Journal of Computer Science

Original Research Paper

Measuring Test Data Uniformity in Acceptance Tests for the FitNesse and Gherkin Notations

Douglas Hiura Longo, Patrícia Vilain and Lucas Pereira da Silva

Department of Informatics and Statistics, Federal University of Santa Catarina, Florianópolis, Brazil

Article history
Received: 23-11-2020
Revised: 24-02-2021
Accepted: 25-02-2021

Corresponding Author:
Douglas Hiura Longo
Department of Informatics and Statistics, Federal University of Santa Catarina, Florianópolis, Brazil
Email: douglashiura@gmail.com

Abstract: This paper presents two metrics designed to measure the data uniformity of acceptance tests in FitNesse and Gherkin notations. The objective is to measure the data uniformity of acceptance tests in order to identify projects with lots of random and meaningless data. Random data in acceptance tests hinder communication between stakeholders and increase the volume of glue code. The main contribution of this paper is the implementation of the proposed metrics. This paper also evaluates the uniformity of test data from several FitNesse and Gherkin projects found on GitHub, as a means to verify if the metrics are applicable. First, the metrics were applied to 18 FitNesse project repositories and 18 Gherkin project repositories. The measurements taken from these repositories were used to present cases of irregular and uniform test data. Then, we have compared the notations from FitNesse and Gherkin in terms of projects and features. In terms of projects, no significant difference was observed, that is, FitNesse projects have a level of uniformity similar to Gherkin projects. However, in terms of features and test documents, there was a significant difference. The uniformity scores of FitNesse and Gherkin features are 0.16 and 0.26, respectively. These uniformity scores are very low, which means that test data for both notations are very irregular. Thus, we can infer that test data are more irregular in FitNesse features than in Gherkin features. The evaluation also shows that 28 of 36 projects (78%) did not reach the minimum recommended measure, i.e., 0.45 of test data uniformity. In general, we can observe that there are still many challenges in improving the quality of acceptance tests, especially in relation to the uniformity of test data.

Keywords: Software Testing, Acceptance Test, Agile Software Development, Uniformity, Metric, Gherkin, FitNesse, Glue Code, Automated Tests, Cucumber

Introduction

Analogous to Test-Driven Development (TDD) (Beck, 2003), Acceptance Test-Driven Development (ATDD) includes different stakeholders (client, developer, tester) who collaborate to write acceptance tests before implementing system functionality (Gärtner, 2012). Teams involved with ATDD generally find that, only by defining acceptance tests and discussing test specifications, there will be a better understanding of the requirements. This happens because acceptance tests tend to force the need for a solid agreement on the exact behavior that is expected from a software (Hendrickson, 2008). According to Santos, (Longo and Vilain, 2018), there are 21 techniques used to specify acceptance tests. Acceptance tests specifications can be done by using semi-structured formats, tables, diagrams, or other Domain-Specific Languages (DSLs).

It is estimated that 85% of software defects originate from ambiguous, incomplete and illusory requirements (Torchiano *et al*., 2007). Specifying software requirements using acceptance tests is an attempt to improve the quality of requirements. However, several problems can arise from the specification of requirements using acceptance tests, as it also happens with the specification of requirements using natural language. For example, by using natural language, readers and writers may use the same word to name different concepts, or even express the same concept in completely different ways (Sommerville, 2011).

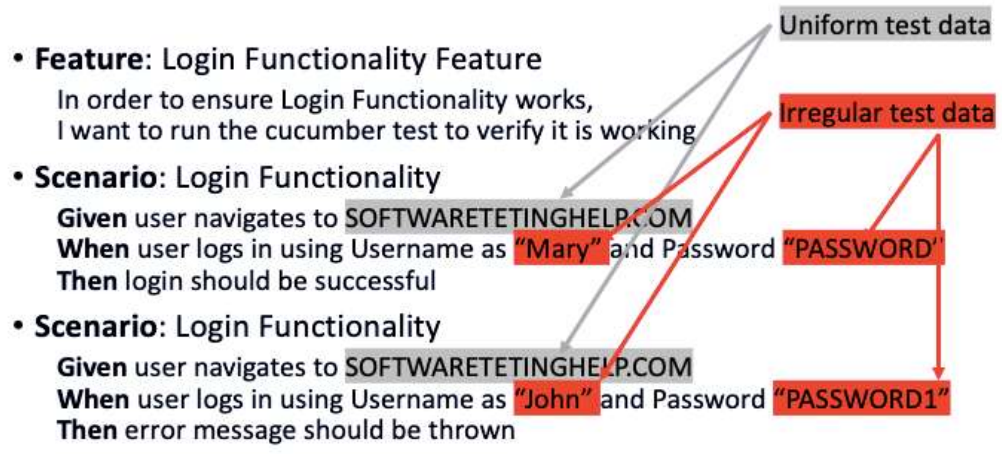
Most notations of acceptance tests are composed of functional data and test data (Druk and Kropp, 2013). Functional data is an artifact that is used to connect test data to the System Under Test (SUT). The connection is done through glue code, which must follow the template of the framework that is being used to execute the tests. Test data are used to set up the SUT and the output data expected from the SUT. Test data are represented by words or expressions, usually with a flag to differentiate it from functional data. Longo and Vilain (2018; Longo *et al*., 2019) define that the test data can be either uniform or irregular. Uniform test data are expressions that are common to various test documents and irregular test data are composed by single expressions that are not repeated through test documents.

Figure 1 shows an example of an acceptance test in Gherkin notation with uniform and irregular test data. This acceptance test in the Gherkin notation deals with the login functionality feature and has two scenarios. The uniformity and irregularity of test data can be verified by comparing the test data from the two scenarios. For example, the test data value 'SOFTWARETETINGHELP.COM' appears in both scenarios and is considered, therefore, uniform. Nevertheless, the test data values 'Mary' and 'John' and 'PASSWORD' and 'PASSWORD1' are considered irregular because they are not repeated in both scenarios.

In general, several features are specified for the development of an application in which acceptance tests are included. These features are usually organized into separate documents, they can be specified at various times throughout the development process and their specifications can be done by different people. In this way, maintaining the uniformity of test data can be a challenge because there can be a lot of test data that are expressed in completely different ways but with the same meaning. For those involved in specifying a test, communication between tests using irregular data may be feasible, with little or no information loss, since humans are able to interpret the irregularities of test data and understand the meaning of the test. However, there may be problems associated with unintentionally irregular data and glue code reuse for test automation (Longo *et al*., 2019). For example, in order to automate the scenarios of the login functionality feature (Fig. 1), some settings of the SUT are required. It is necessary to set up 'Mary' for the first scenario and 'John' for the second scenario. Nonetheless, the setup could be more easily reused if test data were uniform. According to Greiler *et al*. (2013a), test code duplication should be avoided by code reuse. Code reuse can facilitate maintenance activities, as a smaller volume of code is easier to handle.

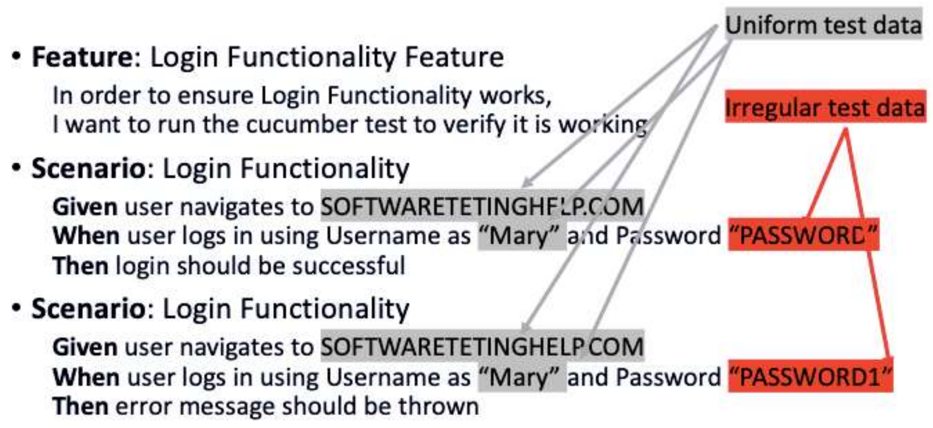
Example of Improving Data Uniformity and Decreasing Glue Code

Figure 2 shows an improvement in the uniformity of the test data presented in Fig. 1. The improvement refers to the uniformization of usernames 'Mary' and 'John'. In the example in Fig. 1, the test can be understood and performed with unformalized test data. More uniform test data can cause a reduction in the fixture settings in the glue code. Figure 3 shows the glue code for the examples in Fig. 1 and 2. In the glue code example, lines 6 and 7 are highlighted because they can be discarded for the example with the most uniform data (Fig. 1).


Uniform test data
Irregular test data

- **Feature**: Login Functionality Feature
  In order to ensure Login Functionality works,
  I want to run the cucumber test to verify it is working
- **Scenario**: Login Functionality
  Given user navigates to SOFTWARETETINGHELP.COM
  When user logs in using Username as "Mary" and Password "PASSWORD"
  Then login should be successful
- **Scenario**: Login Functionality
  Given user navigates to SOFTWARETETINGHELP.COM
  When user logs in using Username as "John" and Password "PASSWORD1"
  Then error message should be thrown

Fig. 1: Sample of acceptance test in Gherkin with uniform and irregular test data (adaptation from Softwaretestinghelp, 2020)



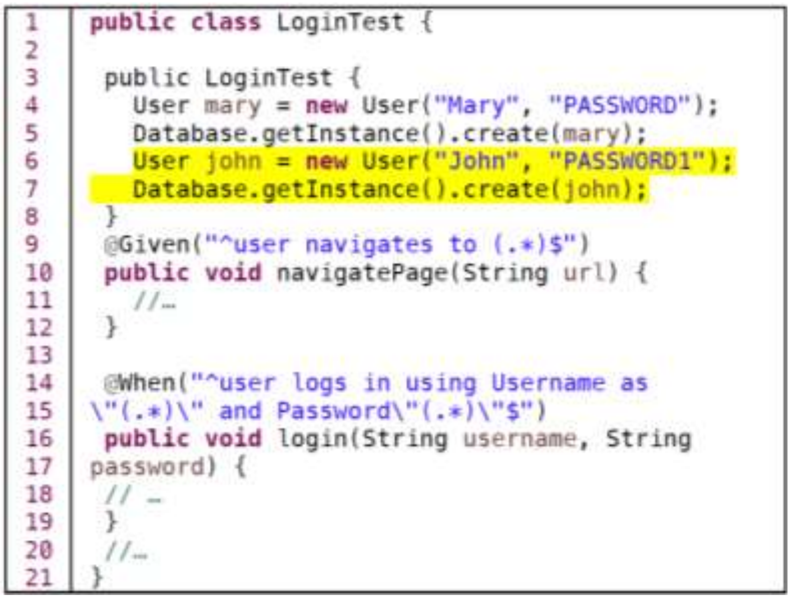
- **Feature**: Login Functionality Feature

  In order to ensure Login Functionality works,
  I want to run the cucumber test to verify it is working
- **Scenario**: Login Functionality

  Given user navigates to SOFTWARETETINGHELP.COM
  When user logs in using Username as "Mary" and Password "PASSWORD"
  Then login should be successful
- **Scenario**: Login Functionality

  Given user navigates to SOFTWARETETINGHELP.COM
  When user logs in using Username as "Mary" and Password "PASSWORD1"
  Then error message should be thrown

Fig. 2: The acceptance test example from Fig. 1 with improved data uniformity (adaptation from Softwaretestinghelp, 2020)



```
public class LoginTest {

  public LoginTest {
    User mary = new User("Mary", "PASSWORD");
    Database.getInstance().create(mary);
    User john = new User("John", "PASSWORD1");
    Database.getInstance().create(john);
  }
  @Given("^user navigates to (.*)$")
  public void navigatePage(String url) {
    //…
  }

  @When("^user logs in using Username as
\"(.*)\" and Password\"(.*)\"$")
  public void login(String username, String
password) {
  // …
  }
  //…
}
```

Fig. 3: Glue code for the tests from Fig. 1 and 2 with highlighting of lines neglected by the better data uniformity (adaptation from Softwaretestinghelp, 2020)

This lower volume of glue code achieved by more uniform test data means less development and maintenance effort. Still, some gain in communication can be obtained, because if we compare the test example in Fig. 2 and the test example in Fig. 1, we can observe that the more uniform test data clarifies the meaning of the scenarios in Fig. 1: The first scenario with a successful authentication system and the second with an incorrect password failure. In the test example in Fig. 2, this communication by test is vague and cannot be perceived through the test data. However, it is worth mentioning that irregular test data is important in certain occasions. For example, the fact that test data 'PASSWORD' and 'PASSWORD1' are irregular helps to understand the difference between the first and second scenarios. Hence, typically, irregular test data should be avoided, unless there is a strong semantic motivation to distinguish one test data from another.

