

УДК 681.5.01

Версионирование исходного кода в системах управления конфигурацией

Фролова Вероника Ивановна

Волосатова Тамара Михайловна (*научный руководитель*)

МГТУ им. Н.Э. Баумана, Москва, Россия

Статья посвящена таким аспектам контроля конфигурации, как версионирование и обеспечение прослеживаемости. Рассмотрена разработка модуля микросервиса для восстановления связей «требование» — «элемент кода» при загрузке новых версий коммита на примере Автоматизированной Информационной Системы управления Требованиями, Трассировкой и Технологией (АИС-Т). Рассмотрены некоторые алгоритмы сравнения символьных последовательностей: расстояние Левенштейна, косинусная близость и т. д. Решение позволяет сократить количество ручной работы по переносу связей при загрузке нового коммита в систему контроля версий, а также упростить работу код-ревьюеров.

Ключевые слова: *система управления конфигурацией, автоматизация переноса связей, сравнение символьных последовательностей, система контроля версий, алгоритмы нечеткого сравнения*

Введение. Современные проекты, особенно в высокотехнологичных отраслях, отличаются высокой сложностью и требуют вовлечения множества специалистов — проектировщиков, инженеров, программистов, маркетологов и др. Их жизненный цикл, независимо от типа продукта (материальное изделие или программное решение), включает итерационные этапы: от маркетинговых исследований и проектирования до эксплуатации, поддержки и утилизации. Например, разработка программного обеспечения предполагает этапы проектирования, тестирования, внедрения и сопровождения, тогда как создание материальных изделий дополнительно охватывает производственные и логистические процессы.

Эффективное управление такими проектами невозможно без строгого контроля артефактов — требований, документации, исходного кода, чертежей и связей между ними. Однако существующие PLM-системы (Product Lifecycle Management) и инструменты управления конфигурацией, несмотря на их широкие возможности, сталкиваются с критическим ограничением: отсутствием автоматизированных механизмов для отслеживания и переноса связей между артефактами в условиях динамично изменяющихся версий файлов. Например, при модификации параметров функции в коде или обновлении требований современные системы требуют ручного подтверждения соответствия, что увеличивает риски ошибок, потери прослеживаемости и трудозатраты.

Особенно остро эта проблема проявляется в контексте итеративной разработки, где частые изменения кода и требований делают ручное управление связями неэффективным.

Контроль конфигурации. Для успешной работы со сложной системой или, по-другому, конфигурацией, особенно если разработка ведется группой специалистов, необходимо верно выстроить процессы управления и контроля. Выделяются четыре основных способа контроля:

1) идентификация — однозначное определение артефакта в системе, будь то уникальное имя класса или `uid` документа;

2) прослеживаемость — обеспечение согласованной обработки связанных артефактов, а также комплексного анализа и оценки рисков проекта;

3) версионирование — инкрементное изменение артефактов проекта во времени, обеспечивающее логическую целостность артефактов;

4) регламент — набор правил и стандартов для соблюдения технологии, позволяющий проконтролировать внесение изменений, оценить трудоемкость и сроки реализации проекта [1].

В контексте прослеживаемости необходимо также упомянуть трассировку требований — способность выявлять происхождение требования, включая его связи с другими требованиями. Это может быть использовано в процессе разработки для определения приоритета требования, а также уже после выполнения работы для проверки его выполнения. Прослеживаемость может принести следующие преимущества [2].

Анализ влияния изменений — если определенное требование изменилось, необходимо соответствующим образом изменить связанные с ним артефакты.

Анализ покрытия — трассировка может показать полноту выполнения тех или иных требований.

Анализ статуса проекта.

Повторное использование артефактов, связанных с несколькими требованиями, путем вынесения их в отдельные группы (пакеты).

Сохранность связей — нередко информация о проекте или некоторой его части может находиться у конкретного человека. При использовании прослеживаемости данная информация сохраняется благодаря визуализации связей между различными артефактами.

Оптимизация тестов — связывая требования, исходный код и тесты (тестовые сценарии и результаты), можно определить затронутые части кода при неудачном тесте, а также избежать избыточных тестовых сценариев.

