УДК 004.416.2:004.415

Н. В. Кононенко

А. А. Яровий

А. М. Яровий

ПІДТРИМКА ПРОГРАМНИХ ДОДАТКІВ ІЗ ЗАСТАРІЛИМ СТАНДАРТОМ У КОНТЕКСТІ ВЕРСІОНУВАННЯ

Вінницький національний технічний університет

Анотація

Проведено аналіз різноманітних методів версіонування додатків. Визначено основні переваги і недоліки кожного з них. Проведено дослідження предметної області та враховано усі етапи розробки та підтримки програмних додатків.

Ключові слова: програмний додаток, версіонування, контроль версій, підтримка ПЗ.

Abstract

An analysis of various application versioning methods was carried out. The main advantages and disadvantages of each of them are determined. A study of the subject area was carried out and all stages of development and support of the application were taken into account

Keywords: software application, versioning, version control, software support.

Вступ

Синтаксис програмування, як відомо, постійно розвивається, виходять нові фреймворки, стандарти коду, бібліотеки, нові пакети ПЗ. Окрім цього, є чимало старих пакетів, що регулярно оновлюються, у них налагоджують старі дефекти (баги), додають нові можливості. Із зростанням системи та поступової інтеграції до неї оновлених пакетів є ризик зіштовхнутись із таким поняттям як пекло залежностей (англ. dependency hell) [1].

Буквально проект складатиметься із купи залежних один від одного пакетів, що майже неможливо організовано впорядкувати. До прикладу, якщо специфікація залежності занадто жорстка, то існує ймовірність зіштовхнутись з таким поняттям як блокування пакетів, неможливість отримати оновлення, тому що потрібно оновити кожен старий залежний пакет. З іншого боку, неможливо давати надто вільну специфікацію, інакше ми обов'язково отримаємо безлад у версіях, витрачаючи ресурси на більшу кількість версій, ніж це взагалі потрібно [1].

Результати досліджень

Головна проблема версіонування полягає в тому, що часто при інтеграції нових можливостей, потрібно також підтримувати старий функціонал. Буквально нові пакети, так чи інакше, залежать від старих. Це призводить, по-перше, до підтримки застарілого та непотрібного для користувача функціоналу. По-друге, застосунок стає більш ресурсо потребуючим.

Постає логічне запитання: чому просто не відмовитись від версіонування та раз у певний термін не показувати демо результат? Справа в тому, що коли різні частини додатку розроблятимуть декілька команд, вони не будуть враховувати залежності певних частин одна від одної. Буквально, при оновленні, у кожної команди все працює, проте при запуску додатку, враховуючи всі оновлення – отримуємо купу багів або взагалі непрацездатну версію. Повернути версію назад буде неможливо, адже ніхто не знає, які версії сервісів працюють належним чином між собою.

Саме для уникнення таких ситуацій були зроблені системи версіонування, виділяють їх три основні види [2]:

1. Семантичне версіонування.

2. Версіонування за допомогою дати (використовують схема ISO “рік-місяць-день”).
3. Коли вказують стадію розробки (напр. 1.5 alpha 5).

Семантичне версіонування

Семантичне версіонування (англ. Semantic Versioning), також відоме як semver (семвер) — специфікація про те, як привласнювати версії випускам програмного забезпечення користуючись поняттями major, minor, patch [3].

Проект, що використовує семантичне керування версіями зазвичай має 3 основних пункти для їх класифікації:

- major – головна версія, у якій можна міняти поведінку застосунку, API, видаляти старий код. Збільшується на 1, коли API безпосередньо додатку змінюється на несумісний з минулими версіями. Наприклад: версії 18.13.0 та 19.4.0 бібліотеки Node.js [1].
- minor – номер версії, коли у додатку додають нові можливості до API, не порушуючи при цьому сумісність з іншими.
- patch – номер версії, який збільшується, коли у додатку виправляють існуючі баги. Ці зміни кардинально не впливають на роботу додатку, але щось нове не додають. При зміні “major” чи “minor” прийнято округляти “patch” до 0.

Рідше використовують поле “label”, де розробники записують зміни. Як правило, це роблять у невеликих компаніях, де не ведеться офіційна документація змін (changelog).

Головною проблемою такого підходу є те, що не кожен розробник розуміє головний принцип такого виду версіонування. Програміст знищує зворотну сумісність, тому на комп’ютері автора все працює, а на пристрої користувача при встановлені з нуля – ні.

Версіонування по даті

Версіонування по даті – специфікація про те, як привласнювати версії випускам програмного забезпечення, користуючись датою останньої версії, що працює [4].

Цей підхід широко застосовується у невеликих компаніях чи коли розробник працює сам. Він власноруч проставляє дату, коли він зробив зміни у проект.

Проте цей підхід, по-перше, немає зворотної сумісності. Буквально зміни потрібно зберігати в голові, чи кудись записувати, а робити бекап до попередньої версії дуже часто не є можливим.