Acceptance test in FitNesse and Gherkin notations are widely adopted according to (Park and Maurer, 2008; Dos Santos *et al*., 2018; Coutinho *et al*., 2019). Thus, in this study we propose specific metrics to be used for these notations. These metrics are based on the one proposed to the User Scenarios through User Interaction Diagram (USUID notation) (Longo and Vilain, 2018). We also evaluate the uniformity of the acceptance test data of several projects that use these notations and present a comparison of the uniformity between FitNesse and Gherkin.

The evaluated acceptance tests were collected from GitHub, a platform that hosts millions of open-source projects. Thirty-six projects from GitHub were extracted, 18 projects using FitNesse notation and 18 projects using Gherkin notation. For each project, we collect data uniformity measures and descriptive measures as well. Then, from the collected measures for each project, we compare FitNesse and Gherkin notations in order to investigate if there is a difference between the uniformity of these notations.

This article is organized as follows: Section two presents some related works. Sections three and four present the metrics proposed for the FitNesse and Gherkin notations. Sections five and six present two case studies showing the applicability of the proposed metrics. Section seven presents a comparison between these case studies. Section eight presents potential threats to the validity. Section nine presents the paper's discussions and conclusions.

Related Works

The two main related works are (Longo and Vilain, 2018; Longo *et al.*, 2019). Longo and Vilain (2018) propose a kind of metric for measuring data uniformity in automated acceptance tests in the notation of User Scenarios through User Interaction Diagrams (US-UIDs). Longo *et al.* (2019), the authors elaborate an experiment with the treatment of data uniformity as the control factor. The conclusions were that with the treatment of data uniformity, both the required volume of glue code and the time spent to automate the tests were reduced.

Some studies that are focused on acceptance tests investigate whether non-technical individuals could write executable specifications based on notations like FitNesse, US-UIDs and Gherkin (Melnik and Maurer, 2005; Alvestad, 2007; Longo and Vilain, 2015a; 2015b; Dos Santos and Vilain, 2018). Other studies focused exclusively on FitNesse notation were conducted by (Ricca *et al.*, 2008; 2009). Most of these studies use qualitative measures as expressed in (IEEE Std 830, 1998) or quantitative measures such as time, for evaluation or comparison. Metrics that are more accurate for evaluating user stories were proposed by (Lucassen *et al.*, 2015; 2016) and applied by (Lucassen *et al.*, 2017). However, these kinds of metrics are specific to the user story format and have not been adapted for automated acceptance test notations.

Other studies, such as (Greiler *et al.*, 2013a; 2013b), focus on problems in automated tests known as bad code smells. The solution to identifying bad code smells is usually the generation of a report with a set of specific measures. With the help of these reports, programmers can make balanced decisions and refactor test code in order to avoid bad smells.

These previous studies have focused on general metrics looking to evaluate and compare automated acceptance tests, as well as to identify test problems. Yet, none of them have proposed metrics for data uniformity that are specific to FitNesse and Gherkin notations. In other words, to the best of our knowledge, no other objective metrics to assess the uniformity of acceptance test data has been found, other than (Longo and Vilain, 2018; Longo *et al.*, 2019), especially for FitNesse and Gherkin techniques. In addition, there is lack of basic studies comparing the different notations of automated acceptance tests that consider a large volume of projects.

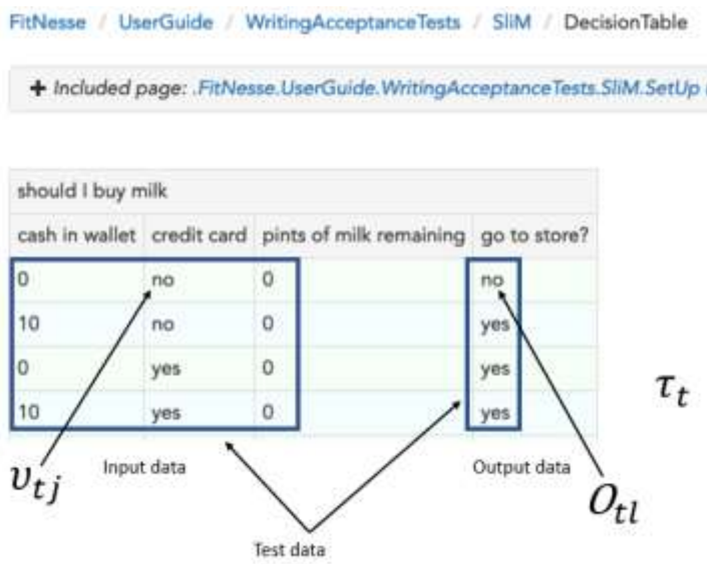
Metrics of Data Uniformity for FitNesse

As mentioned before, this paper proposes two kinds of metrics for measuring the uniformity of acceptance test data for the FitNesse and Gherkin notations, respectively. For the FitNesse notation, in general, the metrics are applied to a set of wiki pages that represents acceptance tests. A set of pair-wise wiki pages is generated from the page set. The uniformity values for each pair of wiki pages are calculated by counting the number of uniform test data that are common to both wiki pages and the number of test data that are only presented in one of them. Finally, general uniformity is the average of uniformity of the pairs in the set.

This section presents the metrics for measuring uniformity of tests in FitNesse notation. The proposed metrics are based on the work of (Longo and Vilain, 2018) in which a metric for calculating uniformity of tests using the US-UID notation is proposed. We present the metric to calculate the data uniformity for Fitnesse through a math model. By applying this math model we can calculate the data uniformity of each pair of Fitnesse feature and the data uniformity of entire project as well. The Table 1 shows the related works.

Table 1: Related works

Reference	Requirement format	Applied technique	Quality criteria assessed
Greiler *et al.* (2013a; 2013b)	Unit test (Java)	Metricsbased framework	Bad code smells
Lucassen *et al.* (2015; 2016; 2017)	User story	Metricsbased framework	Atomic, Minimum, well formed, uniform and etc.
Longo and Vilain (2018)	US-UIDs	Metric	Uniformity
Longo *et al.* (2019)	US-UIDs	Metric	Uniformity and Glue Code
This paper	FitNesse and Gherkin	Metric	Uniformity



FitNesse / UserGuide / WritingAcceptanceTests / SliM / DecisionTable

✚ *Included page: .FitNesse.UserGuide.WritingAcceptanceTests.SliM.SetUp (edit)*

should I buy milk			
cash in wallet	credit card	pints of milk remaining	go to store?
0	no	0	no
10	no	0	yes
0	yes	0	yes
10	yes	0	yes

Fig. 4: Example of a FitNesse feature with input and output data (adapted from FitNesse, 2019)

Metric Input

Metric input is a set of FitNesse features. A FitNesse feature is a wiki page with tests. This set of features is used to measure data uniformity. Figure 4 presents an example of a feature and the indication of input and output elements of the test data that will be used as input for the proposed metrics. The set of features that will be the input of the metrics is represented by the following equation:

$$\omega = \{\tau_1, \tau_2, \tau_t, \ldots, \tau_d\}\left(\forall t\left(t = 1; d\right)\right) \wedge d > 1 \tag{1}$$

Where:

ω	=	A set of features
τ_t	=	The *t*-eth feature of set ω and must be denoted according to Eq. 2
d	=	The number of features within set ω

FitNesse Feature

Test data of the features are organized into tables, as seen on Fig. 4. The table caption (i.e., "should I buy milk") and column headers (e.g., "cash in wallet" and "go to store?") consists of functional data. The proposed metric does not use functional data. The input and output data are shown in the body of the table. Thus, a feature is represented as follows:

$$\tau_t = \begin{Bmatrix} \upsilon_{t1}, \upsilon_{t2}, \upsilon_{tj}, \ldots, \upsilon_{tn} \\ o_{t1}, o_{t2}, o_{tj}, \ldots, o_{tm} \end{Bmatrix}, \left(\forall j\left(j = 1; n\right)\right), \tag{2}$$

Where:

υ_{tj}	=	The *j*-eth input data of the *t*-eth feature of set *ω*
o_{tl}	=	The *l*-eth output data of the *t*-eth feature of set *ω*
n	=	The number of output data of τ_t
m	=	The number of input data of τ_t

Feature Pairs Generation

A set of pairwise combination of features is generated from the input ω. The pairs in this set are used later for calculating the metrics. The set of pairs is denoted as follows:

$$\psi = \begin{Bmatrix} (\tau_1, \tau_2), (\tau_1, \tau_3), \ldots, (\tau_t, \tau_{t'}), \ldots, \\ (\tau_{(d-1)}, \tau_d), \ldots, (\tau_d, \tau_{(d-1)}) \end{Bmatrix}, \tag{3}$$
$$\left(\forall t\left(t = 1; d\right)\right), \left(\forall q\left(t' = 1; d\right)\right), t \neq t'$$

Where:

ψ	=	The set of feature pairs generated from ω
$(\tau_t, \tau_{t'})$	=	A pair of features generated from different features that belong to ω
$\tau_{t'}$	=	The *t*′-eth feature that belongs to set *ω*. The variable *t*′ assumes the same values as *t*, that is, $(\forall q(t' = 1; d))$

$t \neq t'$ determines that the pair must consist of different *t*-eth and *t*′-eth features. For each pair $(\tau_t, \tau_{t'})$, auxiliary metrics of absolute uniformities is obtained. Auxiliary metrics of absolute uniformities are obtained by counting uniform and irregular data. In this way, these auxiliary measurements are used for the creation of the metric for relative uniformity, which is applied to each pair of features. The goal of the relative uniformity metric is to obtain a uniformity value that can be applied in the comparison between different pairs of features. The pairs of features will be used in the metrics of relative and absolute uniformity and that is why they were defined before the metrics.

Auxiliary Metrics of Absolute Uniformities

The metrics of absolute uniformity are sectioned by input and output data. The metric for absolute uniformity of input data is represented by the following equation:

$$UniformInput_{(\tau_t,\tau_{t'})} = \sum_{j=1}^{n} \begin{matrix} 1 if\ \upsilon_{tj} \in \tau_{t'} \\ 0 otherwise \end{matrix}, \quad \left(\forall \upsilon_{tj} \mid \upsilon_{tj} \in \tau_t\right) \tag{4}$$

The metric for absolute uniformity of the input data is the sum of the input data from the τ_t test page that also belongs to $\tau_{t'}$. The expression $\upsilon_{tj} \in \tau_{t'}$ means that input data υ_{tj} which belongs to τ_t, also belongs to $\tau_{t'}$.

