

7. Управление на конфигурации- Резюме

Поради факта, че софтуерът се променя често, системите (С) могат да бъдат разглеждани като **набор от версии**, всяка от които трябва да бъде поддържана и управлявана.

Версията реализират предложения за промени, корекции на грешки и адаптации за различни хардуерни и операционни системи.

Управлението на конфигурацията (УК) се занимава с **политики, процеси и инструменти за управление на променящи се софт**. С. Необходимо ви е УК, защото е лесно да се загуби следа от това какви промени и версии на компоненти са били вкл. във всяка версия на С.

Действия в управлението на конфигурацията:

- **Управление на промените**- Проследяване на **заявки за промени** в софтуера **от клиенти и разработчици**, анализ на разходите и въздействието на промените, и вземане на решения дали промените да бъдат внедрени.
- **Управление на версията**- Проследяване на многообразни версии на компоненти на системата и гарантиране, че промените, направени от различни разработчици, не се намесват помежду си.
- **Изграждане С**- Процесът на сглобяване на програмни компоненти, данни и библиотеки и тяхното компилиране, за да се създаде изпълнима С.
- **Управление на Release**- Подготвяне на софтуер за **външно пускане и проследяване на версията на системата**, които са били пуснати за използване от клиентите.

Configuration item (Конфигурационен елемент) or **software configuration item (SCI)** - Това е всеки елемент, свързан със

софтуерен проект, който се управлява и проследява. Това може да бъде:

- Код (source code)
- Документация
- Дизайн файлове

Тестови данни

Всеки configuration item (SCI) има уникално име и различни версии. Например, файл login.js може да има версии v1.0, v1.1 и т.н.

Configuration control - Това е процесът на управление на промените в конфигурационните елементи. Целта е да се осигури, че всяка промяна е одобрена, документирана и контролирана.

◆ Пример: Един програмист прави промени в login.js.; Промените преминават преглед (code review).; Ако всичко е наред, те се одобряват и стават част от проекта.

Version - Екземпляр на configuration item, който се различава по някакъв начин от други екземпляри на този елемент. Всяка version има уникален идентификатор, който често се състои от **името на configuration item** плюс номер на версията.

Baseline е **колекция от версии** на компоненти, които изграждат **дадена система**. **Baselines** са контролирани, което означава, че версийте на компонентите, съставляващи системата, не могат да бъдат променяни. Това гарантира, че винаги трябва да е възможно възстановяването на **baseline** от неговите съставни компоненти.

Codeline е набор от версии на софтуерен компонент и други configuration items, от които този компонент зависи.

Mainline - **Последователност от baselines**, представляващи различни версии на дадена система.

Release- Версия на С, пусната за клиенти и др. за използване.

Workspace - Лично работно пространство, в което софтуерът може да бъде модифициран, без да се засягат други разработчици, които също могат да го използват или променят.

Branching - Създаването на нова **codeline** от версия в съществуваща **codeline**. Новата **codeline** и съществуващата **codeline** след това могат да се развиват независимо една от друга.

Merging -Създаването на нова версия на софтуерен компонент чрез сливане на отделни версии от различни **codelines**. Тези **codelines** може да са били създадени от предишен **branch** на някоя от участващите **codelines**.

CodeLine (Кодова линия) - Това е основната линия на разработка на кода- **основната версия**, върху която работят програмистите.

2. Baseline (Базова версия)-**Това е стабилна версия на кода, която е "замразена" – използва се като отправна точка за бъдещи промени.** Например, ако имате версия 1.0 на приложение, тя може да бъде базова версия, а след това да се надгражда.

3. Configuration Management (Управление на конфигурацията)-

процесът на **контролиране на разл. версии на кода**. Позволява **да се проследяват и управляват промените**, така че да няма хаос.

4. Version (Версия) -**Всеки път, когато кодът се променя и записва официално, се създава нова версия.** Например: v1.0, v1.1, v2.0.

5. Mainline (Основна линия)-**Това е главната версия на кода, върху която се базират всички други версии и промени.** - **main** или **master**.

6. Branching (Разклоняване)-процесът на създаване на нов клон (branch) от главната версия, за да се разработят нови функции или корекции, без да се засяга основния код. Например, feature-login може да е клон за нова система за вход.

7. Merging (Сливане) - След като приключите работа по даден клон, го сливате (merge) обратно в главната линия (main). Това обединява новите промени с основния код.

