КОМПЬЮТЕРНАЯ МАТЕМАТИКА

Инструментальные средства информационных технологий

УДК 681.3

А.В. КОГУТ

ПРО ПРИНЦИПИ CI НА ПАЛЬЦЯХ

Пояснюється найпростішим чином, що таке неперервна інтеграція. Розглядаються основні принципи практики неперервної інтеграції у командах розробників.

© А.В. Когут, 2016

Вступ. Уявимо, що у нас є кілька програмістів, які працюють над різними модулями додатку. Розроблений функціонал прекрасно працює у кожного із програмістів локально. В кінці ітерації розробники намагаються змусити працювати всі 3 модуля і тут з'являється багато проблем з інтеграцією. Програмісти швидко намагаються розібратись з проблемою і вирішити її. Ну або ж знайти винного в усіх бідах. При цьому тестувальники не приймають ніяких мір, щоб почати тестування системи. Як результат команда тестувальників має нашвидкоруч працювати, щоб встигнути до призначеного терміну.

Ці обидва рішення можуть виявитись досить важкими для команди. Вирішення проблеми насправді лежить на поверхні – проводити інтеграцію максимально часто, можна сказати неперервно. Таким чином ми автоматично вирішуємо багато проблем з інтеграцією. Фактично після кожного коміту ми бачимо і можемо починати виправляти помилки, які пов'язані з інтеграцією коду. Це дозволяє вирішувати проблеми своєчасно. Крім цього, у команди тестувальників завжди є актуальна версія додатку, що дозволяє економити час. Часто скрипти запускаються вручну, що також негативно впливає на весь процес – всю рутину потрібно автоматизувати. Це зекономить час, не дозволить відволікатись на запуск скриптів та вирішить проблему людського фактора. Всьому цьому дано термін – неперервна інтеграція [1 – 3].

Що таке неперервна інтеграція (Continuous Integration) – це практика розробки програмного забезпечення, яка полягає у виконанні частих автоматизованих збірок проекту для найшвидшого виявлення та вирішення інтеграційних проблем.

Простіше кажучи, система CI – це деяка програма, яка слідкує за вашим Source Control, і при появі там змін автоматично підтягує їх, робить білд, проходить тести і, можливо, дещо ще, в залежності від конфігурацій. У випадку невдачі вона дає про це знати всім зацікавленим людям, у першу чергу – останньому розробнику, який робив коміт.

Що це дає? Те, що в будь-який момент часу ми маємо достовірну інформацію про стан вихідних кодів у системі. Якщо останній білд був невдалим, то брати свіжу версію із системи контролю версій не можна, оскільки вона може навіть не скомпілюватись. По-друге, дуже просто знайти того, хто все зламав – зазвичай, це розробник, який робив останній коміт.

Якщо ж на проекті є юніт-тести, то проганяння їх при кожному білді дає деяку гарантію відсутності регресійних багів (коли в одному місці помилку виправили, а в іншому місці з'явилась нова).

Крім того, можна включити в білд різноманітні метрики якості коду, наприклад, покриття, статичний аналіз, пошук коду, який дублюється, автоматизувати установку на тестову машину і т. д.

Continuous Integration відноситься до agile-практик. Agile передбачає ітераційний процес, з великою кількістю повторень циклу розробки, тестування та видачі продукту.

Розглянемо, що таке Continuous Integration більш детально.

Принципи CI. Якщо хоча б один з принципів не виконується, то процес не можна називати неперервною інтеграцією.

Build однією командою. «Натиснути одну кнопку – і отримати білд (деплой) додатку». Це означає, що розробник повинен мати можливість отримати білд свого додатку вводом однієї команди і скрипт виконає всі необхідні дії. Деякі проекти використовують власні білд-скрипти або ж деякі команди, які вводять в консольці, щоб отримати білд. Це не є хорошою практикою. Найкраще мати єдиний, універсальний скрипт для всіх проектів компанії, який має виконувати всі процеси необхідні для білда автоматично.