The metric for absolute uniformity of output data is represented by the following equation:

$$UniformOutput_{(\tau_t,\tau_{t'})} = \sum_{l=1}^{m} \begin{matrix} 1 if\ o_{tl} \in \tau_{t'} \\ 0 otherwise \end{matrix}, \quad \left(\forall o_{tl} \mid o_{tl} \in \tau_t\right) \tag{5}$$

The metric for absolute uniformity of the output data is the sum of all outputs of τ_t that also belong to $\tau_{t'}$. The expression $o_{tl} \in \tau_{t'}$ means that the output data of the test o_{tl}, which belongs to τ_t also belongs to $\tau_{t'}$.

Relative Uniformity Metric

The relative uniformity metric is defined from the auxiliary metrics of absolute uniformity. Its goal is to assign a numerical value to the uniformity of the data. The metric is applied to a pair of features and the result is the ratio between the sum of the absolute uniformity metrics and the amount of input and output data for a pair of features. Thus, the relative uniformity metric for a given pair of features is represented by the following equation:

$$Unifomity_{(\tau_t,\tau_{t'})} = \frac{UniformInput_{(\tau_t,\tau_{t'})} + UniformOutput_{(\tau_t,\tau_{t'})}}{n+m}, \quad (n+m) > 0, \left(\forall n \mid n \in \tau_t\right), \left(\forall m \mid m \in \tau_t\right) \tag{6}$$

The relative uniformity metric is a value within the [0, 1] interval (zero to one interval). Relative uniformity metric always assumes values in the 0 to 1 interval, regardless of the number of inputs and outputs contained in the features and for this reason, it is called relative uniformity. Value 1 (one) represents the maximum uniformity and value 0 (zero) represents the maximum irregularity. If $(n + m) = 0$, then the uniformity value is 1, i.e., in the case that there is no test data on the feature, then a uniformity value of 1 is adopted. The main goal of relative uniformity is to create a normalized scale that enables the comparison between distinct pairs of features.

The relative uniformity metric is calculated for each pair of features, so the arithmetic mean between them can be adopted as the descriptive measure of uniformity for all pairs of a project. Thus, from the relative uniformity metric for feature pairs, we propose the relative uniformity metric for the entire project. The relative uniformity metric for a project is the sum of the uniformity metric values of each pair of features divided by the total number of pairs. The relative uniformity metric for a project is obtained from the following equation:

$$\overline{UniformInput_{(\tau_t,\tau_{t'})}} = \frac{1}{|\psi|} \sum Unifomity_{(\tau_t,\tau_{t'})}, \quad \left(\forall (\tau_t,\tau_{t'}) \mid (\tau_t,\tau_{t'}) \in \psi\right) \tag{7}$$

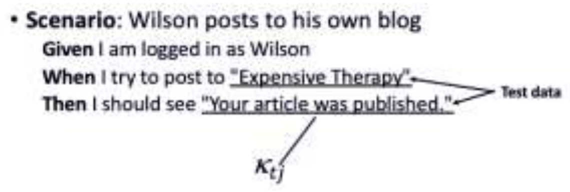
Metric Implementation

The implementation[1] of the metrics is performed with the FitNesse tool, making it possible to extract the uniformity measures by running the FitNesse tool. The entire code for calculating data uniformity was written in the Java programming language. In the source code of the FitNesse tool, the class *fitnesse.testsystems. TestSystemListener* allows the interception and monitoring of the execution of tests. Thus, this class was used to obtain input and output data of the tests. After obtaining the test data, the proposed metrics are applied and the uniformity measures of each FitNesse project is extracted. In order to obtain the test input and output data, it is necessary to run the tests using FitNesse. However, there can be some computational costs for the processing of the tests.

Metrics of Data Uniformity for Gherkin

Data uniformity metrics for the Gherkin notation is similar to the uniformity metrics for the FitNesse notation, except that there is no classification of input data and output data in the Gherkin notation. Input and output data in the Gherkin notation are only classified by the developer according to the meaning of the test information. Thus, Gherkin itself does not distinguish between one type of test data from another, i.e., for Gherkin, everything is just test data. Figure 5 shows an example of a feature in Gherkin notation with a highlight in the test data.

[1] https://github.com/douglashiura/fitnesse-uniformity-data.git



• **Scenario:** Wilson posts to his own blog
Given I am logged in as Wilson
When I try to post to "Expensive Therapy"
Then I should see "Your article was published."

Fig. 5: Example of a Gherkin feature with test data (adapted from Cucumber, 2019)

Test data are encapsulated within the descriptions and they are identified by being in quotes, by its formatting or by the developer's understanding upon reading the test. An example of test data, in Fig. 5, is the expression "Expensive Therapy". This test data appears along with the text describing the keyword When and it is enclosed in quotation marks, which identify it as a test data.

We present the metric to calculate the data uniformity for Gherkin through a math model. By applying this math model we can calculate the data uniformity of each pair of Gherkin feature and the data uniformity of entire project as well. The metrics can be applied to a set of features with acceptance tests. Then, a set of pairwise feature combinations is generated from the set of features. The uniformity of each feature pair is calculated by counting the uniform and irregular data points and applying them to an equation. The equation is the ratio between irregular test data and the total amount of test data. The total uniformity for a pair of features is the arithmetic mean between the uniformity of each feature from the pair.

Metric Input

A metric input is any set of features. Data uniformity metrics is extracted from these features. The notation for the set of input features is:

$$\chi = \{\beta_1, \beta_2, \beta_t, \ldots, \beta_d\} \left(\forall t (t = 1; d)\right) \wedge d > 1 \tag{8}$$

Where:

χ	=	Any set of features in the Gherkin notation
β_t	=	The *t*-eth feature in set and must be denoted according to Eq. 9
d	=	The number of features in χ

Gherkin Feature

A Gherkin feature is a test document and is denoted as follows:

$$\beta_t = \{\kappa_{t1}, \kappa_{t2}, \kappa_{tj}, \ldots, \kappa_{tn}\} \left(\forall j (j = 1; n)\right) \tag{9}$$

Where:

κ_{tj}	=	The *j*-eth test data of the *t*-eth feature
n	=	The number of test data in β_t

Feature Pairs Generation

A set of pairwise combination of features is generated from the ω input. The pairs in this set are used later for the metrics calculation. The set of feature pairs is denoted as follows:

$$\varphi = \left\{ \begin{matrix} (\beta_1, \beta_2), (\beta_1, \beta_3), \ldots, (\beta_t, \beta_{t'}), \ldots, \\ (\beta_{(d-1)}, \beta_d), \ldots, (\beta_d, \beta_{(d-1)}) \end{matrix} \right\}, \\ \left(\forall t (t = 1; d)\right), \\ \left(\forall q (t' = 1; d)\right), t \neq t' \tag{10}$$

Where:

φ	=	The set of feature pairs generated from χ
$(\beta_t, \beta_{t'})$	=	A pair of features generated from different features that belong to set χ
$\beta_{t'}$	=	The t'-eth feature that belongs to set χ. The variable t' assumes the same values as t, that is, $(\forall q\ (t' = 1; d))$
$t \neq t'$	=	A restriction rule, that is, a pair must be formed by distinct features.

For each pair (β_t, $\beta_{t'}$) the auxiliary metric, called metric of absolute uniformity, is obtained. Then, using the auxiliary metric, the metric for relative uniformity is formulated. The objective of the relative uniformity metric is to obtain a uniformity value that can be applied to compare different pairs of features.

Absolute Uniformity Metric

The absolute uniformity metric is represented by the following equation:

$$AbsolutUniform_{(\beta_t, \beta_{t'})} = \sum_{j=1}^{n} \begin{matrix} 1 if\ \kappa_{tj} \in \beta_{t'} \\ 0 otherwise \end{matrix}, \\ \left(\forall \kappa_{tj} \mid \kappa_{tj} \in \beta_t\right) \tag{11}$$

The absolute uniformity metric is the sum of test data from feature β_t, which also belongs to $\beta_{t'}$. The expression $\kappa_{tj} \in \beta_{t'}$ means that test data κ_{tj}, which belongs to β_t, also belongs to $\beta_{t'}$.

Relative Uniformity Metric

The relative uniformity metric for Gherkin features is elaborated from the absolute uniformity metric. The objective of the relative uniformity metric is to summarize the uniformity of the data in a numerical value. The metric corresponds to the ratio between the absolute uniformity metric and the bulk of data. Thus, the relative uniformity metric is represented by the equation:

$$Unifomity_{(\beta_t,\beta_{t'})} = \frac{AbsolutUniform_{(\beta_t,\beta_{t'})}}{n}, \quad n > 0, (\forall n \mid n \in \beta_t) \tag{12}$$

The relative uniformity metric is a value within the [0, 1] interval (zero to one interval). The relative uniformity metric always assumes values in the 0 to 1 interval, regardless of the amount of data that is contained in the test documents. Because of that, it is called relative uniformity. A metric value of 1 (one) represents the maximum uniformity and a metric value of 0 (zero) represents the maximum irregularity. If $n = 0$, the uniformity value is 1. If there is no test data on the feature, the uniformity value 1 is adopted.

The relative uniformity metric is calculated for each pair of features, so the arithmetic mean between all values can be adopted for measuring uniformity in a project. The relative uniformity metric for a project is the sum of the uniformity metric values of each pair of features divided by the total number of pairs. Thus, the relative uniformity metric for a project can be obtained by the following equation:

$$\overline{Unifomity_{(\beta_t,\beta_{t'})}} = \frac{1}{|\varphi|}\sum Unifomity_{(\beta_t,\beta_{t'})}, \quad (\forall(\beta_t,\beta_{t'}) \mid (\beta_t,\beta_{t'}) \in \varphi) \tag{13}$$

Metric Implementation

The metric is implemented[2] in the Cucumber tool, so, it is possible to extract the uniformity metrics through computing. The entire code for calculating data uniformity was written in the Java programming language. The data for each test document is extracted with the help of the Cucumber implementation. Cucumber processes the documents and sets up the tests. Then, a listener that collects the test data is implemented in the *cucumber.runner.TesteCase* class. The metrics are applied after the test data is collected by the listener. It is necessary to run the tests in order to collect the test data.

Case Study I

The first case study investigates the uniformity of data from FitNesse projects in the GitHub repository through the application of the first proposed metrics. Figure 6 shows the general process of searching for FitNesse projects in the GitHub repository. The value inside each rectangle corresponds to the number of repositories found in each step. The process consists of four activities presented in the following subsections.

Searching FitNesse Projects on GitHub

The search for FitNesse projects was carried out on the GitHub platform. GitHub was used because it houses a wide variety of open-source projects. GitHub provides a word search function. The search was carried out with the word "FitNesse" and a filter (size > 1KB). The search result returned 577 projects, of which 274 projects were developed with the Java language, 57 projects with JavaScript and 39 projects with C#. Still, all projects add up to 12K of commits. The search and data collection on GitHub took place in February 2019.