👉 Пример: Започвате с mainline (main). Създавате branch (feature-new). Работите по новата функция и правите промени. Сливате (merge) новите промени обратно в main, след като всичко е тествано.

System building - Създаването на изпълнима версия на система чрез компилиране и свързване на подходящите версии на компонентите и библиотеките, които съставляват системата.

Управление на промените(УП)-Организационните нужди и И се променят през целия жизнен цикъл на С, грешките трябва да бъдат коригирани, а С – адаптиран към промени в тяхната среда.

-УП има за цел да гарантира, че еволюцията на системата е контролиран процес и че се дава приоритет на най-спешните и икономически ефективни промени.

-Процесът на УП е свързан с анализ на разходите и ползите от предложените промени, одобряване на онези, които са оправдани, и проследяване на компонентите в С, които са били променени.

Фактори при анализа на промените: Последствията от неизвършването на промяната; Ползите от промяната; Броят на потребителите, засегнати от промяната; Разходите за извършване на промяната; Цикълът на пускане на продукта

Управлението на версиите (Version Management - VM) е процесът на проследяване на различните версии на софт. компоненти/**configuration items**, както и на С, в които компонент. се използват.

Включва също така **гарантиране**, че промените, направени от различни разработчици в тези версии, не си пречат взаимно.

-УП - процес на управление на **codelines** и **baselines**.

Codeline е последователност от версии на изходен код, като по-късните версии са производни на по-ранните версии. **Codelines** обикновено се отнасят до компоненти на системите, така че за всеки компонент съществуват различни версии.

Baseline представлява дефиниция на конкретна система. **Baseline** - версии на компонентите, вкл. в C, както и спецификацията на използ. библиотеки, конфигурац. файлове и др. **Baselines** могат да бъдат определяни чрез **configuration language**, който позволява дефиниране на компонентите, включени в дадена версия на конкретна система. **Baselines** са важни, тъй като често се налага възстановяване на определена версия на цялата система.

Системи за управление на версии

-Идентификация на версии и release - Управляваните версии получават уникални идентификатори, когато бъдат качени в C.

Управление на съхранението - За да се намали необходимото дисково пространство за множество версии на компоненти, които се различават минимално, системите за управление на версии обикновено предоставят **механизми за оптимизирано съхранение**.

Запис на историята на промените - Всички **промени**, направени в кода на дадена C или компонент, се записват и съхраняват в регистър.

Независима разработка - С за управление на версии следи кои компоненти са били **checkout-нати** за редактиране и гарантира, че промените, направени от различни разработчици, не си пречат.

Поддръжка на проекти- С за управление на версии може да поддържа разработката на **няколко проекта, които споделят едни и същи компоненти**.

Codeline branches- Обикновено, когато се разработва софтуер, промените в кода се записват като линейна последователност от версии. Например:  v1.0 → v1.1 → v1.2 → v2.0 Но в много проекти не се следва само една линейна версия, а се създават различни паралелни версии на кода. Това се нарича Codeline Branches – няколко независими линии на развитие, които могат да се развиват едновременно.-В даден момент може да се наложи **сливане на codeline разклонения**, за да се създаде нова версия на компонент, която включва всички направени промени.

Ако промените засягат различни части на кода, версийте на компонента могат да бъдат слети **автоматично**, като се комбинират **делтите** (разликите), които се отнасят до кода.

Управлението на конфигурацията (Configuration Management - УК) е процесът на управление на развиваща се софтуерна система. При поддръжка на система се сформира **УК екип**, който гарантира, че **промените се интегрират контролирано** и че **се води документация с детайли за всички извършени промени**.

Основните процеси на управление на конфигурацията са:

- **Управление на промените** (Change Management)
- **Управление на версийте** (Version Management)
- **Сглобяване на системата** (System Building)
- **Управление на изданията** (Release Management)

Управлението на промените включва оценяване на предложенията за промени от клиенти и други заинтересовани страни и вземане на

решение дали е икономически ефективно тези промени да бъдат внедрени в нова версия на системата.

Управлението на версии се занимава с проследяването на разл. версии на софт. компоненти, докато върху тях се правят промени.

Управление на конфигурацията и изграждане на системата

- **Изграждането на С** е процесът на създаване на цялостна, изпълнима С чрез компилиране и свързване на компонентите на С, външните библиотеки, конфигурационните файлове и др.
- Инструментите за изграждане С и инструментите за управление на версии трябва да комуникират, тъй като процесът на изграждане включва извличане на версии на компоненти от **хранилището**, управлявано от С за управление на версии.
- Конфигурационното описание, използвано за идентифициране на **базовата линия**, също се използва от инструмента за изграждане на С.