Важным аспектом версионирования является не только фиксация изменений в артефакте, но и сравнение версий артефактов между собой. При этом под термином «версия» понимается идентифицированный экземпляр объекта в пределах совокупности свойств и состояния такого объекта, который удовлетворяет некоторой функции целевого применения и может быть однозначно идентифицирован в данный момент времени (идентифицированный экземпляр составной части конфигурации) [3]. У каждой версии существует свой `uid`, нередко составной: части идентификатора отвечают за «уровень» и «значимость» внесенных изменений.

Система контроля версий (англ. VCS, Version Control System) — программное обеспечение, позволяющее работать с изменяющейся информаци-

ей, храня несколько версий объекта и историю изменений для каждого пользователя [4].

Такие системы широко применяются при разработке программного обеспечения для хранения исходного кода программы. Также очень широко их использование в САПР, особенно в составе систем управления данными об изделии (PDM). В контексте обозреваемого в данной работе вопроса конфигурационного управления принято говорить о Software Configuration Management Tools (SCM tools) — инструментах конфигурационного управления [4].

Систем управления версиями немало, и у каждой из них есть свои специфические особенности: в порядке работы администраторов и пользователей, в наборе команд и т. д. Однако основные принципы VCS для большинства систем неизменны:

- проект хранится на удаленном сервере (репозитории), к которому разработчику выдается доступ;
- есть возможность создать разные варианты одного документа — ветки — с общей историей изменений до ветвления и различной после него;
- можно узнать, какой пользователь и когда добавил и/или изменил конкретные строки в файле;
- ведется журнал изменений, где пользователи могут пояснять свои правки;
- права доступа пользователей контролируются.

На данный момент самыми распространенными системами контроля версий являются Git и Subversion (SVN). По исследованиям 2024 года, Git используется в 79 % проектов, SVN — в 17 % [5].

Аналоги разрабатываемого механизма в существующих системах. Разрабатываемая система должна предоставлять возможность работы с требованиями и связями в различных версиях документа. Функционал работы с требованиями, их выставления и проверки реализован в нескольких продуктах.

Первый продукт — IBM DOORS. Данный продукт легко масштабируем, предоставляет возможность трассировки требований, а также повторного использования требований в последующих проектах. Среди его минусов можно отметить высокую стоимость, а также отсутствие поддержки для требований, не связанных с программным обеспечением, например нормативных.

Второй продукт — Visure Solutions. Он предоставляет инструменты для тестирования и управления требованиями, однако из недостатков системы можно выделить слабую интеграцию с внешними сервисами для контроля конфигурации, такими как Git.

Третий продукт — Requality. Он наиболее близко подошел к решению поставленной задачи, так как в данном инструменте есть возможность переноса требований в новую версию спецификации. Однако объект, к которому ведет требование, будет обнаружен только в том случае, если он не изменился относительно предыдущей версии. При сравнении двух версий спецификаций, в одной из которых есть требования, для каждого из них может быть

три варианта: Verify, Add Locations и Find Locations. Результат Verify выдается, если текст, к которому относилось требование, полностью найден в новой версии, и пользователю остается только подтвердить перенос. Add Locations говорит о том, что найдена только часть текста, к которому относится требование (например, если изначально данный текст состоял из нескольких частей), и пользователю предлагают самостоятельно найти недостающие отрывки спецификации. Find Locations обозначает, что искомый текст не найден в новой версии. Такого функционала для поставленной задачи недостаточно, так как в процессе разработки элементы кода могут меняться незначительно (например, изменится один из нескольких параметров функции или одно поле пользовательского класса), неся прежний функционал — соответственно, требование должно быть перенесено автоматически при определенном коэффициенте соответствия.

Система управления конфигурацией АИС-Т. Поставленная задача выполнена с использованием имеющейся кодовой базы системы управления конфигурацией АИС-Т. АИС-Т (Автоматизированная Информационная Система управления Требованиями, Трассировкой и Технологиями) — это специализированная платформа нового поколения, предназначенная для управления конфигурацией и обеспечения безопасности в процессе разработки критически важных систем, требующих сертификации. Она применяется в авионике, авиастроении и других ответственных отраслях, где соблюдение стандартов качества и безопасности является обязательным (например, в проектах с участием AIRBUS, Thales, Honeywell).