Filtering FitNesse Projects with Relevant Tests

The 577 repositories found in the previous activity were manually filtered according to four steps:

- First step: Filter and select all projects that contain the 'FitNesseRoot' directory, '*.wiki' files, or 'content.txt' files, because such files contain acceptance tests. Therefore, 392 repositories remained
- Second step: Filter and select all projects that contain more than four features and that are not the tests of FitNesse itself. Thereby, 68 repositories remained
- Third step: Remove duplicate projects. Consequently, 22 projects remained
- Fourth step: Filter and select only executable projects. As 4 projects were removed, because FitNesse (release 20180127) did not execute them properly, 18 projects remained. Table 2 presents the 18 selected projects

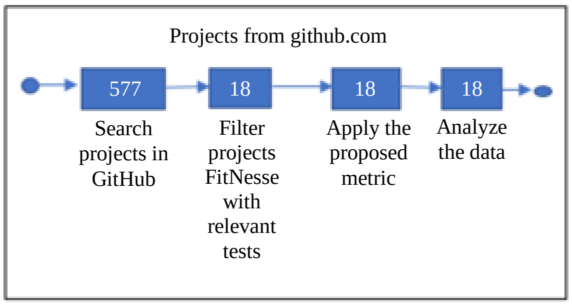

Projects from github.com
577
Search projects in GitHub
18
Filter projects FitNesse with relevant tests
18
Apply the proposed metric
18
Analyze the data

Fig. 6: General search and analysis process for FitNesse projects

Table 2: FitNesse projects selected from GitHub

Project	URL
P1	.../ManishDua90/DemoQASiteAutomation.git
P2	.../andrealbinop/fitnesse-selenium-slim.git
P3	https://...com/HeIsIdeus/CarRental_fitnesse.git
P4	…github.com/ernanics/bankapp-FitNesse.git
P5	https://github.com/yarec/fitnis.git
P6	https://github.com/Suptzs/OnlineLottery.git
P7	…github.com/ohardeng/fitnesseworkshop.git
P8	https://...com/bzon/PetClinicFitnesseSamples.git
P9	...fredericmarchand/AgricolaFitNesseTesting.git
P10	…github.com/lsu-ub-uu/diva-cora-fitnesse.git
P11	..github.com/RobteSpenke/FirstFitnesse.git
P12	https://github.com/8thlight/cob_spec.git
P13	https://github.com/xebia/Xebium.git
P14	.../PennAssuranceSoftware/inspro-fitnesse-tests.git
P15	https://...com/paytonrules/HuntTheWumpus.git
P16	…/pwojtkow/fitnesseTestsWithOracleExpress.git
P17	…github/maikelgithub/WorkshopCTFitNesse.git
P18	...github.com/lsu-ub-uu/cora-fitnesse.git

[2]https://github.com/douglashiura/cucumber-data-uniformity.git

Application of the Proposed Metrics and Data Analysis

The application of the proposed metrics was performed using the FitNesse framework and the implementation of the proposed metrics. The projects were executed and the uniformity metrics, number of features and number of test data per feature were collected. The collected data were presented in graphic charts. The charts and analyses are presented in the next subsection.

Results of the FitNesse Projects

Descriptive Measures of the FitNesse Projects

Descriptive measures consist of general information about the projects. Figure 8 shows the number of test data for each project. Project P1 is the smallest project and consists of only 14 test data (input and output data). Project P18 is the largest project and consists of 12707 test data. Projects are ranked by the number of test data, from the smallest to the largest amount.

Figure 9 presents the descriptive metrics of FitNesse projects. Descriptive metrics are defined as the average of the test data per feature (input and output data) and the total number of features for each project. Regarding the average test data per feature, project P1 has an average of 2.8 test data per feature and is composed of only five features, thus being the smallest project. Project P18 has an average of 186.87 test data per feature and is composed of 68 features, so, it has the highest average of test data. Project P16 consists of 160 features, which is the largest one in number of features.

There is a wide variety in the number of features per project, as the projects differ a lot in terms of domain, number of people involved and total commits. For example, project P18 has 4 contributors and 469 commits, while project P1 has only one contributor and 6 commits. Thus, the average number of test data per feature may also vary significantly. As an example, project P16 has many features (160) and a small number for the average of test data per feature (9.01), when compared to P18 which has less features (68) and a high number for the average of test data per feature (186,87). This means that the size (number of words written in the document) of the features in project P18 is larger than the one in project P16.

In addition, eight projects (44% of total) have between 10 and 41 test data per feature. Project P18 has an outlying average number of test data per feature and this indicates that having this amount of data in features is not a common thing. Each project has an average of 34 test data per feature and a median of 22 test data per feature. Regarding the total number of features for each project, nine projects (50% of the total projects) have between 9 and 36 features. Each project has an average of 31 features and a median of 16 features. Projects P11, P16 and P18 have distinct amounts of features, that is, they have many more features than the other projects.

Data Uniformity on FitNesse Projects

Average data uniformity was measured by the metric proposed in Eq. 7. Figure 10 shows the average data uniformity for each project. Project P2 has the lowest uniformity rate (0.03) and project P10, the highest one (0.73). Only four projects (22% of total) show data uniformity rates above 0.5. Five projects (28%) are less than 0.1 data uniformity rates, that is, extremely low compared to the recommended value (Longo and Vilain, 2018). The average uniformity rate is 0.31 and the median uniformity rate is 0.27.

Informal Assessment

In order to point out evidence that the use of the proposed metric helps to measure the uniformity of pairs of features with irregular or uniform test data in FitNesse projects, an informal assessment was carried out. The evaluation considers some pairs of features from three projects (P5, P6 and P18). The three projects were selected at random. Table 3 shows the result of this informal assessment. The objective was to find out if the measured value of the feature pair, using the proposed metric, is related to an informal assessment that intends to classify the feature pairs as irregular or uniform.

The informal assessment was carried out more easily for the P5 and P6 projects in relation to the P18 project, as the features of the P5 and P6 project are smaller in relation to the features of the P18 project. The P5 project, which has a uniformity of 0.08, showed irregularity in all three of its evaluated pairs. The P18 project, with 0.66 uniformity, showed uniformity in its three pairs, but one of its pairs (N and O) showed both uniformity and irregularity. The value 0 represented the maximum irregularity and is consistent with the metric proposal. The highest uniformity value assessed informally was a pair with 0.81. The value 0.49 with the large size of a feature indicated a gray area.

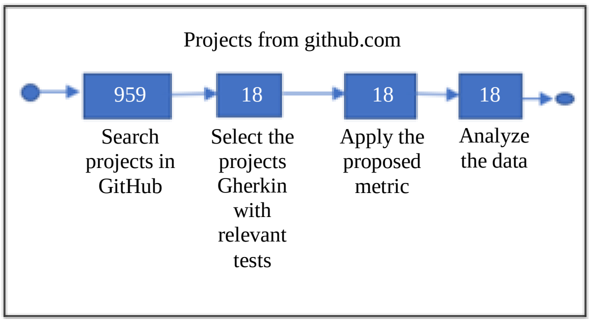

Projects from github.com
959
18
18
18
Search projects in GitHub
Select the projects Gherkin with relevant tests
Apply the proposed metric
Analyze the data

Fig. 7: General search and analysis process of Gherkin projects

Table 3: Informal assessment of data uniformity for some pairs of features

Project	Project uniformity	Pair uniformity	Checked uniformity	Pair
P5	0,08	0,00	Uniformity is very low.	A[1] and B[2]
		0,33	The features are small and have a uniform test data.	C[3] and D[4]
		0,00	The features are relatively large to be evaluated manually with precision, however they look very irregular.	E[5] and F[6]
P6	0,26	0,00	The test data is irregular and variables are used in the test instead of the test data.	G[7] and H[8]
		0,13	They don't have a lot of test data, but the data that exists that can be uniformized is already uniformized. It is important to note that there is no strong relationship between the tests.	I[9] and J[10]
		0,44	An important part of the data is uniform, which reinforces a relationship between the two tests, but the data that are irregular are numbers and are associated with complex rules (lottery system) that are difficult to understand to improve uniformity.	L[11] and M[12]
P18	0,66	0,49	The features are large, making manual evaluation difficult, but you can see that some data is repeated. The test data "OK" is used a lot. However, there are irregular data, which makes it difficult to decide between uniformity and irregularity.	N[13] and O[14]
		0,81	The feature P is small in relation to J, this reason favors the pair to be more uniform. The inverse pair (Q and P) has less uniformity (0.25), closely linked to the difference in size of the features.	P[15] and Q[16]
		0,65	The features are large, but uniformity prevails.	R[17] and S[18]

[1]https://github.com/yarec/fitnis/blob/master/ExTest/ExampleHtmlTest/HtmlTest/content.txt
[2]https://github.com/yarec/fitnis/blob/master/ExTest/ExampleHtmlTest/RestTest/content.txt
[3]https://github.com/yarec/fitnis/blob/master/ExTest/MysqlTest/ExecuteTest/content.txt
[4]https://github.com/yarec/fitnis/blob/master/ExTest/MysqlTest/ConnectTest/content.txt
[5]https://github.com/yarec/fitnis/blob/master/ExTest/ExampleProgramTest/DateUtilTest/content.txt
[6]https://github.com/yarec/fitnis/blob/master/ExTest/MysqlTest/QueryTest/content.txt
[7]https://github.com/Suptzs/OnlineLottery/blob/Development/FitNesseRoot/OnlineLottery/PlayerRegistration.wiki
[8]https://github.com/Suptzs/OnlineLottery/blob/Development/FitNesseRoot/OnlineLottery/PurchaseTicketTestSuite/BasicCase.wiki
[9]https://github.com/Suptzs/OnlineLottery/blob/Development/FitNesseRoot/OnlineLottery/PrizeCalculation.wiki
[10]https://github.com/Suptzs/OnlineLottery/blob/Development/FitNesseRoot/OnlineLottery/TicketReview/TwoAccountsOneDraw.wiki
[11]https://github.com/Suptzs/OnlineLottery/blob/Development/FitNesseRoot/OnlineLottery/TicketReview/SeveralTicketsOneDraw.wiki
[12]https://github.com/Suptzs/OnlineLottery/blob/Development/FitNesseRoot/OnlineLottery/PurchaseTicketTestSuite/NotEnoughMoney.wiki
[13]https://github.com/lsu-ub-uu/cora-fitnesse/blob/master/FitNesseRoot/TheRestTests/CallThroughJavaCode/Authorization/Rule/content.txt
[14]https://github.com/lsu-ub-uu/cora-fitnesse/blob/master/FitNesseRoot/TheRestTests/CallThroughJavaCode/Filter/Filter.wiki
[15]https://github.com/lsu-ub-uu/corafitnesse/blob/master/FitNesseRoot/TheRestTests/CallThroughRest/TheRestTest/content.txt
[16]https://github.com/lsu-ub-uu/cora-fitnesse/blob/master/FitNesseRoot/TheRestTests/CallThroughJavaCode/Search/Index.wiki
[17]https://github.com/lsu-ub-uu/cora-fitnesse/blob/master/FitNesseRoot/TheRestTests/CallThroughJavaCode/BuiltInPresentation/BasicPresentation/PresentationGroupAndContainer/content.txt
[18]https://github.com/lsu-ub-uu/cora-fitnesse/blob/master/FitNesseRoot/TheRestTests/CallThroughJavaCode/BuiltInMetadata/PreDefinedMetadata/Login/content.txt