Функционалност на Build System (Система за изграждане):
Генериране на build скриптове, Интеграция със система за управление на версии, Минимална повторна компилация, Създаване на изпълнима система, Автоматизация на тестовете, Отчитане (Reporting), Генериране на документация

Минимизиране на повторната компилация

- Инструментите за изграждане на система обикновено са създадени, за да **минимизират броя на необходимите компилации**.
- Те правят това, като проверяват дали вече съществува компилирана версия на даден компонент. Ако такава версия е налична, **няма нужда от повторна компилация**.

- **Уникален подпись (signature)** идентифицира всяка версия на сурс кода и обектния код и се променя, когато сурс кодът бъде редактиран.
- Чрез **сравняване на подписите** на сурс и обектните кодове може да се определи дали даден сурс код е бил използван за генериране на обектен код.
- **Всеки файл с код има уикален подпись (signature)** – това е като „пръстов отпечатък“, който се променя, когато кодът се редактира.
- **Какво означава това?** Ако направите промяна в сурс кода (изходния код), неговият подпись също се променя.
- **Зашто е полезно?** Ако сравним подписите на сурс кода и обектния код (компилираната версия), можем да разберем дали даденият сурс код е бил използван за създаване на този обектен код.
- **Пример:**
1Имате файл program.c и го компилирате → получавате program.o.
2Ако след това промените program.c, неговият подпись ще се промени.
3Сравнявайки подписите, можем да проверим дали program.o е актуален или трябва да се компилира отново.
- Това помага за автоматизацията на build системите и гарантира, че винаги работим с правилната версия на кода. 

Идентифициране на файлове

Модификационни времеви марки (Timestamps)-Подписът на сурс кодовия файл представлява **часть и датата на последната му модификация**. Ако сурс кодовият файл на даден компонент е бил модифициран след последната компилация на обектния файл, **системата предполага, че е необходима повторна компилация**.

Контролни суми на сурс кода (Checksums) -Подписът на сурс кодовия файл може да бъде **контролна сума (checksum)**, изчислена от съдържанието на файла. Дори **една малка промяна в сурс кода** води до различна контролна сума, което гарантира, че файловете с различни контролни суми са наистина различни.

Timestamps срещу Checksums – Обяснение на прост български

✓ Timestamps (Времеви отметки)-Това е времето (**дата и час**), когато даден файл е бил последно променен.Build системите често използват timestamps, за да проверят дали файлът е обновен и трябва ли да се компилира отново. ◆ Пример:

1 Ако файлът program.c е бил **променен в 10:00, но компилиран в 9:30**, значи трябва да се компилира пак.
2 Build системата вижда, че **timestamp-ът е по-нов, и прави нова компилация**.

! Проблем: Ако **прехвърлите файла на друг компютър или часовникът на системата се промени, timestamp-ът може да не е надежден.**

✓ Checksums (Контролни суми)-Това е уникален код (число или низ), генериран от съдържанието на файла. Ако дори един символ в кода се промени, checksum-ът също ще се промени.



Пример:

1 program.c има checksum abc123.

2 Добавяте нов ред код → checksum-ът става xyz789.
3 Build системата вижда, че checksum-ът е различен, и знае, че файлът трябва да се компилира отново.

! Предимство: Checksums са по-надеждни от timestamps, защото не зависят от датата и часа на системата

Кой метод е по-добър?

- ➡️ Timestamps са по-бързи, но могат да бъдат неточни.
- ➡️ Checksums са по-надеждни, но изискват повече ресурси за изчисление. Много build системи използват комбинация от двете, за да гарантират точност и ефективност. 🚀

Agile Building

1. Извличане на основната система (**mainline system**) от **системата за управление на версии** в личното работно пространство на разработчика.
2. **Изграждане на системата** и изпълнение на автоматизирани тестове. Ако тестовете се провалят, **билдът се счита за неуспешен** и трябва да се уведоми разработчикът, който е качил последната версия.
3. Извършване на промени в компонентите на системата.
4. **Изграждане на системата в личното работно пространство** и повторно изпълнение на тестовете. Ако тестовете не преминат, **редактирането продължава**.
5. След като системата премине тестовете, **тя се изпраща към билд системата, но не се комитва като нова baseline версия**.
6. **Изграждане на системата на билд сървъра** и изпълнение на тестовете, за да се гарантира, че **други разработчици не са направили промени, които могат да доведат до грешки**.
7. Ако всички тестове преминат успешно, промените се комитват в основната версия (**mainline**) като нова baseline версия.