Для разработки необходимого механизма потребовалось взаимодействие с несколькими микросервисами.

Библиотека Jservicetools. Выступает связующим звеном между микросервисами. В ней реализовано формирование запросов к микросервисам на выполнение запросов.

Микросервис Sourcecode. Содержит кодоанализатор и методы для доступа к репозиториям, предварительно загруженным в систему. При помощи метода found-outlines позволяет для конкретного файла получить список объектов кода. Клиент SourcecodeClient реализован в библиотеке Jservicetools.

Микросервис Binder. В нем реализован функционал создания и удаления связей и их типов. Написан на Python. Клиент BinderClient реализован в библиотеке Jservicetools.

Микросервис Diff. В нем реализована логика сравнения кода и спецификаций. Основная работа по решению поставленной задачи ведется в нем.

Выбор алгоритма вычисления значения меры сходства. Для сопоставления элементов кода, полученных из файлов старого и нового коммитов после применения метода found-outlines, были рассмотрены три алгоритма:

- косинусная близость, объединение в векторы на основе «мешка слов»;
- коэффициент Танимото;
- расстояние Левенштейна.

Косинусная близость основана на вычислении косинуса угла между векторами [6]

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}| \cdot |\mathbf{B}|}, \quad (1)$$

где $\mathbf{A} \cdot \mathbf{B}$ — скалярное произведение векторов; $|\mathbf{A}|$, $|\mathbf{B}|$ — длины векторов $\mathbf{A}$ и $\mathbf{B}$ соответственно.

При этом может получиться значение от -1 до 1 , где 1 означает, что векторы совпали, 0 — вектора ортогональны и -1 — вектора направлены в противоположные стороны.

Чтобы вычислить вектор фразы, необходимо воспользоваться, например, методом «мешок слов» («bag of words»), схема которого имеет следующий вид:

1) разбить пару сравниваемых фраз на уникальные слова (каждое слово записывается в порядке появления);

2) сопоставить каждому слову количество появлений во фразе;

3) из полученных чисел составить вектор-столбец фразы.

Коэффициент Танимото также описывает схожесть двух множеств [7]. В результате может получиться значение от 0 до 1 , где 0 — сходство не обнаружено, 1 — полное сходство. Формула для нахождения коэффициента Танимото имеет вид

$$k = \frac{c}{a+b-c}, \quad (2)$$

где k — коэффициент Танимото; a , b — количество элементов в первом и втором множествах соответственно; c — количество совпадающих элементов в двух множествах.

Расстояние Левенштейна (редакционное расстояние) позволяет измерить по модулю разность между двумя последовательностями символов — сколько символов необходимо убрать, сколько добавить и сколько заменить (в некоторых алгоритмах замены не учитываются), чтобы получить одинаковые последовательности [8]. Формула для нахождения данного расстояния имеет вид

$$r = a + b - 2c, \quad (3)$$

где r — расстояние Левенштейна; a , b — длины последовательностей; c — длина наибольшей общей последовательности.

