Business Value Driven Regression Testing for Agile Methods

1st Mondal
SCAI
Arizona State University
USA
amondal8@asu.edu

1st Das
SCAI
Arizona State University
USA
sdas76@asu.edu

2nd Jain
SCAI
Arizona State University
USA
ajain342@asu.edu

3rd Gary
SCAI
Arizona State University
USA
kgary@asu.edu

***Abstract*—This paper introduces an intelligent value-driven approach to regression test selection to enhance the effectiveness of regression testing in agile development. By selecting test cases associated with higher business value, this approach ensures that critical existing business functionalities are thoroughly retested in case of any changes during the testing process. Unlike traditional approaches focusing on technical coverage, this method integrates business priorities, resulting in a more targeted and strategic test selection. Simulations comparing this approach to random selection demonstrate that it preserves more business value while reducing the number of test cases executed. The results indicate that this value-driven regression testing selection method strikes an optimal balance between testing efficiency and safeguarding high-impact features, making it a practical and effective solution for agile environments.**

***Index Terms*—agile, CI/CD, regression testing, business value**

I. Introduction

This paper introduces a Regression Test Selection (RTS) approach tailored for agile methods. Agile methods are the dominant software development method, with over 89% [1] organizations following some agile variants. The quick nature of agile comes with challenges, and testing is one of the significant ones [2]. The challenge becomes more evident after an update/change to ensure delivered functionalities continue to work as intended, suggesting that the software has not regressed backward.

Regression testing is the process that aims to ensure that software changes do not break existing accepted functionalities [3]. The swift deliveries promised by agile methodologies and CI/CD pipelines can compromise quality, resulting in businesses losing money and face. The CrowdStrike incident in 2024 summer is a stark reminder of the potential risks of incomplete regression testing, demonstrating how it can lead to critical failures [4], [5]. This incident underscores the need for thorough testing, as incomplete regression testing under the pressure of fast delivery can lead to reliance on automated tests set up by engineers who may lack a comprehensive understanding of the system's functionality from a business perspective that is valuable to the customer.

Agile, as per the manifesto [6], strongly emphasizes the *value* of the software product delivered, which is the primary success metric of the deliverable. This value, as highlighted by Hidenberg et al. [2012] [7], is not just financial but also non-financial. Moreover, lean principles state that any activity that does not provide value to the customer/organization should be eliminated [8]. We extend this concept to regression testing, which, while not directly contributing value, plays a crucial role in checking the amount of value the product holds after changes. It ensures that new increments do not introduce defects that could diminish the value of the already delivered software; thus, regression testing is a value-preserving activity.

Requirements represent value, and researchers have explored requirement-based regression testing [9] [10]. Various techniques are used to rank requirements, followed by selecting and/or prioritizing test cases associated with the requirements for regression testing. While these approaches report prominent results, they have certain limitations regarding the evaluation metric. Even though promising, these metrics used to evaluate the effectiveness of regression testing have yet to capture the value of the requirement selected and/or prioritized from a business point of view. For example, the software updates a sorting algorithm in a component, and the component is thoroughly regression tested. However, the changes are irrelevant to the end user's business needs, i.e., the change does not affect the value of the software. For large enterprise systems such as CrowdStrike, this change will not regress the whole system even though the component might not have regressed. Thus, regression testing post any increment must include metrics that signify the business value of requirements. This metric must be used simultaneously with metrics such as coverage/fault detection rate to ensure that the change has not impacted anything technical or the business.

Hence, we suggest an approach in which a requirement's business value is considered when selecting test cases for regression testing. If a requirement already adds value to the system, it has fulfilled all the necessary acceptance criteria, addressed the business needs, and met the definition of done (DoD) [11]. Therefore, from a regression testing standpoint, the goal is to ensure that all requirements that meet the definition of done remain in the done state.

Given the role of regression testing in agile environments, it is essential to ensure technical and component-level functional correctness and safeguard the software's business value. While current regression testing approaches excel at testing technical changes, they may overlook the broader implications of how

these changes impact the system as a whole. By integrating business value into the test selection process, regression testing can evolve into a more strategic tool that ensures the software's stability and continued alignment with business priorities. This refined approach opens the door to a deeper exploration of how value-based testing can enhance the effectiveness of regression testing in today's fast-paced development landscape.

The rest of this paper is organized as follows: section II lays the foundation for understanding how regression test selection strategies can be effectively applied in agile environments. Section III discusses the design rationale and implementation of our algorithm. Section IV describes the experimental approach for validation of the algorithm, and section V present the results of that experimentation. Section VI discusses our interpretation of the results, and we conclude with some final thoughts in section VIII.

II. Background

A regression test selection (RTS) algorithm selects a subset of tests from the complete set of available tests, reducing the size of the regression test suite and improving testing efficiency [12]. Many different criteria have been presented in the literature for guiding RTS, such as [13]:

- selecting tests associated with requirements that have higher significance
- selecting tests that are more likely to detect defects (based on execution history)
- selecting tests closely related to new requirements introduced since the last version of the software
- selecting tests based on what code has changed since the last version

A related concept is regression test prioritization (RTP), which extends RTS by requiring an ordered subset, presumably with the motivation of finding (more and/or significant) defects earlier. For the purposes of this research, we consider the mathematically weaker notion of RTS instead of RTP, for reasons discussed in section II-C. RTS and RTP algorithms have been studied for a long time, with established approaches identified by [14], [15], [3] among others. However, agile software processes, with their emphasis on short (or continuous) release cycles of smaller software versions, suggest revisiting these algorithms and approaches to address the nature of what is now the predominant way to develop and deliver software.

