entities participate in test case generation and preconditions to be met for the requirement, post-conditions to be validated. Template also includes attribute range a variable can hold, also the test sequence extracted from the process maps if any test sequence exists for the combination of entity and action participating in the test case generation.

An Algorithm is written with logical template rules to identify different combinations possible to extract test cases. Template generated will take care of hierarchy of the entities present in the requirement which will reduce the false positives in the generated test cases. By prioritizing the sequence, redundancy in the generated test cases will be eliminated. For any false positives the accuracy of the information extracted from the Knowledge base is cross verified and will be corrected for any missing information.

A false positive is identified as test case which is not part of test case to be verified and is identified by algorithm. A false negative is test case which is not generated by the algorithm but has to be verified.

Positive Test Cases: Positive test case verifies for the action stated in the requirement sentence. Positive test cases generated were shown in Table III.

Negative Test Cases: Negative test case will identify all conditions where the affirmative action of the system should not happen. Negative test cases Generated were shown in Table IV.

For every test case, the expected output is cross checked along with post-conditions with actual output and actual post conditions of the application being tested.

VI. VALIDATION AND DISCUSSION

Req2Test was run on the requirement statements and the generated test cases were analyzed manually for accuracy. The accuracy of parser is measured by its ability to link grammatically correct sentences and by the number of correct (and intelligible) test cases as a percentage of the total number of test cases generated and to the overall test coverage.

The generated Test cases showed a significant percent where the information extracted from the Knowledge Graphs, Behavior Trees and Process Maps regarding entities participating in the testable intent is complete. The completeness in the test cases is dependent on the Test Template.

Req2Test includes better test granularity through Test Intents which helps to ensure all aspects are tested and better covered through positive conditions and negative conditions. Req2Test could able to generate test cases that were missed by the human analysts. Req2Test has applicability to Business Analysts too, who can estimate the complexity of projects through the number of Positive and Negative test cases generated. The test coverage included in the requirements selected is completed for

Table III. Req2test Positive Test Cases Generated For Requirement Sentence

No	Test case	Expected Result
1	Wiper.Humidity sensor = 0 AND wiper.Drive Motor = On AND Drive Motor.Voltage Range = 9v to 14v	Wiper=On
2	Wiper.Humidity sensor = 0 AND wiper.Drive Motor = Off AND Drive Motor.Voltage Range = 9v to 14v	Wiper=On
3	Wiper.Humidity sensor = 1 AND wiper.Drive Motor = On AND Drive Motor.Voltage Range = 9v to 14v	Wiper=On
4	Wiper.Humidity sensor = 1 AND wiper.Drive Motor = Off AND Drive Motor.Voltage Range = 9v to 14v	Wiper=On

Table IV. Req2test Negative Test Cases Generated For Requirement Sentence

No	Test Case	Expected Result
1	Wiper.Humidity sensor <> 0 AND wiper.Drive Motor = On AND Drive Motor.Voltage Range = 9v to 14v	Wiper=Off
2	Wiper.Humidity sensor = 0 AND wiper.Drive Motor <> Off AND Drive Motor.Voltage Range = 9v to 14v	Wiper=Off
3	Wiper.Humidity sensor = 0 AND wiper.Drive Motor = Off AND Drive Motor.Voltage Range <> 9v to 14v	Wiper=Off
4	Wiper.Humidity sensor <> 1 AND wiper.Drive Motor = On AND Drive Motor.Voltage Range = 9v to 14v	Wiper=Off
5	Wiper.Humidity sensor = 1 AND wiper.Drive Motor <> Off AND Drive Motor.Voltage Range = 9v to 14v	Wiper=Off
6	Wiper.Humidity sensor = 1 AND wiper.Drive Motor = Off AND Drive Motor.Voltage Range <> 9v to 14v	Wiper=Off
7	Wiper.Humidity sensor <> 1 AND wiper.Drive Motor <> On AND Drive Motor.Voltage Range = 9v to 14v	Wiper=Off
8	Wiper.Humidity sensor = 1 AND wiper.Drive Motor <> Off AND Drive Motor.Voltage Range <> 9v to 14v	Wiper=Off
9	Wiper.Humidity sensor <> 1 AND wiper.Drive Motor = Off AND Drive Motor.Voltage Range <> 9v to 14v	Wiper=Off

the requirement statements where parser could able to identify the testable intent for single line requirement

statements. For multi line requirement statements the test coverage is comparatively less than single line requirement statement because of multiple entities to be considered as a single state which Req2Test couldn't achieve.

