A Method for Change Impact Analysis of C# Projects

Hoang-Viet Tran, Nguyen Thi Mai Loan, Doan Duc Kien, Nguyen Ha Trang, Le Van Huy, Pham Ngoc Hung
VNU University of Engineering and Technology, 144 Xuan Thuy str., Cau Giay dist., Hanoi, Vietnam

Correspondence: Hoang-Viet Tran, thv@vnu.edu.vn
Communication: received 21 Jun 2023, revised 15 Oct 2023, accepted 26 Nov 2023
DOI: 10.32913/mic-ict-research.v2024.n1.1211

***Abstract*: Change Impact Analysis (CIA) is a common technique for identifying the potential effects of a change in software on other components. Despite the popularity of C# in the software industry, there have not been any CIA methods for C# projects. This paper proposes a new method for performing static CIA in C# projects based on existing methods for object-oriented languages. The main idea behind the method is constructing a dependency graph by doing static source code analysis. This enables the use of WaveCIA, a CIA algorithm leveraging dependency graphs. The implementation of this method in a CIA tool has proven its effectiveness in determining impacted components in C# projects, which can be further used for regression testing.**

Change Impact Analysis (CIA) is a common technique for identifying the potential effects of a change in software on other components. This solution has been made to analyze the impact of code changes in different languages like Java, C/C++, etc. Despite the popularity of C# in the software industry, there have not been any CIA methods for C# projects. This paper proposes a new method for performing static CIA in C# projects based on existing methods for object-oriented languages, to generate dependency graphs of that project's source code. The key idea behind the method is to construct a dependency graph by doing static source code analysis. This enables the use of Wave-CIA, a CIA method leveraging dependency graphs. The results of analyzing the source code to build the dependency graph between the components in the project have been shown in the CIA4CS tool. We did some tool testing with several C# projects and the results showed high coverage for components and dependencies, that has proven its effectiveness in determining impacted components in C# projects. Finally, we provide some methodological discussion at the end of the paper.

Keywords: *Internet of Things (IoT), deep learning, IoT device identification, attention features fusion method (AFF), IoT device detection, feature fusion; image processing.*

I. Introduction

In the software life cycle, changing requirements often happens because of various reasons including adding new features, changing customer requirements, improving performance, etc. However, a small change in the source code can have some unpredictable effects and hidden bugs deep inside the application that even programmers and testers are not aware of. As a result, change requests introduce several challenges for software maintainers. The developers have many difficulties in identifying which modules need to be modified to accomplish a change. Testers must minimize the number of test cases for regression testing. The project manager and higher-level managers are faced with reducing costs and time for change management. As a result, determining how changing source code components affects other components is one of the main problems in the software development project and the software life cycle. The optimal and effective solution to solve the above problem is the analysis of the effects of changes (Change Impact Analysis - CIA [1], [2]). CIA helps identify the affected parts of the software application when making changes to its source code in a new version. For this reason, the CIA is an important solution in software development and maintenance, especially in reducing the maintenance effort in many senses.

Many researchers published many papers addressing the CIA problem [3–5]. In 2007, Huang and Song proposed a dynamic impact analysis based on program executions [4]. They considered the characteristics of object-oriented programs and performed dependency analysis by removing elements that do not have a dependency on the changed elements. In 2010, Sun et al. presented a static CIA method that considers different impact mechanisms and rules of different change types, to calculate the impact sets [3]. They present an intermediate representation for object-oriented Java programs, then perform CIA based on the impact rules of different change types. In 2020, Mondal et al. [5] proposed a method to find impact sets using rule association and code clone. The rule association relies on the previous change history of program entities. Code

clones are detected using a clone detector. Mondal et al. performed the investigation on some projects written in Java, C++, and C#. Although many methods have been proposed representing many different viewpoints addressing the CIA problems, there is no detailed method that applies Wave-CIA for C# projects.

In practice, the CIA solution has been implemented to analyze projects in many different languages on a number of tools. First, JRipples[1] is a tool for program comprehension during incremental change. It manages the organization of the steps that comprise the impact analysis and subsequent change propagation [6]. However, this tool only supports Java programming language, not C#. Second, JArchitect[2] is a tool that supports a large number of code metrics and allows visualization of dependencies using directed graphs and a dependency matrix. The tool also performs code base snapshots comparison and validation of architectural and quality rules. JArchitect also supports Java, not C#. Third, JCIA is a tool for change impact analysis of Java EE applications. This tool analyzes the source code of Java EE applications for building the dependency graphs (called JDG) [7]. Fourth, CodeSurfer [8] is a popular code visualization tool used to explain and display source code clearly and understandably. It can be used to create charts and graphs to help users visualize and track the flow of data and the execution process of the source code. It supports programming languages like C, C++, Java, and Ada. However, CodeSurfer does not support C#. Finally, Tochal [9] is a tool that implements a DOM-sensitive hybrid change impact analysis technique for JavaScript through a combination of static and dynamic analysis. Tochal also does not support C#. Consequently, the above tools support many programming languages, not C#, which is one of the most popular programming languages.

This paper introduces a Wave-CIA-based method, named CIA4CS, for analysis of the impact of changes in C# projects. It is based on WAVE-CIA, a novel CIA approach based on call graph mining, which until now, no one has applied and implemented for C#. In CIA4CS, two versions of the project source code are analyzed and converted into two dependency graphs, respectively. Then, CIA4CS compares the two dependency graphs to find the components that are changed between the two versions of the source code. Those components will be put into the execution of a CIA method named WAVE-CIA to find the affected components. There are two main contributions of CIA4CS. First, CIA4CS provides detailed algorithms for the process from code analysis to change computation. Second, CIA4CS applies a well-known method named WAVE-CIA in C# projects that there has been no published research before. A tool supporting the CIA4CS has been built and applied for testing a number of C# projects. The test results show that CIA4CS gives high accuracy in analyzing the effect of changing one component on other components.

The rest of this paper is organized as follows. Section II presents some background concepts being used in this paper. Next, Section III shows the details of CIA4CS methods. Then, Section IV describes the CIA4CS Tool which includes the implementation of CIA4CS. Initial experiment results and discussions are presented in Section V. After that, related works to CIA4CS are discussed in Section VI. Finally, we conclude the paper in Section VII.

II. BACKGROUNDS

In this section, we present some background concepts which will be used in this paper.

1. Change Impact Analysis

Change Impact Analysis (CIA) [1, 10] is an important solution in the process of software development and maintenance. The basis of this approach is to evaluate whether a component is affected or not based on its dependency on the changed component when analyzing the source code. From there, we can predict the scope and impact of the change on the code and recommend components that need to be retested. This is an effective solution for all stakeholders in software projects. For testers and developers, CIA is used to identify test cases that involve changed components when the source code changes. As a result, it is possible to reduce the number of test cases that need to be performed, reducing the time, effort, and cost required for testing while ensuring high-quality software. For project managers, based on the results of the analyzed impact, they can evaluate the level of impact to estimate time, costs, etc. In addition, it allows investors and customers to determine the total cost to invest in the change.

2. An Overview of C#

C# (or C sharp) is a simple, modern, object-oriented, powerful, and versatile programming language developed by Microsoft based on C++ and Java [11]. C# has a syntax quite similar to those languages. However, it has been developed and improved to be simpler and more accessible. C# comes along with and is an important part of the .NET framework. Currently, C# is an open-source programming language, is fast, and can be compiled on different computer platforms (Windows, Linux, and Mac OS). Although it is an object-oriented programming language, the latest versions of C# also support some features of functional languages

[1] https://en.wikipedia.org/wiki/JRipples
[2] https://www.jarchitect.com/

such as *immutable data types*, *delegate*, *local function*, etc. For this reason, analyzing C# source code becomes more difficult than analyzing the source code of other object-oriented languages, which only have *class*, *method*, and *field*. In fact, C# source code is managed with two levels: *project* and *solution*.

