Predictive Test Selection

RedHat

Rubi Arviv, Alon Mannor

Product Requirements Document

Revision History

Version	Description of Change	Author	Date

Approval

List relevant people from the company that need to approve the requirements.

Department/Mentor	Name & Email	Date
Mentor - Red Hat research supervisor	Ilya Kolchinsky ikolchin@redhat.com	

Table of Contents

1. Purpose & Scope

The purpose of this project is to minimize the time a new code change takes to reach production, while maximizing the level of confidence in the new code.

This will be achieved by an ML model that will try to predict which tests will fail, given a code change. This will be accomplished by utilizing predefined features (e.g. number of files changed, number of files dependent on changed file etc.).

This will be achieved by recreating the model described in the article: *Predictive Test Selection (by Facebook research)* (hereinafter "**The Facebook article**")

2. Motivation

This project aims to minimize the time it takes to deploy new code to production while maintaining a (predefined) high-enough level of probability that the code won't introduce regression.

Motivation A

Software developers are always writing and deploying code to production. This code can include bug fixes, new features, new enhancements to existing features and much more.
Since every such piece of new code improves the end-product in some way (if it does what it's supposed to :)), there's a clear motivation to add it to the codebase as quickly as possible.

Motivation B

Our 2nd motivation is making sure we don't break stuff in our effort to move fast[1].
The code base includes tests that potentially check things as small as micro code units and single lines all the way to huge end-to-end and integrations of whole mega components of code, long complex processes - and everything in between.
In the effort to deploy fast it's infeasible to run all tests on every code change.

Thus, our 2nd motivation is to create a tool that selects in a "smart" way which tests to run - given a code change. This is in order to ensure fast delivery of code while not compromising on the probability that it won't introduce bugs \ regressions etc.

[1] https://www.businessinsider.com/mark-zuckerberg-on-facebooks-new-motto-2014-5

3. Assumptions, Global Design Constraints and External Dependencies

3.1. Critical Success Factors

- *We'll be able to come up with a satisfying way to produce the relevant data, with an emphasis on enabling data creation with features identical (or otherwise very similar) to the one in the FB article*
- *We'll create a codebase that fetches the data, prepares it and feeds it into the ML implementation. This will be done in a way s.t. a future RedHat team will be able to seamlessly continue where we left off.*

3.2. Design and Implementation Constraints

3.2.1. External Standards

At the very minimum, the project is required to recreate the model (and method described in the Facebook article.

Either way, we have no limitations on things like technologies, programming languages etc.

3.2.2. Regulatory Standards

NA.

3.2.3. Localization/Internationalization Standards

NA.

3.2.4. Backwards Compatibility

NA.

3.3. Additional Assumptions and Dependencies

The project we are working on is RedHat OpenShift

The data we need is a mapping between failed tests and the code changes that failed them.

Since no such relevant dataset exists for this project, we assume we can create one in the given timeframe.

4. Related Documents

The project will recreate the article *Predictive Test Selection (by Facebook research)*

5. Product Features/Functionalities

The end-model's input will be a code change and its output will be a list of tests - each with a numbered priority. The higher the priority - the more likely this test will fail on the input code change.

5.1. Requirement A

A good enough understanding of the resources from which we fetch the data (the Github project for openshift and the CI artifacts dashboard and repo)

6. Security requirement

NA.

7. Management/Integration

This project is a POC (originally planned for a year) and as such isn't planned to "meet" production any time soon.

This project will be monitored and evaluated by Ilya Kolchinsky and Gil Klein from Redhat. They will decide if to push up the backlog for a future handoff to a team in Redhat.

7.1. Logs/Performance Events & Alerts

NA.

7.2. Admin Dashboards

NA.

7.3. Role based access control (RBAC)/User management

NA.

7.4. Integrations with external systems

We'll fetch data from these open sources:

- GitHub
- Openshift's CI dashboard and repo

8. Scalability & Performance

NA

9. Data

This is a critical section for those developing ML/DL model or any project that relates to the company's data.

You must detail in thorough:

- What is the data?
- Where is it coming from?
- Add examples of the data (anonymize if needed but the format should be clear)
- Relate to the size of the data available to you.
- Do you need to synthesize some data?
- If there are possible issues dealing with large volumes of data, please state them.

