# Ciągła Integracja

**Krystian Kaczor**

Pomyśl ile czasu i energii można zaoszczędzić budując nową wersję aplikacji zawsze w ten sam sposób i natychmiast po wprowadzeniu zmian do kodu. Do tego możesz automatycznie wykonać wszystkie testy w stabilnym środowisku testowym, powiadomić zespół rezultacie i opublikować wyniki wraz z aplikacją gotową do pobrania online. Przestań zastanawiać się, dlaczego wcześniej tego nie zrobiłeś i poznaj Hudson CI już dziś.

**Dowiesz się...**

- Na czym polega Ciągła Integracja
- Jaka jest rola Ciągłej Integracji na projektach prowadzonych metodami agile
- Poznasz aplikację Hudson CI
- Jak skonfigurować prosty projekt w Hudson CI

**Powinieneś wiedzieć...**

- Na czym polega prowadzenie projektu metodami agile
- Jak zainstalować aplikację webową z pliku WAR

## Co to jest  Ciągła Integracja

Każdy z członków zespołów IT zastanawia się jak zoptymalizować codzienne zajęcia i zautomatyzować powtarzające się, często nudne czynności. Tutaj z pomocą przychodzi nam Ciągła Integracja (*Continuous Integration*).

Co to jest  Ciągła Integracja i dlaczego jest taka ważna w projektach programistycznych, a w szczególności w tych korzystających z metod zwinnych (*agile*)?

W skrócie można powiedzieć, że Ciągła Integracja jest to praktyka polegająca na jak najczęstszej integracji zmian z istniejącym już systemem.

Częsta integracja kodu zmniejsza ilość pracy potrzebnej do łączenia zmian z istniejącym kodem aplikacji i zapewnia wczesne wykrywanie konfliktów, błędów w kompilacji i defektów w kodzie. Wraz z każdą zmianą w kodzie,  w krótkim odstępie czasu do testowania dostępna jest nowa wersja oprogramowania.

### Agile – krótkie przypomnienie

Programowanie zwinne ((ang.) Agile software development) – grupa metodyk wytwarzania oprogramowania opartego o programowanie iteracyjne (model przyrostowy). Wymagania oraz rozwiązania ewoluują przy współpracy samozarządzalnych zespołów, których celem jest przeprowadzanie procesów wytwarzania oprogramowania. Pojęcie zwinnego programowania zostało zaproponowane w 2001 w Agile Manifesto.