The subsequent subsections will explore some of the significant contributions of requirement-based RTS, which selects and prioritizes test cases aligned with essential requirements; textual similarity-based RTS, which groups test cases based on the similarity of requirements, enabling efficient selection; and agile-specific regression testing techniques, which adapt regression testing practices to the rapid cycles of continuous integration and delivery (CI/CD) in agile methodologies.

A. Requirements-based Regression Test Selection

The approach we present in this paper is based on requirements significance (1st bullet above), and thus fits into the class of RTS algorithms based on requirements priority. Multiple studies [13], [9], [10] address requirements-based regression testing. One of the studies that motivated this paper is the technique proposed by Chittimalli et al. [2008]. The study provides a way to use system requirements and their associations with test cases to perform RTS. In addition, the proposed technique uses this criticality to order the selected test cases. Another study by Srivastava et al. [2008] proposed a requirements-based regression test prioritization technique that explores two factors: 1) the priority of requirements given by customers, developers, and managers, and 2) the severity and probability of risk factors that occur in requirements. Khalid et al. [2019] [16] collected a dataset of black box test cases using a simple ranking method, and assigned user priority weights according to the priority assigned during the requirement elicitation phase.

B. Textual Similarity Based Regression Test Selection

For this paper, the studies on requirement clustering and measuring textual similarity between requirements closely align with our research and provide significant motivation. Notably, our work resembles the study by Arafeen et al. [2013] [13], which uses textual similarity to cluster requirements for test case prioritization. Their approach involves calculating textual resemblance between requirements and applying k-means clustering to group similar requirements. The clusters are then prioritized based on two factors: 1) code commits by developers and 2) the extent of code modifications. Test cases are selected from these prioritized clusters using a requirement-test case traceability matrix. The technique's effectiveness was validated using the Average Percentage of Fault Detection, showing improvement rates between 55.52% and 65.37%. However, the method's reliance on accessing source code, especially in large applications, presents challenges.

For our research, textual similarity is used as a means to relate new requirements to similar past requirements, and thus provide one dimension of our selection algorithm. This way of localizing the impact of changes in the requirements space (compared to the code space, 4th bullet above) is less intensive and does not even require the source code. To be clear, we are using an off-the-shelf (OTS) textual similarity library, not contributing a new such algorithm. From a test process standpoint, better textual similarity algorithms, including those customized to the software requirements space, can simply be "plugged in" to our process to improve test selection results.

C. Agile Regression Testing

With agile methodologies gaining widespread adoption, the demand for rapid, iterative software testing has increased, shifting away from traditional models such as waterfall [14]. The rapid iterative nature of agile methods, combined with an emphasis on automated testing and pipeline delivery, suggests there simply isn't time to correct defects found during the testing process. Gone are "test-and-fix" *phases* on large software versions (large meaning "lots of new features and fixes"), replaced by automated pipelines that pass or fail smaller

software *increments* more often. For this reason we emphasize RTS and not RTP in our research as it reflects modern practice.

Kandil et al. [2017] [17] proposed clustering similar test cases to select regression tests, showing improved fault detection, though limitations like potential bias in selecting representative test cases remain. Ali et al. [2020] [19] employed code coverage and change impact analysis for test selection, while Koivuniemi [2017] [20] explored RTS based on code coverage and dependencies, leading to reduced feedback times without compromising test robustness. Elbaum et al. [2014] [21] discussed continuous integration (CI) specific challenges in optimizing regression testing, showing cost-effective improvements in large-scale projects. Shi et al. [2019] [22] contrasted module- and class-level RTS, finding that while module-level RTS selects more test cases with minimal overhead, class-level RTS is more selective but time-intensive. Their study using Travis CI on open-source Java projects showed RTS reduced testing time compared to running all tests, though cloud-based overhead remains. Yu and Wang [2018] [23] noted that while RTS may be redundant for infrequent commits, it becomes crucial for rapid changes, as frequent commits often affect a small portion of the source code. For most Java projects, class-level RTS proved more efficient than method-level RTS or exhaustive testing. These advances in regression testing, from code coverage to machine learning integration, highlight the need for continuous improvements as CI becomes more prevalent.

Various regression test selection approaches, including requirements-based, textual similarity-based, and agile-specific techniques, are invaluable in upholding software quality. Agile methodologies, which emphasize the importance of delivering incremental business value to stakeholders, serve as a reassuring reminder of their effectiveness. 'Value' is the cornerstone of software success. Regression testing methods play a crucial role in swiftly identifying multiple defects; however, a defect indicates that the software is not in the 'done' state, thereby diminishing its value. Therefore, in agile environments, regression testing must also ensure that changes do not disrupt any business requirements, technical accuracy, and the preservation of value post-change. Sustaining the business value and technical integrity of software is a pivotal aspect of agile development. Nevertheless, the existing literature on agile regression testing has yet to integrate business value as a metric for evaluating regression testing. As such, our motivation for this paper is to address the research question:

RQ: How does value-based regression test selection compare to random selection in the context of agile?

