Поддержка мультиверсионности в контексте разработки программного обеспечения

А. А. Копий

***Аннотация*— Процессы разработки и эксплуатации программного обеспечения требуют фиксации стабильных версий. В случае поддержки кроссплатформенности, разных версий языков программирования или разных версий компиляторов возникает необходимость выделения под-версий программ. Современные методы версионирования предназначены для линейной фиксации изменений кода, то есть архитектурно не предоставляют возможности манипулирования под-версиями. Как следствие, необходимость поддержки нескольких под-версий сопряжена с дополнительными временными затратами на ручную синхронизацию под-версий, либо с увеличивающимся объемом исходного кода с семантически идентичными участками кода. В ходе анализа современных систем контроля версий, были выявлены их недостатки для задачи поддержки мультиверсионности программного обеспечения. На основе существующих подходов к контролю под-версий был предложен альтернативный подход. Предложенное решение предполагает хранение контекстно зависимых участков кода отдельно от основной ревизии. Таким образом, каждая ревизия может содержать в себе несколько версий кода, которые находятся в самой ревизии, однако не дублируют исходный код программы. Задача синхронизации под-версий была перенесена в один из модулей предлагаемого решения, в результате чего была нивелирована необходимость ручной синхронизации под-версий. Данное решение позволяет уменьшить время разработки и выпуска новых версий программного обеспечения. Таким образом, предложенный подход к выделению под-версий является более эффективным методом решения поставленной задачи в сравнении с существующими.**

***Ключевые слова*— Версионирование программного обеспечения, поддержка мультиверсионности, разработка программного обеспечения, системы контроля версий.**

I. Введение

В процессах разработки и эксплуатации какой-либо программы выявляются ее ошибки, добавляются новые бизнес-требования, возникает потребность в изменении бизнес-логики. Любое изменение программного продукта требует выпуска новых версий [1].

Для обеспечения версионности ПО в процессе его разработки используются VCS (системы контроля версий) [2, 3]. VCS предполагает линейный процесс разработки программ и, как следствие, линейное выделение версий [4]. То есть, под одной выделенной версией конечного продукта в VCS стоит одна версия исходного кода.

Подобный линейный подход является подходящим для последовательного версионирования проекта, однако не учитывает вариативность в рамках обособленных версий, необходимость в которых возникает в случае поддержки разных версий языка, библиотек, компиляторов; обеспечения кроссплатформенности и т.д.

В случае поддержки мультиверсионности использование систем контроля версий не отличается удобством, поскольку все настоящие методы предполагают ручное манипулирование исходным кодом, что увеличивает время поставки ПО и, как следствие, его стоимость [5 - 8].

Автором статьи рассмотрены существующие методы поддержки мультиверсионности и предложен новый метод, обеспечивающий более удобный контроль выпуска ПО.

II. Современные методы параллельного версионирования

Существуют разные реализации VCS [9, 10]. Они отличаются синтаксисом, некоторыми подходами хранения версий, однако принцип их работы примерно одинаков [5]. Он заключается в последовательном выделении обособленных версий некоторого набора файлов в области видимости рабочей директории путем фиксации изменений [11]. Общий принцип работы позволяет абстрагироваться от конкретных реализаций систем контроля версий и рассматривать общие методы поддержки мультиверсионности программного обеспечения:

1. Выделение отдельных веток в VCS [5]
2. Использование препроцессорных директив [6] и обработка контекстных исключений [7]
3. Использование технологии банчей [8]

При выделении отдельных веток в VCS возникает необходимость ручной синхронизации ревизий [12]. Линейный подход к контролю версий представляет собой последовательность изменений в проекте в виде однонаправленного графа, каждый узел которого является версией проекта, что позволяет визуализировать данный подход (рисунок 1) [13]. Следствием подобных ручных действий является уменьшение скорости разработки ПО.

Manuscript received May 04, 2020.

Copiy Anna Aleksandrovna – undergraduate of the sub-department «Informatics and computer engineering» of Moscow State Technological University «STANKIN» (e-mail: ncopiy@ya.ru).

