УДК 004.4'234

# МЕТОД АНАЛИЗА ВЛИЯНИЯ ИЗМЕНЕНИЙ В ТИПАХ ДАННЫХ НА ФУНКЦИИ РАЗРАБАТЫВАЕМОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

**Леохин Ю.Л., Сыроежко А.А.**

*Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования «Московский технический университет связи и информатики» (МТУСИ), 111024, г. Москва, ул. Авиамоторная, 8А, e-mail: a.a.syroezhko@edu.mtuci.ru, y.l.leokhin@mtuci.ru*

**В статье рассматривается проблема анализа изменений в функциях программного обеспечения вследствие изменений типов данных и предлагается метод статического анализа исходного кода, направленный на её решение. Разработанный метод основывается на обходе абстрактного синтаксического дерева исходного кода на языке программирования C++. Также предложены подходы для применения данного метода в процессах разработки, которые опираются на гибкие методологии.**

Ключевые слова: AST, зависимости в исходном коде, статический анализ, Agile, регрессионное тестирование.

# METHOD FOR ANALYZING THE IMPACT OF CHANGES IN DATA TYPES ON THE FUNCTIONS OF SOFTWARE BEING DEVELOPED

**Leokhin Y.L., Syroezhko A.A.**

*Moscow Technical University of Communications and Informatics (MTUCI), 111024, Moscow, st. Aviamotornaya, 8A, e-mail: a.a.syroezhko@edu.mtuci.ru, y.l.leokhin@mtuci.ru*

**The article examines the problem of analyzing changes in software functions due to changes in data types and proposes a method of static source code analysis aimed at solving it. The developed method is based on traversing the abstract syntax tree of source code in the C++ programming language. Approaches for applying this method in development processes that based on flexible methodologies are also proposed.**

Ключевые слова: AST, source code dependencies, static analysis, Agile, regression testing.

**Введение**

В настоящее время одним из наиболее эффективных подходов к разработке программного обеспечения (ПО) является подход, использующий методологию гибкого управления проектами — agile. Согласно отчету об исследовании Agile в России доля компаний, которые используют данный подход, составила 68% [2].

Agile-процессы управления проектами в сфере разработки ПО направлены на обеспечение максимальной гибкости за счет коротких этапов разработки по 2-4 недели каждый, включающих в себя планирование, анализ проекта, разработку, тестирование и развертывание продукта. Такая итеративность процесса разработки ПО способствует быстрому устареванию программной документации. Неактуальная программная документация затрудняет процессы планирования, разработки, тестирования и снижает их качество. При этом возрастает вероятность появления новых ошибок в программном коде.

Таким образом, применение agile-подхода на практике позволяет быстро реагировать на риски, связанные с изменениями требований клиента и рыночных условий. Однако при этом возникают собственные риски в процессе разработки [3].

**Недостатки agile-подход**

Несмотря на очевидные преимущества agile-подхода, он имеет ряд недостатков:

- занижение сложности реализации задач при их оценивании на этапе планирования;
- возникновение ошибок в функциях ПО, изменение которых не планировалось в рамках выполняемой задачи;
- ложноположительный результат верификации ПО при регрессионном тестировании.

Занижение оценок сложности реализации задач может возникать при поверхностном анализе предстоящего объема работы и, в частности, того, какие функции программы могут быть подвержены влиянию изменений в

исходном коде, требуемых для выполнения задачи. Факторами возникновения данной проблемы могут являться как неопытность специалистов, которым ещё не приходилось решать подобные задачи, так и неактуальная программная документация. С учетом этих факторов внесения изменений в ПО в сжатые сроки может отразиться на качестве кода и на появление новых ошибок в программном коде.

При внесении изменений в повторно используемые модули кода, которые могут применяться для реализации множества функций ПО, возникает вероятность появления ошибок в тех функциях, изменение которых не планировалось поставленной задачей или спецификацией. Основными факторами данной проблемы являются человеческая ошибка при проведении нетривиального анализа зависимостей между типами данных в коде, которые могут быть замаскированы различными интерфейсами для абстрагирования от деталей реализации при применении парадигмы объектно-ориентированного программирования, и не покрытие функций приложения юнит-тестами. Последствия будут выражаться в некорректной работе программы, которые при благоприятном развитии событий будут обнаружены на этапе тестирования.