III. RTS Algorithm

This section presents our design rationale and algorithm for RTS based on a set of agile factors. We start by giving an agile primer to ensure agile-specific language is consistently interpreted, and then present the algorithm.

A. An Agile Primer

We presume the reader has a basic understanding of agile software development, but to ensure the language used in describing our approach is consistently understood, we summarize these concepts briefly here. First, we use the term *agile software development* to refer to methodology not a specific process model; agile methods emphasize incremental and iterative development. For this research it is not important which specific process model is used, though it may be helpful to the reader to assume a *Scrum*-like model [1]. Consistent with Scrum, we assume requirements are written as *user stories*, though this is not strictly necessary. The more important piece is that user stories have 1) a defined acceptance test, and 2) an associated *business value* assigned by a *product owner*. The acceptance tests are candidate regression tests as they are based on observable behaviors of the application; we do not consider unit tests as these (whether black-box or white-box) are based on the implemented code. In practice business values are *interval scale* measures used by Scrum teams to prioritize feature delivery. Different lightweight estimation techniques exist, such as *planning poker*, which apply convergence-based estimating concepts using interval scales such as a Fibonacci sequence or "T-shirt sizing". For our research we assume Fibonacci numbers. Iterations are commonly referred to as *sprints*, which originally in practice were suggested to be roughly one month [18] but in modern practice are much shorter (1-2 weeks). Some organizations reduce sprints to a single day or even a single change event (e.g. feature completion leading to continuous deployment). The Scrum team collaboratively determines which user stories to implement in a new sprint through a planning process, and completes a sprint with a *review* with the Product Owner of all stories passing acceptance tests. From our research perspective the length of a sprint impacts the amount of time available for regression test execution, but otherwise is inconsequential.

B. Algorithm Design Factors

The rationale for applying these factors is as follows. In agile processes we seek to maximize the value delivered. This factor prioritizes the work an agile team does on a daily basis, and therefore should influence the tests that are performed. The first bullet is self-evident as business value is the primary way to prioritize current requirements for agile teams. From this prioritized list we try to identify those requirements that are most like these requirements. By doing so, we identify a set of already implemented requirements whose features are more likely to be adversely impacted by the new software version. Incorporating the business value of those already delivered user stories gives us an indication of risk - if a new feature breaks existing features, just how critical or valuable were those existing features? Note that anecdotally, this might be an analysis function performed by a quality assurance team or the development team itself (criticality of a defect). But we are not concerned with the time required to fix a defect, we are concerned whether existing value inherent in the software is negatively impacted. Finally, the time available to test and

the time needed for each test is obviously important, as these define the inherent constraints of the RTS problem in an agile environment.

C. Regression Test Selection Implementation

The RTS algorithm, as shown in algorithm 1, combines textual similarity, business value, and test execution time to guide the selection of test cases. The first step is to create a matrix of current and prior user stories, with each cell storing a textual similarity score. This gives us a strength-oriented relationship between each pair of stories, current and prior. Then for each row in the matrix (the SimilarityList), we compute two additional values; a sum of the business values of each story pair, and the summed execution time of all test cases associated with the prior story. Recall user stories have one or more acceptance tests, so the union of all acceptance tests from all prior (already delivered) user stories are the candidate set for regression test selection. Following agile traceability guidelines, every user story ships with acceptance tests that formalize its acceptance criteria We therefore select at story level and execute all linked tests, guaranteeing that high-value business functionality is always exercised [24].

This substep simply determines how long it would take to run all tests associated with a prior acceptance test. To complete step 2 we compute what we refer to as the *Importance Value* for each prior user story as the product of similarity and value, normalized by the execution time of all that user stories' acceptance test cases. Effectively we determine a per unit time of *value preserved* from previous sprints using this ratio. The final step, step 3, is simply a greedy algorithm based on a sorted list of Importance Values. The intelligence of this algorithm comes from the combination of similarity and value as the guiding heuristics. The algorithm seeks to protect as much delivered value as possible, which is in stark contrast to typical RTS approaches that seek to maximize defects discovered. The RTS problem reduces to the 0–1 knapsack; exact ILP or DP solvers are infeasible in a CI loop. Greedy ratio selection provides a proven $\frac{1}{2}$-approximation bound and, in practice, reaches within 2–5% of optimal at $\mathcal{O}(n \log n)$ cost [25].

Our implementation was done in Python, with the spaCy Python library used to calculate the Similarity Index (SI), comparing the text of user stories from current releases with those from prior releases. This similarity metric helped identify the user stories most relevant to current changes. As mentioned above, spaCy is an OTS library for text similarity, and suitable library or algorithm could be substituted to improve results, though this was not a focus of our research at this stage. Business value (BV) prioritized user stories important to the business, ensuring that test cases linked to high-value functionalities were prioritized.