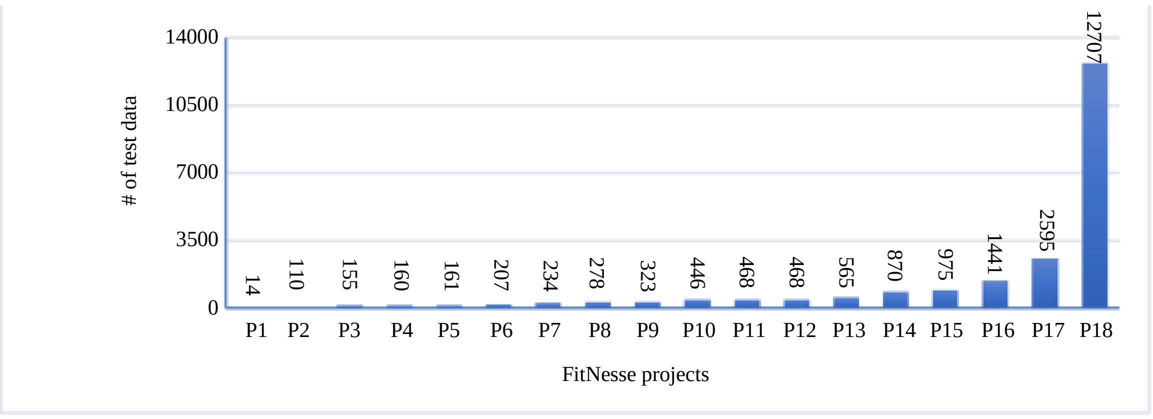


Fig. 8: Total number of test data for each project

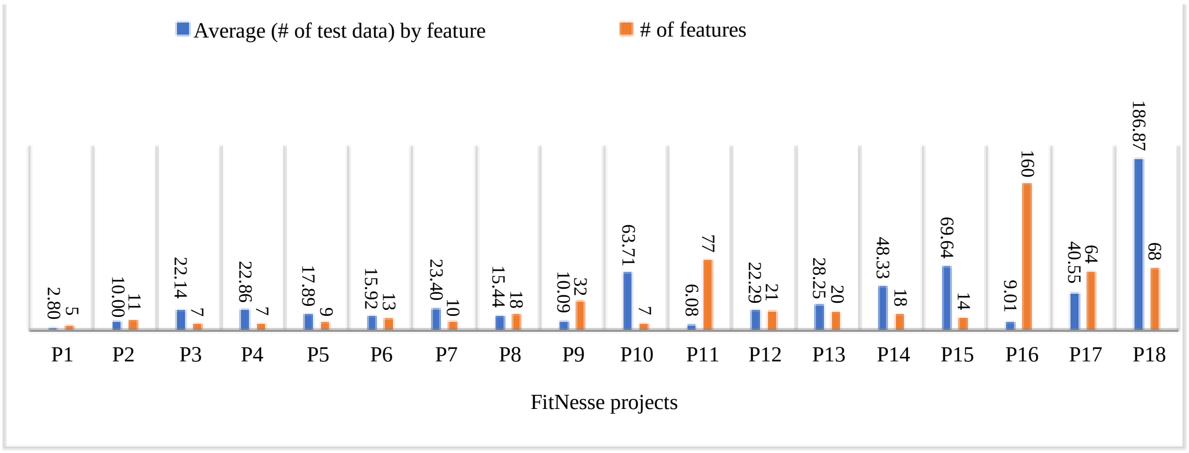


Fig. 9: Descriptive metrics of FitNesse projects

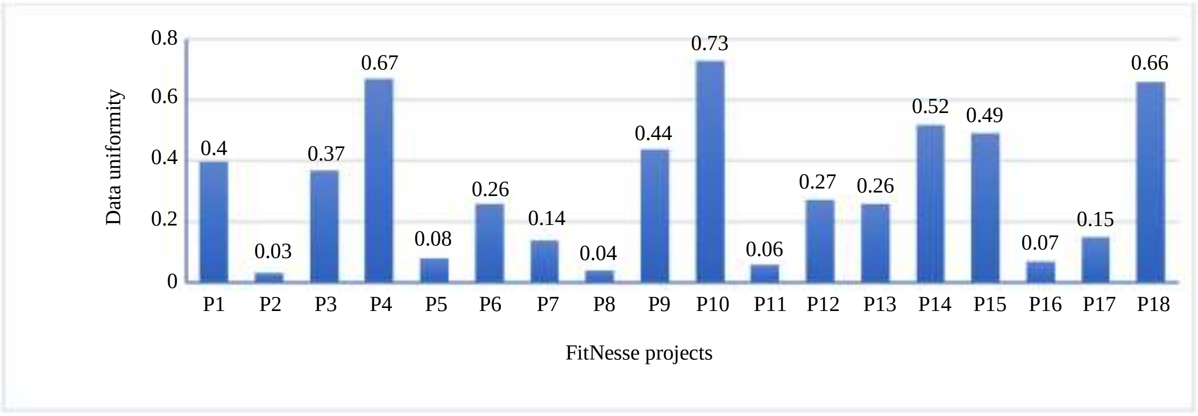


Fig. 10: Average test data uniformity rate for each project

Case Study II

Case study II investigates the uniformity of data from Gherkin notation projects in the GitHub repository. In this case study, the second proposed metrics are applied. Figure 7 shows the general process of searching for Gherkin projects in GitHub. The process consists of four activities presented in the following subsections.

Searching Gherkin Projects on GitHub

The search for Gherkin projects was carried out in GitHub. GitHub repository was selected because it houses a wide variety of projects and many of them with public access. The repository provides an advanced search function that is specific to the Gherkin language and a filter (size > 1KB), which caused the search to return only Gherkin language repositories. The search found 959 projects, ranked by "Best match". Search and data collection on GitHub were carried out in June of 2019.

Filtering Gherkin Projects with Relevant Tests

The activity of selecting Gherkin projects with tests was limited to selecting only the first 18 projects from the 959 repositories found in the previous activity, ranked by "Best match". The limit of 18 projects was applied in order to reduce the research effort requirements and to have the same number of projects that were found for the FitNesse notation. Thus, from the projects ranked by "Best match", only the first 18 repositories with more than four features were selected. In addition, it was required that the project ran properly with the Cucumber framework. Table 4 presents the 18 selected projects.

Application of the Proposed Metrics and Data Analysis

The application of the proposed metrics was carried out with the Cucumber framework and the implementation of the metrics. The projects were executed and uniformity metrics, number of total features and total test data information were collected.

The collected data was presented in graphic charts. These results, analyses and charts are presented in the next subsection.

Results of the Gherkin Projects

Descriptive Measures of the Gherkin Projects

Descriptive metrics consists of general information about the measured projects. Figure 11 shows the number of test data for each project. Project C1 is the smallest project and consists of only 10 test data. Project C18 is the largest project and consists of 6,085 test data. The projects are ranked by the amount of test data.

Figure 12 shows the descriptive metrics of the projects. These are the average test data per feature and the total of features for each project. Project C1 has an average of two test data per feature and it consists of only five features, making it the smallest project. Project C18 has an average of 132.28 test data per feature and it is composed of 46 features, thus being the highest average of test data. Project C16 is composed of 109 features, which makes it the largest one in number of features.

Table 4: Gherkin projects selected from GitHub

C1	https://.../danbuckland/crudecumber.git
C2	https:/.../CU-CommunityApps/kuality-kfs-cu.git
C3	https://github.com/bulletproofnetworks/ript.git
C4	https://../ucsf-drupal/ucsf_installprofile.git
C5	VaishnaviGunwante/selenium-cucumber-java.git
C6	https://...FreeFeed/acceptance-tests.git
C7	https://../MorkovkAs/SmokeTestsRiskGap.git
C8	https://github.com/ajspadial/canciella.git
C9	https://github.com/rejeep/ruby-tools.el.git
C10	/deepstreamIO/deepstream.io-client-specs.git
C11	ucsf-web-services/ucsf_www.ucsf.edu_tests.git
C12	/douglashiura/ucsf_www.ucsf.edu_tests.git
C13	https://git.../SoftServeUniversity/yunakquiz.git
C14	https://github.com/deformio/cli-deform.git
C15	https://github.com/jakobmattsson/locke-api.git
C16	https://../pumbaEO/vanessa-behavior-tets.git
C17	https://github.com/Vardot/varbase-behat.git
C18	https://github.com/IntersectAustralia/dc21.git

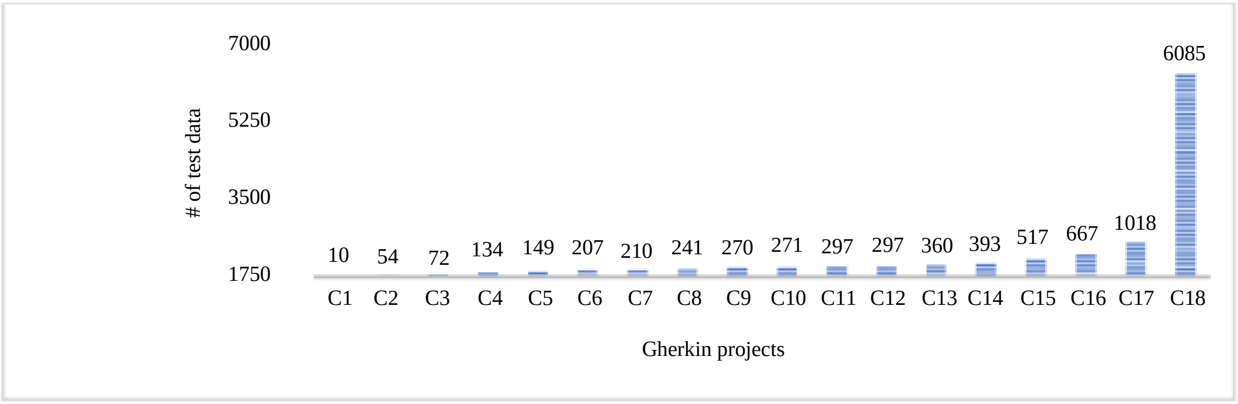


Fig. 11: Total test data for each Gherkin project

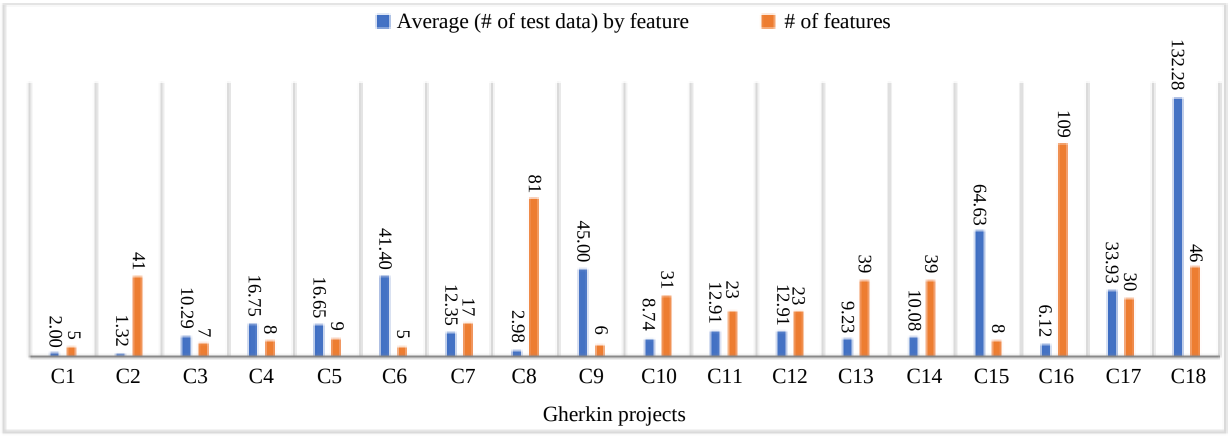


Fig. 12: Descriptive metrics of Gherkin projects

The wide diversity in the number of features per project is likely to be caused by the domain of each project, the number of people involved and the total commits. As an example, project P18 has 18 contributors and 1.642 commits while project C1 has only one contributor and 129 commits. The average number of test data per feature is significantly variable, as well. For instance, project C16 has a combination of many features (109) with a small average of test data per feature (6.12), when compared to C18. In project C18, however, this relationship is reversed, as it has an average of 132.28 test data and just 46 features. This means that, in project C16, the features are smaller (few words in the document); while in project C18, the features are bigger (many words in the document).