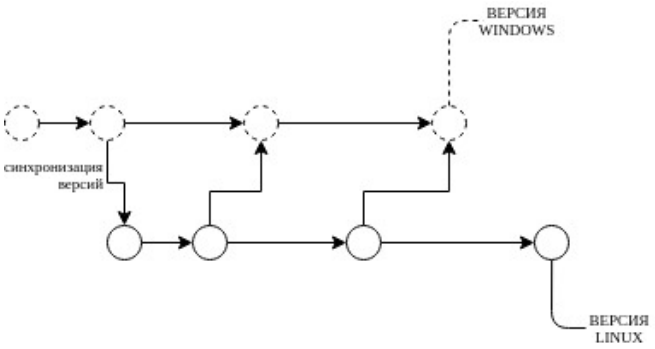

Рис. 1. Синхронизация версий в процессе разработки программного обеспечения для операционных систем Windows и Linux

Следующими методами являются использование препроцессорных директив (листинг 1) и обработка контекстных исключений (листинг 2). Данные методы выделены в отдельный подход, поскольку предполагают внесение контекстных изменений в исходный код некоторого ПО [14, 15].

```
#ifdef _WIN32
// do something windows specific
#endif
#ifdef linux
// do something linux specific
#endif
```

Листинг 1. Использование препроцессорных директив языка программирования C++

```
try:
    from x import y  # python 3
except ImportError:
    from old_x import y  # python 2
```

Листинг 2. Обработка контекстных исключений для поддержки двух версий языка программирования Python

В результате изменений возникает дублирование семантически идентичных фрагментов кода, что является прямым нарушением принципа DRY [16].
Использование технологии банчей представляет собой сочетание предыдущих подходов, поскольку предполагает ручное дублирование файлов исходного кода, а также постоянную синхронизацию версии исходного кода программ в рамках идентичных файлов [8]. Дублирующихся файлов становится все больше, структура проекта разрастается (рисунок 2). Как следствие, по проекту становится сложнее ориентироваться, и скорость внесения правок в код увеличивается [17].

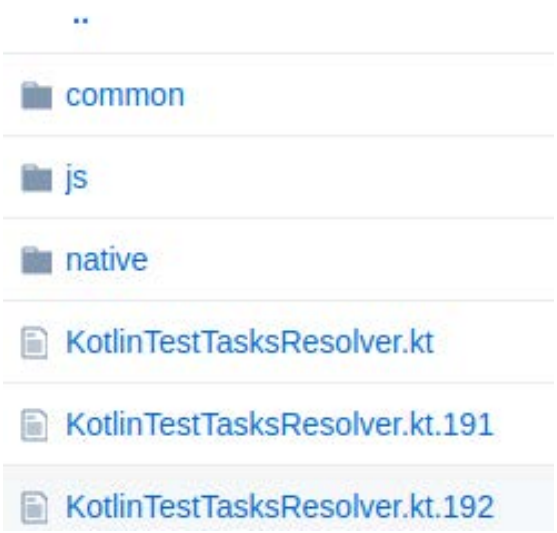

Рис. 2. Структура проекта при использовании технологии банчей

Таким образом, методы обеспечения вариативности в рамках обособленных версий, которыми ограничиваются современные VCS, требуют дополнительных временных затрат на ручную синхронизацию версий либо предполагают внесение семантически идентичных правок в исходный код программы. Это увеличивает временные затраты на разработку ПО и, как следствие, негативно сказывается на стоимости конечного продукта [18].

III. Альтернативный метод поддержки параллельного версионирования

Одной из основных целей программного обеспечения является автоматизация труда человека [19]. Автор статьи полагает, что ручные операции по синхронизации исходного кода можно автоматизировать, поэтому разработала технологию HIH (hide inside of history). Данная технология предполагает расширение функционала VCS и предлагает более гибкое решение поддержки мультиверсионности проекта.
Независимо от причины возникновения контекстно зависимых правок в исходном коде, они представляют собой ряд последовательных семантически идентичных участков кода. Именно эту разницу в коде предлагается хранить в рамках версий внутри ревизии.
Хранение разницы контекстно зависимых участков кода обеспечивается механизмом хранения ревизий в таких VCS, как mercurial и CVS [20]. Данные системы контроля версий в рамках ревизии фиксируют только разницу в файлах с предыдущей версией, что позволяет использовать этот же механизм для обеспечения мультиверсионности (рисунок 3) [20, 21].