Algorithm 1 Value-Preserving Regression Test Selection

Input: CurrentReleaseStories[], PriorReleaseStories[], BusinessValue[], ExecutionWindow
Output: TotalBusinessValue
Initialize `SelectedStories[]`
Set `RemainingExecutionTime` to `ExecutionWindow`
Set `TotalBusinessValue` to 0
Initialize `ImportanceValue[]`
Step 1: Text Similarity Calculation
for each story s_i in CurrentReleaseStories[] **do**
  for each story s_j in PriorReleaseStories[] **do**
    Calculate SI between s_i and s_j using spaCy
    Store the similarity score, s_i, and s_j in SimilarityList[]
  end for
end for
Step 2: Importance Value Calculation
for each pair in SimilarityList[] **do**
  Get the business values BV_i and BV_j
  Compute $BV_{sum} = BV_i + BV_j$
  Calculate total execution time for all test cases associated with s_j: ET_{total}
  Compute Importance Value (IV) as:

$$ImportanceValue[] = \frac{\text{SI} \times BV_{sum}}{ET_{total}}$$

end for
Step 3: Select Prior Release Stories within Execution Window
for each prior release story s_j in PriorReleaseStories[] (sorted by descending ImportanceValue[]) **do**
  while $ExecutionTime - ET_{total} \geq 0$ **do**
    Add s_j to SelectedStories[]
    Add BV_j to TotalBusinessValue
    Subtract ET_{total} from ExecutionTime
  end while
end for
Return TotalBusinessValue =0

IV. Evaluation

This section details the methodology for evaluating the value-preserving regression test selection (RTS) method. Our experimental approach is quasi-experimental ([26], Ch. 6), where we executed a number of simulations varying execution windows to determine the sensitivity to time on the ability of the algorithm to preserve value. In this section we first describe our approach for preparing data by augmenting an existing established dataset with new attributes, and then discuss the criteria for evaluation before presenting the results in the next section.

A. Data Preparation

The simulations were meticulously performed by synthetically augmenting the attributes of the extracted user stories derived from the TAWOS data set [27], which contains real-world user stories articulated in natural language. This is important due to the critical nature of applying a text similarity library in our approach. These user stories were systematically

categorized into two distinct groups: those about current software releases and those associated with prior releases.

The TAWOS data, however, does not include business value as an attribute of the stories. To simulate real-world scenarios that reflect varying business priorities, we augmented the dataset with specific business values assigned to each user story, ranging from 1 to 89, following the Fibonacci series [28] as discussed above. These values were generated right-skewed, as in real-world best practice, user stories are refined (*groomed*) to be small and well-defined so they may be implemented and tested within a single sprint. Smaller stories contribute less value.

The next step was to augment the dataset with test cases that represent the acceptance test cases for each user story. As the actual test case script text is not an input variable to the algorithm, we were more concerned with randomly generating differing cardinalities of user story-to-test case relationships, and with varying test case execution times, as both of these impact the input variables of the algorithm, The execution times for these test cases were modeled to follow a right-skewed distribution just like the user stories, which is representative of the typical variation observed in actual test case execution durations. This intentional skewness towards the right was designed to mirror reality, acknowledging that test cases with longer execution times are generally fewer in number.

Furthermore, the total time allocated for regression testing, called the execution window, was varied between different simulation scenarios. This variation was instrumental in replicating various testing environments that are time-constrained and commonly encountered in agile frameworks. Table 1 below illustrates the specific variables utilized within these simulations, detailing the parameters that influenced the outcomes and the overall testing dynamics.

TABLE I
KEY VARIABLES USED IN THE SIMULATIONS

Variable	Description
Business Value	Values ranging from 1-89
User Story(s)/Release	Count of user stories/release
Execution Time	Execution time of each test case
Execution Window	Available time for regression testing

B. Performance Measure

The evaluation of the RTS algorithm was conducted in comparison to two alternative approaches to illuminate the efficacy of RTS in preserving overall business value under constraints. The primary benchmark used for this assessment was the total retained business value, determined by aggregating the business values associated with the user stories whose corresponding test cases were selected for regression testing. This comprehensive analysis aimed to assess the effectiveness of the RTS method in safeguarding business value compared to the two baseline approaches, while navigating the restrictions imposed by the allotted execution window.

The first alternative method, known as the Random Selection Process (RSP), operated on a principle of chance. It randomly selected test cases within the available time frame, without factoring in either textual similarity or the underlying business value of the test cases. This approach served as a critical point of reference, allowing for an exploration of whether the value-driven RTS method offered a marked improvement over the arbitrariness of random selection.

The second baseline method employed was IV (Importance Value), which methodically chose test cases based on their importance value in a greedy fashion. This value was computed as the product of the similarity index (SI) and business value (BV) associated with each test case. Notably, this approach did not account for execution times, providing a clear lens through which to analyze the implications of incorporating test case execution times into the selection process. This isolation facilitated a deeper understanding of the potential benefits of a time-sensitive approach, highlighting the strategic importance of aligning testing efforts with both business priorities and test execution realities. The replication package can be found here.[1]

V. RESULTS

This section presents the results of the RTS approach using two importance value formulations, comparing them with the random selection process discussed in the previous section. We focus on the preserved value relative to the maximum preserved value, which is the total business value of all user stories determined by the selection process. We utilized two variations to compute importance values. It is important to remember that we are conducting RTS, not regression test prioritization. Even though our approach could facilitate prioritization, we are not considering it, as we need to consider defect severity and time for fixing and retesting. Therefore, if a regression test fails, the corresponding change causing the failure should be excluded from the upcoming increment. To illustrate that we first see Figure 1 shows variability across different simulation factors, a bar chart of 20 simulation runs is presented in Figure 1 [*Color codes:- Blue: RSP, Orange: IV = (SI x BV), Grey: IV = (SI x BV)/Sum of test case exec time, Yellow: Max Possible BV*]. These grouped bar charts display the total business value available in previous releases and the maximum possible value preserved. Figure 1 plots preserved business value for each of the 20 runs (bars grouped by run). The yellow bar in every group is the theoretical upper bound; blue, orange and grey bars are the three methods under test, in that order. In Figure 2 shows the percentage of test cases sleeted for each of the approach [*Color codes:- Blue: RSP, Orange: IV = (SI x BV), Grey: IV = (SI x BV)/Sum of test case exec time*]. Because all three lines share the same x-axis ordering as Figure 1, readers can visually connect "fewer tests yet higher BV" decisions to each run—exactly the cost-vs-value trade-off our RQ examines. Because all three