Іншої проблемою цього підходу є занадто вільна форма специфікації оновлень. Дуже важко зрозуміти, чому вийшла нова версія. До прикладу, розробник виправив баги чи повністю переробив функціонал. Готовий користувач побачить лише дату нової збірки. До того ж, багато розробників нумерують версії по власному вільному стандарту, не користуючись при цьому схемою ISO.

Версіонування по стадії розробки

Версіонування по стадії розробки – специфікація про те, як привласнювати версії випускам програмного забезпечення, користуючись стадіями розробки, що відображають кількість нових реалізованих функцій, які заплановані до виходу певної версії додатку [5].

Термінологія вперше була запроваджена у комерційній компанії “IBM” та поділялась на 3 підвиди [5]:

1. Тест «A» представляв собою перевірку нового продукту перед публічним оголошенням.
2. Тест «B» був перевіркою перед випуском продукту у виробництво.
3. Тест «C» був остаточним випробуванням перед загальною доступністю продукту.

Протягом розвитку програмування ці етапи були реформовані та відредаговані, таким чином виділили 5 основних етапів:

– Pre-alpha – початкова стадія розробки; період часу зі старту розробки ПЗ до виходу стадії «Alpha»;

– Alpha – стадія розробки, коли відбувається безпосередньо тестування додатку фахівцями-тестувальниками;
– Beta – стадія, під час якої ПЗ підлягає публічному тестуванню; стадія активного бета-тестування і налагодження програми.

Головною проблемою цього виду версіонування є те, що користувач розуміє, які зміни були виконані на різних стадіях. Можливо, потрібний користувачу функціонал забрали із додатку і вже немає необхідності купувати його.

Проблема підтримки додатків із застарілим стандартом

Створення однієї програми

На перший погляд, здається, що це ультимативне рішення, яке повністю вирішує покликану проблему, проте це зовсім не так. Чимало додатків постійно оновлюються та отримують новий функціонал, при цьому старі версії також потрібно зберігати для певного відсотка незвичайних користувачів. Окрім цього, потрібна велика кількість розробників, що будуть підтримувати кожну із версій до та після випуску. Розробники також повинні володіти старими стандартами коду та знати всі залежності пакетів у проекті. Саме тому, це потребує величезної кількості ресурсів, починаючи від потужних серверів, закінчуючи значним людським ресурсом.

Випуск незалежних застосунків

Ще з самого початку потрібно розуміти, що випуск нового, а в даному випадку оновленого продукту потребує значну кількість ресурсів. Головною проблемою таких застосунків є те, що позитивний результат не завжди можна гарантувати, адже за декілька років мова програмування могла повністю оновитись, деякі експериментальні рішення взагалі можна було б прибрати.

Висновки

Під час проведення дослідження розглянуто проблеми версіонування високонавантажених додатків. Охарактеризовано їх основні види, проаналізовано переваги та недоліки кожного з них.

Дослідження поділено на декілька логічних розділів, у яких висвітлена проблематика, особливості кожного із підходів версіонування, порушено питання безпосередньо підтримки програмних додатків із застарілим стандартом.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Том Престон-Вернер. Семантичне Версіонування 2.0.0. Semantic Versioning. URL: https://semver.org/lang/uk.
2. Некрасов А. Як ми зіштовхнулися з версіонуванням і усвідомили, що варіант просто проставити цифри не працює. Хабр. URL: https://habr.com/ru/companies/mts_ai/articles/674370
3. Walker J. What is Semantic Versioning?. How-To Geek. URL: https://www.howtogeek.com/devops/what-is-semantic-versioning/
4. Найпростіший спосіб версіонування файлів. Backup-Software - Jetzt Daten sichern | Langmeier Backup Datensicherung 12. URL: https://www.langmeier-software.com/uk/seiten/news/die-einfachste-art-der-dateiversionierung
5. Учасники проектів Вікімедіа. Стадії випуску програмного забезпечення – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Стадії_випуску_програмного_забезпечення

Кононенко Назарій Валерійович – студент групи 2КН-21б, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, Вінниця, e-mail: nazarella8@gmail.com.

Яровий Андрій Анатолійович – завідувач кафедри комп'ютерних наук, Вінницький національний технічний університет, Вінниця, e-mail: a.yarovyy@vntu.edu.ua.

Яровий Анатолій Михайлович – доцент кафедри комп'ютерних наук, Вінницький національний технічний університет, Вінниця.

Nazarii V. Kononenko – Faculty of intelligent information technologies and automation, Vinnytsia National Technical University, Vinnytsia, e-mail: nazarella8@gmail.com.

Andrii A. Yarovyi – Head of the Department for Computer Science, Vinnytsia National Technical University, Vinnytsia, e-mail: a.yarovyy@vntu.edu.ua.

Anatolii A. Yarovyi – Professor Assistant of the Department for Computer Science, Vinnytsia National Technical University, Vinnytsia.