## SWPC 2020 Presentation Submission

|  |  |
| --- | --- |
| Title | Change-based UEFI validation using keyphrase extraction techniques |
| Author Names | Abhishek Mishra |
| WWIDs | 11796709 |

## Summary/Key Takeaway

Change-based validation is an approach that allows cost-saving and early bug capture in the testing process. Current change-based validation approaches either involve manual identification of the impacted tests or require huge mapping effort between the changed files and impacted test cases. This work discusses how there is a lack of accuracy and scalability in current approaches and aims to solve the problem using Natural language processing.

In this presentation, we go over the hybrid model of heuristic and graphical approaches used to solve the problem. We discuss the insights in data that are useful in designing such a system and how the ideology can be borrowed in similar applications.

## Opportunity/Problem

UEFI firmware has high design complexity and thus requires high release cadence. Every binary requires exhaustive testing to ensure quality delivery. The cost of validation on such scale forms a significant portion of Intel investment in the project. Several optimization approaches exist which aim to reduce cost by either yield-based optimization (identifying tests which yield no bugs and deprioritizing their testing) or removing redundant testing by identifying duplication of efforts. Yield-based optimization ignores the possibility that some recent change can fail the low yield test cases. Similarly, redundancy removal is a one-time optimization that is not designed to identify possible failures and prioritize them.

Change-based validation is an approach with the principle that all expected changes in validation result directly from the software changes [1]. There is a direct correlation between the changes going inside a binary and the possible failures or pending verifications. The task of identifying tests based on incoming changes is very challenging. It requires one to assess incoming changes, map them to impacted features, and then identify which tests cover the features. The number of test cases in UEFI firmware validation can easily average over 1000 tests per project. The number of incoming changes for every project can also be huge, thus increasing the resource required to perform the assessment manually. It requires one to have broad domain expertise and project background to make the decisions on which features are being affected and identify all the impacted tests. Moreover, the methodology chosen to implement change-based validation needs to be scalable and have defined risk-assessment (independence from human errors). How can we have a widely deployable system to identify the impacted test cases accurately at a low cost?

## Solution

Every firmware change information is either a bug-fix description or implementation of a new requirement. It contains keywords and key phrases that are linked to the feature being impacted by the change. Similarly, every test case contains information that is linked to the features or bug-fixes being verified. These relations can be extracted and utilized to identify the impacted test cases with every change, see Fig. 1.

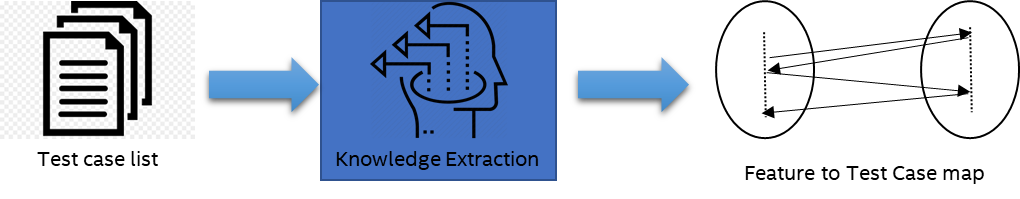


Fig. 1 Extracting the feature to test case mapping from the test case list.

We can divide the solution into two phases. Phase 1 is extracting the knowledge of feature to test case mapping and building what we call here a knowledge base. Phase 2 is using the knowledge base to recommend test cases, identify the impacted domains and rank the impacted domains in order of impact. See Fig. 2 and 3. The task of extracting the feature to test case mapping is the same as the task of extracting the BIOS changes to feature mapping. This process can be broken down into three parts - tokenization, data pre-processing, and keyphrase extraction.

