Szemantikus verziószámozás

Jeszenszky Péter

2024.03.05.

Konfiguráció menedzsment (1)

Ez a rész az alábbi könyvön alapul:

- Ian Sommerville. *Software Engineering*. 10th ed. Pearson, 2015. https://software-engineering-book.com/
  - Chapter 25: Configuration Management (p. 730–755)

Konfiguráció menedzsment (2)

- A szoftverfejlesztésben állandóan változás történik.
- A konfiguráció menedzsment egy fejlődő szoftverrendszerben történő változások kezelésének folyamata.
- Az alábbi négy alapvető és szorosan kapcsolódó tevékenységet foglalja magában:
  - Verziókezelés
  - Rendszerépítés
  - Változáskezelés
  - Kiadáskezelés

Konfiguráció menedzsment (3)

Szoftver konfiguráció elem, röviden **konfiguráció elem**: bármi, ami egy szoftverprojekthez tartozik (például terv, kód, tesztadatok, dokumentum) és konfiguráció menedzsment alá esik.

- Tehát változásnak van kitéve, ahol a változás egy konfiguráció menedzsment rendszerben felügyelhető.
- Mindig van egy egyedi azonosítója.

Verziózási fogalmak: verziózás

Definíció (SEVOCAB):

Egyedi verzionevek vagy verziószámok hozzárendelése szoftver konfiguráció elemek egyedi állapotaihoz, általában egy bizonyos célra, mint például a szoftvertermék egy kiadása egy külső csoport számára vagy pedig egy bizonyos termékvonal azonosítása.

- A **verzió** kifejezés egy szoftver konfigurációs elem egyedi és megkülönböztethető állapotait jelenti.

Verziózási fogalmak: verzióazonosító

Definíció (SEVOCAB):
Kiegészítő információ egy konfiguráció elem verzióinak megkülönböztetéséhez.

- A verzióazonosítókat általában növekvő sorrendben rendelik hozzá egy szoftver konfigurációs elem különböző verzióihoz.
- Ez azt jelenti, hogy egy rendezés van értelmezve a verzióazonosítók halmazán.

Verziózási alapfogalmak: kiadás

- Egy **szoftver (rendszer) kiadás** (***software release***, ***system release***) egy szoftver (szoftverrendszer) egy olyan verziója, melyet elérhetővé tesznek az ügyfelek számára.
- A kiadások két fajtája: fő kiadás (*major release*) és *minor release*.

Verziózási fogalmak: Changelog (1)

- Egy *changelog* egy olyan állomány, mely egy gondosan összeállított és időrendbe rendezett listát tartalmaz arról, hogy egy projekt egyes verzióiban milyen említésre méltó változások történtek.
  - Forrás: Olivier Lacan. *Keep a Changelog*. https://keepachangelog.com/ https://github.com/olivierlacan/keep-a-changelog
- Changelog formátum konvenciók:
  - https://github.com/olivierlacan/keep-a-changelog/blob/main/CHANGELOG.md
  - Vincent Weevers. *Common Changelog*. https://common-changelog.org/ https://github.com/vweevers/common-changelog

Verziózási fogalmak: Changelog (2)

- Eszközök:
  - conventional-changelog-cli (programozási nyelv: JavaScript; license: MIT License) https://github.com/conventional-changelog/conventional-changelog/tree/master/packages/conventional-changelog-cli
  - git-cliff (programozási nyelv: Rust; license: Apache License 2.0/MIT License) https://git-cliff.org/ https://github.com/orhun/git-cliff
  - github-changelog-generator (programozási nyelv: Ruby; license: MIT License) https://github.com/github-changelog-generator/github-changelog-generator
- GitHub támogatás: Automatically generated release notes

Verziókezelés

Definíció (Sommerville):

Egy szoftverrendszerben és komponenseiben történő változások kezelésének folyamata, ahol tudható, hogy a komponens/rendszer egyes verzióiban milyen változások történtek, és vissza lehet állítani a komponensek/rendszer korábbi verzióit.

- Más néven: *revision control*, *source control*, *source code management*

Verziókezelő rendszerek (1)

Definíció (Sommerville):
A verziókezelési folyamatok támogatására fejlesztett szoftvereszközök.

- A verziókezelő rendszerek két fajtája: centralizált (például Subversion) és elosztott (például Git).