We validated Req2Test against different industrial requirement statements on Automatic Wiper Control module. Requirement statements were classified for validation, as single line requirements having few entities and relations making a test sequence to be executed and multiple line requirements having multiple entities and actions form a test sequence within same domain. The accuracy for multi line requirement statements is lower than that of single line requirement statements. We found the accuracy of such sentences is low because of test coverage should include sequences and pre-conditions of multiple entities as a single entity, which couldn't be retrieved by Req2Test. Req2Test Validation results are shown in figure 9.

Requirement Class	No. of Requirements	Positive Test Cases	Negative Test Cases
Single Line	20	110	240
Multi Line	10	250	460

Fig. 9. Experimental Results of Req2Test on Industrial Requirement statements

VII. CONCLUSIONS

In this paper, we presented Req2Test methodology and tool to generate functional test cases from natural language

(English) functional requirement specification document. Req2Test identifies the entities using named entity recognition model. The sentence is then parsed to extract information which result in triplets in the form of *subject - predicate/action - object*. These entities with their actions to be performed were cross checked with domain ontology for testability. Information regarding the entities that needs to be extracted and how depth the graph to be traversed to extract knowledge of entities along with their attributes,pre-conditions and post-conditions were identified. Procedural knowledge for performing tasks were extracted from process maps. We have presented the accuracy of the tool over industrial project requirements. Req2Test has received positive feedback in its practical review from test engineers and business analysts. Our future research will be focusing on improving the accuracy, deployment by improving domain knowledge representation and identifying and extracting pre and post conditions when multiple domain components involve for test case generation.

VIII. APPENDIX

The following is the dataset processed to construct knowledge graph, domain ontology for the requirement statement processed in the paper.

”Wiper control system play a key role in keeping driver safe in times of precipitation. Wiper system has seen a change in approach over a period of time from traditional wiper system to automated sensor based system. Sensors like Humidity, precipitation and temperature are used to detect the environment conditions and switch on the wiper automatically. Wiper system is activated and controlled by Engine control unit(ECU) in tandem with Body control unit (BCU). Controller area network (CAN) is used as a primary form of communication between ECU and other components for automated wiper control like sensors, micro controllers etc. The motor drive primarily depends on the Pulse width modulator (PWM) for the drive voltage and control of speed. The wiper switch on and off is subjected to various conditions like sensor condition of humidity, visibility, range of temperatures and vehicle conditions as given by the ECU through data CAN protocol. Further to the automated switching of wipers is checked for the ON and OFF condition of the driver motor. The visibility of the vehicle driver is sensed through the image sensor and the speed of the wiper driver motor is varied by the PWM based on these inputs. For every start condition the wiper driver motor voltage range is checked, this is done to protect from heat and stress conditions of the wiper driver motor. The BCM module is coupled with the ECU so that vehicle conditions and the wiper play conditions are coupled. The ECU sends the signals through the data CAN to the BCM module which in turn checks the TRUE or FALSE condition of the all prevalent condition like sensor inputs, drive voltage, motor speed and status, required variation speed parameters for the PWM etc. The micro controller works in slave condition to the ECU and connected by CAN protocol. for the automatic start of wiper, humidity sensor condition is checked for 1 or 0, driver motor condition if checked for ON or OFF, driver motor voltage range is set between 9v to 14v. Driver motor over-voltage condition is checked during the running condition.”

REFERENCES

[1] Charles P. Morgan, Mark l. Gillenson, Xihui Zhang, Son N. Bui and Euntae “TED” Lee, “ATCG: An Automated Test Case Generator”,Journal of Information Technology Management, vol. XXVII, no. 3, pp.112-120, 2016.

[2] Priyanka Kulkarni and Yashada Joglekar, “Generating and analyzing test cases from software requirements using NLP and Hadoop”, International Journal of Current Engineering and Technology, vol. 4, no.6, pp.3934-3937, 2014.