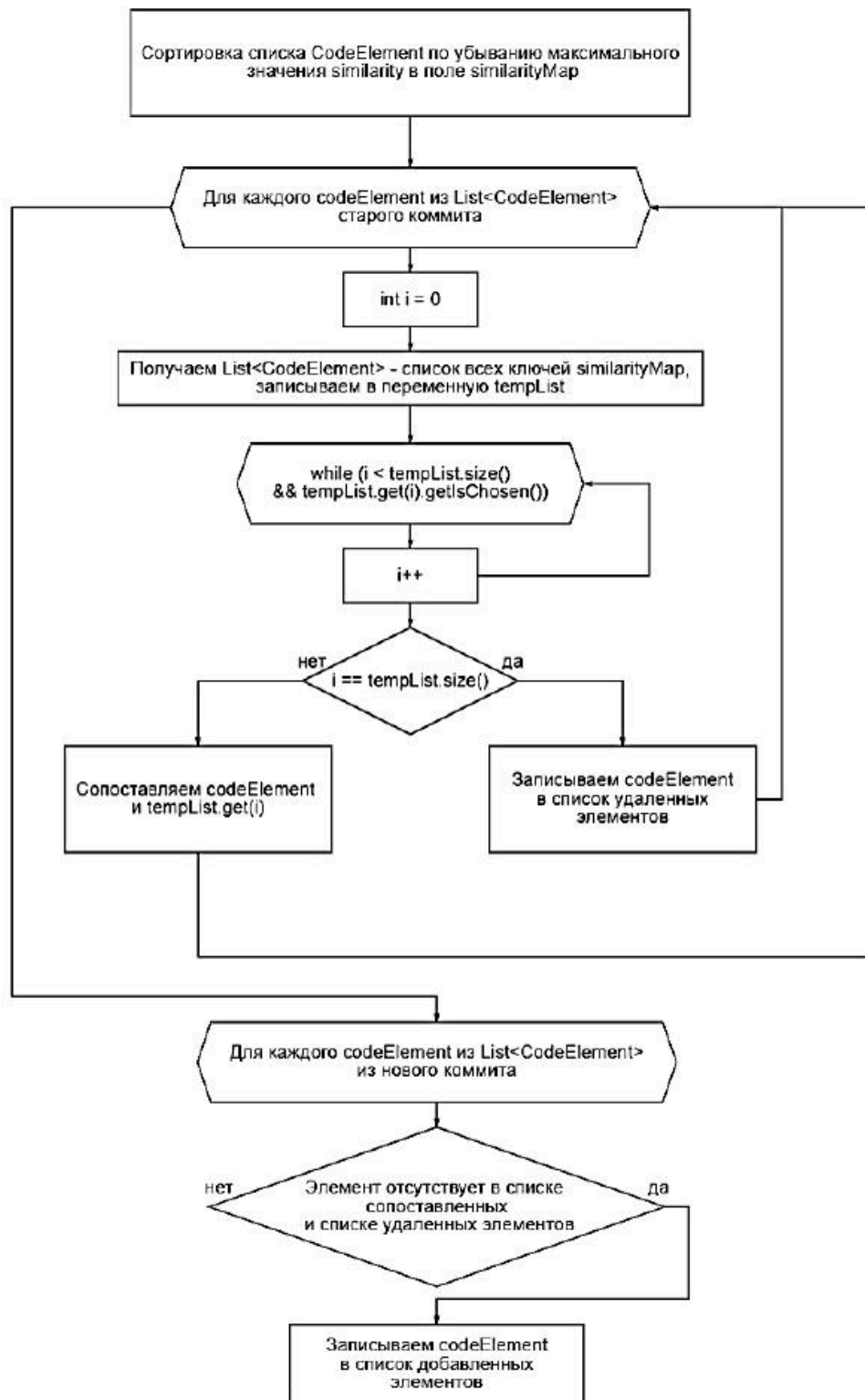
Поскольку необходимо получить не расстояние, а меру сходства (число от 0 до 1), данная формула может быть доработана следующим образом:

$$k = \frac{a+b-r}{a+b}, \quad (4)$$

где k — процент сходства между последовательностями.

Поскольку косинусная близость не учитывает четкий порядок слов (если слово встречается несколько раз, у него будет большее значение в векторе, но его место во фразе не учитывается), а коэффициент Танимото более применим к последовательностям слов, нежели чем к последовательностям символов, для решения задачи было выбрано расстояние Левенштейна.

Сопоставление элементов с посчитанной мерой сходства. Для сопоставления элементов был разработан алгоритм, блок-схема которого представлена на рисунке.