Daily Building (Ежедневно изграждане)

- Организацията задава **фиксирано време за доставка на компонентите** (например 14:00).
- Разработчиците трябва да предоставят **нови версии на компонентите до този час**.
- От тези компоненти се **изгражда нова версия на системата**.
- След това системата се **предоставя на екипа за тестване**, който изпълнява **предварително дефинирани тестове**.
- **Откритите грешки** се документират и се връщат на разработчиците за корекция.

Управление на Release(Release Management) -Системен release представлява версия на софтуерна система, която се разпространява до клиентите.

В масовия пазар на софтуер има **два основни типа release**:

- Major releases** – предлагат значителни нови функционалности.
 - Minor releases** – съдържат поправки на грешки и подобрения, базирани на обратна връзка от клиенти.
- При персонализиран софтуер**, всяка версия може да бъде различна за различни клиенти, като **едновременно могат да се поддържат няколко версии на системата**.

Release Tracking (Проследяване на версии)- В случай на проблем, може да бъде необходимо **възпроизвеждане на точно определена версия** на софтуера, предоставена на даден клиент. Когато се създава release, той **трябва да бъде документиран**, за да може по-късно да бъде възстановен в същата форма. Това е особено важно за **вградени**

системи с дълъг жизнен цикъл, които могат да бъдат използвани в продължение на много години.

Release Reproduction (Възпроизвеждане на версия)-За да се документира release, трябва да се запазят:

- Специфичните версии на сорс кодовите компоненти
- Изпълнимите файлове (executables), конфигурационни и данни файлове
- Версията на операционната система, библиотеките и инструментите, използвани при изграждането

Release Planning (Планиране на release)-Освен техническата работа, свързана със създаването на release, трябва да се подготвят **рекламни и маркетингови материали**, за да се убедят клиентите да преминат към новата версия. **Честота на release-ите**:

-Твърде чести release-и могат да отблъснат клиентите, особено ако изискват **хардуерни ъпгрейди** или **допълнителни разходи**.

-Твърде редки release-и могат да доведат до **загуба на пазарен дял**, тъй като клиентите могат да преминат към конкурентни продукти.

Release Components (Компоненти на release)-Освен изпълнимия код, release-ът може да включва: Конфигурационни файлове; Файлове с данни (например съобщения за грешки); Инсталационна програма; Документация (електронна и хартиена); Опаковка и рекламни материали

Фактори, влияещи върху пускането на нови версии (Release Management):

 **Техническо качество на системата** -Ако бъдат открити сериозни грешки, които засягат много клиенти, може да се наложи

спешно пускане на нова версия за поправка на грешките (fault repair release). По-малки грешки се поправят с patches (кръпки), които често се разпространяват през интернет и се добавят към тек. версия.

- ✓ Промени в платформата-Ако излезе нова версия на операционната система, може да е необходимо да се направи нова версия на софтуера, която да е съвместима с нея.
- ✓ Петият закон на Леман-Колкото повече нови функции се добавят към системата, толкова по-голяма е вероятността да се появят нови бъгове. Затова след голямо обновяване често следва версия, фокусирана върху поправяне на грешки и подобряване на производителността.
- ✓ Конкуренция-Ако конкурентен продукт предлага нови функционалности, може да се наложи нова версия, за да не се загубят клиенти.
- ✓ Маркетингови изисквания-Понякога маркетинговият отдел обещава, че новата версия ще бъде налична на определена дата, което налага спазване на срокове.
- ✓ Изисквания на клиенти-За персонализирани системи клиентите могат да поискат и заплатят за конкретни промени. Новата версия се пуска веднага след като тези промени са готови.
- 💡 Накратко: Нови версии се пускат поради грешки, нови технологии, конкуренция, маркетингови ангажименти или клиентски заявки.

Изграждането на система включва компилиране и свързване на компонентите в изпълним файл.

Софтуерът трябва **често да се компилира и тества**, за да се откриват грешки на ранен етап.

Управлението на release-ите включва **решения за дати на release, подготовка на документи и архивиране на версии**.

Непрекъсната интеграция (Continuous Integration - CI) – практика в софтуерната разработка, при която промените в кода се интегрират (сливат) често – обикновено няколко пъти на ден.