Проблема ложноположительного результата верификации ПО после внесенных изменений напрямую возникает из предыдущей проблемы. Так как полное тестирование всех функций приложения после внесения любого изменения в исходный код не представляется возможным, то специалисты по обеспечению качества будут производит тестирование тех функций, об изменении которых их предупредил разработчик. Следовательно, если множество протестированных функций будет меньше множества функций, поведение которых действительно изменилось, то результат верификации будет ложноположительным. Тогда обнаружение ошибки может произойти в конце очередной итерации процесса разработки ПО во время полного тестирования, или случайно в рамках других задач, или при эксплуатации приложения.

**Средства поддержки процесса разработки с использованием agile-подхода**

Рассмотрим программные средства для поддержки процессов разработки ПО с использованием agile-подхода.

Doxygen — программное обеспечение для автоматической генерации документации на основе исходного кода и комментариев к нему. Полученная документация будет содержать информацию о типах данных с их описанием на основе извлеченных комментариев. Doxygen способен строит диаграммы: наследования классов, кооперации классов, граф вызова функций, граф зависимостей между файлами исходного кода. Таким образом, Doxygen позволяет автоматизировать процесс создания документации, включая информацию о зависимостях типов на уровне наследования, композиции и агрегации, и облегчает поддержание её актуальности. Однако, Doxygen не способен извлекать информацию о глубоких зависимостях между типами, так как ограничивается информацией, извлекаемой из сигнатур методов и определений классов в результате парсинга, что не позволяет в полней мере установить все зависимости. Также сгенерированная программная документация не выделяет функции приложения как отдельные элементы, реализуемые множеством классов, что делает невозможным проведение анализа влияния изменений в типах данных на функции.

Средства поиска зависимостей, встроенные в IDE, основываются на полноценном синтаксическом анализе исходного кода программы, что позволяет с наибольшей достоверностью определять зависимости между типами данных, однако они не позволяют связывать типы данных и реализуемые ими функции приложения. Следовательно, они не предусматривают анализ того, в какие типы были внесены изменения и на какие функции они могут повлиять.

Процесс тестирования является основным способом верификации ПО после внесения изменений в его исходный код, который позволяет определить, в каких функциях возникли ошибки. Однако корректность результата зависит от множества факторов: корректность тестовых сценариев; степень покрытия исходного кода автоматизированными тестами [1]; корректность оценки влияния изменений на функции ПО и т. д., а установление факта ошибок происходит уже после реализации задачи.

Также существуют различные средства проектирования с использованием универсального языка моделирования UML, которые позволяют в полной мере описать поведение приложения, каждой отдельной его функции и отношения между ними. Однако простроенные UML диаграммы не будут иметь связи с исходным кодом и будут требовать ручного анализа разработчика при сопоставлении их с кодом.

Таким образом, представленные выше средства по отдельности не способны решить проблемы разработки ПО, связанные с применением agile-подхода, а их совместное использование осуществляется в результате ручного анализа разработчиком функций и типов данных, их реализующих. Следовательно, разработка нового метода и программных средств для определения влияния изменений в типах данных на функции разрабатываемого программного обеспечения является актуальной.

**Анализ влияния изменений в типах данных на функции**

Рассмотрим предлагаемый метод для устранения рассмотренных недостатков agile-подхода на примере языка

программирования C++. Для простоты рассмотрения его можно разделить на 2 этапа: сбор данных о зависимостях между типами данных и функциями, ими реализуемыми; анализ собранных данных. Зависимость определим следующим образом: тип A является зависимостью для типа B, если изменения, вносимые в тип A способны внести изменения в логику функционирования типа B. Исходя из этого определения в рамках языка C++ имеет смысл рассмотрения зависимостей, реализуемых через:

- наследование;
- определение полей;
- определение аргументов функций;
- определение переменных в реализациях функций;
- спецификацию шаблонов.

Стадия сбора данных направлена на извлечение информации обо всех типах данных, их зависимостях и функциях приложения за счет выполнения следующего алгоритма:

Шаг 1. Препроцессинг исходного кода. Первая стадия стандартного процесса компиляции программ на языке C++, на которой происходит исполнение директив препроцессора, таких как включение заголовочных файлов и раскрытие макросов [4].