Verziókezelő rendszerek (2)

Példák verziókezelő rendszerekre:

- Apache Subversion https://subversion.apache.org/
- Fossil https://www.fossil-scm.org/
- Git https://git-scm.com/
- Mercurial https://www.mercurial-scm.org/

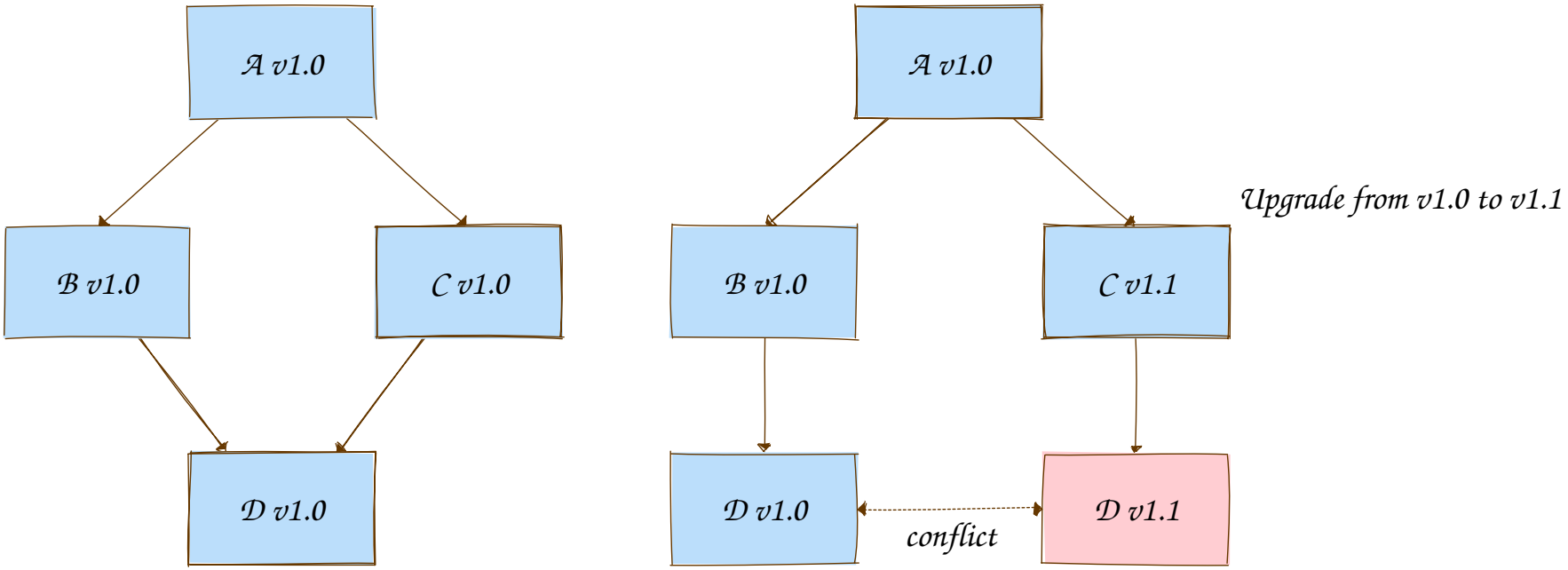
SemVer bevezető

Miért van szükség a verziószámokhoz egy formális specifikációra?

- A függőségkezeléshez pontosan meghatározott verziószámok szükségesek, melyek világos és rugalmas függőség-specifikációkat tesznek lehetővé.
- Kapcsolódó fogalom: verziópokol (*version hell*), függőségi pokol (*dependency hell*)

Verziópokol

Példa:



Mi a szemantikus verziószámozás?

- Az általános bevett gyakorlaton alapuló egyszerű szabályok és követelmények a verziószámok kiosztásához és növeléséhez.
- Tetszőleges olyan szoftverhez használható, mely nyilvános API-val rendelkezik.
  - Az API változásai a verziószámának növelésével kerülnek kifejezésre.
- Webhely: https://semver.org/
- Tároló: https://github.com/semver/semver

SemVer specifikáció

- Aktuális specifikáció: *Semantic Versioning 2.0.0*
  https://semver.org/spec/v2.0.0.html
  - Számos nyelven elérhető.
  - Magyar fordítás: https://semver.org/lang/hu/spec/v2.0.0.html
- A Szemantikus verziószámozás specifikáció eredeti szerzője Tom Preston-Werner, a GitHub egyik társalapítója.