[1]https://github.com/amondal8/VPRTS/blob/aj/updating_code/README.md

lines share the same x-axis ordering as Figure 1, readers can visually connect "fewer tests yet higher BV" decisions to each run—exactly the cost-vs-value trade-off our RQ examines.

Note: We limited the study to 20 seeds because (i) empirical-software-engineering surveys report a median 10 runs for randomized algorithms, with only 37% of papers applying statistical tests at all and (ii) after 15 seeds our BV curve stabilized ($\sigma < 1\%$), indicating further replication would not alter the conclusion [29].

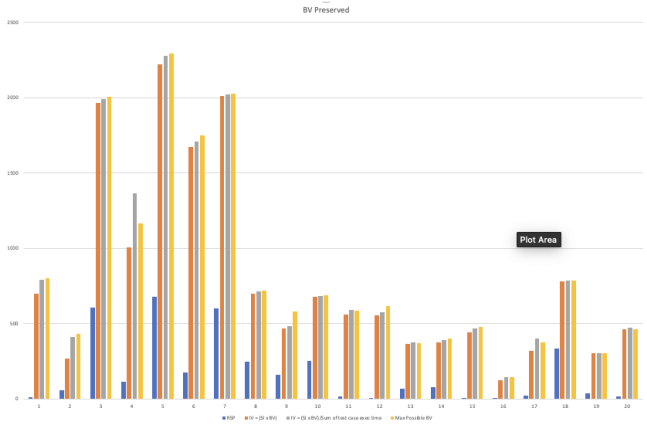

Fig. 1. BV Preserved

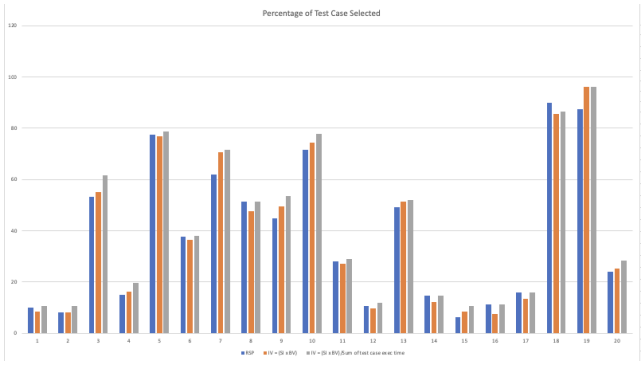

Fig. 2. Percent of TC selected

The results of the 20 simulations, presented in Figures 1 and 2, demonstrate apparent differences in the effectiveness of the three test case selection methods—Random Selection Process (RSP), IV = SI × BV, and IV = SI × BV / Sum of Test Case Execution Time—in terms of both business value preserved and the percentage of test cases selected. These differences are strongly influenced by the available time for regression testing in each simulation.

In terms of business value preserved, as illustrated in Figure 1, the RSP method consistently under-performs across all simulations. The randomness inherent in this method results in an inefficient selection of test cases, leading to the lowest amount of business value preserved in nearly every instance. For example, in simulations 4, 5, and 6, where the available time is relatively high, RSP preserves significantly less business value than the other methods. This trend is consistent across both scenarios with abundant and constrained execution time, highlighting the fundamental inefficiency of RSP when business value is not a guiding factor in the selection process.

In contrast, the IV = SI × BV method, which prioritizes test cases based on their business value and similarity to prior release user stories, performs significantly better in preserving business value, especially when the execution time is not a limiting factor. In simulations with ample available time, such as runs 5 and 7, this method preserves business value close to the Maximum Possible BV, demonstrating its effectiveness in selecting high-value test cases when time constraints are relaxed. However, in simulations with more restricted time, such as runs 10 and 12, this method shows a noticeable decline in performance. The reduction in business value preserved in these scenarios suggests that while this approach effectively prioritizes high-value test cases, it fails to account for execution time, making it less efficient when time is a critical constraint.

The IV = SI × BV / Sum of Test Case Execution Time method, on the other hand, demonstrates consistent performance across both time-rich and time-constrained simulations. By incorporating execution time into the Importance Value calculation, this method balances preserving business value and adhering to the available execution window. In simulations with limited time, such as runs 10 and 12, this approach outperforms RSP and IV = SI × BV by selecting test cases that maximize business value while fitting within the restricted time frame. Even in simulations with more available time, such as runs 5 and 9, the IV = SI × BV / Execution Time method remains competitive, preserving business value close to the maximum possible. This indicates that including execution time in the selection process provides a significant advantage, particularly in time-constrained environments.