Шаг 2. Парсинг исходного кода. Производится токенизация единиц трансляций исходного кода для последующей генерации AST (Abstract syntax tree).

Шаг 3. Построение AST. Производится построение AST для последующего статического анализа [5].

Шаг 4. Определение типов данных для исключения. Формируется список типов, которые будут исключаться из анализа. К ним относятся типы: из пространства имен стандартной библиотеки языка std, тривиальные и определяемые во внешних библиотеках типы.

Шаг 5. Обход AST. В процессе обхода AST извлекаются все определения классов и структур. Для каждого определенного типа происходит поиск типов, которые могут влиять на его поведение.

Шаг 6. Раскрытие шаблонов. Все типы, порождаемые шаблонами, раскладываются на типы, которые используются в качестве спецификаторов данного шаблона, что позволяет установить неявную зависимость от типов аргументов. Например, класс использующий шаблонный класс *C<A,B>* будет иметь зависимости от типов *A, B, C*. Т.е. он явно зависит от класса *C* и неявно от классов *A* и *B*, так как они могу определять логику класса *C*. Без использования данного подхода информация о зависимостях от *A* и *B* была бы не учтена.

Шаг 7. Извлечение метаинформации. Производится получение дополнительной информации для типов. В данной реализации метода извлекается комментарий, записанный в формате: // Функция_1, …, Функция_N перед определением типа, который содержит перечисление функций приложения, за реализацию которых он отвечает.

Шаг 8. Сохранение данных. Производится запись собранной информации в БД для удобства проведения последующего анализа средствами СУБД и кеширования данных.

Первые 3 этапа выполняются с использованием транслятора Clang из фреймворка LLVM. Их результатом является AST без применения оптимизаций к исходному коду. Для получения AST в текстовом представлении достаточно выполнить команду: clang -Xclang -ast-dump -fsyntax-only SourceFile.cpp. В целях оптимизации процесса сбора данных вместо текстового представления AST используется предоставляемый Clang API [6], который позволяет построить дерево и выполнить его обход. Стоит отметить, что программа не должна содержать синтаксических ошибок, так как они делают невозможным построение корректного дерева.

Перед выполнением обхода AST производится определение типов данных для исключения из анализа на основе переданных аргументов, которые определяют каталоги для включения сторонних библиотек. Игнорирование стандартной и внешних библиотек позволяет исключить из анализа типы, изменения которых со стороны разработчика обычно не подразумеваются. Передача путей для каталогов включения также необходима ввиду того, что они требуются для стадии препроцессинга кода и нахождения определений всех используемых типов, без чего корректное построение AST не является возможным.

Обход AST выполняется через предлагаемый Clang API, который подразумевает использование паттерна проектирования — посетитель. Для каждого типа узлов, которые необходимо обработать реализуется метод, вызов которого осуществляет API в процессе итерирования узлов дерева. В данном алгоритме происходит обход узлов с определением классов и структур с их дочерних узлов с определениями переменных как в их объявлениях, так и в реализациях. При посещении узла определения нового типа он заносится в ассоциативный контейнер, если не содержится в списке исключений, где ключом является сам тип, а значением кортеж, состоящий из списка базовых типов и множества типов, которые являются его зависимостями. Заполнение

множества зависимостей происходит при обходе дочерних узлов определения нового типа. Результатом выполнения данного шага является заполненный контейнер, содержащий все типы, используемые в программе и их зависимости, за исключением игнорируемых.

Если в процессе обхода дерева производится обработка шаблонного типа, то происходит формирование множества зависимостей, которое будет состоять из типа самого шаблона и типов его спецификаторов. Далее будет произведено его объединение с множеством зависимостей в уже сформированном ассоциативном контейнере.

Для получения метаданных о функциях в данной реализации алгоритма производится обход узлов, содержащих комментарии перед определением анализируемых типов. Комментарии добавляются в ассоциативный контейнер, где ключом является тип, а значением строка с комментарием. В зависимости от используемого подхода к разделению функций между типами данных реализация алгоритма извлечения метаданных может отличаться. Например, в работе [7] предлагается располагать файлы исходного кода по каталогам, исходя из реализуемых ими функций. Тогда вместо комментария можно фиксировать название каталога в качестве значения.