- *Project* is the basic level of management. Each project consists of one or more C# source code files. After the compilation process, each project is corresponding to a generated executable file or library. Project information is contained in a file with the extension *.csproj*. Normally, all project components are located in one folder.
- *Solution* is a higher level of source code management. A solution consists of one or more projects. Projects in the same solution can reference each other, allowing one project to reuse source code from another project.

3. Abstract Syntax Tree

An abstract syntax tree (AST) is a data structure that represents the structure of the source code. AST focuses on representing the logical components of the source code. Each node in the tree represents a structural object that appears in the source code, such as a name, function, etc. Nodes can contain child nodes which are more specific components of the parent node. For example, an expression node can include child nodes such as name, type, and value. AST is commonly used in compilers and is often the result of the source code parser's parsing phase. AST can be seen as an intermediate structure to represent various forms of source code.

4. Project Component Structure Tree

The structure tree is the result of analyzing the project source code. Each element in the structure tree represents an actual component of the project. On the branch level, the root represents the root of the C# project followed by sub-components at each level, and so on until the most granular level. Component types are defined in accordance with the components defined in the C# project.

Table I lists all the component types in the C# project tree and their meaning in a software project.

TABLE I
COMPONENT TYPES IN THE C# PROJECT COMPONENT TREE STRUCTURE

ID	Component	Define
1	Unknown	Unknown components
2	Class	Classes in C# project
3	Interface	Interface in a C# project
4	Struct	Representing the struct data type in C# project
5	Enum	Representing the enum data type in C# project
6	Field	Field in a C# project
7	Property	Property in a C# project
8	Method	Method in a C# project
9	Delegate	Representing delegate in C# project
10	LocalFunction	Representing Local Function in C# project
11	File	Files in C# project
12	Folder	Folders in C# project
13	Project	C# Project Root Component

TABLE II
C# PROJECT DEPENDENCIES

ID	Dependency	Description
1	*member*	Expressed when the wrapper component contains another component in the project.
2	*inheritance*	A class inherits another class, a class implements an interface, an interface inherits another interface
3	*use*	A method uses another component in the project.
4	*invocation*	A method calls another method
5	*override*	Represents when a method overrides another

5. Dependency Graph

The project component structure tree contains only the components. For this reason, it only describes the architecture of components of a project. Thus, it is not enough information to use the project component structure tree for CIA purpose. The reason is that it does not show the dependency relationship between two arbitrary components. To visualize the components in the project and the relationships between those components, we represent them with a dependency graph.

Definition: *The dependency graph G is defined as a pair $\{V, E\}$ where $V = \{v_0, v_1, \ldots, v_n\}$ is a list of nodes that represent components like class, method, property, etc. and $E = \{(v_i, v_j) \mid v_i, v_j \in V\} \subset V \times V$ is a list of directed edges. Each edge (v_i, v_j) represents a dependency between v_i and v_j which means v_i depends on v_j.*

All dependency types are presented in Table II. These types of dependencies represent relationships between components on the dependency graph. If component A has one of those five dependencies with component B, then A depends on B. For example, if A uses B, there will be an arrow from node A to node B on the dependency graph.

6. Libraries Used In Source Code Analysis

a) Microsoft.CodeAnalysis.CSharp

Microsoft.CodeAnalysis.CSharp is a library published by Microsoft. This library provides API to generate abstract syntax trees from C# source code, interacts with the nodes in the tree, and provides API for analyzing static bindings.

We can use this library to find the definition of an identifier in the source code. For example, we can find where the definition of a function that is being called in the function call, or we can find the definition of a data type when we declare a variable. These features are necessary for building the dependency graph for analyzing the impact of change. The dependency graph is generated by traversing the vertices of the abstract syntax tree. The edges of the graph are generated by defining the associations in the statements.

b) Microsoft.CodeAnalysis.CSharp.Workspaces

Microsoft.CodeAnalysis.CSharp.Workspaces is a library developed based on Microsoft.CodeAnalysis.CSharp. It has extensive features to help process the source code of C# solutions and projects. Using this library is necessary since C# projects are often organized into solutions and projects whilst Microsoft.CodeAnalysis.CSharp mainly works with individual pieces of C# code. In our tool implementing CIA4CS, this library is used to determine and obtain the AST of C# source code files in a solution when the path to that solution file is known.

7. Change type

When comparing two versions of source code, we need to find the difference between them, specifically the set of changed components. The set of changed nodes is called the change set (CS). Changes can be classified into three categories: addition, deletion, and syntax change. In particular, the type of "syntax change" covers all types of component changes (except additions and deletions). This includes body changes, return types, declarations, etc. These types of components are shown in Table III

TABLE III
NODE CHANGE TYPES

No.	Change type	Description	Example
1	Deletion	Delete a component of the old version	Remove a property in class C
2	Addition	Add a component that the old version does not have	Add a method in class C
3	Syntax change	Change a component that is already in an old version	Change the return type of method M

In addition to the change nodes, we also have impact nodes. These nodes represent the components affected by the change from the previous version.

8. WAVE-CIA

WAVE-CIA is a CIA method proposed by Li et al. [10] in 2013. The method is to compute a set of impacted components from a change set and dependency graph. The original idea of the WAVE-CIA method is "the propagation of water waves". By throwing some stones into a quiet water surface to create a center of propagation, waves are formed on the surface of the water. These water waves interfere with each other and the intersection points become new centers of propagation. Let's consider the dependency graph as a water surface and the change set as the set of stones thrown into the water. The key idea of WAVE-CIA method is that a component is more likely to be impacted if it has dependency relationships with more changing components. These changed components will propagate the impact to other components in the graph. The intersection component mentioned above is called the *core* set defined as follow:

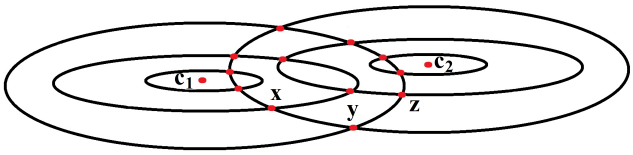


Figure 1. An example of the *core* set.

Definition 1: (Core set) Core set is a set of components that are cohesive and more likely impacted by multiple changed components in the change set.

In Figure 1, let components c_1 and c_2 be two components in the change set. Then, they propagate their impact waves to other components in the given graph. Components $\{x, y, z\}$ are put in the core set because both c_1 and c_2 can reach them.

To compute the core set from a changed set and a dependency graph, we use the neighbor set (NS) of vertices that do not belong to the changed set in the dependency graph. If the neighbor set of a vertex has multiple components that belong to the changed set, then the vertex belongs to the core. The neighbor set of a component is defined as follows:

Definition 2: (Neighbor set) Given a vertex in a dependency graph, its neighbor set is a set of vertices that are reachable from it within a given distance. The distance between two vertices in a dependency graph is defined to be the least number of edges needed to move from one vertex to the other.

III. CIA4CS: A CIA METHOD FOR C# PROJECTS

This section gives a detailed description of the proposed method for performing change impact analysis of C# projects. For a given C# project source code, the method consists of three main phases. First, an AST is obtained using Microsoft.CodeAnalysis.CSharp library. Next, a dependency graph is generated by traversing the AST in the

depth-first search order. For the comparison of two source code versions of the same project, two corresponding dependency graphs are generated and compared to identify the change set. Finally, the execution of a CIA algorithm with the change set as input determines the core set and impact set. In this paper, we employ the Wave-CIA [10] method to perform the change impact analysis. The overview of CIA4CS is shown in Figure 2. The following sections present each phase in more detail.

