September 2022

AUTOMATIC SMART FEATURE ASSIGNMENT FOR CONTINUOUS INTEGRATION FRAMEWORKS

Chiara Troiani

Erwan Zerhouni

AUTOMATIC SMART FEATURE ASSIGNMENT FOR CONTINUOUS INTEGRATION FRAMEWORKS

AUTHORS:
Chiara Troiani
Erwan Zerhouni

ABSTRACT

Continuous integration of software among a variety of developers and teams engenders challenges involving task prioritization and efficient resource allocation. Presented herein are techniques that provide an overview of semantic and codebase dependencies for new features with respect to open bugs. Further, techniques presented herein provide an efficient method through which the development of new features can be assigned by leveraging the relationship between such features and open software bugs.

DETAILED DESCRIPTION

In current software integration environments, when a new software feature is requested, there is no straightforward way to assess either the new feature's dependency on any open bug(s) or the affinity of developers with respect to an affected codebase. Additionally, bugs are often misclassified, resulting frequently as new features.

Presented herein are techniques that provide for achieving various objectives, such as providing for the ability to automatically assess the complexity of new features across a project and resolve dependencies between features and open bugs based on complexity, business requirements, source code and current developers work assignment. Advantageously, a system configured in accordance with the techniques prescribed herein can automatically suggest an efficient "next new feature" development to: developers working on bugs that are semantically related to the new feature, developers working on bugs that touch at the same part of the code as the new feature, and/or developers who have enough bandwidth to work on the new feature. Such a system can also provide an overview of the dependencies between new features and open bugs, which can be visually represented in an interactive dashboard.

The techniques of this proposal can be implemented through a machine learning (ML) model that can write code and can utilize code repositories under a versioning system that is implemented with a corresponding ticketing system.

For example, a Complexity Code Assessment Unit (CCAU) can be provided for the system in which the CCAU is responsible for estimating the time needed for a feature request. This component can receive, as an input, a description of the feature request through collaborative version control software, such as GitHub or GitLab, for example.

Multiple metrics can be utilized to evaluate the feature request complexity. For example, code complexity can first be assessed using a pre-trained ML model that is finetuned to resolve competitive programming problems. Various code factors can then be measured, such as time complexity, memory complexity, and the number of lines of the proposed solution. These metrics can be used to determine the overall code complexity of the feature request.

Next, the impact of the new code within existing code can be estimated. Stated differently, the number of modules impacted by the new feature request can be estimated through which a Code Based Knowledge Graph (CBKG) can be constructed. The knowledge graph can be leveraged by locating the nodes (i.e., the code files) at which the feature request will be implemented by using a key word search in the feature request description. The nodes can be extracted in order to count the number of adjacent vertices, which corresponds to the number of files that might be impacted by the feature request.

Finally, the time needed for developing the feature request can be estimated using, for example, the formula as shown below in Equation 1, in which 'C' represents the code complexity, 'N' represents the number of nodes located within the CBKG, and 'V' represents the number of adjacent vertices of node 'N$_k$'.

$$R_{CCAU} = C \cdot \sum_{k=0}^{K} N_k + \frac{V_k}{2}$$

Equation 1

An Automatic Work Assignment Unit (AWAU) can also be provided for the system in which the AWAU is responsible for resolving dependencies between features request and open bugs for optimizing the development workflow. The AWAU can take, as an input: the feature complexity assessment R_{CCAU}, the CDKG from the CCAU, the code repositories under a versioning system with its corresponding ticketing environment, the developers' current work assignments, and, potentially, business priorities.

First, using the code repositories, a developer contribution map can be computed that encodes the ratio of lines that each developer contributed for each of the module in the code repository. This can provide for the ability to detect affinities between code and developers.

Second, the nodes corresponding to the new features and all the open bugs from the CDKG can be identified. Subgraphs containing these nodes and their adjacent nodes can be extracted (the depth of these subgraphs can be set as parameter) in order to generate three categories of subgraphs: 1) feature requests without dependencies on open bugs (Pure features), 2) open bugs without dependencies on any new feature request (Pure bugs), and 3) new feature requests and open bugs which are related to each other, semantically or from a codebase perspective (Hybrid).

Third, the workload for each of the three aforementioned categories can be computed. The workload for the ‘Pure features’ category will correspond to the ranking generated with the CCAU component. The workload for the ‘Pure bugs’ category workload will correspond to the size assignment from the ticketing system. The ‘Hybrid’ workload will be a combination of the CCAU ranking and the size assignment of each bug related to that feature, as shown in Equation 2, below, in which R_{CCAU} represents the ranking from the previous component, and S_k is the size of the open bugs related to the feature request.

$$R_{CCAU} + \frac{1}{2} \cdot \sum_{k=0}^{K} S_k$$

Equation 2

Finally, the optimized assignment of new features can be computed considering business priorities, the affinity of developers with the codebase, the complexity and dependencies of the features, and the available workload for each developer. In one example, the optimized assignment can be computed using the formula as shown below in Equation 3.

$$\begin{gathered} argmax_A W_B \cdot L^T \cdot A \cdot \mathbf{1} \\ \text{subject to} \, \|A \cdot F_B\| \leq U \end{gathered}$$

Equation 3

For Equation 3, 'W' represents the matrix of business priorities and is of size 1xB. Further, 'L' represents the matrix of affinities between a developer and a business (feature) request (based on how much lines he wrote) and is of size N x B where 'N' is the number of developers. Additionally, 'A' represents the assignment matrix and is of size N x B, 'F' represents the workload for a business request and is of size B x 1, and 'U' represents the unit of time available for each developer and is of size N x 1 where '1' is the unit vector of size B x 1.

Accordingly, the system can provide for the ability to automatically assess the complexity of new features across a project and resolve dependencies between features and open bugs based on complexity, business requirements, source code and current developers work assignment. The system can automatically suggest an efficient "next new feature" development to any combination of developers working on bugs that are semantically related to the new feature, developers working on bugs that touch at the same part of the code as the new feature, and/or developers who have enough bandwidth to work on the new feature. The system can also provide an overview of the dependencies between new features and open bugs (e.g., the subgraphs constructed by the AWAU), which can be visually represented in an interactive dashboard.