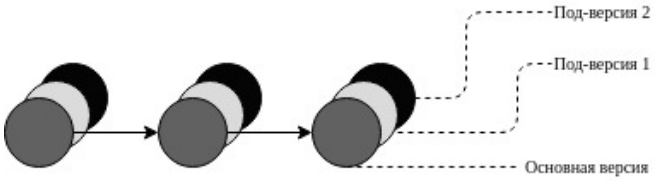

Рис. 3. Структура проекта при использовании технологии HIH

Таким образом, отличием в исходном коде среди под-версий является только контекстно зависимые участки кода, то есть хранится только разница между основной версией и ее под-версиями. Сами же под-версии представляют собой ревизии, стоящие за основной версией кода. Как следствие, структура проекта не усложняется, а код в рамках под-версий не дублируется.
Также предлагается упразднить ручной процесс синхронизации под-версий и делегировать это действие одной из составляющей частей HIH - пресинхронизационной процедуре. Данная часть HIH является настраиваемым модулем, для работы которого необходимо подключить программное обеспечение, отслеживающее необходимость внесения контекстных изменений. Настоящий подход является гибким решением, поскольку предполагает пользовательскую настройку, тем самым абстрагируется от первоначальной причины возникновения потребности в мультиверсионности.

IV. Заключение

Современные средства поддержки версионности в рамках разработки программного обеспечения не решают актуальную проблему поддержки мультиверсионности. Более того, настоящие подходы влекут за собой дополнительные временные затраты на синхронизацию исходного кода в под-версиях.
Автором был предложен альтернативный подход к мультиверсионированию. Данный подход не требует ручной синхронизации исходного кода в подверсиях, что уменьшает время на фиксацию новых версий программного продукта. Также настоящий подход предполагает хранение только контекстно зависимой разницы исходного кода программы между под-версиями. Как следствие технология HIH справляется с поставленной задачей более эффективно.

Библиография

[1] Loeliger J., McCullough M. Version Control with Git: Powerful tools and techniques for collaborative software development. O'Reilly Media, Inc., 2012
[2] Spinellis D. Version control systems. IEEE Software, 2005, Т. 22., №. 5., 108-109 p.
[3] Pilato C. M., Collins-Sussman B., Fitzpatrick B. W. Version control with subversion: next generation open source version control. O'Reilly Media, Inc., 2008.
[4] Lebanon G., El-Geish M. A Few More Things About Programming. Computing with Data, Springer, Cham, 2018, 441-470 p.
[5] Meyer S. Quality assurance for open source software configuration management. 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. IEEE, 2013, 454-461 p.
[6] Stroustrup B. The C++ programming language. Pearson Education India, 2000.
[7] Lutz M. Programming python. O'Reilly Media, Inc., 2001.
[8] Jetbrains/Bunches. Set of utils for supporting patchsets branches. https://github.com/JetBrains/bunches (accessed: 4 April 2020).
[9] Knittl-Frank D. Analysis and comparison of distributed version control systems. Bachelorarbeit, University of Applied Sciences, Upper Austria. 2010.
[10] Lanubile F. Collaboration tools for global software engineering. IEEE software, 2010, Т. 27., №. 2., 52-55 p.
[11] Rana A. I., Arfi M. W. Software release methodology: A case study. Student Conference on Engineering Sciences and Technology. IEEE 2005, 1-10 p.
[12] Swicegood T. Pragmatic version control using Git. Pragmatic Bookshelf, 2008.
[13] Collberg C. A system for graph-based visualization of the evolution of software. ACM symposium on Software visualization, 2003.
[14] Logan S. Cross-platform development in c++: building mac os x, linux, and windows applications. Pearson Education, 2007.
[15] Nanjekye J. Package Imports. Python 2 and 3 Compatibility. Apress, Berkeley, CA, 2017, 47-52 p.
[16] Yu Y., Leite J. C. S. P., Mylopoulos J. From goals to aspects: discovering aspects from requirements goal models. 12th IEEE International Requirements Engineering Conference, 2004, 38-47 p.
[17] Krinke J. Identifying similar code with program dependence graphs. Eighth Working Conference on Reverse Engineering, IEEE, 2001, 301-309 p.
[18] Brooks F. P. Mythical Man-Month T. Essays in Software Engineering. Adisson-Wesley, Reading, Mass, 1975.
[19] Jones N. D. On modeling and programming. International Symposium on Leveraging Applications of Formal Methods, Springer, Cham, 2018. 22-34 p.
[20] Chacon S., Straub B. Git and Other Systems. Pro Git. Apress, Berkeley, CA, 2014, 307-356 p.
[21] Blischak J. D., Davenport E. R., Wilson G. A quick introduction to version control with Git and GitHub. PLoS computational biology, 2016, Т. 12, №. 1