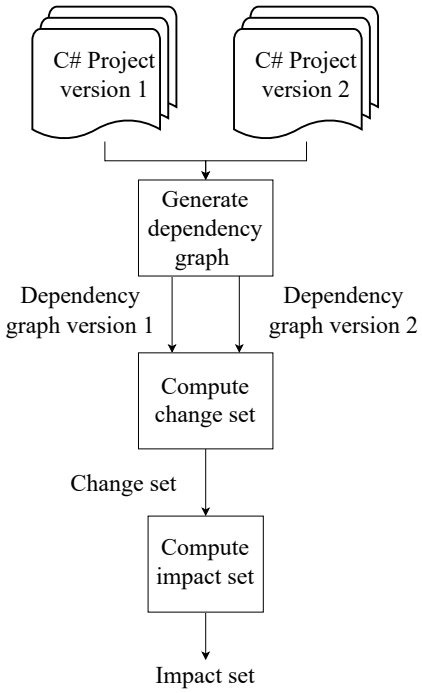


Figure 2. An overview of CIA4CS.

1. Dependency Graph Generation

In this phase, a dependency graph is constructed based on the AST of C# project source code. The construction process contains three main steps which are shown in Figure 3.

- Step 1: Starting from the project directory, CIA4CS filters out the C# source files (*.cs) and builds a tree of directories and files.
- Step 2: For each source code file, the Microsoft.CodeAnalysis.CSharp library is used to generate the corresponding AST.
- Step 3: Traverse the generated AST to obtain the necessary information about the components such as classes, methods, properties, etc. For each appropriate node in the AST, a new node is added to the dependency graph. Then, its dependencies with other nodes are created.

In Step 1, the path to a C# project directory, which consists of C# source files needed for analysis, is used to

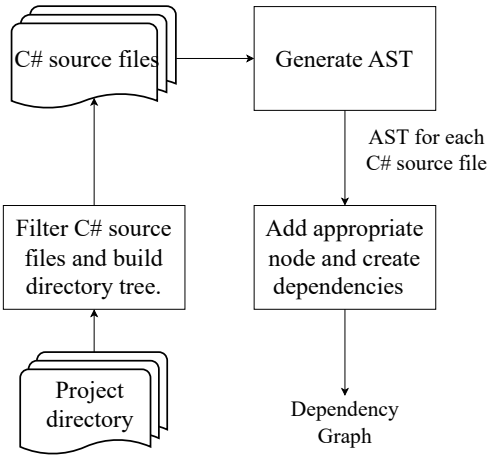


Figure 3. Dependency graph generation.

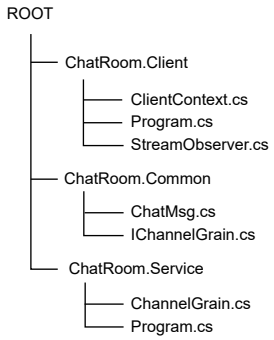


Figure 4. Directory tree for the ChatRoom project

construct a directory tree. A C# project directory is usually composed of a .sln file and one or more .csproj files, which contain information about the project's dependencies and runtime. The directory tree is built with the directory that contains the .sln file as root and other nodes corresponding to the files and directories hierarchy. Figure 4 illustrates a directory tree of ChatRoom[3] project, a sample C# online chat application.

In Step 2, the AST for each C# source file is constructed using Microsoft.CodeAnalysis.CSharp. This process contains two following smaller steps. First, a concrete syntax tree is produced by calling the method CSharpSyntaxTree.ParseText, which takes C# source code as an argument. A concrete syntax tree is a complete representation of a program's source code, including nodes corresponding to keywords, parentheses, operators, etc., referred to as syntax tokens in the library. Second, the concrete syntax tree is pruned to obtain a more compact version which is the corresponding abstract syntax tree (AST). This can be done by removing syntax tokens from the concrete syntax tree.

[3]https://github.com/dotnet/samples/tree/main/orleans/ChatRoom

```
1  int gcd(int a, int b)
2  {
3    if (b == 0)
4      return a;
5    return gcd(b, a % b);
6  }
```

Listing 1: Example program

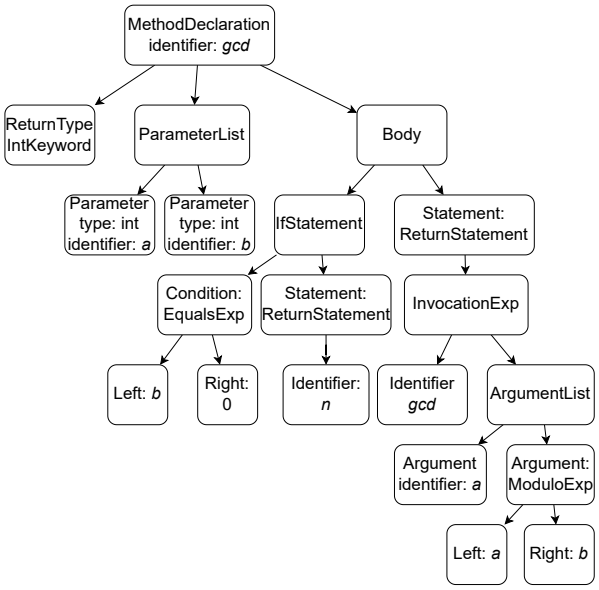


Figure 5. AST for the example program.

To illustrate this process, a simple C# method that finds the greatest common divisor (GCD) of two integers is given in Listing 1. The corresponding AST is shown in figure 5.

In Step 3, for each node in the AST, a corresponding node is added to the dependency graph. The detailed process is described in Algorithm 1. The main idea of the algorithm is to traverse the AST tree in the depth-first search order. For each node in the tree, CIA4CS creates an edge connecting that node and its parent, then performs static binding to locate dependent nodes, and recursively processes all child nodes.

The algorithm starts by checking if there are nodes in the dependency graph corresponding to the node we are traversing in the AST tree (lines 3-4). If not, the algorithm creates a new node with the identifier (ID) being the fully qualified name of the node in the AST tree (lines 5 - 6). The ID of a node is generated by Microsoft.CodeAnalysis.CSharp library via binding resolution. For instance, if field B is inside class A, which is wrapped in namespace N, its ID will be N.A.B. Since there cannot be two fields with the same name inside the same class, this hierarchical naming system results in uniquely defined IDs. For methods, in addition to

Algorithm 1: Generate dependency graph

```
1  Input: ast_node: root of the AST; parent_node:
     parent node in dependency graph; dep_graph: the
     dependency graph to be generated
2  begin
3      node_id ← fully qualified name of ast_node;
4      graph_node ← find node with ID node_id in
         dep_graph;
5      if graph_node = NULL then
6          graph_node ← create new node in dep_graph
             with ID node_id;
7      end
8      if parent_node! = NULL then
9          Create new edge from parent_node to
             graph_node with MEMBER dependency type;
10     end
11     Process inner statements, return types, etc. to handle
         other dependency types;
12     foreach child_node in ast_node.children do
13         call Algorithm 1
             (child_node, graph_node, dep_graph);
14     end
15 end
```

their name, their parameter types are also present in their ID. This can prevent two overloads of a method from being assigned identical IDs. For example, suppose two methods named C are members of class A mentioned above. The first one accepts a string as a parameter, making its ID N.A.C(string), while the ID of the second one might be N.A.C(int, int) if it requires two integer inputs. Back to the algorithm, Then, we proceed to create an edge between the parent node and the child node to represent the MEMBER type dependency between the parent and the child (lines 8 - 9). Thus, a project component structure tree is a tree that includes all vertices and edges whose dependency type is MEMBER. This is followed by processing inner statements for methods, or return types for variables, properties, and so on to create edges in the graph with other types of dependencies (line 11). This step is primarily concerned with a technique called semantic analysis. The semantic analysis attempts to obtain extra information about an AST expression or statement node, particularly what declaration node a variable or a method call refers to. This is useful to the algorithm as the ability to determine which node a method call or a type refers to enables the addition of appropriate edges. In Microsoft.CodeAnalysis.CSharp, semantic analysis is provided via the SemanticModel class, which contains the semantic information of a C# source file. Specifically, each node in the AST has an associated “symbol”, which contains information regarding its declarations, type, references, etc. The symbol can be accessed by calling the method SemanticModel.GetSymbolInfo, which takes a node in the syntax tree as input. The result is a generic

symbol object that can be further cast to a specific symbol type such as method, class, or interface to obtain more concrete information. For instance, a symbol associated with a method call provides pieces of information about the corresponding method such as its return type, parameters, and parent class. Finally, the algorithm recursively calls itself with each children node of *graph_node* (lines 12 - 13). Each child node, referred to as *child_node* in the algorithm, is a child of the AST node used as input. To generate the dependency graph, Algorithm 1 is executed with three inputs: the root of AST (*ast_node*), *NULL* (*parent_node*), the required dependency graph (*dep_graph*).