At its simpleset, the data is a long list of items, each of which is a mapping between a code changeset and the results of the tests that were run as part of the CI\CD process for that given changeset (as a prerequisite for deployment).
Both a changeset as well as test\s can each contain multiple files. In openshift, a test is identified by a string called a "locator" which can be thought of as a special command to run it. This locator can be mapped to a list of test files. Note that most of the time it'll be a list of a single test file, but not necessarily. There were times when we weren't successful at mapping between a test locator and any test file.

The data is composed of a few features that are fetched using an agent that also binds the changeset to tests. After the raw data is fetched, we run code that flattens it and also aggregates some data into new features (e.g. breakdowns of changes according to days). The data is not anonymized. The data is computed for the last 56 days.
Our data-producing code looks at a changeset and a list of tests that ran on the project's entire code containing that changeset. It then breaks it down into single data records - each for a specific file in the changeset and a specific test file (if one was found for the given test locator).

The data schema is as follows (name in parentheses is the name in the FB article, if applicable):

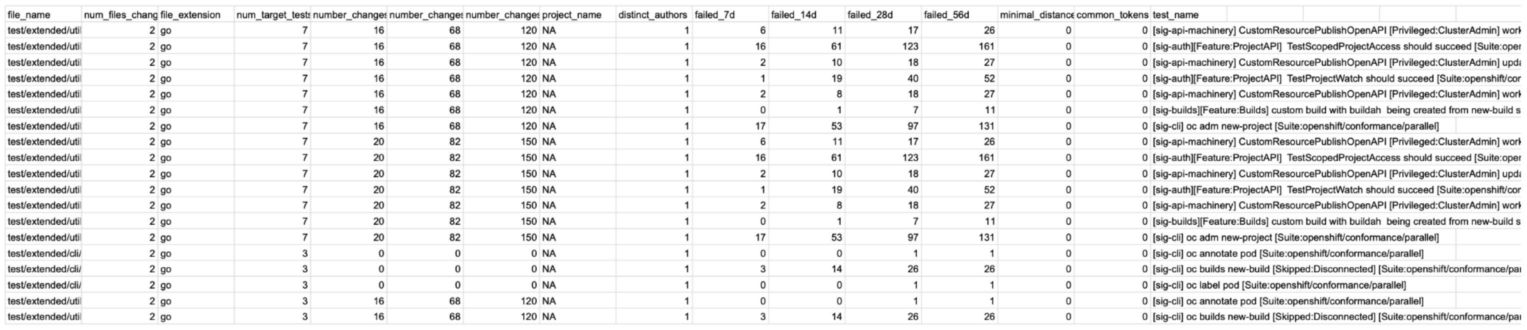
```
"schema_fields": {
 "file_name": The full path of the changed file,
 "num_files_changed" (File Cardinality): Number of files in this changeset,
 "file_extension": The extension of the file,
 "num_target_tests" (Target Cardinality): Number of overall tests that were ran for this changeset,
 "number_changes_3d": Number of changes made to this file in the last 3 days,
 "number_changes_14d": Number of changes made to this file in the last 14 days,
 "number_changes_56d": Number of changes made to this file in the last 56 days,,
 "project_name": The project's name (we ran our code on a single project so this field was ignored),
 "distinct_authors": Number of distinct authors of changeset,
 "failed_7d": Number of times this test failed in the last 7 days,
 "failed_14d": Number of times this test failed in the last 14 days,
 "failed_28d": Number of times this test failed in the last 28 days,
 "failed_56d": Number of times this test failed in the last 56 days,
 "minimal_distance": The minimal distance needed to traverse to get from the test file to the changed file
(files in the same directory get 0),
 "common_tokens": Number of shared folders, starting from the root directory, between the changed file and
the test file,
 "test_file" : The full path of the test file,
 "test_name": The test's name (i.e. it's locator)
}
```

Note: as for the features “minimal_distance” and “common_tokens” - they are framework \ language \ implementation dependent. These features were chosen by facebook because:

- It “...approximates how close are changes to a given target and the significance of the impact on it” (in the case of minimal distance)
- It Proxies “human perceived relevance” (in the case of common tokens)

In order to get similar insight from these features and to decouple the dependency in specific language etc., one should consider replacing \ adding a feature like “code coverage”. This means running a tool that, given a changeset, will output (approximately) how much code of the entire project is affected.