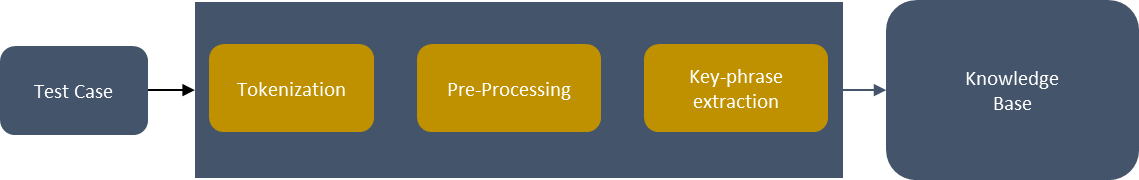


Fig. 2 Building a knowledge base by extracting feature-to-test mapping from the test case list

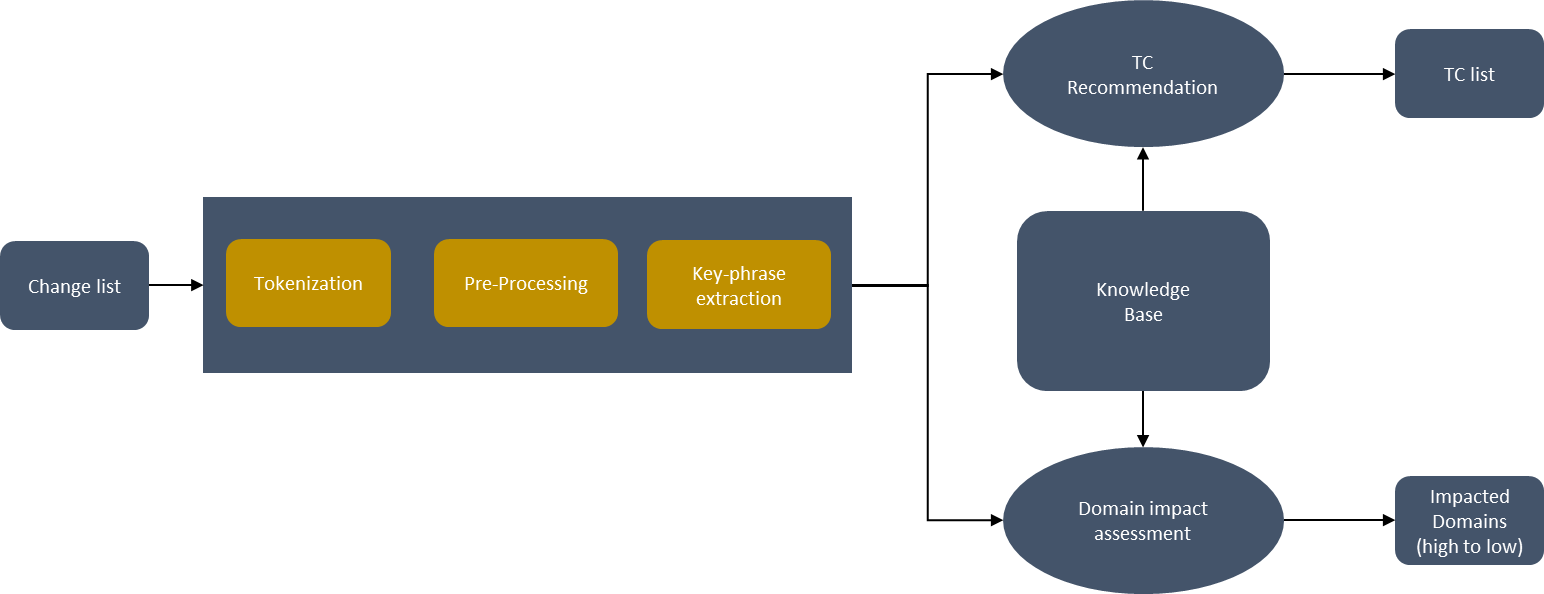


Fig. 3 Extracting change-to-feature mapping from change list and using the knowledge base to recommend test cases, identify impacted domains and rank domains in terms of impact

**Tokenization and Data pre-processing**

In Phase 1, see Fig. 2, we take a test case list as input and tokenize the test case titles. Tokenization is a process of breaking down a sentence in words. The natural language processing tasks which we are going to utilize in this research require sentences to be broken down in words. The words are known as tokens, and we need to pre-process them for our algorithms to be effective. Data pre-processing involves the following steps:

* Removing Stopwords

Stopwords are words which do not add any value to the document in terms of the intended application. For example, “the,” “is,” “on” are Stopwords.