2. Change Set Computation

After generating dependency graphs for the two versions of the source code, CIA4CS finds a set of nodes that are changed in the new version (hereafter called version 2) in comparison with the old version (hereafter called version 1). Algorithm 2 describes the process in detail. The algorithm takes two dependency graphs A and B generated from version 1 and version 2 of the source code as inputs, respectively. Some examples of each type of change are given in Table III.

Algorithm 2 begins by declaring sets of nodes (*change_set*, *deletion_nodes*, *changed_nodes*, *added_nodes*) corresponding to the change types and initializing them as an empty set (∅) (lines 4 - 7). Then, for each node in graph A, the algorithm finds a node with the same ID in graph B (lines 8 - 9). It is possible that graph B contains nodes with the same IDs as class A as long as their signatures do not change. In other words, if a node and its parents keep their name in the new version, which means its ID is unchanged, it can be found using its ID. If this is correct, the corresponding syntax tree of the two nodes are compared to see if there is a change in syntax (lines 10 - 11). If there is a change in syntax, the node is changed in version 2 in comparison with version 1. The algorithm adds it to *changed_nodes*. If the node cannot be found, it is added to the set of deleted nodes (*deleted_nodes*) (lines 14 - 15). Similarly, the algorithm iterates through all nodes in graph B to find the added nodes and adds them to *added_nodes* (lines 18 - 23). Finally, the algorithm returns the union of *added_nodes*, *deleted_nodes*, and *changed_nodes* as the required change set.

3. Compute The Impact Set

In this paper, we use the WAVE-CIA which is a well-known CIA algorithm proposed by Li et al. [10] in 2013. The algorithm is to compute a set of impacted components from a change set and a dependency graph. Given

Algorithm 2: Change set calculation

```
1  Input: A: dependency graph of version 1; B: dependency graph of version 2.
2  Output: change_set: the set of nodes which was changed/deleted/added in the second version.
3  begin
4      change_set ← ∅ ;
5      deleted_nodes ← ∅;
6      changed_nodes ← ∅;
7      added_nodes ← ∅ ;
8      foreach nodeA in A.nodes do
9          nodeB ← find node in B with the same ID as nodeA ;
10         if nodeB != NULL then
11             if the syntax tree of nodeA and nodeB do not match then
12                 changed_nodes ← changed_nodes ∪ {nodeA};
13             end
14         else
15             deleted_nodes ← deleted_nodes ∪ nodeA;
16         end
17     end
18     foreach nodeB in B.nodes do
19         nodeA ← find node in A with the same ID as nodeB;
20         if nodeA = NULL then
21             added_nodes ← added_nodes ∪ {nodeB};
22         end
23     end
24     change_set ← added_nodes ∪ deleted_nodes ∪ changed_nodes;
25     return change_set;
26 end
```

a change set (CS) and a dependency graph, WAVE-CIA algorithm consists of two steps to compute the impact set as follows: Step 1: Core set computation; Step 2: Impact set computation. Sections below present more details about the computation process.

a) Compute Core Set

As mentioned above, the core set consists of components that are likely to be impacted by multiple changes. The computation of the core set is described in Algorithm 3. The algorithm accepts the dependency graph (*dep_graph*), the change set (*change_set*), and the given limited distance (*depth*) as inputs. After processing, the algorithm returns the core set (*core_set*) as its output. For each vertex in the dependency graph (*dep_graph*), the algorithm retrieves its set of neighbors with a limited distance of depth (*depth*) (line 5 - 7). If the number of neighbors in the change set of a vertex is greater than one, the vertex will be pushed into the temporary core set (*temp_set*) (lines 9 - 10). The final core set is the union of the temporary core set (*temp_set*) and the change set (*change_set*) (line 14).

Core Set Computation Example

Algorithm 3: Core computation

```
1  Input: dep_graph: dependency graph; change_set: set
     of changed components; depth: limited distance.
2  Output: core_set: core set.
3  begin
4      temp_set ← ∅;
5      foreach vertex in dep_graph.V do
6          if vertex ∉ change_set then
7              neighbor_set ← get_neighbor_set(vertex, depth) ;
8              neighbor_change_set ← neighbor_set ∩ change_set;
9              if |neighbor_change_set| > 1 then
10                 temp_set ← temp_set ∪ {vertex};
11             end
12         end
13     end
14     core_set = change_set ∪ temp_set;
15     return core_set;
16 end
```

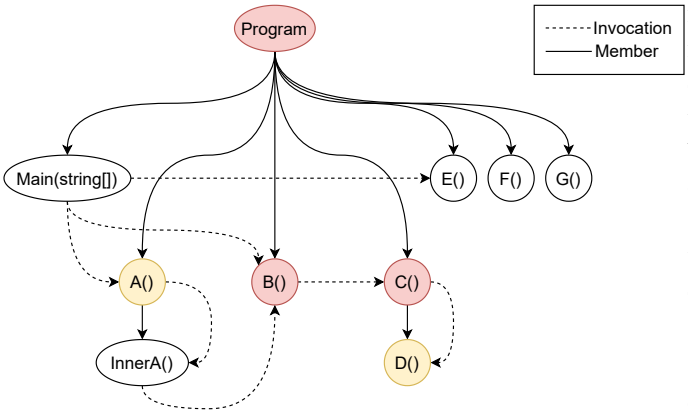


Figure 6. An example of core set.

Let's consider an example shown in Figure 6. In this case, we set the limit distance $depth$ to 2 and the change set to $A(), D()$ (in yellow). We compute the core set for the dependency graph in this example using the following steps:

1) We identify the set of neighbors of vertices that are not in the change set (i.e., $E()$, $F()$, $G()$, $B()$, $C()$, $Main(string[])$, $Program$, $InnerA()$). For example, $E()$ has a neighbor set of $\{C(); A(); Main(string[]); Program; E(); F(); G()\}$. The reason is that these vertices can be reached from $E()$ within 2 edges. Similarly, the neighbor set of $B()$ is $\{C();\ D();\ A();\ InnerA();\ Main(string[]);\ Program;\ E();\ F();\ G()\}$, and so on.
2) We identify all vertices belonging to the neighbor set which are in the change set. For instance, the neighbor set of $E()$ contains $A()$ which is in the change set. In the meantime, the neighbor set of $B()$ has $A()$ and $D()$ which are in the change set, etc.
3) Since the number of neighbors in the change set of $E()$ is 1 (not greater than 1), $E()$ is not in the core set. On the other hand, $B()$ has 2 neighbor vertices in the change set (greater than 1), so $B()$ is in the core set. Similar computation for the remaining vertices, we obtain a core set as follows $\{B(), C(), Program\}$ (in red).

Identifying the core set helps increase the accuracy of the impact set. The core set obtained from vertices not in the changed component set will be combined with the change set to compute the impact set. In the above example, the core set will be $\{B(), C(), Program, A(), D()\}$.

b) Compute Impact Set

Although the core set can give the impact set with few false positives, there are other components that are really impacted but not included in the core. To reduce false negatives, we need to expand from the core set to find the impact set. To obtain a complete set of impacted components, the key idea of the process is as follows. In the beginning, the impact set is initialized as the core set. With each vertex that is not in the impact set, we can obtain the set of components that depend on it and the set of components it depends on. If both of these sets exist vertices belonging to the impact set, the components under consideration will be added to the impact set.