The percentage of test cases selected, as shown in Figure 2, further illustrates the adaptability of each method to varying time constraints. The RSP method, due to its random nature, exhibits erratic behavior, with the percentage of test cases selected varying widely from simulation to simulation. This inconsistency reflects the lack of strategic prioritization in the RSP approach, resulting in inefficient test case selection regardless of the available time. In contrast, the IV = SI × BV method selects a higher percentage of test cases when more time is available, as seen in simulations such as runs 5 and 7, where it selects more than 70% of the test cases. However, in simulations with tighter time constraints, such as runs 10 and 12, the percentage of selected test cases drops significantly, illustrating the method's difficulty in optimizing under limited execution time.

The IV = SI × BV / Sum of Test Case Execution Time method consistently selects a moderate to high percentage of test cases, regardless of the available time. Even in simulations with severe time constraints, this method can select many test cases while ensuring they fit within the execution window. For instance, in runs 10 and 12, where the available time is minimal, this method selects a higher percentage of test cases than the other two, reflecting its

efficiency in balancing business value and time constraints.

The influence of the execution time on the performance of these methods is evident across all simulations. In scenarios with ample time for regression testing, such as simulations 5, 9, and 19, both IV-based methods perform similarly, as the larger execution window allows for more comprehensive test case selection. However, in simulations with tighter execution windows, such as runs 10, 12, and 15, the differences between the methods become more pronounced. The IV = SI × BV method, which does not factor in execution time, struggles to maintain high performance in these scenarios. In contrast, the IV = SI × BV / Execution Time method consistently outperforms the other approaches by selecting the most valuable test cases that can be executed within the limited time.

In summary, the results show that while IV = SI × BV is highly effective when execution time is abundant, it becomes less efficient under time constraints. In short, IV = SI × BV / Execution Time won 20 / 20 runs against random and 15 / 20 against IV = SI × BV, confirming that time-aware value weighting is the decisive factor. The IV = SI × BV / Sum of Test Case Execution Time method, which accounts for execution time in the selection process, is more adaptable and efficient across time-rich and time-constrained scenarios. The RSP method, on the other hand, consistently performs poorly, demonstrating the inefficiency of random selection when business value and execution time are not considered.

VI. Discussion

This section answers the research question: *How does value-based regression test selection compare to random selection in the context of agile?*

By analyzing two Importance Value (IV) formulations—IV = SI × BV and IV = SI × BV / Execution Time—, we examined how business value (BV) and execution time influenced the selection of test cases across 20 simulations. By focusing on both business value preservation and the efficiency of test execution, the RTS algorithms aimed to improve the regression testing process in agile environments. The results across 12 analyzed simulations offer a clear picture of the strategies employed by each IV formulation, and these insights can be generalized to the remaining eight simulations, highlighting the practical trade-offs made by the algorithms in selecting test cases under varying time constraints.

The selection of user stories and test cases across the 12 analyzed simulations reflects the decision-making of the regression test selection (RTS) algorithms based on two Importance Value (IV) formulations: IV = SI × BV and IV = SI × BV / Execution Time. Both formulations aim to maximize business value (BV) during regression testing but with differing priorities. Analyzing these 12 simulations provides insight that can be generalized to the remaining eight simulations due to the consistent pattern of selections observed.

In the IV = SI × BV / Execution Time formulation, execution time plays a crucial role in the algorithm's selection of test cases. This approach prioritizes test cases with the best balance between business value and execution time, ensuring that the most valuable test cases are selected within the constraints of the available testing window. For example, in simulations such as 1 and 2, many test cases with moderate business value but relatively low execution times were selected, allowing the RTS algorithm to efficiently use the available time while preserving as much business value as possible. This pattern is particularly evident in simulation 3, where test cases like TC186 and TC193 were chosen not because they had the highest business value but because their shorter execution times allowed them to fit within the limited testing window. This selection strategy enables the algorithm to prioritize the overall business impact of the regression testing process without exceeding time limits.

Execution time significantly influences the test case selection process under the IV = SI × BV / Execution Time. In cases where available time is limited, the algorithm leans towards test cases with a favorable business value-to-execution time ratio. This ensures that as many valuable test cases as possible can be run within the available testing window, maximizing business value retention while adhering to time constraints. The selections in simulations 3 and 4 demonstrate this, where test cases with moderate business value but shorter execution times were preferred, ensuring that the testing process could proceed efficiently. By contrast, if execution time had not been considered, longer-running test cases might have been chosen, limiting the number of test cases that could be executed and thus reducing overall business value preservation.

On the other hand, the IV = SI × BV formulation focuses solely on business value and the similarity index (SI) between user stories without factoring in execution time. While this approach would prioritize test cases that maximize business value, it risks overloading the testing process with long-running test cases, especially in time-constrained scenarios. In such cases, the lack of consideration for execution time would lead to inefficient use of the available testing window. This explains why, in the analyzed simulations, no rows were populated under this formulation—indicating that time constraints significantly guided test case selection in these scenarios.

The observed selections across the 12 simulations demonstrate the effectiveness of IV = SI × BV / Execution Time in balancing business value preservation with execution efficiency. The consistent selection of test cases that optimize business value and time reflects the algorithm's ability to make practical trade-offs, ensuring that the testing window is used as efficiently as possible. This approach proves particularly advantageous when execution time is limited, as seen in simulation 5, where shorter test cases with lower business value were selected to ensure that the available time was fully utilized.