В целях упрощения последующего анализа собранных данных за счет использования средств СУБД они сохраняются в виде базы данных SQLite. Также это предоставляет возможность организовать кеширование, которое позволит пропускать стадию сбора данных, если изменений в коде не было. ER-схема разработанной БД представлена на рисунке 1.

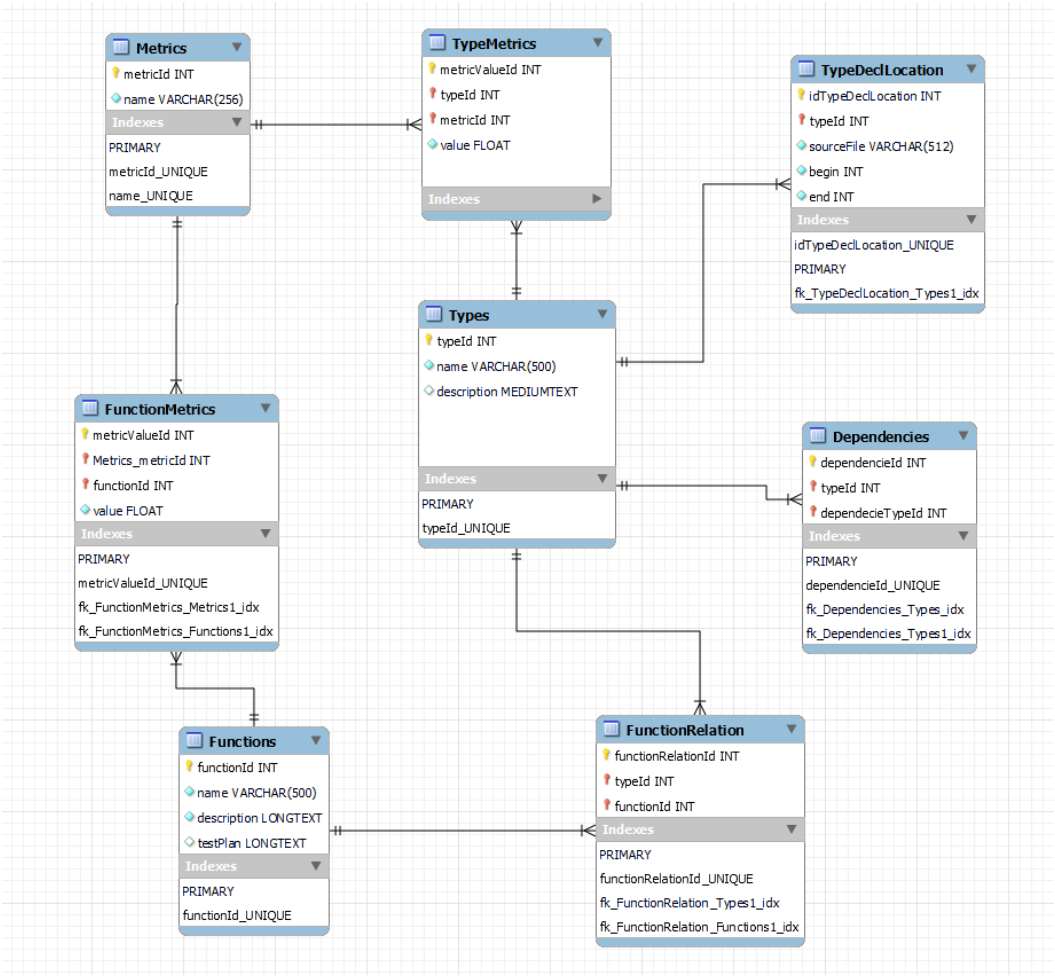


Рисунок 1. ER-схема разработанной базы данных

Для выполнения стадии анализа на языке программирования C# был реализован фронтенд, который отвечает за построение запросов к сформированной базе данных и отображение требуемой информации в графическом виде. В нём реализованы 2 вида анализа влияния изменений - при изменении типов в исходном коде и при изменении спецификаций функций приложения.

Для определения функций, которые необходимо подвергнуть регрессионному тестированию после внесенных изменений в типы, пользователю достаточно указать типы данных, в которые фактически были внесены

изменения. За счет реализации отдельного модуля, который будет определять на основе данных из системы контроля версий изменённые типы, данный процесс возможно автоматизировать полностью. Для каждого измененного типа строится запрос на получение всех типов, для которых он является зависимостью. Они объединяются в единое множество типов, поведение которых могло измениться. Остаётся для каждого из них произвести запрос метаданных, которые описывают реализуемые типом функции, и отобразить их пользователю в наглядном виде.