Details of the impact set computation process are described in Algorithm 4. The algorithm accepts the dependency graph (dep_graph) and the core set ($core\_set$) as its inputs. When the algorithm stops, it returns the impact set ($impact\_set$) as its output. When the algorithm starts, it initializes the impact set ($impact\_set$) as core set ($core\_set$) (line 4). For each vertex ($vertex$) that does not belong to the impact set, the algorithm finds the set of components that depend on it ($depender\_set$) and the set of components it depends on ($dependee\_set$) (lines 8 - 9). If both sets contain some impacted components then that vertex is added to the impact set (lines 10 - 11). These steps are repeated until the impact set remains unchanged and the final impact set ($impact\_set$) is returned.

Impact Set Calculation Example

Let's consider an example shown in Figure 7 with the given core set $\{B(), C(), Program, A(), D()\}$ (in red). To calculate the impacted component set, we consider the core set as the initial impact set. Then, we iterate over the vertices that are not in the impact set.

In the first iteration, the vertex $Main(String[])$ depends on vertex $A()$ and is depended upon by vertex $Program$. Since both vertices are already in the impact

Algorithm 4: Impact Set Computation

```
1  Input: dep_graph: dependency graph; core_set: core
     set.
2  Output: impact_set: set of impact components.
3  begin
4      impact_set ← core_set;
5      while impact_set is unstable do
6          foreach vertex in dep_graph do
7              if vertex ∉ impact_set then
8                  depender_set ←
                     get_depender_set(vertex);
9                  dependee_set ←
                     get_dependee_set(vertex);
10                 if (impact_set ∩ depender_set ≠ ∅) ∧
                     (impact_set ∩ dependee_set ≠ ∅) then
11                     impact_set ← impact_set ∪
                         {vertex};
12                 end
13             end
14         end
15     end
16     return impact_set;
17 end
```

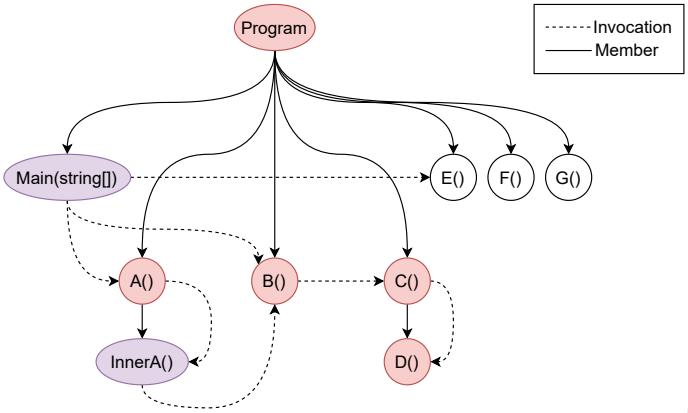


Figure 7. Impact set of dependency graph.

set, $Main(String[])$ belongs to the impact set. Similarly, the vertex $InnerA()$ also belongs to the impact set. For this reason, the two vertices $\{Main(String[]), InnerA()\}$ (in purple) are added to the impact set.

In the second iteration, since there is no more vertex that can be added to the impact set, the algorithm ends and gets the final impact set as $\{B(), C(), Program, A(), D(), Main(String[]), InnerA\}$. The impact set includes $Main(String[])$ and $InnerA()$. This result shows that a more accurate impact set has been constructed.

IV. Implementation

We have implemented the CIA4CS method in a tool named CIA4CS Tool by using C# programming language. CIA4CS Tool can compute the impact set of a certain change set in a given C# project. For a given project, when the user uploads the project folder to CIA4CS Tool, it analyzes and displays the project structure and the dependency graph of components in the project.

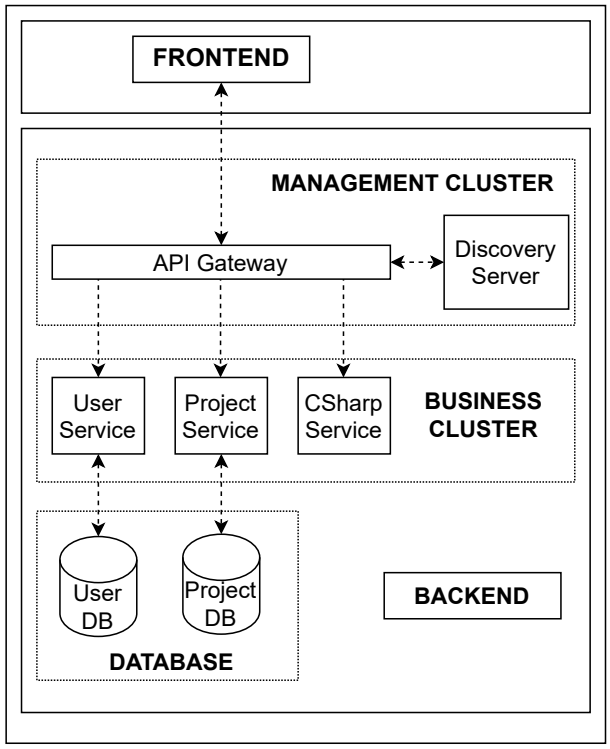


Figure 8. System architecture.

CIA4CS Tool is implemented using the microservices architecture which is based on the JCIA architecture [7]. The architecture of CIA4CS Tool is shown in Figure 8. The tool is built as a website, where the front-end and back-end communicate with each other using API via an API gateway. The back-end of CIA4CS Tool includes Management Cluster and Business Cluster. In which, Management Cluster contains services related to the network navigation, coordination, and management. In the meantime, Business Cluster contains services that directly affect the user interface:

- **User Service:** This service provides user-related functions, including account registration, login, and logout.
- **Project Service:** This service is responsible for managing projects uploaded to CIA4CS Tool. The provided functions include saving and getting the list of projects and versions.
- **CSharp Service:** This service is used to analyze C# source code. The provided functions include comparing versions and computing the impact set.

In CIA4CS Tool, CIA4CS is implemented in CSharp Service. Other services are inherited from the JCIA tool [7].

CIA4CS Tool allows the user to upload the source code of a C# project as a .zip file or a link of a Git project.

Figure 9 shows a screenshot of a CIA4CS session. The project is represented as a dependency graph. We can see that in the Dependency graph section in figure 9. The rectangle nodes represent the components of the C# project. In the normal state, these nodes are black. With the CIA4CS tool, users can select any node and set it as a changed node. After being set, these nodes will be displayed in the Change set section. In addition, in the Dependency graph section, selected changed components are represented by yellow nodes. Then, we can determine the impact set. In the Dependency graph section, nodes in purple are the components impacted by the selected change element. These components are displayed as a list in the Impact set section.

In addition, the CIA4CS Tool has a function to compare two given source code versions of a project. A screenshot of this function is shown in Figure 10. The project and its components are shown in the Version Compare section. Normally, project nodes are shown in black. When a node has a different color, this means that the node is either changed or impacted by some changes. In Figure 10, nodes in green are added whilst nodes in red are deleted and nodes in yellow are changed nodes. Finally, nodes in purple represent the impact set of the new version in comparison with the old version. In addition, the impact set is also displayed in the Impact nodes section.

V. Experiments

We have performed experiments to evaluate the method with C# projects. The experiment was conducted on one open-source library and one project which are publicly available on GitHub[4], one of the world's most famous source code-sharing websites. The tested source code is as follows:

- **eShopOnWeb**[5]: This is a Microsoft's reference ASP.NET Core project. The project is close to industrial projects and large in size.
- **CoreSearch**[6]: These are search library projects that are small and easy to control.
- **LiteNetLib**[7]: This is a UDP library for Mono and .NET.

For each of the above GitHub repositories, two versions are compared at the *project* level (which corresponds to .csproj files) to obtain the impact set. For eShopOnWeb, the two versions tagged as netcore2.1 and netcore2.2 are chosen as the old and new versions, respectively. For LiteNetLib, the candidate versions are 0.9.5.2 and 1.0.1.1. For CoreSearch, since there is no published release, we have tested the two commits: 64dd4858cfb37016f954096110a340d0382df212 (for the old version) and cab1aa05ce9c62b571fe075f8c3f0b18868805ed (for the new version). In these experiments, we use $depth = 2$ when computing the core set. The effectiveness of the proposed method is evaluated through some key indicators: the number of detected impacted components, the percentage of impacted nodes, the time required, and the memory consumed. Performance metrics are measured using the BenchmarkDotNet[8] library. The number of C# source code files and the number of lines of code denote the size of the project.