Źródło: *http://pl.wikipedia.org/wiki/Agile*

## Dlaczego Ciągła Integracja jest taka ważna w agile?

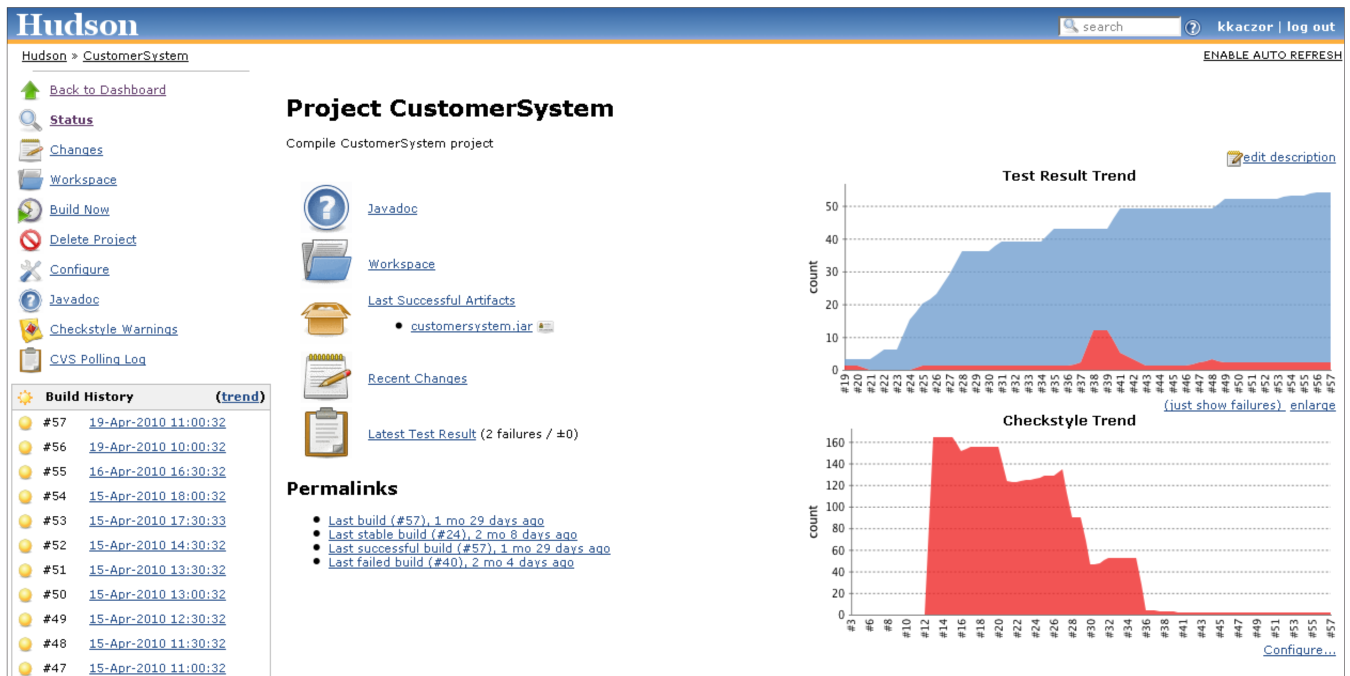
Ze względu na utrzymanie kodu i minimalizację ryzyka utraty pracy programiści powinni jak najczęściej zgłaszać nowy kod do repozytorium kodu. W ten sposób

**Rysunek 1.** *Logo Hudson CI*

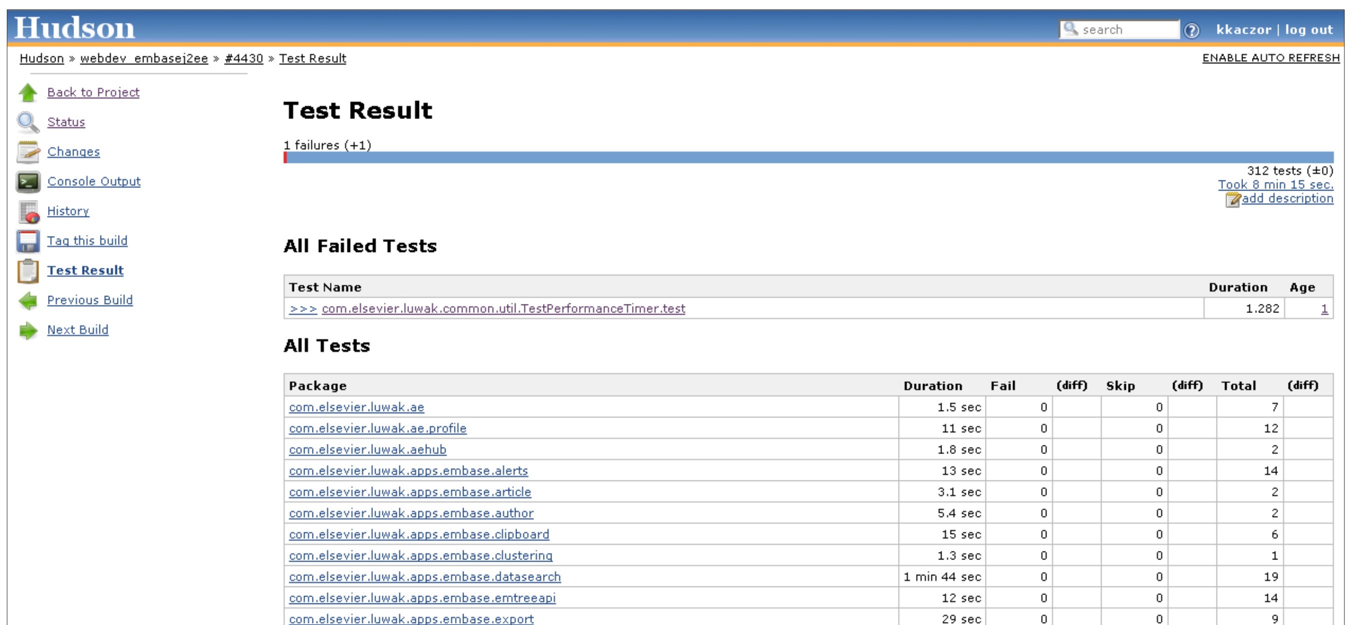
również programiści pracujący nad tymi samymi lub zależnymi komponentami, mogą wcześnie zacząć pracować z nowym kodem.