Data Uniformity on Gherkin Projects

The average data uniformity was measured by the metric proposed in Eq. 13. Figure 13 shows the average data uniformity for each project. Projects C3, C11 and C12 have the smallest uniformity (0.02) and project C1 has the highest uniformity (0.64). Six (33%) projects (C3, C4, C11, C12, C14 and C17) present data uniformity levels below 0.01, that is, data uniformity in these projects is very low and it was probably careless during the specification of the tests. Only two projects (C1 and C2) have the uniformity score greater than 0.5.

Informal Assessment

In order to point out evidence that the use of the metric helps to measure the uniformity of irregular or uniform feature pairs, an informal assessment was carried out. The evaluation was done on some pairs of features of three Gherkin projects (C5, C6 and C18). Table 5 presents the informal assessment of the uniformity of some pairs of features. The objective was to find out if the measured value of the feature pair is related to an informal assessment that intends to classify the feature pairs as irregular or uniform.

Table 5: Informal assessment of data uniformity for some pairs of features

Project	Project uniformity	Pair uniformity	Pair	Checked uniformity
C5	0,42	0,03	A[1] and B[2]	One feature is huge in relation to the other and therefore uniformity is low.
	0,57		C[3] and A	The first feature is small and with similar or uniform data.
		0,40	B and D[4]	One feature is small and another is large and some data is uniform. It presents a cloudy area between uniformity and irregularity.
C6	0,45	0,40	E[5] and F[6]	There is uniform data, but there is also irregular data. However, it could be more uniform if the test was less vague or avoided using expressions like "When I enter incorrect information" and as an alternative, define what the test data is for the expression "incorrect information".
		0,00	E and G[7]	Feature G is incomplete and without data.
		0,01	H[8] and E	There is some uniform data, however in both tests many scenarios are elaborated and the scenarios attempt to cover the Registration and Authentication features. The coverage is in the sense of testing a wide range of data possibilities. These coverage scenarios are different from the acceptance testing proposal.
C18	0,15	0,12	I[9] and J[10]	Both features are large and difficult to evaluate. Apparently, the uniformity is low.
		0,24	L[11] and M[12]	There is a lot of test data in the features. There is uniform data, but visually it is difficult to say that the features are uniform.
		0,00	N[13] and O[14]	The features do not have a very large size, which facilitates an evaluation. The data is very irregular, however, similar data such as "Facility" and "Facility0" are used, which could be uniformized.

[1]https://github.com/VaishnaviGunwante/selenium-cucumber-java/blob/master/target/test-classes/features/AssertSteps.feature
[2]https://github.com/VaishnaviGunwante/selenium-cucumber-java/blob/master/target/test-classes/features/ClickSteps.feature
[3]https://github.com/VaishnaviGunwante/selenium-cucumber-java/blob/master/target/test-classes/features/progressSteps.feature
[4]https://github.com/VaishnaviGunwante/selenium-cucumber-java/blob/master/target/test-classes/features/navigationSteps.feature
[5]https://github.com/FreeFeed/acceptance-tests/blob/master/features/authorization.feature
[6]https://github.com/FreeFeed/acceptance-tests/blob/master/features/Pages.feature
[7]https://github.com/FreeFeed/acceptance-tests/blob/master/features/viewfeed.feature
[8]https://github.com/FreeFeed/acceptance-tests/blob/master/features/registration.feature
[9]https://github.com/IntersectAustralia/dc21/blob/master/features/aaf_login.feature
[10]https://github.com/IntersectAustralia/dc21/blob/master/features/add_to_cart.feature
[11]https://github.com/IntersectAustralia/dc21/blob/master/features/api_package_create.feature
[12]https://github.com/IntersectAustralia/dc21/blob/master/features/publish_collection.feature
[13]https://github.com/IntersectAustralia/dc21/blob/master/features/view_data_file.feature
[14]https://github.com/IntersectAustralia/dc21/blob/master/features/api_facility_and_experiment_list.feature

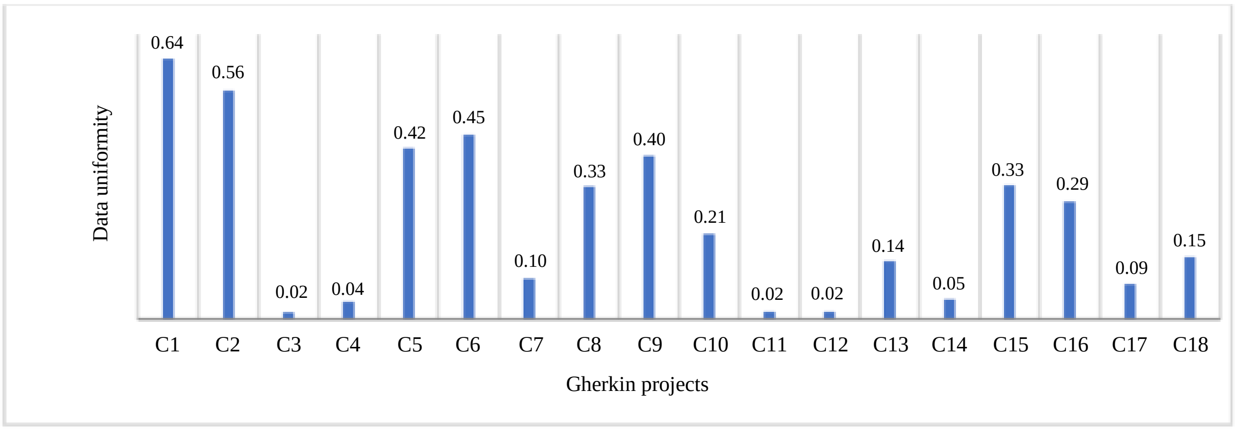


Fig. 13: Test data uniformity average for each project

The informal assessment was carried out with some difficulty, especially because the application domains of the tests are not familiar and, in some cases, because of the size of the features. When the size of the features increases, human evaluation can be impaired by limiting the mental capacity to memorize data. However, the value 0 represented the maximum irregularity and was easily perceived. The highest uniformity value assessed informally was a pair with 0.57. The value 0.40 associated with the large size of a feature indicated a nebulous zone.

FitNesse Vs. Gherkin

The application of the uniformity metrics has yielded quantitative data from both case studies. In this way, quantitative data, such as test data uniformity, can be compared in order to determine whether there are differences between notations. This comparison can be done between the descriptive metrics of the two case studies looking to identify the projects similarities. Additionally, for a deeper investigation on uniformity, the uniformity of the two notations is compared against each other to determine whether there is any relationship between descriptive metrics and uniformity. With the result of the investigation, we look to answer the following questions:

RQ1: Are there any difference between the numbers of features in the samples of the two acceptance test notations?
RQ2: Is there a difference in the average of test data per feature between the notations?
RQ3: Is there a difference in data uniformity between the two notations?
RQ4: Is there a correlation between number of features, average of test data by feature and data uniformity? That is, does the size (features and test data) of the project influence data uniformity?

These research questions are linked to the case studies and they are intended to further deepen the study by comparing the two notations. In this way, decisions that are more assertive can be done regarding the uniformity of the two notations. Boxplot diagrams and a hypothesis test are used to analyze the differences between notations in questions RQ1, RQ2 and RQ3. So, by using the boxplot diagram, we look to present data for a visual assessment and by using the hypothesis test, we aim to determine if there are any differences between the samples. The RQ4 question is analyzed using a scatter diagram of the variables number of features, average number of test data per features and data uniformity. The RQ4 question is also analyzed using a regression model.

Answering RQ1

In order to answer RQ1, regarding the number of features per project of case studies I and II, FitNesse and Gherkin, respectively, must be used. Figure 14 shows the visual comparison among the numbers of features of the projects between the two notations. The number of features per project is considered (Fig. 9 and 12), with 18 FitNesse projects and 18 Gherkin projects. FitNesse projects have a median of 16 features per project, an average of 31.16 features per project, with a standard deviation of 39.19 features. Gherkin projects have a median of 23 features per project and an average of 29.27 features per project, with a standard deviation of 28.07 features. However, the distribution of the number of features per project was analyzed via a T-Test, at a significance level of (α = 0.05) and no statistically significant difference between the number of features of the two notations (P_{value} = 0.8691) was found. Thus, regarding the number of features, it was not possible to observe statistically significant changes between the projects of each notation.

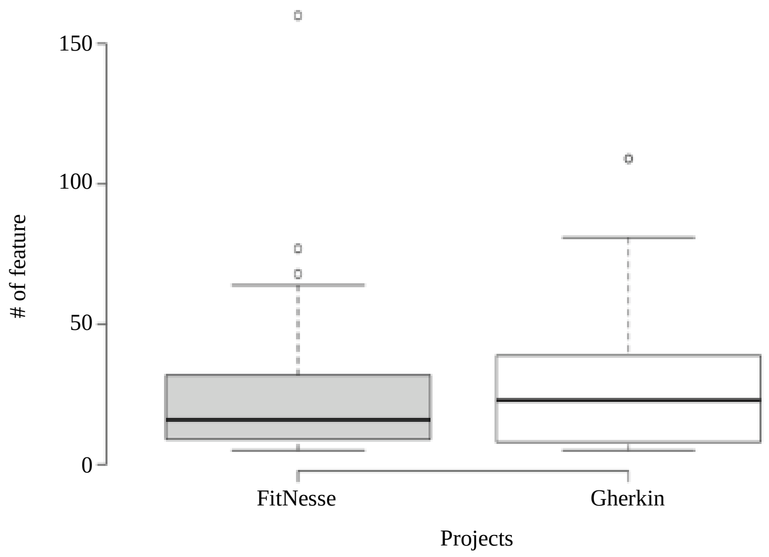


Fig. 14: Visual comparison among the number of features between notations

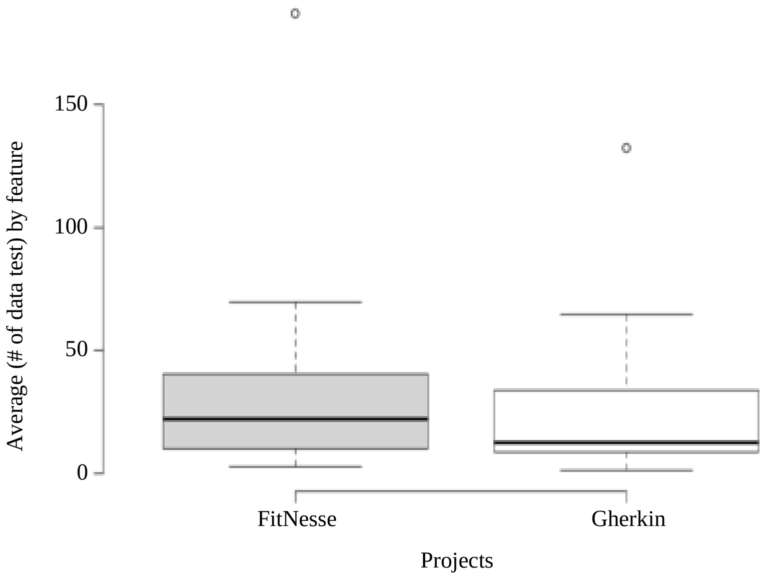


Fig. 15: Visual comparison between the number of features for each notation

Answering RQ2

In order to answer RQ2, the average number of test data per feature of each project is used, respectively, in FitNesse and Gherkin notations, from the case studies previously mentioned. Figure 15 shows the visual comparison between the average numbers of data per features for each notation.

The average number of test data per feature is considered per project (Fig. 9 and 12), with 18 FitNesse projects and 18 Gherkin projects. FitNesse notation has a median of 22.21 test data per feature, an average of 34.18 and a standard deviation of 42.54. Gherkin notation has a median of 12.63 test data per feature, an average of 24.41 and a standard deviation of 31.89. The distribution of the average number of test data per feature was analyzed via a T-Test, at a significance level of ($\alpha = 0.05$), with the result that there is no statistically significant difference in the average number of test data per feature between the two notations ($P_{value} = 0.4417$). Thus, regarding the average number of test data per feature, it was not possible to observe statistically significant changes between the projects of each notation. The conclusion reached is that the amount or number of test data in each feature is similar for both notations.