* Removing punctuation

We remove punctuations because we are dealing with tokens only, and keeping punctuations as tokens does not add any value.

* Noise removal

The noise here refers to symbols and characters which are just part of the text as formatting but are not required.

* Change to lowercase

This removes duplicacy in the data. Our data may contain the same word in lower case and upper case, which will increase the size of the corpus and reduce efficiency.

* Lemmatization

Lemmatization is the process of reducing a word to its lemma (root). It also serves the purpose of removing data redundancy. For example, “run,” “ran,” and “running” all occur in a text and have the root as “run.” We only need to store “run” to identify the occurrence of this specific term in the corpus.

**Keyphrase extraction and building the knowledge base**

Every document contains specific keywords or group of keywords which we call keyphrases. Keyphrases are important in identifying the subject of a document. Intuitively, humans understand a sentence by breaking a sentence into keyphrases. The co-occurrence, frequency, and uniqueness of the keyphrases help distinguish the context of a document. In our application, we want to use the knowledge base to perform two tasks: Identify the test cases that must be tested to cover the code changes and identify the impacted domains and rank them in order of impact.

We use the n-gram extraction approach to extract keyphrases from the test case names. N-gram extraction means extracting a group of n words from the sentence [2]. Keyphrases, rather than keywords, are better at identifying the context of a document. For example, the keyphrase “memory correctable error” will convey more precise information than using just “memory,” “correctable,” or “error” separately. Empirically, we have observed that most UEFI firmware domain terms can be extracted if we consider unigram, bigram and trigrams. The extracted knowledge base looks similar to the structure shown in Table 1. In this approach, we find specific keywords or keyphrases that occur in many domains and test cases [3]. Using these keywords or keyphrases ends up in recommending a large portion of the test plan or they might identify all domains as impacted domains. Hence we keep a “ignore” dataset which helps us filter out the list of keyphrases that are highly prevalent across the dataset.

|  |  |  |
| --- | --- | --- |
| **Domain 1** | **Domain 2** | **Domain 3** |
| |  |  | | --- | --- | | Test Case ID | Extracted n-grams | | |  |  | | --- | --- | | Test Case ID | Extracted n-grams | | |  |  | | --- | --- | | Test Case ID | Extracted n-grams | |

Table 1 A toy example showing the domain-wise extracted information of test cases. A test case to extracted keyphrase mapping exists inside every test case.

**Helpful and Unhelpful keyphrases**

As mentioned in the last section, specific keywords do not contribute positively to the task of identifying impacted domains and test cases. Identifying such keywords/keyphrases is a challenging task as they score highly on the most favored metric of measuring importance, i.e., frequency. Other measures of the importance of keyphrases exist like TF-IDF (statistical), word-degree-to-frequency ratio (graphical). However, they fail to identify the keyphrases that can be useful for the recommendation of test cases and favor the keyphrases which fall in the middle between the frequency of occurrence and specificity.

We solve this problem by first creating a word graph [4]. In this graph, we populate each node as a word and connect it with neighboring words in a sliding window of size 5. This way, every word is connected to a co-occurring word. We know the names of our domains and eliminate any keywords which are the same as the domain name. We calculate the degree of connectivity of every keyphrase and use the following criteria to rank the keyphrases.

1. Identify and eliminate keyphrases that are connected to more than 30% of the total count of domains.

This ensures that no keyphrase can recommend or highlight a large part of the test suite.

1. With the remaining keywords, identify the number of test cases impacted in every domain. The higher the count, the higher the rank of the keyphrase within the domain.