Jednakże, samo zgłoszenie nowego kodu do repozytorium może jedynie wykryć konflikty w samym kodzie. To nie wystarcza, żeby określić, czy zmiana integruje się z kodem istniejącym w repozytorium. Kolejne problemy mogą pojawić się przy kompilacji, uruchamianiu czy korzystaniu z aplikacji.

Spójrzmy jakie problemy taka sytuacja stwarza w zespole. Przede wszystkim późno wychwycony błąd może spowodować, że cały zespół będzie pracował na kodzie zawierającym błąd i budował kolejne funkcjonalność w oparciu o wadliwe działanie aplikacji propagując błąd w systemie. Może także się okazać, że nie można uruchomić aplikacji i szukanie przyczyny usterki zajmie dużo czasu. Testerzy czy osoby odpowiedzialne za umieszczanie nowych wersji stracą czas na pobranie kodu z błędami, próbę kompilacji i uruchomienia aplikacji w środowisku testowym, czy w reszcie na próbie przetestowania aplikacji i znalezieniu błędu krytycznego typu *NullPointerException* i działanie aplikacji zostanie natychmiast przerwane. Spójrzmy też na ilość czasu traconą na pobieranie, kompilowanie, umieszczanie aplikacji w środowisku testowym w szczególności, kiedy aplikacja i tak nie działa, a te same czynności są wykonywane przez 3, 5, a może nawet 20 osób. Jest to dosko-




**Rysunek 2.** *Przykładowy widok projektu w Hudson CI*




| Test Name | Duration | Age |
|---|---|---|
| >>> com.elsevier.luwak.common.util.TestPerformanceTimer.test | 1.282 | 1 |

| Package | Duration | Fail | (diff) | Skip | (diff) | Total | (diff) |
|---|---|---|---|---|---|---|---|
| com.elsevier.luwak.ae | 1.5 sec | 0 | | 0 | | 7 | |
| com.elsevier.luwak.ae.profile | 11 sec | 0 | | 0 | | 12 | |
| com.elsevier.luwak.aehub | 1.8 sec | 0 | | 0 | | 2 | |
| com.elsevier.luwak.apps.embase.alerts | 13 sec | 0 | | 0 | | 14 | |
| com.elsevier.luwak.apps.embase.article | 3.1 sec | 0 | | 0 | | 2 | |
| com.elsevier.luwak.apps.embase.author | 5.4 sec | 0 | | 0 | | 2 | |
| com.elsevier.luwak.apps.embase.clipboard | 15 sec | 0 | | 0 | | 6 | |
| com.elsevier.luwak.apps.embase.clustering | 1.3 sec | 0 | | 0 | | 1 | |
| com.elsevier.luwak.apps.embase.datasearch | 1 min 44 sec | 0 | | 0 | | 19 | |
| com.elsevier.luwak.apps.embase.emtreeapi | 12 sec | 0 | | 0 | | 14 | |
| com.elsevier.luwak.apps.embase.export | 29 sec | 0 | | 0 | | 9 | |