Support for multi-versioning in the context of software development

A. A. Copiy

***Abstract* — Software development and exploitation requires to fix stable versions. In the case of support of cross-platform development, usage of different programming language versions or different compilers versions, it becomes necessary to allocate sub-versions of programs. Modern versioning methods are designed to linearly commit code changes, that is, architecturally do not provide the ability to manipulate sub-versions. As a result, the requirement of supporting several sub-versions is associated with additional time costs for manual synchronization of sub-versions, or with an increasing amount of semantically identical sections of code. In the analysis of modern version control systems, their shortcomings were identified for the task of supporting multiversional software. Based on existing approaches of sub-version control, an alternative approach was proposed. The solution involves storing context-sensitive sections of code separately from the main revision. Consequently, each revision can contain several versions of the code that are in the revision itself, but do not duplicate the full program source code. The task of synchronizing sub-versions was transferred to one of the modules of the proposed solution, as a result of which the need for manual synchronization of sub-versions was leveled. This solution allows to reduce software versions development and release time. As a result, the proposed sub-versions allocation approach is a more effective method of solving the problem in comparison with the existing ones.**

***Keywords* — Multiversion support, software development, software versioning, version control systems.**

REFERENCES

[1] Loeliger J., McCullough M. Version Control with Git: Powerful tools and techniques for collaborative software development. O'Reilly Media, Inc., 2012

[2] Spinellis D. Version control systems. IEEE Software, 2005, T. 22., №. 5., 108-109 p.

[3] Pilato C. M., Collins-Sussman B., Fitzpatrick B. W. Version control with subversion: next generation open source version control. O'Reilly Media, Inc., 2008.

[4] Lebanon G., El-Geish M. A Few More Things About Programming. Computing with Data, Springer, Cham, 2018, 441-470 p.

[5] Meyer S. Quality assurance for open source software configuration management. 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. IEEE, 2013, 454-461 p.

[6] Stroustrup B. The C++ programming language. Pearson Education India, 2000.

[7] Lutz M. Programming python. O'Reilly Media, Inc., 2001.

[8] Jetbrains/Bunches. Set of utils for supporting patchsets branches. https://github.com/JetBrains/bunches (accessed: 4 April 2020).

[9] Knittl-Frank D. Analysis and comparison of distributed version control systems. Bachelorarbeit, University of Applied Sciences, Upper Austria. 2010.

[10] Lanubile F. Collaboration tools for global software engineering. IEEE software, 2010, T. 27., №. 2., 52-55 p.

[11] Rana A. I., Arfi M. W. Software release methodology: A case study. Student Conference on Engineering Sciences and Technology. IEEE 2005, 1-10 p.

[12] Swicegood T. Pragmatic version control using Git. Pragmatic Bookshelf, 2008.

[13] Collberg C. A system for graph-based visualization of the evolution of software. ACM symposium on Software visualization, 2003.

[14] Logan S. Cross-platform development in c++: building mac os x, linux, and windows applications. Pearson Education, 2007.

[15] Nanjekye J. Package Imports. Python 2 and 3 Compatibility. Apress, Berkeley, CA, 2017, 47-52 p.

[16] Yu Y., Leite J. C. S. P., Mylopoulos J. From goals to aspects: discovering aspects from requirements goal models. 12th IEEE International Requirements Engineering Conference, 2004, 38-47 p.

[17] Krinke J. Identifying similar code with program dependence graphs. Eighth Working Conference on Reverse Engineering, IEEE, 2001, 301-309 p.

[18] Brooks F. P. Mythical Man-Month T. Essays in Software Engineering. Adisson-Wesley, Reading, Mass, 1975.

[19] Jones N. D. On modeling and programming. International Symposium on Leveraging Applications of Formal Methods, Springer, Cham, 2018. 22-34 p.

[20] Chacon S., Straub B. Git and Other Systems. Pro Git. Apress, Berkeley, CA, 2014, 307-356 p.

[21] Blischak J. D., Davenport E. R., Wilson G. A quick introduction to version control with Git and GitHub. PLoS computational biology, 2016, T. 12, №. 1