Sample data: link to CSV (below is an excerpt of that data)



file_name	num_files_chang	file_extension	num_target_tests	number_changes	number_changes	number_changes	project_name	distinct_authors	failed_7d	failed_14d	failed_28d	failed_56d	minimal_distance	common_tokens	test_name
test/extended/util	2	go	7	16	68	120	NA	1	6	11	17	26	0	0	[sig-api-machinery] CustomResourcePublishOpenAPI [Privileged:ClusterAdmin] work
test/extended/util	2	go	7	16	68	120	NA	1	16	61	123	161	0	0	[sig-auth][Feature:ProjectAPI] TestScopedProjectAccess should succeed [Suite:oper
test/extended/util	2	go	7	16	68	120	NA	1	2	10	18	27	0	0	[sig-api-machinery] CustomResourcePublishOpenAPI [Privileged:ClusterAdmin] upda
test/extended/util	2	go	7	16	68	120	NA	1	1	19	40	52	0	0	[sig-auth][Feature:ProjectAPI] TestProjectWatch should succeed [Suite:openshift/cor
test/extended/util	2	go	7	16	68	120	NA	1	2	8	18	27	0	0	[sig-api-machinery] CustomResourcePublishOpenAPI [Privileged:ClusterAdmin] work
test/extended/util	2	go	7	16	68	120	NA	1	0	1	7	11	0	0	[sig-builds][Feature:Builds] custom build with buildah being created from new-build s
test/extended/util	2	go	7	16	68	120	NA	1	17	53	97	131	0	0	[sig-cli] oc adm new-project [Suite:openshift/conformance/parallel]
test/extended/util	2	go	7	20	82	150	NA	1	6	11	17	26	0	0	[sig-api-machinery] CustomResourcePublishOpenAPI [Privileged:ClusterAdmin] work
test/extended/util	2	go	7	20	82	150	NA	1	16	61	123	161	0	0	[sig-auth][Feature:ProjectAPI] TestScopedProjectAccess should succeed [Suite:oper
test/extended/util	2	go	7	20	82	150	NA	1	2	10	18	27	0	0	[sig-api-machinery] CustomResourcePublishOpenAPI [Privileged:ClusterAdmin] upda
test/extended/util	2	go	7	20	82	150	NA	1	1	19	40	52	0	0	[sig-auth][Feature:ProjectAPI] TestProjectWatch should succeed [Suite:openshift/cor
test/extended/util	2	go	7	20	82	150	NA	1	2	8	18	27	0	0	[sig-api-machinery] CustomResourcePublishOpenAPI [Privileged:ClusterAdmin] work
test/extended/util	2	go	7	20	82	150	NA	1	0	1	7	11	0	0	[sig-builds][Feature:Builds] custom build with buildah being created from new-build s
test/extended/util	2	go	7	20	82	150	NA	1	17	53	97	131	0	0	[sig-cli] oc adm new-project [Suite:openshift/conformance/parallel]
test/extended/cli/	2	go	3	0	0	0	NA	1	0	0	1	1	0	0	[sig-cli] oc annotate pod [Suite:openshift/conformance/parallel]
test/extended/cli/	2	go	3	0	0	0	NA	1	3	14	26	26	0	0	[sig-cli] oc builds new-build [Skipped:Disconnected] [Suite:openshift/conformance/pa
test/extended/cli/	2	go	3	0	0	0	NA	1	0	0	1	1	0	0	[sig-cli] oc label pod [Suite:openshift/conformance/parallel]
test/extended/util	2	go	3	16	68	120	NA	1	0	0	1	1	0	0	[sig-cli] oc annotate pod [Suite:openshift/conformance/parallel]
test/extended/util	2	go	3	16	68	120	NA	1	3	14	26	26	0	0	[sig-cli] oc builds new-build [Skipped:Disconnected] [Suite:openshift/conformance/pa

The fetching of the data has 2 parts:

- Mapping between code changes and failed tests - The code change is fetched from both openshift’s CI system as well as Github. The failed tests data is fetched from openshift’s CI system and is the “heavy lifting” part of the entire data fetching code. This process takes 30-40 for a set of ~4000 records of CI (20 per batch).
  The part that takes the most time by far is the mapping between each test’s locator and the respective file\s location in the openshift Github repo.

- Fetching files changes history - This is fetched from Github. The only bottleneck is the limit Github has in successive API requests rate (5k per hour for a “regular” user like we have). There are just over 18k files in the repo so this takes 3.5-4 hours.

For further explanation on this, please see the project’s description in the README file on the Github repo

10. User Documentation/Project delivery

Per instructions from Ilya (the mentor from redhat) - the code is to be in Github with proper documentation.
Our repo is:
https://github.com/Amannor/redhat_final_proj/

10.1. User Manuals

NA