[3] Anurag Dwarakanathand Shubhashis Sengupta, “Litmus: Generation of test cases from functional requirements in natural language” in Proceedings of NLDB'12 Proceedings of the 17th international conference on Applications of Natural Language Processing and Information Systems, pp.58-69, 2012.

[4] Mich Luisa, Franch Mariangela and Novi Inverardi Pierluigi, “Market research for requirements analysis using linguistic tools”,Requirements Eng, pp.40-56, 2004.

[5] Colin JNeill and Phillip A Laplante, “Requirements engineering: the state of the practice”,in IEEE Software, vol. 20, no. 6, pp. 40-45, 2003.

[6] Kirti Nagpal and Raman Chawla, “Improvement of Software Development Process: A new SDLC Model”, International Journal of Latest Research in Science and Technology, vol.1, no.3, pp. 217-224,2012.

[7] N Vijay,“Little Joe Model of Software Testing”, Software Solutions Lab, Honeywell, Bangalore, India, pp.1-12, 2001.

[8] Elaine Weyuker, Tar& Goradia, and Ashutosh Singh, “Automatically Generating Test Data from a Boolean Specification”, IEEE Transactions on Software Engineering, vol 20, no. 5, pp. 353-363, 1994.

[9] “Requirements Based Testing Process Overview”, Bender RBT Inc. Queensbury, New York, USA, pp. 1-18, 2009.

[10] James Martin, An information systems manifesto, Prentice Hall, p.300, 1984.

[11] L. H. Tahat, B. Vaysburg, B. Korel and A. J. Bader, "Requirement-based automated black-box test generation," in25th Annual International Computer Software and Applications Conference. COMPSAC 2001, Chicago, IL, pp. 489-495, 2001.

[12] Prabakaran P.M. (2012) “How automatic wiper control works in modernCar”,[Online].Available:https://www.engineersgarage.com/contribution/how-automatic-wiper-control-works-modern-car

[13] Matthew Honnibal and Mark Johnson, “An improved non-monotonic transition system for dependency parsing”, in Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing,Lisbon, Portugal,pp. 1373–1378, 2015.

[14] Google’s new artificial intelligence can’t understand these sentences. Can you?. Washington Post. Retrieved 2016-12-18.

[15] IEEE Recommended Practice for Software Requirements Specifications,IEEE Std. 830-1998, pp. 1-40, 1998.

[16] R. G Dromey, “For Requirements to Design: Formalizing the Key Steps”, in Proceedings of First International Conference onSoftware Engineering and Formal Methods, Brisbane, Queensland, Australia, pp. 2-11, 2003.

[17] Cesar Gonzalez, Brian Hendreson-Sellers and Geoff Dromey, “A Meta model for the Behavior Trees Modeling Technique”, in Proceedings of Third International Conference on Information Technology and Applications (ICITA'05), vol.1, pp. 35-39, 2005.

[18] John R. Anderson, Rules of the Mind,Psychology Press; 1st ed. p. 336, 1993.

[19] Chun-Hung Lu, Shih-Hung Wu, LiongYu Tu and Wen-Lian Hsu, "The design of an intelligent tutoring system based on the ontology of procedural knowledge," in Proceedings of IEEE International Conference on Advanced Learning Technologies, pp. 525-529, 2004.

[20] Zhanfang Zhao, Sung-Kook Han and In-Mi So, “Architecture of Knowledge Graph Construction Techniques”, International Journal of Pure and Applied Mathematics, vol. 118, no. 19, pp. 1869-1883, 2018.

[21] LeonidKof,Ricardo Gacitua, Mark Rouncefield and Peter Sawyer, “Ontology and model alignment as a means for requirements validation”, in Proceedings of IEEE Fourth International Conference on Semantic Computing, Pittsburgh, PA, USA,pp.46-51, 2010.

[22] F A Rezaur Rahman Chowdhury, Chao Ma, Md Rakibul Islam, Mohammad Hossein Namaki, Mohammad Omar Faruk and JanardhanRao Doppa, “Select-and-Evaluate: A Learning Framework for Large-Scale Knowledge Graph Search”, in Proceedings of the Ninth Asian Conference on Machine Learning, vol. 77, pp. 129-144, 2017.