**Rysunek 3.** *Wyniki testów w Hudson CI*

nała ilustracja czegoś, co w Lean Software Development nazywa się Stratą (*Waste*). Jedną z zasad Lean Software Development jest Minimalizacja Strat (*Minimazing Waste*). System *Continuous Integration* pozwala nam w pełni zautomatyzować wcześniej wymienione kroki, dzięki czemu zespół może spożytkować zaoszczędzony czas na dużo bardziej twórcze zadania. Dodatkową zaletą automatycznego procesu Ciągłej Integracji jest z punktu widzenia testowania są automatyczne testy regresji, które dają nam dużą dozę pewności, że wcześniej dostarczona funkcjonalność po wprowadzeniu zmian nadal działa poprawnie. W projektach prowadzonych przy użyciu metodyk zwinnych każda zmiana powinna być natychmiast zintegrowana i dostępna do testów a na testy regresji nie ma po prostu czasu w standardowych Sprintach (inaczej jest w *Release Sprint*, czy *End Game*). Tutaj automatyczny system ciągłej integracji jest po prostu niezbędny.

Kolejnym elementem usprawniającym pracę zespołu i podnoszącym efektywność jest odpowiednie ustawienie priorytetów:

- Naprawa defektów
- Przegląd kodu
- Tworzenie nowego kodu

Ewentualne defekty naprawia osoba zgłaszająca kod lub jeśli jej akurat tej osoby nie ma to ktoś inny z zespołu. Jedną z zasad pracy w projektach agile jest to, ze kod należy do zespołu a nie indywidualnych osób.

## Co jest potrzebne, żeby mówić o Ciągłej Integracji?

Zobaczmy, z jakich elementów musi składać się system Ciągłej Integracji.

Potrzebujemy:

- Repozytorium kodu takie jak ClearCase, CVS, Subversion, Team Foundation Server.
- Skrypt umożliwiający automatyczne budowanie aplikacji (build script) taki jak Maven, Ant czy make.
- Testy Jednostkowe (*Unit Tests*) napisane w tym samym języku, w którym jest pisana aplikacja.
- Narzędzia do analizy statycznej kodu.
- Opcjonalnie automatyczne umieszczanie i uruchamianie aplikacji w środowisku testowym (*deployment*)
- Opcjonalnie Testy akceptacyjne/GUI
- Wyzwalacz (*trigger*) w postaci włącznika czasowego lub wykrywania zmiany w kodzie, albo połączenia obu.
- System powiadamiania o wynikach procesu i ewentualnych problemach
- Zbiór wyników i interfejs dostępny dla każdego w dowolnym momencie

Można samemu zbudować taki system, ale po co wymyślać koło od nowa. Na rynku istnieją gotowe rozwiązania takie jak Hudson CI, Anthill Pro, Apache Continuum, Apache Gump, Bamboo, CruiseControl,

**Rysunek 4.** *Podgląd konsoli systemowej w Hudson CI*

Team Foundation Server, Rational Team Concert. Część tych rozwiązań jest darmowa.

W tym artykule będę omawiał Hudson CI, ponieważ z tego rozwiązania korzystam na co dzień.