We can generalize these findings by examining these 12 simulations to the remaining eight. The selection patterns remain consistent: in time-constrained environments, the algorithm favors test cases with a favorable business value-to-execution time ratio, maximizing the number of test cases that can be executed while preserving essential business functionality.

This strategy results in a more comprehensive and efficient regression testing process, particularly under time restrictions, and ensures that the business impact of regression testing is maximized without exceeding time limits.

In conclusion, the inclusion of execution time in the IV = SI × BV / Execution Time formulation significantly impacts the selection of test cases. This formulation allows the algorithm to prioritize test cases with the highest overall benefit, optimizing for business value and execution time. In contrast, the IV = SI × BV formulation would prioritize high-value test cases but may lead to inefficiencies in time-constrained scenarios. The RTS algorithm's adaptability in selecting test cases with lower execution times, especially in scenarios with restricted time, ensures that regression testing remains efficient and effective. This analysis of 12 simulations confirms that these patterns can be extended to the rest, making the algorithm a valuable tool for maintaining regression quality under varying time constraints.

VII. Limitations

This preliminary study is based on simulated scenarios with certain limitations. We made key assumptions using Fibonacci numbers to assign business value and the right-skewed distribution of execution times, which were not based on empirical data but on widely accepted practices and theoretical considerations. While these choices help model a practical regression test selection environment, they may only partially represent real-world dynamics. Other simulation factors, such as the specific characteristics of test cases and execution windows, were derived from the authors' interpretations and understanding of typical agile development processes. These factors are open to different interpretations and may vary in real-world settings. For instance, we did not consider factors like setup and tear-down times for test cases, which are common in practice and could influence test case selection. Future research would benefit from validating these assumptions and extending the scope to include more comprehensive, real-world data to strengthen the practical applicability of the proposed method.

VIII. Conclusions and Future Work

This study introduced a value-driven approach to regression test case selection, offering a more strategic method for optimizing the regression testing process. Focusing on business value allows teams to prioritize user stories that deliver the most significant impact, even when testing resources are limited. The comparison with random selection methods confirmed that this value-based strategy could ensure the preservation of critical functionality with fewer test cases, making it a robust alternative for regression testing in agile environments. The method's ability to balance preserving business value and optimizing test execution opens the door for more efficient and targeted regression suites, contributing to higher software quality and more efficient use of testing resources.

However, this work inspires several fascinating possibilities for further refinement and enhancement. One promising avenue for future research involves expanding the method's consideration of time factors. While the current approach focuses solely on test case execution time, real-world testing environments often involve additional overheads, such as setup and tear-down times. These factors can significantly impact the overall efficiency of the regression testing process and warrant inclusion in future iterations of the method. By accounting for these additional time constraints, future work could offer a more precise and holistic approach to test case selection, further enhancing the algorithm's utility in real-world applications.

Moreover, there is considerable potential in exploring the partial selection of test cases within user stories. In this study, we approached each user story holistically, selecting all associated test cases based on business value. However, introducing the ability to prioritize and select only the most critical test cases within a user story could provide a more flexible and granular approach. This partial selection would allow teams to test partial business value when resources are constrained, optimizing the regression test suite for agile environments where time and resources are scarce.

In summary, the proposed value-driven RTS approach has demonstrated the first step towards a novel perspective on using business value as a metric for measuring regression testing effectiveness. It also highlights potential avenues for improvement and emphasizes the importance of business value, as stated in the recent review study by Das and Gary [2025] [30]. Further research is needed to integrate BV-driven RTS into delivery pipelines, which can later be extended to AI/ML pipelines. Fundamentally, these systems are also software systems with a different goal, paving the way for AI/ML systems to measure the direct business outcome that can be measured in various ways, as stated in the review study by Chakraborty et al, [2024] [31]. Additionally, future work can build upon this foundation by incorporating additional time factors, enabling partial test case selection, time to gather, train, and label data for AI/ML systems, and leveraging historical metadata of test cases. Together, these enhancements promise to refine the approach and bring it closer to practical, real-world application, where the demands of agile development necessitate both efficiency and precision in testing strategies. This potential should inspire and motivate further exploration of the value-driven approach in the field of software testing and agile development.

Acknowledgment

This work acknowledges the assistance of a Large Language Model (LLM) used explicitly for editing/drafting purposes. The authors have reviewed all content to ensure originality, accuracy, integrity, and correctness.

References

[1] Institute of Project Management, "16th Annual State of Agile Report," 2022. [Online]. Available: https://instituteprojectmanagement.com/wp-content/uploads/2023/01/AR-SA-2022-16th-Annual-State-Of-Agile-Report.pdf. [Accessed: Sep. 12, 2024].

[2] S. Das and K. Gary, “Agile transformation at scale: A tertiary study,” in *Agile Processes in Software Engineering and Extreme Programming–Workshops*, vol. 22, pp. 3–11, Springer, 2021.

[3] G. Rothermel and M. J. Harrold, “A safe, efficient regression test selection technique,” *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 2, pp. 173–210, 1997.

[4] CrowdStrike, “Channel File 291 Incident Root Cause Analysis,” Tech. Rep., CrowdStrike, Inc., Austin, TX, 2024. [Online]. Available: https://www.crowdstrike.com/wp-content/uploads/2024/08/Channel-File-291-Incident-Root-Cause-Analysis-08.06.2024.pdf.