Переваги автоматизованого білда:

– покращення якості продукту;
– прискорення компіляції;
– звести до мінімуму «погані» білди;
– виключити (або звести до мінімуму) конфлікти залежностей;
– зберігати історію релізів та білдів.

Фактично, перевагами є покращення якості продукту та уникнення ризиків. Такі скрипти називаються білд-системами. Сьогодні є досить багато таких систем. Найпопулярнішими – це Ant, Maven, CMake.

Автоматизування білдів

Є три типи білдів:

1) локальний (локально на машині розробника перед комітом; впевнитись, що нема «поганого» коду);

2) інтеграційний (збирається на спеціальній машині; передбачає послідовність потрібних операцій; коли білд готовий, відправляється звіт);

3) релізний (збирається на спеціальній машині; збірка надсилається команді тестувальників; потрібні артефакти публікуються в репозиторій).

Є чотири типи білд-механізмів:

1) on demand (Розробник вручну запускає CI білд);

2) scheduled (так звані «нічні білди». Є фіксовані моменти часу, коли повинен бути запущений білд);

3) polling for Changes (CI сервер запитує у системи контролю версій, чи були внесені зміни. Якщо зміни є, то збирається збірка. Частота запитів налаштовується);

4) build by Event (деяка зовнішня подія запускає білд. Наприклад, коміт в систему контролю версій).

Існує деяка залежність між типом білда та його механізмом. Локальні білди запускаються on demand; для інтеграційних застосовний будь-який механізм; релізні виконуються on demand або ж scheduled.

Відокремлення білд-скрипту від IDE. IDE може залежати від білд-скрипту, але він не має бути залежним від IDE. Багато IDE мають власні вбудовані білд-механізми і розробники часто їх використовують. Для CI це не є поганою практикою, потрібно лише сконфігурувати окремий білд-скрипт так, щоб його можна було запустити з IDE. Така практика підтримує повністю ідентичний білд процес на різних машинах розробників та інтеграційних серверах (запобігає виникненню проблеми «на моїй машині це працює»).

Централізування доступу до проекту. Має бути можливість отримати всю необхідну інформацію у будь-який час та в будь-якому місці. Неможливо забезпечити гнучкі білди без того, щоб тримати коди проекту та залежності в окремому репозиторії. Машина білда має працювати як термінал: отримувати всю необхідну інформацію з системи контролю версій. Дана практика дозволяє кожному білду бути прозорим та незалежним.

Якщо падати, то швидко. Якщо система виконує багато різних перевірок, то та частина з них, яка найбільш ймовірно, що падатиме, має виконуватись у першу чергу. Також це стосується і тестів: найбільш критичні тести та ті, які падають найчастіше, мають виконуватися першими. Ця практика дозволяє зменшити час білда і, як результат, час на отримання звіту.

Білд для будь-якого середовища. Білд має бути незалежним від платформи з окремим білд-скриптом. Тобто, де б не стартували білди: на власній локальній машині, CI сервері чи машині іншого розробника – на виході мають отриматись ідентичні файли.

Використання окремої інтеграційної машини для білда. Інколи вся команда використовує машину певного розробника для того, щоб збирати інтеграційні білди. Це не є хорошою практикою тому, що

1. Машина розробника не має достатньо ресурсів для цього.
2. Розробник має підтримувати репозиторій (тобто, витрачати час на нерозробницьку діяльність).
3. Така машина не може гарантувати правильного середовища.
4. Активність одного розробника впливає на всю команду.

Щоб уникнути таких проблем, потрібно використовувати окремий сервер. Сьогодні існує багато таких серверів: Jenkins, Bamboo, TeamCity, CruiseControl та ін. Їх можна розширювати плагінами, вони підримують більшість існуючих систем контролю версій, мають різні бекап механізми.