Answering RQ3

Question RQ3 can be interpreted in two ways. The first way is whether there are differences in data uniformity between the projects of each acceptance test notation. The second way is whether there are differences in data uniformity between the pairs of features of the FitNesse and Gherkin notations, regardless of the projects. In order to answer the first interpretation of the question, if there is a difference in uniformity between FitNesse and Gherkin projects, the average uniformity metrics for each project must be used (Fig. 10 and 13). Figure 16 shows the visual comparison of the average uniformity between each project of the two notations.

Figure 16 considers the uniformity of 18 FitNesse projects and 18 Gherkin projects. The FitNesse notation has a median of data uniformity of 0.26, an average of 0.31 and a standard deviation of 0.23. The Gherkin notation has a median of data uniformity of 0.18 an average of 0.24 and a standard deviation of 0.19. The uniformity of the 36 projects analyzed does not have a regular distribution. Therefore, a transformation must be applied to the data. The transformation applied was the square root. After data was transformed from the application of the square root, distribution was normalized and the variances were equal. Thus, the distribution of data uniformity per project was analyzed via a T-Test, at a significance level of ($\alpha = 0.05$), with the result that there is no statistically significant difference in data uniformity per project between the two notations ($P_{value} = 0.2961$). So, regarding data uniformity per project, it was not possible to observe statistically significant changes between notations. The conclusion reached is that test data uniformity of the projects is similar in both notations.

Longo and Vilain (2018) advocate a 0.45 uniformity value as the minimum value to be reached before the automation of the tests. As Fig. 16 displays, most projects tend to be irregular rather than uniform when using this minimum value. Only 8 (22%) of the 36 projects reached the minimum measure of 0.45 data uniformity.

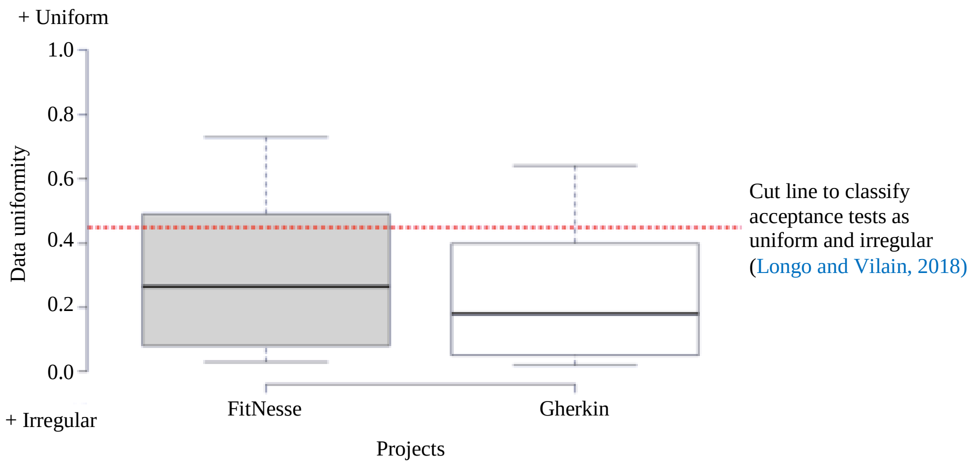


Fig. 16: Visual comparison of the average uniformity between each project of the two notations

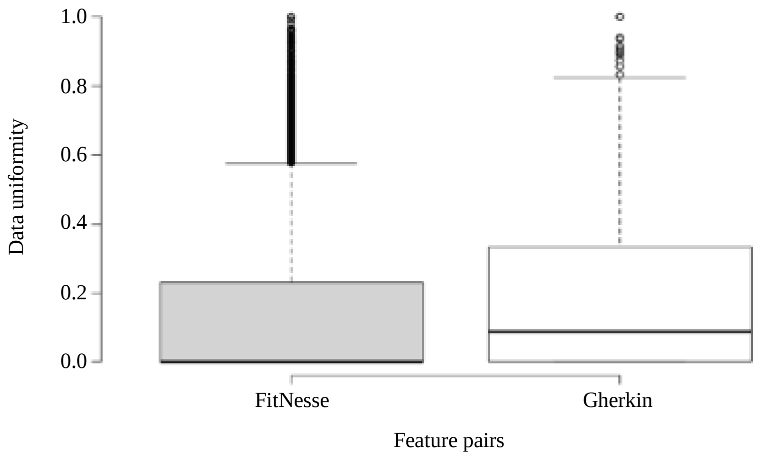


Fig. 17: Visual comparison of the uniformity of pairs of features between projects of the two notations

In order to answer the second interpretation of the question, if there is a difference in uniformity between the pairs of features of FitNesse and Gherkin, the uniformity metrics of all feature pairs for all projects must be used. Figure 17 shows the visual comparison of the uniformity distribution of each feature pair for each project split by notations. For the FitNesse notation, 43.040 pairs of features extracted from 18 projects are considered. For the Gherkin notation, 28.306 pairs of features are considered.

Analyzing the pairs of features, the FitNesse notation has a median of uniformity of 0.00, an average of 0.16 and a standard deviation of 0.27. The 0.00 median of uniformity is justified by the fact that most of the FitNesse feature pairs are irregular between each other. The Gherkin notation has a median of uniformity of 0.08, an average of 0.26 and a standard deviation of 0.34. Thus, in a preliminary analysis, the uniformity metrics value is lower than 0.5. According to the scale, one is the most uniform value and zero is the most irregular value. Therefore, it can be stated that the projects from both notations are more irregular than regular.

The distribution of data uniformity of the feature pairs of each notation was analyzed via a Z-Test, at a significance level of (α = 0.05), resulting on a statistically significant difference in the uniformity of the features between the two notations (P_{value} = 0.0008967). Thus, regarding data uniformity by pairs of features, it was possible to observe statistically significant changes between notations. Since the uniformity metrics values are lower than 0.5, the conclusion reached is that the features are more irregular in the FitNesse notation when compared to the Gherkin notation.

To summarize the answer to question RQ3, the conclusion reached is that the data uniformity of the projects is similar between FitNesse and Gherkin notations. When the analysis is expanded to features, it can be said that the data of the FitNesse features is more irregular than the data of the Gherkin features.

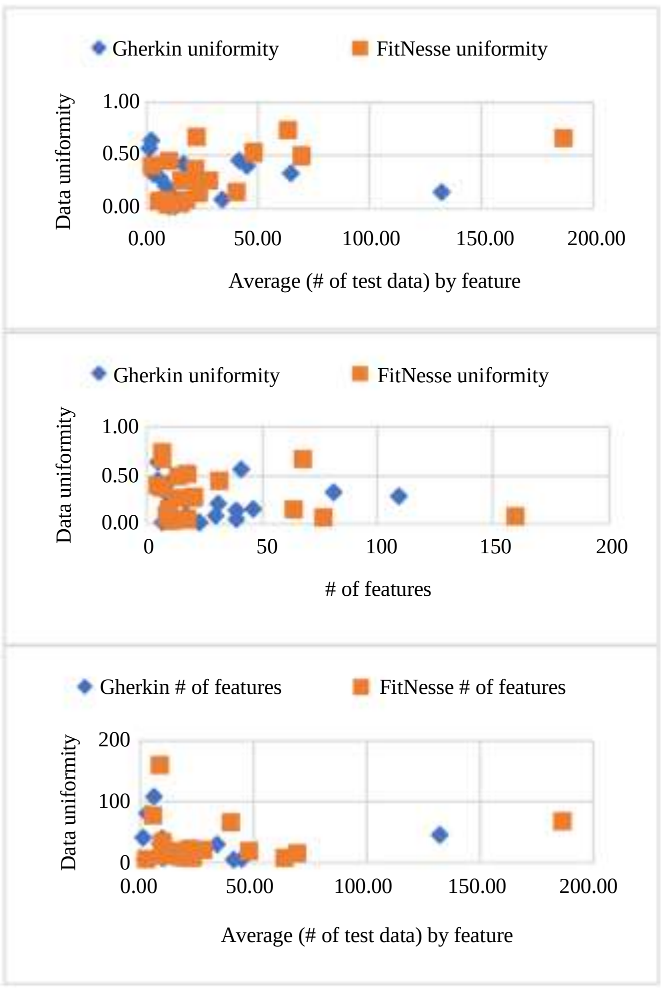


Fig. 18: Scatter diagrams of the variables number of features, average number of test data per feature and data uniformity of the 36 FitNesse and Gherkin projects

Answering RQ4

In order to answer question RQ4, the variables of number of features, average number of test data per by feature and data uniformity of each project are used. Figure 18 presents the dispersion diagrams for the variables of the 36 projects. From a visual perspective, the scatter diagrams do not show correlation between the variables. Linear regression was used for the analysis and discrepant data were excluded at first. These outlying data are from projects, P11, P16, P17, P18, C16 and C18. Therefore, an attempt was done to build a suitable model. However, none of the independent variables, number of features and average number of test data per feature have a significant correlation with the variable response (data uniformity). At a significance level of (α = 0.05), with P_{value} = 0.1361, first-degree linear regression is not adequate, namely, there is no model to characterize the data. Thus, it is concluded that the variables number of feature and average number of test data per feature have no correlation with data uniformity. That is, the size of the project does not influence the uniformity of the data.

Threats to Validity

The case studies are incomplete without discussing the concerns that can threaten the validity of the results. Internal validity refers to causal inferences based on experimental data (Yin, 2003). As for the case studies, the scope of the work was limited to projects found exclusively as public GitHub repositories. The GitHub platform was chosen because it has more than 10 million repositories and is becoming one of the most important sources of software artifacts on the Internet (Kalliamvakou *et al*., 2014). The GitHub platform was used in order to mitigate any bias regarding size, human qualification and type of the repository. In this way, the GitHub platform provided projects from all around the world. Above all, only public projects were used, which may represent a difference in results when compared to private projects. In both case studies, only projects with more than five features were selected. This restriction was intended to avoid extremely small and/or premature projects. The sample of projects for case study I was composed of all executable projects that could have the metrics applied on them. So, for case study I, accurate decisions can be done regarding FitNesse projects on GitHub.

The sample of case study II is composed only of the 18 executable repositories that could have the metrics applied on them, so not all GitHub repositories, but a sample of 18 out of 959. This sample was limited by the application effort of the metrics, because the metrics was applied with the Cucumber framework and, depending on the project, several interventions were necessary for the correct execution. Still, there is some bias to case study II, that is, the first 18 repositories ranked by "Best match" were selected. This bias occurred because, initially, the intention was to investigate all GitHub repositories, but the time and effort required to do so discouraged a complete investigation. Above all, it is highlighted that there is no difference between FitNesse and Gherkin between the number of features and average number of test data per feature of the projects. This equality between FitNesse and Gherkin projects mitigates the bias in the selection of Gherkin projects and contributes to the validity of the conclusions.

The construction of validity refers to the appropriate use of metrics and evaluation measurements (Yin, 2003). In the case studies, uniformity measures were obtained according to the proposed metrics. In order to calculate other statistical metrics, statistics were used upon agreement and the recommended practices for applying them were scrupulously followed.