[5] CrowdStrike, “What is CI/CD?,” 2024. [Online]. Available: https://www.crowdstrike.com/cybersecurity-101/security-it-operations/what-is-ci-cd/. [Accessed: Sep. 12, 2024].

[6] Agile Alliance, “Manifesto for Agile Software Development,” 2001. [Online]. Available: https://agilemanifesto.org. [Accessed: Sep. 12, 2024].

[7] J. Heidenberg et al., “A model for business value in large-scale agile and lean software development,” in *EuroSPI 2012, Vienna*, vol. 19, pp. 49–60, Springer, 2012.

[8] M. Poppendieck, “Principles of Lean Thinking,” *IT Management Select*, vol. 18, pp. 1–7, 2011.

[9] P. K. Chittimalli and M. J. Harrold, “Regression test selection on system requirements,” in *Proc. 1st India Software Engineering Conference*, pp. 87–96, ACM, Feb. 2008.

[10] P. R. Srivastva, K. Kumar, and G. Raghurama, “Test case prioritization based on requirements and risk factors,” *ACM SIGSOFT Software Engineering Notes*, vol. 33, no. 4, pp. 1–5, 2008.

[11] S. Nidhra and J. Dondeti, “Black Box and White Box Testing Techniques: A Literature Review,” *Int. J. Embedded Syst. Appl. (IJESA)*, vol. 2, no. 2, pp. 29–50, 2012.

[12] M. Azizi and H. Do, “Retest: A cost-effective test case selection technique for modern software development,” in *Proc. 2018 IEEE 29th Int. Symp. Software Reliability Engineering (ISSRE)*, pp. 144–154, Oct. 2018.

[13] M. J. Arafeen and H. Do, “Test case prioritization using requirements-based clustering,” in *Proceedings of the 2013 IEEE 6th International Conference on Software Testing, Verification and Validation*, pp. 312–321, IEEE, Mar. 2013.

[14] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, “Reinforcement learning for automatic test case prioritization and selection in continuous integration,” in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 12–22, ACM, New York, NY, 2017.

[15] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, “Test case selection and prioritization using machine learning: a systematic literature review,” *Empirical Software Engineering*, vol. 27, no. 2, p. 29, 2022.

[16] Z. Khalid and U. Qamar, “Weight and cluster based test case prioritization technique,” in *Proceedings of the 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 1013–1022, IEEE, Oct. 2019.

[17] P. Kandil, S. Moussa, and N. Badr, “Cluster-based test cases prioritization and selection technique for agile regression testing,” *Journal of Software: Evolution and Process*, vol. 29, no. 6, p. e1794, 2017.

[18] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[19] S. Ali, Y. Hafeez, S. Hussain, and S. Yang, “Enhanced regression testing technique for agile software development and continuous integration strategies,” *Software Quality Journal*, vol. 28, pp. 397–423, 2020.

[20] J. Koivuniemi, *Shortening feedback time in continuous integration environment in large-scale embedded software development with test selection*, Master’s Thesis, 2017.

[21] S. Elbaum, G. Rothermel, and J. Penix, “Techniques for improving regression testing in continuous integration development environments,” in *Proc. 22nd ACM SIGSOFT Int. Symp. Foundations of Software Engineering*, pp. 235–245, ACM, Nov. 2014.

[22] A. Shi, P. Zhao, and D. Marinov, “Understanding and improving regression test selection in continuous integration,” in *Proc. 2019 IEEE 30th Int. Symp. Software Reliability Engineering (ISSRE)*, pp. 228–238, Oct. 2019.

[23] T. Yu and T. Wang, “A study of regression test selection in continuous integration environments,” in *Proc. 2018 IEEE 29th Int. Symp. Software Reliability Engineering (ISSRE)*, pp. 135–143, Oct. 2018.

[24] TestRail Team, “4 Ways to Use Acceptance Criteria,” *TestRail Blog*, Feb. 12, 2019. [Online]. Available: https://www.testrail.com/blog/use-acceptance-criteria/. [Accessed: Jul. 13, 2025].

[25] Y. Akçay, H. Li, and S. H. Xu, “Greedy algorithm for the general multidimensional knapsack problem,” *Annals of Operations Research*, vol. 150, no. 1, pp. 17–29, 2007.

[26] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, vol. 236. Berlin, Germany: Springer, 2012.

[27] SOLAR-group, “GitHub - SOLAR-group/TAWOS: The Tawosi Agile Webhosted Open-Source Issues dataset,” [Online]. Available: https://github.com/SOLAR-group/TAWOS. [Accessed: Sep. 12, 2024].

[28] C. Chauhan, M. Dakoliya, and R. K. Sharma, “Use of Fibonacci Sequence in Project Estimation,” *Indian Journal of Science and Technology*, vol. 16, no. 33, pp. 2649–2652, 2023

[29] A. Arcuri and L. Briand, “A hitchhiker’s guide to statistical tests for assessing randomized algorithms in software engineering,” *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.

[30] S. Das and K. Gary, “Regression testing in agile—A systematic mapping study,” *Software*, vol. 4, no. 2, pp. 9, 2025.

[31] A. Chakraborty, S. Das, and K. Gary, “Machine learning operations: A mapping study,” in *Proc. World Congress in Computer Science, Computer Engineering & Applied Computing*, Cham: Springer Nature Switzerland, Jul. 2024, pp. 3–21.