SemVer: normál verziószámok

A normál verziószámok $X.Y.Z$ formájúak, ahol X, Y és Z nemnegatív egész számok:

- X: főverzió (*major version*),
- Y: alverzió (*minor version*),
- Z: *patch* verzió.

SemVer összefoglalás

Növeld

- a főverziót, amikor a korábbi verzióval inkompatibilis módon változik az API;
- az alverziót, amikor a korábbi verzióval kompatibilis módon vezetünk be új funkcionalitást;
- a *patch* verziót, amikor a korábbi verzióval kompatibilis hibajavítások történnek.

SemVer részletek

- Miután kiadásra került egy verziózott csomag, a verzió tartalma nem módosítható. Bármilyen módosítást egy új verzióként kell kiadni.
- A nulla főverzió a kezdeti fejlesztéshez van fenntartva.
  - Bármilyen változás történhet, az API nem tekinthető stabilnak.
- A *patch* verziót 0-ra kell visszaállítani az alverzió növelésekor (például `1.2.3` $\rightarrow$ `1.3.0`).
- A *patch* verziót és az alverziót 0-ra kell visszaállítani a főverzió növelésekor (például `0.9.15` $\rightarrow$ `1.0.0`).

Semver: breaking changes

Kapcsolódó fogalom:

- ***Breaking change***: nem visszafelé kompatibilis változás egy nyilvános API-ban.
  - Az API kliensei számára fordításidejű, szerkesztésidejű vagy futásidejű hibákat okoz.

SemVer: kiadás előtti verziók

Egy kiadás előtti (*pre-release*) verzió $X.Y.Z\text{-}V$ formájú, ahol V alfanumerikus karakterekből és kötőjelekből (`[0-9A-Za-z-]`) álló azonosítók egy pontokkal elválasztott sorozata.

- A kiadás előtti verziók alacsonyabb precedenciájúak, mint a megfelelő normál verziók.
- Egy kiadás előtti verzió azt jelzi, hogy a verzió instabil és lehet, hogy nem elégíti ki a megfelelő normál verzió tervezett kompatibilitási követelményeit.
- Példák: `1.0.0-alpha`, `1.0.0-alpha.1`, `1.0.0-beta.2`

SemVer: összeállítási metaadatok

Alfanumerikus karakterekből és kötőjelekből (`[0-9A-Za-z-]`) álló azonosítók egy pontokkal elválasztott sorozata követheti egy plusz jel után a *patch* vagy a kiadás előtti verziót.

- Az ilyen összeállítási metaadatokat figyelmen kívül kell hagyni a verzió precedencia megállapításakor.
- Példák: `1.0.0-alpha+001`, `1.0.0+20230304142500`

SemVer: verzió precedencia meghatározása (1)

- A verziószámok felbontása fő-, al- és *patch* verzióra. Az összeállítási metaadatok figyelmen kívül hagyása.
- Az első olyan komponens meghatározása balról jobbra haladva, ahol a verziószámok eltérnek, ez a komponens kerül összehasonlításra a verziószámokból.
  - A fő-, al- és *patch* verziók összehasonlítása numerikusan történik.
    - Példa: `1.0.0` < `1.0.1` < `1.2.0` < `1.10.0` < `1.10.1` < `2.0.0`
  - A fő-, al- és *patch* verziók egyezése esetén egy kiadás előtti verzió kisebb precedenciával bír egy normál verzióhoz képest.
    - Példa: `1.0.0-alpha` < `1.0.0`

SemVer: verzió precedencia meghatározása (2)

- Azonos fő-, al- és *patch* verziójú kiadás előtti verzióknál a pontokkal elválasztott azonosítók összehasonlítása balról jobbra az első eltérésig.
  - Számjegyekből álló azonosítók összehasonlítása numerikusan történik.
  - Betűket vagy kötőjeleket tartalmazó azonosítók összehasonlítása lexikografikusan történik ASCII sorrend szerint.
  - A numerikus azonosítók precedenciája mindig alacsonyabb a nem numerikus azonosítókéhoz képest.
  - *pre-relase* mezők egy hosszabb sorozata nagyobb precedenciájú egy rövidebb sorozatnál, ha a rövidebb sorozat tagjai páronként egyenlőek a hosszabb sorozat megfelelő tagjaival.
- Példa: `1.0.0-alpha` $<$ `1.0.0-alpha.1` $<$ `1.0.0-alpha.beta` $<$ `1.0.0-beta` $<$ `1.0.0-beta.2` $<$ `1.0.0-beta.11` $<$ `1.0.0-rc.1` $<$ `1.0.0`