В случае проведения анализа для определения типов, которые потенциально могут подвергнуться изменениям в результате изменения спецификаций функций приложения, алгоритм является схожим. Пользователь выбирает функции, спецификация которых будет изменяться. Для каждой функции строится запрос типов данных, которые её реализуют. Для каждого типа производится запрос его зависимостей. Собранные данные выводятся пользователю в виде графа типов и их зависимостей, которые потенциально могут быть изменены в целях реализации обновленной спецификации.

**Заключение**

Предложенный метод основан на преимуществах подходов, реализованных в инструментальных средствах Doxygen и IDE. Реализация метода в виде программного комплекса позволяет проводить автоматизированный анализ различных видов зависимостей исходного программного кода. Ограничением предлагаемого метода является медленная скорость сбора данных и анализа, требование к синтаксической корректности анализируемой программы. Однако данные ограничения преодолеваются использованием инструментов непрерывной интеграции.

Таким образом, использование предложенного в статье подхода позволяет повысить эффективность процессов разработки программного обеспечения, основанных на применение методологии гибкого управления проектами.

Список литературы

1. Гратинский В.А., Новиков Е.М., Захаров И.С. Экспертная оценка результатов верификации инструментов верификации моделей программ // Труды института системного программирования РАН. - 2020. - №32. - С. 7-20.

2. Результаты исследования Agile в России 2022 [Электронный ресурс] // Scrumtrek URL: https://agilesurvey.ru/report22 (дата обращения: 01.04.2024).

3. Жамбалов Е.Д., Золотарева Н.А. Гибкая методология управления проектами Agile: сущность, преимущества и недостатки // АКТУАЛЬНЫЕ ПРОБЛЕМЫ АВИАЦИИ И КОСМОНАВТИКИ – 2018. Том 3.. - Красноярск: СибГУим. М. Ф. Решетнева, 2018, 2018. - С. 68-69.

4. Страуструп Б. Язык программирования C++. - 4 изд. - Бином, 2024. - 1216 с.

5. Møller A., Schwartzbach M.I. Static Program Analysis. - Aarhus University, 2023. - 210 с.

6. Introduction to the Clang AST [Электронный ресурс] // Clang 19.0.0git documentation URL: https://clang.llvm.org/docs/IntroductionToTheClangAST.html (дата обращения: 05.03.2024).

7. Peng Z., Chen P., Yang J. Revisiting Test Impact Analysis in Continuous Testing From the Perspective of Code Dependencies // IEEE Transactions on Software Engineering. - 2020. - №48. - С. 1979 - 1993.

References

1. Gratinskij V.A., Novikov E.M., Zaharov I.S. Ekspertnaya ocenka rezul'tatov verifikacii instrumentov verifikacii modelej programm // Trudy instituta sistemnogo programmirovaniya RAN. - 2020. - №32. - S. 7-20.

2. Rezul'taty issledovaniya Agile v Rossii 2022 [Elektronnyj resurs] // Scrumtrek URL: https://agilesurvey.ru/report22 (data obrashcheniya: 01.04.2024).

3. ZHambalov E.D., Zolotareva N.A. Gibkaya metodologiya upravleniya proektami Agile: sushchnost', preimushchestva i nedostatki // AKTUAL'NYE PROBLEMY AVIACII I KOSMONAVTIKI – 2018. Tom 3.. - Krasnoyarsk: SibGUim. M. F. Reshetneva, 2018, 2018. - S. 68-69.

4. Straustrup B. YAzyk programmirovaniya S++. - 4 izd. - Binom, 2024. - 1216 s.

5. Møller A., Schwartzbach M.I. Static Program Analysis. - Aarhus University, 2023. - 210 s.

6. Introduction to the Clang AST [Elektronnyj resurs] // Clang 19.0.0git documentation URL: https://clang.llvm.org/docs/IntroductionToTheClangAST.html (data obrashcheniya: 05.03.2024).

7. Peng Z., Chen P., Yang J. Revisiting Test Impact Analysis in Continuous Testing From the Perspective of Code Dependencies // IEEE Transactions on Software Engineering. - 2020. - №48. - S. 1979 - 1993.