**Using the knowledge base to recommend test cases and identify impacted domains**

The change list is a list of all the code check-ins inside the UEFI BIOS source. The change description contains text which has a vocabulary same as that of the test cases and can be used to identify the test cases which might be impacted by the code changes. We extract the list of all n-grams which are part of our knowledge base and create a dataset similar to Table 2.

|  |  |
| --- | --- |
| **Change ID** | **Extracted n-grams** |
| 1 | “Bios,” “warm reset,” “register.” |
| 2 | “memory correctable error,” “os” |
| 3 | “linux”, “acpi,” “pci enumeration” |

Table 2 A toy example showing how changes in the change list are mapped to extracted n-grams which can be used to identify related test cases and domains.

Once we have this data from change lists, we score every test case for individual changes based on the presence of keyphrases.

------------------------------------------------ Eq. 1

*Ts*is the score calculated for every test case. *Ki* is the total number of test cases impacted by any *i­­th* keyphrase where *i* ranges from *0* to *n-1* for the *n* keyphrases present in the change description. For every change description, we calculate this score for all test cases and list the test cases as a recommendation for that change description. The impact on domains and domain ranking is calculated by counting the number of test cases from each domain.

We have tried multiple approaches to solve the problem of recommending the test cases and identifying and ranking the impacted domains. We have used doc2vec to vectorize the sentences in the change list and identify the most similar sentences in the dataset of test cases [5]. We have used LDA for topic extraction and identify the impacted domains. Domain classification has also been attempted with the help of Logistic Regression and Naïve Bayes classifiers. In either of the approaches mentioned above, the performance of the system is not satisfactory and highly underwhelming compared to the designed statistical model. Thus we recommend the usage of a combination of statistical and graphical NLP approaches for small datasets like the one we are using.

## Conclusion

A change-based validation approach is a modern approach that aims to reduce the cost of regression testing by directing the validation teams to a smaller test dataset. In its implementation so far, we have been able to capture 80% bugs consistently in a program that is in the post Product candidate stage. In future work, we aim to use the results predicted by the statistical module as feedback in the training classifiers and recommender systems. This will enable the trained model to eventually give highly accurate results with the small unsupervised datasets in UEFI firmware validation environments. We highly recommend the hybrid of statistical and graphical NLP techniques for enabling efficient usage of data on small unsupervised datasets.

## Self-Assessment

Summarize on how your submission satisfies each SWPC [review criteria](#_Review_Criteria_for) in a few sentences.

|  |  |
| --- | --- |
| **Criteria** | **Supporting Details** |
| Scope/Interest | The scope of this work is ubiquitous in Intel or outside as it aims to deal with the problem of implementing natural language processing techniques in an unsupervised environment with an unstructured and small dataset. Anybody facing a similar challenge or looking to build a test recommender or impact analysis can borrow from this work. |
| Innovation/Transformation | The methods discussed in this work provide a new path of NLP based validation in Test and validation environments. These methods are also insightful for any future work in improving the performance of recommender systems in the field of natural language processing in general. |
| Result | The author has implemented this method on the validation suite of a program in the post Program candidate stage and has captured over 80% bugs consistently in the recommended tests. This method is now in process of implementation across the entire “IAGS SFP FIV” org. It is one of the primary optimization approaches to be considered for cost-saving and early bug capture. |

## References

1. [Change-based Test management](https://www.globalspec.com/reference/55313/203279/what-is-change-based-test-management)
2. Siddiqi, Sifatullah & Sharan, Aditi. (2015). Keyword and Keyphrase Extraction Techniques: A Literature Review. International Journal of Computer Applications. 109. 18-23. 10.5120/19161-0607.
3. Firoozeh, N., Nazarenko, A., Alizon, F., & Daille, B. (2020). Keyword extraction: Issues and methods. Natural Language Engineering, 26(3), 259-291. doi:10.1017/S1351324919000457Statistical and graphical hybrid approaches in NLP
4. Danesh, Soheil, Tamara Sumner, and James H. Martin. "Sgrank: Combining statistical and graphical methods to improve the state of the art in unsupervised keyphrase extraction." Proceedings of the fourth joint conference on lexical and computational semantics. 2015.
5. [Word embeddings](https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/)