External validity refers to the ability to generalize the findings to other domains (Yin, 2003). The external validity of the research poses a threat to both case studies. The threat is that the scope is limited to the public GitHub repositories. In private repositories, there may be longer and larger projects, additionally, there may be better qualified teams involved in the task of treating the data to improve the quality of acceptance tests. However, the sample was made up of public repositories from around the world and for various purposes of applicability of acceptance tests. Reliability refers to the ability of other researchers to replicate a methodology (Massey *et al*., 2014). The metric proposal for the two notations, the evaluation technique of the case studies and its results were detailed. Still, we consider that it is important for other researchers to be able to reproduce the study, for which the implementation and all the collected data were done available on GitHub[3,4].

Discussion

The applicability of the proposed metrics was feasible because their implementation was included in the FitNesse and Cucumber frameworks. The proposed metrics were applied to 18 FitNesse projects and 18 Gherkin projects. Uniformity measurements were done at two levels: At the level of pairs of features and at the project level. At the project level, the uniformity metrics presents a more general view. However, the uniformity metrics at the project level was low, that is, most of the analyzed projects, regardless of their rating, had less than 0.45 of data uniformity. Longo and Vilain (2018) classified projects with less than 0.45 uniformity as

[3]https://github.com/douglashiura/fitnesse-data-uniformity.git
[4]https://github.com/douglashiura/cucumber-data-uniformity.git

irregular. The reason why many projects have irregular data has not been identified; however, it is assumed that data uniformity has not been addressed in these projects.

A simple training can be effective in the treatment of uniformity in the specification of acceptance tests by a single person. However, when a team is responsible for the specification and development, complications in the uniformity can arise. In a team specification, individuals can know different examples of data that can be easily used at the time of specification. However, during the test automation, when the tester implements the glue code, doubts may arise when test data are irregular. Due to not being close to test specifiers, testers can often follow through with their doubts and produce a glue code with some bad smell. Thus, this can accumulate bad smell throughout the development of the tests. If these doubts are associated with the irregularity of the test data, the ideal solution is to review the uniformity before implementing the glue code. This way, the metric can be useful for an overall evaluation of the project and an evaluation of the pairs of features.

Upon evaluating the project, we do not have a specific number to indicate whether a FitNesse or Cucumber project is uniform or not. Longo and Vilain (2018) suggest a minimum uniformity value of 0.45 for US-UID projects. Above all, the impact of uniformity in the automation of tests has already been studied in this other notation. The application of the metrics can be used for any type of project, regardless of the uniformity metrics, but it is unclear which minimum uniformity value could be indicated in the process of test specification and automation in order to obtain better communication advantages and high-quality glue code.

In that sense, it is recommended that the minimum value of 0.45 of data uniformity of a project be reached before starting the test automation process. In addition, during the application of the metric in the case studies, it was possible to observe some quality criteria in the projects with more uniformity. One of the quality criteria was clarity, that is, one can read and understand most tests with greater uniformity. Above all, there are many factors that can influence clarity, such as knowledge of the problem and the language used to write the test. Not all evaluated projects had glue code available, which suggests low quality works[5]. However, even in projects with glue code available, a low quality was observed, that is, the glue code was written with several responsibilities, including loop (for) and deviations (if's). As stated by (Meszaros, 2007), conditional test logic becomes tests harder to understand and should be avoid whenever possible, as it was possible to understand, the deviations were just being used to enable the SUT configuration and were used because of irregular test data.

[5]https://github.com/Vardot/varbase-behat/blob/8.x-4.x/features/bootstrap/SelectorsContext.php

Conclusion

The comparison between FitNesse and Gherkin suggests that there are no differences in project sizes. There is also no difference in uniformity between projects. However, when comparing pairs of features, there is a significant difference. The pairs of features of FitNesse have less uniformity than the pairs of Gherkin features. Above all, data uniformity values in the evaluated projects are low, that is, notations from both projects have high amounts of irregular test data.

The main contributions of this article are the two proposed metrics. These metrics can be applied to any project with Gherkin or FitNesse notation and can be a good quantitative assessment tool. A contribution that can be used in future research is the information collected from the 36 GitHub projects. The metric has potential for applicability throughout the specification of acceptance tests. The teams using the metrics can go through the measurements and make interferences to improve uniformity before the test automation. The other contributions of this work include the findings from the case of studies. First, we found out that there are no statistical differences between FitNesse and Gherkin notations regarding the uniformity. We also found out that the size of the project does not appear to have impact in the uniformity. Above all, we discovered that projects with a greater uniformity tend to have a better glue code with better reuse.

An investigation of the impact of uniformity on communication throughout development is suggested as a future work, as well as experiments evaluating the effort and volume of the glue code for projects with different levels of uniformity. Another potential future work is to investigate the use of test data recommendation systems to assist the construction of tests, by recommending uniform data. Also, the measurement of uniformity could be included in the editing tools of FitNesse and Cucumber tests as a guide to test specifiers. Actually, the application of the proposed metrics took place through implementation that consider the execution of the FitNesse and Cucumber testing frameworks.

Acknowledgment

This work was partially supported by the Brazilian Federal Agency for the Support and Evaluation of Graduate Education (CAPES) and by the Brazilian Council for Scientific and Technological Development (CNPq) through M.Sc. and Ph.D. grants.

Authors Contributions

Douglas Hiura Longo: Defined the metrics, implemented Metrics in Cucumber and FitNesse Frameworks, carried out the study cases and wrote the paper.

Patrícia Vilain and Lucas Pereira da Silva: Proposed and defined the metrics, carried out the study cases and wrote the paper.

Ethics

All authors have been personally and actively involved in substantial work leading to the paper and will take public responsibility for its content.

References

Alvestad, K. (2007). Domain Specific Languages for Executable Specifications (Master's thesis, Institutt for datateknikk og informasjonsvitenskap). https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/250512

Beck, K. (2003). Test-driven development: by example. Addison-Wesley Professional.

Coutinho, J. C., Andrade, W. L., & Machado, P. D. (2019, September). Requirements engineering and software testing in Agile methodologies: A systematic mapping. In Proceedings of the XXXIII Brazilian Symposium on Software Engineering (pp. 322-331). https://doi.org/10.1145/3350768.3352584

Cucumber. (2019). BDD Testing; Tools and techniques that elevate teams to greatness. Cucumber. https://cucumber.io.

Dos Santos, E. C., Vilain, P., & Longo, D. H. (2018, May). Poster: A Systematic Literature Review to Support the Selection of User Acceptance Testing Techniques. In 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion) (pp. 418-419). IEEE.

dos Santos, E. C., & Vilain, P. (2018, May). Automated acceptance tests as software requirements: An experiment to compare the applicability of fit tables and gherkin language. In International Conference on Agile Software Development (pp. 104-119). Springer, Cham. https://doi.org/10.1007/978-3-319-91602-6_7

Druk, M., & Kropp, M. (2013, May). ReFit: A Fit test maintenance plug-in for the Eclipse refactoring plug-in. In 2013 3rd International Workshop on Developing Tools as Plug-Ins (TOPI) (pp. 7-12). IEEE. https://doi.org/10.1109/TOPI.2013.6597187

FitNesse. (2019). FitNesse. http://fitnesse.org

Gärtner, M. (2012). ATDD by example: a practical guide to acceptance test-driven development. Addison-Wesley.

Greiler, M., Van Deursen, A., & Storey, M. A. (2013a, March). Automated detection of test fixture strategies and smells. In 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation (pp. 322-331). IEEE. https://doi.org/10.1109/ICST.2013.45

Greiler, M., Zaidman, A., Van Deursen, A., & Storey, M. A. (2013b, May). Strategies for avoiding text fixture smells during software evolution. In 2013 10th Working Conference on Mining Software Repositories (MSR) (pp. 387-396). IEEE. https://doi.org/10.1109/MSR.2013.6624053

Hendrickson, E. (2008). Driving development with tests: ATDD and TDD. STARWest 2008.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2014, May). The promises and perils of mining github. In Proceedings of the 11th working conference on mining software repositories (pp. 92-101). https://doi.org/10.1145/2597073.2597074

Longo, D. H., & Vilain, P. (2018). Metrics for Data Uniformity of User Scenarios through User Interaction Diagrams (S). In SEKE (pp. 592-591). https://doi.org/10.18293/SEKE2018-075

Longo, D. H., & Vilain, P. (2015a). Creating User Scenarios through User Interaction Diagrams by Non-Technical Customers. In SEKE (pp. 330-335). https://doi.org/10.18293/SEKE2015-179

Longo, D. H., & Vilain, P. (2015b). User scenarios through user interaction diagrams. International Journal of Software Engineering and Knowledge Engineering, 25(09n10), 1771-1775. https://doi.org/10.1142/S0218194015710151

Longo, D. H., Vilain, P., & da Silva, L. P. (2019). Impacts of Data Uniformity in the Reuse of Acceptance Test Glue Code. In SEKE (pp. 129-176). https://doi.org/10.18293/SEKE2019-102

Lucassen, G., Dalpiaz, F., Van Der Werf, J. M. E., & Brinkkemper, S. (2015, August). Forging high-quality user stories: towards a discipline for agile requirements. In 2015 IEEE 23rd international requirements engineering conference (RE) (pp. 126-135). IEEE. https://doi.org/10.1109/RE.2015.7320415

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., & Brinkkemper, S. (2016). Improving agile requirements: the quality user story framework and tool. Requirements Engineering, 21(3), 383-403. https://doi.org/10.1007/s00766-016-0250-x

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., & Brinkkemper, S. (2017, February). Improving user story practice with the Grimm Method: A multiple case study in the software industry. In International Working Conference on Requirements Engineering: Foundation for Software Quality (pp. 235-252). Springer, Cham. https://doi.org/10.1007/978-3-319-54045-0_18

IEEE Std 830. (1998). IEEE 830-1998 - IEEE Recommended Practice for Software Requirements Specifications. https://standards.ieee.org/standard/830-1998.html

Massey, A. K., Rutledge, R. L., Antón, A. I., & Swire, P. P. (2014, August). Identifying and classifying ambiguity for regulatory requirements. In 2014 IEEE 22nd international requirements engineering conference (RE) (pp. 83-92). IEEE. https://doi.org/10.1109/RE.2014.6912250

Melnik, G., & Maurer, F. (2005, October). The practice of specifying requirements using executable acceptance tests in computer science courses. In Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages and applications (pp. 365-370). https://doi.org/10.1145/1094855.1094974

Meszaros, G. (2007). xUnit test patterns: Refactoring test code. Pearson Education.

Park, S. S., & Maurer, F. (2008, May). The benefits and challenges of executable acceptance testing. In Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral (pp. 19-22). https://doi.org/10.1145/1370143.1370148

Ricca, F., Torchiano, M., Di Penta, M., Ceccato, M., & Tonella, P. (2008). The use of executable fit tables to support maintenance and evolution tasks. Electronic Communications of the EASST, 8. https://pdfs.semanticscholar.org/82ed/31abd3c57f87bbfc6f3696980b9b6d1e401e.pdf

Ricca, F., Torchiano, M., Di Penta, M., Ceccato, M., & Tonella, P. (2009). Using acceptance tests as a support for clarifying requirements: A series of experiments. Information and Software Technology, 51(2), 270-283. https://doi.org/10.1016/j.infsof.2008.01.007

Softwaretestinghelp. (2020). Software Testing Help. https://www.softwaretestinghelp.com/

Sommerville, I. (2011). Software Engineering. 9th ed. Pearson, Boston. ISBN-13: 978-0-13-703515-1.

Torchiano, M., Ricca, F., & Di Penta, M. (2007, September). " Talking tests": a Preliminary Experimental Study on Fit User Acceptance Tests. In First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007) (pp. 464-466). IEEE. https://doi.org/10.1109/ESEM.2007.76

Yin, R. K. (2003). Case study research: design and methods (ed.). Applied social research methods series, 5.