Experimental results are shown in Table IV and V. In these Tables, the columns are as follows:

- **Project**: the project under test.
- **F1/F2**: the number of source code files in the old and new versions of the project, respectively.
- **LOC1/LOC2**: the number of lines of code in the old and new versions of the project, respectively.
- **Num**: the number of impacted components detected by CIA4CS Tool.
- **Total**: the number of components in the dependency graph.
- **%**: the percentage of impacted components in the dependency graph.
- **Mem**: average memory (in MB) allocated to complete one analysis phase.
- **Time**: the average time (in ms) required to complete an analysis phase.

From the experimental results shown in Table IV and V, we have the following observations.

For eShopOnWeb project:

- The eShopOnWeb project contains three children projects which are Infrastructure, ApplicationCore, and Web.
- The number of impacted components are 144, 155, and 371 for projects Infrastructure, ApplicationCore, and Web, respectively.
- The percentage of impacted components are 84.21%, 77.11%, and 70.15% for projects Infrastructure, ApplicationCore, and Web, respectively.
- CIA4CS Tool can detect all expected impacted component according to *depth* parameter.

For Coresearch project:

[4]https://github.com/
[5]https://github.com/dotnet-architecture/eShopOnWeb
[6]https://github.com/pilotpirxie/coresearch
[7]https://github.com/RevenantX/LiteNetLib
[8]https://github.com/dotnet/BenchmarkDotNet

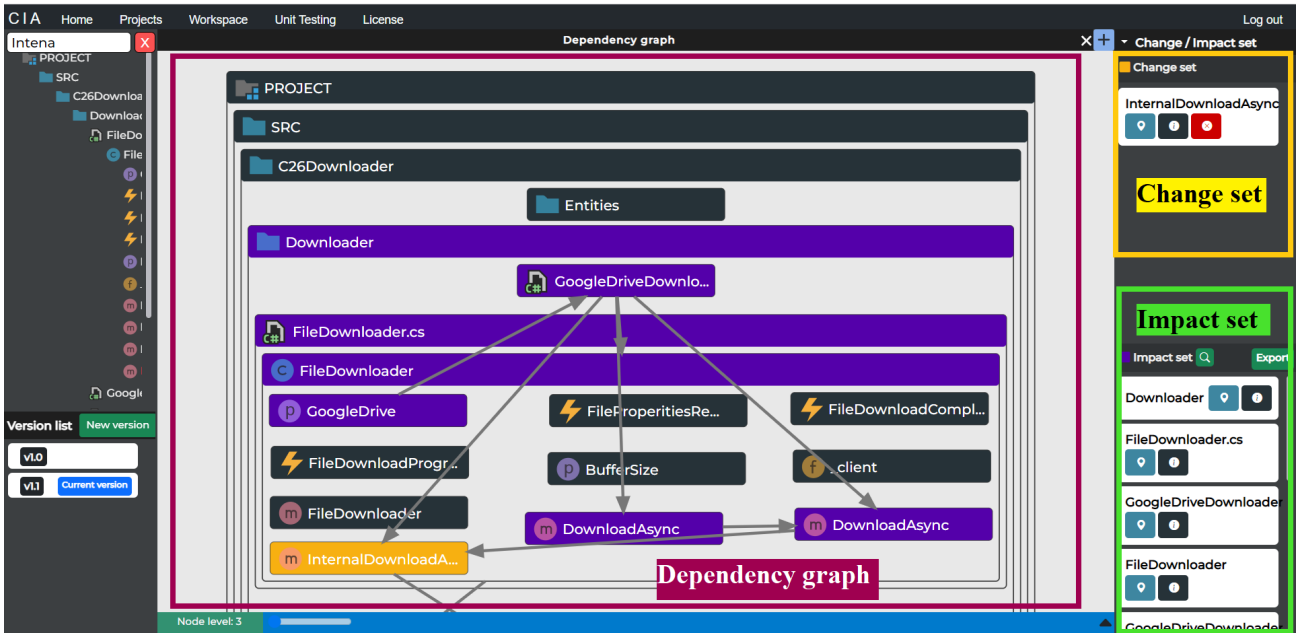


Figure 9. Impact set in CI44CS Tool.

TABLE IV
EXPERIMENTAL RESULTS OF COMPARING TWO VERSIONS OF SOURCE CODE ON ESHOPONWEB

Component	F1	LOC1	F2	LOC2	Num	Total	%	Mem	Time
Infrastructure	17	1664	35	3268	144	171	84.21	81.51	1027.2
ApplicationCore	35	543	35	571	155	201	77.11	82.85	1203.5
Web	42	5985	52	6964	371	529	70.13	156.51	1881.7

TABLE V
EXPERIMENTAL RESULTS OF COMPARING TWO VERSIONS OF SOURCE CODE ON CORESEARCH

Component	F1	LOC1	F2	LOC2	Num	Total	%	Mem	Time
Webserver	3	152	3	167	10	111	9	40.49	542.0
coresearch	6	630	6	645	2	100	2	34.92	468.8

TABLE VI
EXPERIMENTAL RESULTS OF COMPARING TWO VERSIONS OF SOURCE CODE ON LITENETLIB

Component	F1	LOC1	F2	LOC2	Num	Total	%	Mem	Time
LibSample	16	1456	18	1468	150	505	29.70	64.92	975.8
LiteNetLib	34	8368	41	9061	1030	2881	35.75	240.49	2542.0

- The Coresearch project contains two children projects which are Webserver and coresearch.
- The number of impacted components are 10 and 2 for projects Webserver and coresearch, respectively.
- The percentage of impacted components are 9% and 2% for projects Webserver and coresearch, respectively.
- In Webserver, we see the result is not as expected, the actual number of changes is more. The reason is that the Webserver class only changes the namespace. However, the tool considers that one Webserver class is deleted and another Webserver class is added.

For LiteNetLib project:

- The LiteNetLib project contains two children projects which are LibSample and LiteNetLib.
- The number of impacted components are 150 and 1030 for projects LibSample and LiteNetLib, respectively.
- The percentage of impacted components are 29.70% and 35.75% for projects LibSample and LiteNetLib, respectively.
- CIA4CS Tool can detect all expected impacted component according to *depth* parameter.

Regarding the memory and time usage of CIA4CS, we can see that the memory and time used are proportional to