Виконання інтеграційних білдів вручну (необов'язково). Іноді такий білд може бути корисним. Він запускається з CI сервера чи машини розробника. Якщо з машини розробника, то відбувається додаткова перевірка на те, чи робочий код. У випадку CI сервера може знадобитись перезавантаження. Для великих команд цей принцип не дуже добре застосовувати, оскільки кожен ручний крок змушує команду працювати повільніше.

Виконування швидких білдів. Має бути можливість отримати звіт настільки швидко, наскільки це можливо.

Для того, щоб швидко отримати звіт потрібно:

– збільшити потужність машин для інтеграційних білдів;
– часто робити коміти;
– оптимізувати білд-скрипт;
– покращити інфраструктуру;
– покращити виконання тестів;
– покращити свій код та стиль, уникати складнощів коду.

Білд за стадіями. Для того, щоб пришвидшити час білда, часто є корисним розділити білд на кілька частин. Як правило, перша частина є найшвидшою. Вона компілює код та виконує тести та інспекцію коду. Також вона виконує деплой на сервер.

Звіти після проходження цієї частини показують розробнику чи є його код припустимим для подальшого інтеграційного тестування. Отже, такий білд реалізує принцип «швидкого білда». Друга частина – це «важке» інтеграційне тестування та релізи. Фактично, проходження всіх типів тестів. Можуть бути й інші частини. Наприклад, вони можуть збирати релізи, вручну стартувати білди чи специфічні тести.

Негайне виправлення зламаного білда. Зазвичай, якщо певний коміт ламає білд, то вся команда чи кілька людей у паніці намагаються налаштувати систему на робочий лад. І це правильно, тому що система має негайно бути виправлена і розробники мають працювати зі стабільним робочим білдом. Зазвичай

білд падає у тому випадку, якщо розробник був неуважним і закомітив неробочий код або ж несумісний з тією версією, що лежить у спільному репозиторії (наприклад, два розробники одночасно вирішили залити один і той же файл і, як результат, отримали конфлікт).

Як казав Кент Бек: «Ніхто не має важливішою задачі, ніж виправлення самого білда». І це означає, що не існує задачі більш пріоритетної, ніж отримання стабільного робочого білда.

Висновки. Отже, неперервна інтеграція може принести дуже багато переваг для організації, що її використовує. Зокрема, можна попрощатись з довгими та виснажливими інтеграціями, збільшити прозорість виконання завдань та можливості виправлення помилок, зменшити час для відлагоджування та збільшити його для розробки нових ідей.

Які б практики не вводили, що б не робили, не існує ніякого способу дати 100 % гарантію того, що розробники не будуть вносити в систему контролю версій код, який не компілюється чи не збирається. Звичайно, неперервна інтеграція не оберігає такої можливості також, проте вона робить більш простим шлях для виявлення та знешкодження помилок.

Варто зазначити, що процес розробки є індивідуальним для кожної команди. Наведені в статті практики є лише основою. Команда розробників може орієнтуватись на основні принципи та вносити свої доповнення в залежності від ситуацій, з якими їй доводиться зіштовхуватися.

А.В. Когут

О ПРИНЦИПАХ СИ НА ПАЛЬЦАХ

Объясняется самым простым образом, что такое непрерывная интеграция. Рассматриваются основные принципы практики непрерывной интеграции в командах разработчиков.

A.V. Kohut

ABOUT CI PRINCIPLES IN A HAND-WAVING TERMS

The definition and usage of continuous integration are considered in a hand-waving terms. The main principles of continuous integration in the teams of developers are considered.

1. *http://www.martinfowler.com/articles/continuousIntegration.htmlanko*
2. *http://habrahabr.ru/post/229873/*
3. *http://habrahabr.ru/company/icl_services/blog/262173/*

Одержано 30.08.2016

Про автора:

Когут Антоніна Володимірівна,

студентка магістратури факультету кібернетики
Київського національного університету імені Тараса Шевченка.

E-mail: antoshka412@gmail.com