SemVer felhasználások

Csomag ökoszisztémák és csomagkezelők:

- Node.js: npm
  - Lásd: https://docs.npmjs.com/about-semantic-versioning
- PHP: Composer
  - Lásd: https://getcomposer.org/doc/faqs/which-version-numbering-system-does-composer-itself-use.md
- Rust: Cargo
  - Lásd: https://doc.rust-lang.org/cargo/reference/resolver.html#semver-compatibility
- .NET: NuGet
  - Lásd: https://learn.microsoft.com/en-us/nuget/concepts/package-versioning

SemVer eszközök

- composer/semver (programozási nyelv: PHP; licenc: *MIT License*)
  https://github.com/composer/semver
- GitVersion (programozási nyelv: C#; licenc: *MIT License*)
  https://gitversion.net/ https://github.com/GitTools/GitVersion
- semver (programozási nyelv: JavaScript; licenc: *ISC License*)
  https://www.npmjs.com/package/semver
  https://github.com/npm/node-semver
- semver (programozási nyelv: Python; licenc: *New BSD License*)
  https://github.com/python-semver/python-semver
  https://python-semver.readthedocs.io/en/latest/
- Java SemVer (programozási nyelv: Java; licenc: *MIT License*)
  https://github.com/zafarkhaja/jsemver

Más verziószámozási sémák

- *Calendar Versioning*: https://calver.org/ https://github.com/mahmoud/calver
  - Példa:
    - Ubuntu: The Ubuntu lifecycle and release cadence
- *ZeroVer: 0-based Versioning*: https://0ver.org/ https://github.com/mahmoud/zerover
- Python: PEP 440: Version Identification and Dependency Specification
- TEX: a TEX verziószáma a π-hez konvergál, a legutóbbi verzió a `3.141592653` számú.
  - Lásd: https://www.ctan.org/pkg/tex

Apache Maven és a szemantikus verziószámozás (1)

- A Maven verzió rendezési algoritmusa nem kompatibilis a Szemantikus verziószámozás 2.0.0 specifikációéval.
  - Például a Maven nem kezeli a plusz jelet speciális karakterként.
  - A szemantikus verziószámozástól eltérően a Maven nem tulajdonít jelentést a verziószámoknak.
- Lásd:
  - POM Reference - Version Order Specification
  - `ComparableVersion` (Javadoc)

Apache Maven és a szemantikus verziószámozás (2)

Empirikus megfigyelés a központi Maven tárolóra:

- A könyvtár frissítések 83,4%-a megfelel a szemantikus verziószámozásnak.
- A tanulmányhoz felhasznált eszköz:
  - Maracas (programozási nyelv: Java; licenc: *MIT License*)
    https://alien-tools.github.io/maracas/
    https://github.com/alien-tools/maracas
- Lásd:
  - Lina Ochoa, Thomas Degueule, Jean-Rémy Falleri, Jurgen Vinju. *Breaking bad? Semantic versioning and impact of breaking changes in Maven Central: An external and differentiated replication study*. Empirical Software Engineering, 2022, 27 (3).
    https://hal.science/hal-03378089

Kiadáskezelés

Eszközök:

- semantic-release (programozási nyelv: JavaScript; licenc: MIT License)
  https://semantic-release.gitbook.io/semantic-release/
  https://github.com/semantic-release/semantic-release
- GitHub támogatás: Releasing projects on GitHub

Commit üzenet konvenciók (1)

- Számos említett szoftvereszköz (például a semantic-release és a *changelog* generátorok) egy *commit* üzenet konvención alapul.
- Egy ilyen konvenciót határoz meg a Conventional Commits: https://www.conventionalcommits.org/

Commit üzenet konvenciók (2)

Conventional commits:

- Commit üzenet felépítés:

  `<típus> ['(' hatáskör ')'] ['!'] ': ' <leírás>`

- Típusok: `build`, `chore`, `ci`, `docs`, `feat`, `fix`, `style`, `refactor`, `perf`, `test`, ...
- Példák:
  - `build: update dependency versions in pom.xml`
  - `build!: bump minimum JDK version to 21`
  - `docs: fix broken links in README.md`
  - `feat(i18n): add Hungarian language support`