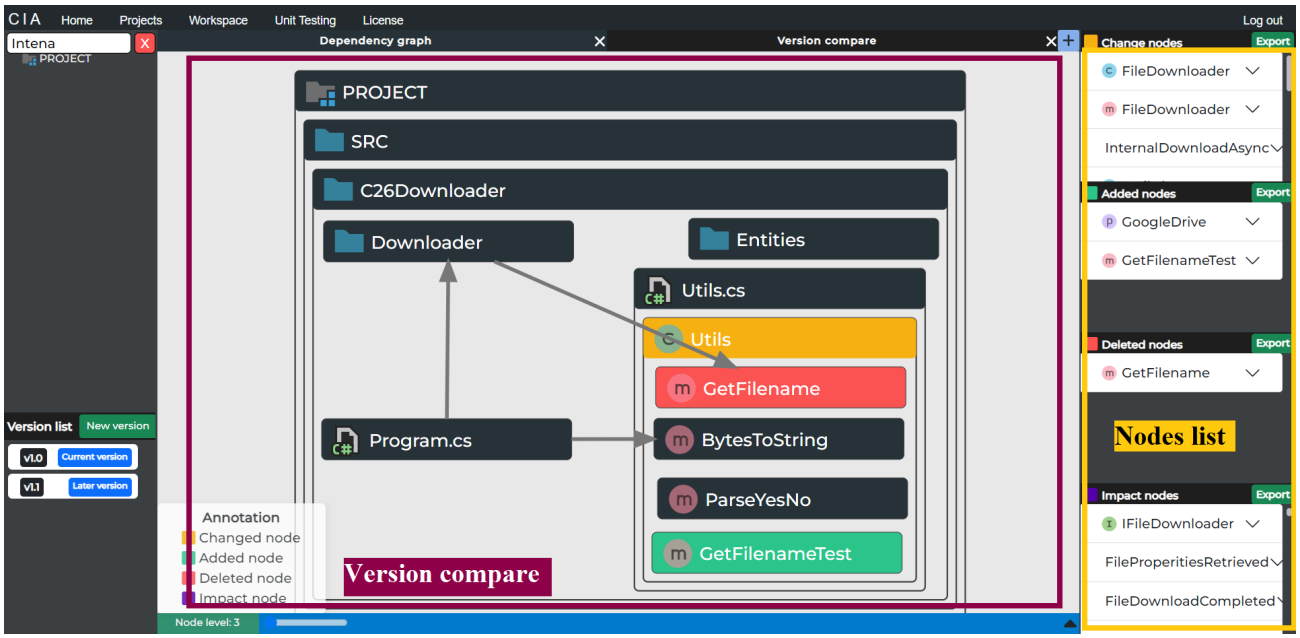


Figure 10. Compare two versions of the source code in the CIA4CS Tool.

the size of the project.

Regarding the acceptance validation of CIA4CS and the experimental results, there are three points as follows.

- CIA4CS is based on Wave-CIA [10] method, which has its own philosophy to consider which components are affected. This should not be compared with other CIA methods. The contribution of CIA4CS is to provide details for C# project source code analysis and dependency graph construction. In turn, the dependency graph will be used as the input for the Wave-CIA method for analysis.
- We do not discuss whether the found impacted components are *true positive* or *false positive* in this paper. The reason is that this depends on the philosophy of the Wave-CIA method. Discussing the correctness of this can be found in the research of Li et al. [10].
- The validation of CIA4CS lies in the source code analysis and dependency graph construction. This is shown through Section III.1.

VI. Related Works

There are many works that have been recently proposed for the CIA method by several authors. Those are CIA methods for many different programming languages such as C++ [12, 13] and other software artifacts [14], etc. In addition, many tools are implemented to support those methods [8], [15], [9], [16], [17], [18].

Chaumun et al. [12] propose a change impact model which is defined at the conceptual level, and map it to the C++ programming language. The conceptual model is comprised of components, relationships, and change types. The impact of a change depends on two main factors the type of change and the type of relationship between both code entities. They are interested in measuring the influence of high-level design on maintainability, and how inter-class dependencies influence changeability. We share the interest in the CIA problem as Chaumun but we focus on solving the problem for C# projects.

Zalewski et al. [13] propose a conceptual change impact analysis (CCIA) which determines the impact of changes in the conceptual specification of generic libraries. The analysis is organized in a pipe-and-filter manner which contains two algorithms. The first one is to detect the impact of changes on the compatibility of different versions of concept specifications. The second one is to detect the impact of changes to the degree of genericity of an algorithm. We share an interest in the CIA problem but we focus on C# projects.

Pirklbauer et al. [14] present a CIA process and a tool that supports many types of visualization methods. The tool is featured with a quality assurance component. In this process, software artifacts such as source code, configuration files, and database scripts are used as the inputs. Then, those artifacts are used to generate a dependency database and to do the CIA process. The process presented by Pirklbauer is a general method and it is not detailed enough for C#. We share the interest of Pirklbauer about the CIA process but we focus on C#.

CodeSurfer [8] is a tool that is available at no cost for C/C++. This tool provides many capabilities to understand a program by providing the results of a static-semantic analysis to the user. It performs a lot of program analyses such as pointer analysis, data flow analysis, and system dependency graphs. However, CodeSurfer does not provide CIA functionalities. We share an interest in program analysis, however, we focus on the CIA functionalities and the C# projects.

Rigi[9] is a visual software understanding tool. Rigi supports reverse engineering, visualization, exploration, and redocumentation [15]. The tool is provided with parsers to retrieve information from the source code. In addition, the tool contains an exchange format to store retrieved, transform, and abstract the information, etc. However, the tool only supports C/C++ and Cobol, not for C#. We share an interest in program analysis but we focus on the CIA and C# projects.

Petrenko et al. [19] are concerned with enhancing impact analysis to cope with a variable granularity of analyzed software artifacts. They argue that a single granularity is insufficient and leads to imprecise analysis. Therefore, they studied how a variable granularity improves the precision of impact analysis, saving the programmers the extra work that is needed to inspect the false positives. They discuss different granularities, including the granularity of classes, the granularity of all program components that include class members, and the granularity of code fragments. They extended the JRipples [20] tool to cope with this variable approach. The algorithms for finding inspected sets at the different granularities were implemented as a plug-in for JRipples. This plug-in substitutes the programmer and simulates all actions from the programmers. We share the interest of CIA for software projects but we focus on units of C# projects by applying the WAVE-CIA method for the dependency graph of C# project.

S.Black et al. [17] proposes a method for the reformulation of Yau and Collofello's ripple-effect algorithm [21]. This method is implemented in the Ripple Effect and Stability Tool (REST) to compute the ripple effect for C programs. REST builds a dependency matrix of the program and applies four different McCabe complexity measures to compute the ripple effect and stability measures for the source code. They performed an evaluation to gauge the correlation between the original algorithm and the REST tool. Then, they concluded that both match, whereas REST achieves more accurate results. We share an interest in improving the quality of maintenance projects but we focus on the CIA of C# projects.

Alimadadi et al.[9] propose a hybrid analysis method for change impact analysis of JavaScript-based web applications that combines the advantages of both static and dynamic analysis techniques to obtain a more complete impact set. This analysis is DOM-sensitive and aware of the event-driven, dynamic and asynchronous entities, and their relationship in JavaScript. It creates a novel graph-based representation capturing these relations, which is used for detecting the impact set of a given code change. Their approach is implemented in a tool called Tochal (TOol for CHange impact AnaLysis)[10]. We share an interest in the CIA problem but we focus on the CIA of C# projects.

Mohamad et al. [22] approach Change Impact Analysis (CIA) method to support the developer to find potential affections or dependency information in source code. The prototype called CIA-V which is a combination of CIA and visualization method based on C++ Object-Oriented language to enhance program understanding ability, will assist the developers to understand the program through diagrams. It also helps to save time to find the affected code. There are three models which are involved to implement the visualization: CATIA [16], Relationship Analysis, and Dependencies Analysis. We share an interest in the CIA problem but we focus on applying the WAVE-CIA [10] for C# projects.

Gwizdala et al. [18] propose the JTracker tool to guide programmers while changing software. This is an extension to the Java environment JBuilder5 and assists a programmer during change propagation. JTracker marks the neighbors which can be impacted when the programmer changes a class and informs the programmers about them. If these changes are necessary, the programmers will make modifications using a standard editor. After that, JTracker automatically marks additional classes. The change propagation continues until no marked classes exist in the program. JTracker scans the source code and creates a database of classes and their dependencies. JTracker was written in Java as an extension of the Borland JBuilder5 environment. We share an interest in the CIA problem. However, we focus on applying the WAVE-CIA [10] for C# projects.

In 2007, Huang and Song introduced a dynamic impact analysis approach that relies on program executions [4]. They took into account the peculiarities of object-oriented programs and conducted a dependency analysis by eliminating components that are not dependent on the modified components. We share an interest in CIA analysis. However, this paper focuses on the Java programming language rather than C#.

In 2010, Sun et al. introduced a static CIA method that

[9]https://rigi.io/

[10]https://github.com/saltlab/tochal

accounts for various impact mechanisms and rules associated with different change types to compute the impact sets [3]. They proposed an intermediate representation for object-oriented Java programs and subsequently performed CIA based on the impact rules of different change types. Sun focused on the Java programming language rather than C#.