Алгоритм сопоставления элементов с посчитанным значением сходства:

CodeElement — класс для десериализации полученных от found-outlines данных; поле similarityMap — поле класса CodeElement типа HashMap<CodeElement, Float> для хранения информации о коэффициенте сходства с целевым элементом (this); метод getIsChosen() — метод класса CodeElement, позволяющий получить информацию о том, был ли элемент сопоставлен с другим элементом, или он доступен для сопоставления

Дальнейшее развитие проекта. В ходе работы над проектом цели и задачи дополнялись. К примеру, была выдвинута идея о необходимости подбора весов для признаков сравнения функций (их названия, типа и имен параметров, длины): совпадение названия должно иметь больший вес, чем совпадение количества параметров, и т. д. Данная задача может быть решена с внедрением искусственного интеллекта для анализа нескольких коммитов большой кодовой базы и отслеживания частоты изменения тех или иных признаков.

Помимо этого, стоит сравнивать также текст функций: поскольку разработка продукта может являться очень долгим процессом, на последних итерациях функции могут сильно отличаться от своих исходных версий, и больше не реализовывать требования, изначально связанные с ними. Для анализа смысла функции может потребоваться как простой расчет цикломатической сложности, так и построение абстрактных синтаксических деревьев.

Также необходимо реализовать возможность сравнения файлов не по одному, а всего коммита целиком. Для этого необходимо извлекать метаданные коммита, содержащие информацию о том, какие файлы были изменены и требуют сравнения, какие остались прежними, и необходимо лишь перенести связи, какие файлы были добавлены и удалены. Необходимо обнаруживать не только измененные, но также и перемещенные, и переименованные файлы. Для разрешения проблем с такими файлами может потребоваться пользовательский интерфейс для ручного сопоставления файлов.

Заключение. В рамках данной статьи достигнута поставленная цель — разработан механизм автоматического управления связями между требованиями и артефактами в условиях динамичных изменений версий файлов. Реализованный подход интегрирует методы управления жизненным циклом изделия, контроля конфигурации и современные практики работы с системами управления версиями (VCS), что позволяет обеспечить перенос требований между версиями с заданным порогом соответствия и сохранение целостности связей в распределенных средах.

Полученные результаты демонстрируют практическую значимость механизма: сокращение трудозатрат на ручное подтверждение соответствия, минимизацию ошибок и повышение надежности прослеживаемости в проектах с частыми итерациями.

Литература

- [1] INCOSE, Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, version 3.2.2. Wiley, International Council on Systems Engineering, 2012.
- [2] Bouillon E., Mäder P., Philippow I. A Survey on Usage Scenarios for Requirements Traceability in Practice. Requirements Engineering: Foundation for Software Quality. Lecture Notes in Computer Science, 2013, vol. 7830, pp. 158–173.
- [3] ГОСТР ИСО/МЭК 12207–2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств. Москва, Стандартинформ, 2010, 105 с.
- [4] Системы управления версиями. URL: <https://kaf401.rloc.ru/TRPO/VCS.pdf> (дата обращения 14.11.2024).

- [5] Compare repositories. URL: <https://openhub.net/repositories/compare> (дата обращения 18.11.2024).
- [6] Jurafsky D., Martin J.H. Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Pearson Education International, 2009, 1024 p.
- [7] *Использование коэффициента Танимото для поиска людей с одинаковыми предпочтениями*. URL: <https://habr.com/ru/articles/104901> (дата обращения 01.12.2024).
- [8] *Задача о редакционном расстоянии, алгоритм Вагнера — Фишера*. URL: <https://neerc.ifmo.ru/wiki/index.php> (дата обращения 01.12.2024).

Versioning source code in configuration management systems

Frolova Veronika Ivanovna

Volosatova Tamara Mikhailovna (*scientific advisor*)

BMSTU, Moscow, Russia

The article is devoted to such aspects of configuration control as versioning and traceability. This article deals with the development of a microservice module for restoring “requirement” — “code element” links when loading new versions of a commit on the example of the Automated Information System for Requirements, Traceability and Technology Management (AIS-T). Some algorithms for comparing symbolic sequences are considered: Levenshtein distance, cosine proximity, etc. The solution allows to reduce the amount of manual work on transferring links when loading a new commit in the version control system, as well as to simplify the work of code-reviewers.

Keywords: configuration management system, link transfer automation, character sequence comparison, version control system, fuzzy comparison algorithms