In 2020, Mondal et al. proposed a method to find impact sets using rule association and code clone detection [5]. The rule association is based on the previous co-change history of program entities. Code clones are detected using a clone detector. Mondal et al. conducted an investigation on several projects written in Java, C++, and C#. While we share an interest in the CIA problem, our focus lies in applying WAVE-CIA [10] to C# projects.

VII. Conclusion

This paper proposes a Wave-CIA-based method, named CIA4CS, to analyze the impact of changed components of C# projects. The method has been implemented in CIA4CS Tool and used to perform some experiments with some common C# projects in the community. Experimental results show that the tool effectively detects the change set and impact set of the given source code. The tool is a solution to support both the development team and the software company in their daily work in software quality assurance.

In the future, several features of CIA4CS Tool will be implemented for the user to have a more complete solution with a better user experience. First, we will add the ability to analyze the dependencies between *.cshtml files in ASP.NET projects or *.xaml files in WPF projects. In addition, other optimizations can also be implemented such as improving the speed of the dependency graph-building process, comparing versions, improving the algorithm to determine the impact set, or defining existing dependencies. Moreover, the representation of components on the graph for large-scale projects needs to be improved and optimized. Last but not least, some features like the report export and CI/CD integration support need to be added to the tool. By implementing these improvements, we aim to help users have a better user experience, and make the tool capable of being used practically in an enterprise environment.

References

[1] M. Kretsou, A. A. Elvira-Maria Arvanitou, I. Deligiannis, and V. C. Gerogiannis, "Change impact analysis: A systematic mapping study," *Journal of Systems and Software*, vol. 174, no. 110892, (2021).

[2] B. Li, X. Sun, H. K. N. Leung, and S. Zhang, "A survey of code-based change impact analysis techniques," *Software Testing*, vol. 23, (2013).

[3] X. Sun, B. Li, C. Tao, W. Wen, and S. Zhang, "Change Impact Analysis Based on a Taxonomy of Change Types," in *2010 IEEE 34th Annual Computer Software and Applications Conference*, (2010): 373–382.

[4] L. Huang and Y.-T. Song, "Precise dynamic impact analysis with dependency analysis for object-oriented programs," in *5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA 2007)*, (2007): 374–384.

[5] M. Mondal, B. Roy, C. K. Roy, and K. A. Schneider, "Associating Code Clones with Association Rules for Change Impact Analysis," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, (2020): 93–103.

[6] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich, "Jripples: A tool for program comprehension during incremental change," 06 (2005): 149 – 152.

[7] L. B. Cuong, V. S. Nguyen, D. A. Nguyen, P. N. Hung, and D. H. Vo, "Jcia: A tool for change impact analysis of java ee applications," in *Information Systems Design and Intelligent Applications*, V. Bhateja, B. L. Nguyen, N. G. Nguyen, S. C. Satapathy, and D.-N. Le, Eds. Singapore: Springer Singapore, (2018): 105–114.

[8] P. Anderson and M. Zarins, "The CodeSurfer software understanding platform," in *13th International Workshop on Program Comprehension (IWPC'05)*, (2005): 147–148.

[9] S.Ãlimadadi, A. Mesbah, and K.P̃attabiraman, "Hybrid DOM-Sensitive Change Impact Analysis for JavaScript," in *29th European Conference on Object-Oriented Programming (ECOOP 2015)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), J. T. Boyland, Ed., vol. 37. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, (2015): 321–345. [Online]. Available:

[10] B. Li, Q. Zhang, X. Sun, and H. Leung, "Wave-cia: A novel cia approach based on call graph mining," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC '13. New York, NY, USA: Association for Computing Machinery, (2013): 1000–1005. [Online]. Available:

[11] ECMA-334, *C# Language Specification*. Ecma Internationa, 4th Edition, 06 (2006).

[12] M. Chaumun, H. Kabaili, R. K. Keller, and F. Lustman, "A change impact model for changeability assessment in object-oriented software systems," *Science of Computer Programming*, vol. 45, no. 2, (2002): 155–174, special Issue on Software Maintenance and Reengineering (CSMR 99). [Online]. Available:

[13] M. Zalewski and S. Schupp, "Change Impact Analysis for Generic Libraries," in *2006 22nd IEEE International Conference on Software Maintenance*, (2006): 35–44.

[14] G. Pirklbauer, C. Fasching, and W. Kurschl, "Improving change impact analysis with a tight integrated process and tool," in *2010 Seventh International Conference on Information Technology: New Generations*, (2010): 956–961.

[15] H. Kienle and H. Müller, "Rigi—an environment for software reverse engineering, exploration, visualization, and redocumentation," *Science of Computer Programming*, vol. 75, 04 (2010): 247–263.

[16] S. Ibrahim, M. Munro, and A. Deraman, "A requirements traceability to support change impact analysis," *Asian J Inform Technol*, vol. 4, 01 (2005).

[17] S. Black, "Computing ripple effect for software maintenance," *Journal of Software Maintenance*, vol. 13, 07 (2001): 263–279.

[18] S. Gwizdala, Y. Jiang, and V. Rajlich, "JTracker - A Tool for Change Propagation in Java." 04 (2003): 223 – 229.

[19] M. Petrenko and V. Rajlich, "Variable granularity for improving precision of impact analysis," in *2009 IEEE 17th In-*

ternational Conference on Program Comprehension, (2009): 10–19.
[20] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich, "JRipples: a tool for program comprehension during incremental change," in *13th International Workshop on Program Comprehension (IWPC'05)*, (2005): 149–152.
[21] S. Yau, J. Collofello, and T. MacGregor, "Ripple effect analysis of software maintenance," in *The IEEE Computer Society's Second International Computer Software and Applications Conference, 1978. COMPSAC '78.*, (1978): 60–65.
[22] R. Mohamad, "A change impact analysis approach using visualization method," 01 (2010).

Hoang-Viet Tran received his B.E. from Hanoi University of Science and Technology in 2005, MSc., and Ph.D. from VNU University of Engineering and Technology (2014, 2021). He is now a lecturer at VNU University of Engineering and Technology. His research interests include model checking, assume-guarantee verification, software testing, software evolution, and artificial intelligence for software engineering.

Email: thv@vnu.edu.vn

Nguyen Thi Mai Loan received a Bachelor's degree in Information Technology from VNU University of Engineering and Technology, Hanoi, Vietnam, in 01/2024. She is now a developer at Rikkeisoft Corporation. Her research interests include software testing, mobile applications, and web development.

Email: 20020114@vnu.edu.vn

Doan Duc Kien is currently a third-year Computer Science student at VNU University Of Engineering and Technology, Hanoi, Vietnam. His research interests include automated software testing and applying machine learning methods in software quality assurance.

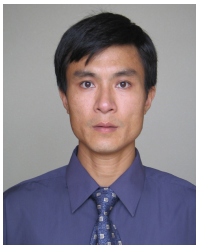
Email: 21020207@vnu.edu.vn

Nguyen Ha Trang received a Bachelor's degree in Information Technology from VNU University of Engineering and Technology, Hanoi, Vietnam, in 2024. She is now a software developer at FPT Software, Hanoi. Her research interests include web development, software quality assurance, and automated testing.

Email: 20020482@vnu.edu.vn

Le Van Huy is currently a 4^{th} year Information Technology student at VNU University of Engineering and Technology, Ha Noi, Vietnam. His research interests include web, cloud computing, and automated testing.

Email: 20020197@vnu.edu.vn



Pham Ngoc Hung received his B.S. degree from VNU University of Engineering and Technology in 2002, M.S. and PhD. degrees from Japan Advanced Institute of Science and Technology (2006, 2009). He is now an assoc. prof. at VNU University of Engineering and Technology. His research interests include model checking, assume-guarantee verification, model-based testing, and software evolution.

Email: hungpn@vnu.edu.vn