## Kilka słów o Hudson CI

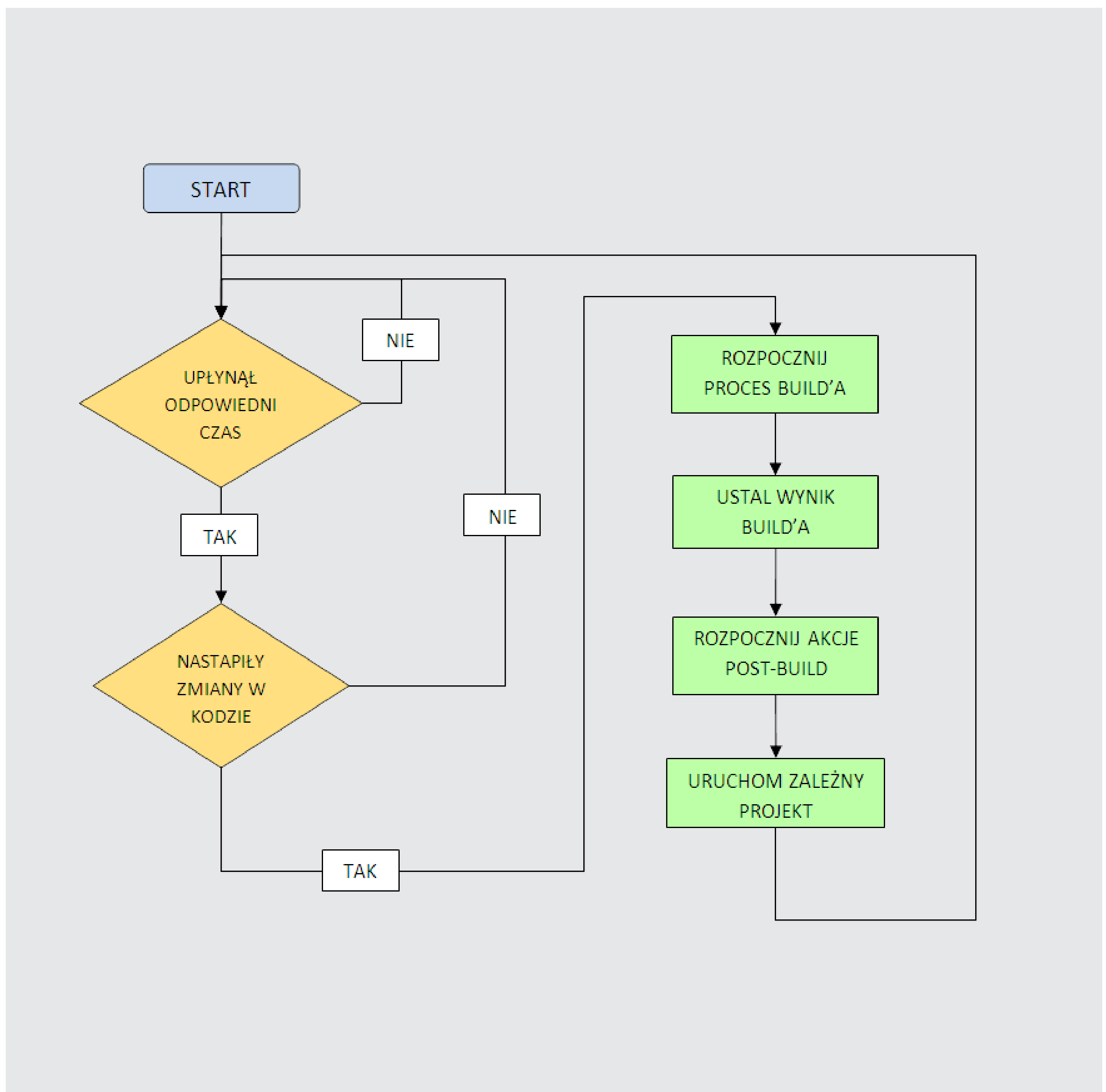
Jest to aplikacja dostarczana w postaci paczki WAR. Paczkę można uruchomić z linii poleceń wpisując `java -jar hudson.war` lub umieścić na serwerze obsługującym Servlet 2.4/JSP 2.0, na przykład Glassfish, Tomcat, JBoss czy Jetty. W ten sposób uzyskujemy także niezależność od systemu operacyjnego.

Serwer CI nie potrzebuje bazy danych, ponieważ zapisuje konfigurację i wyniki w plikach XML.

Hudson CI jest łatwy w instalacji oraz dodatkowo jest powszechnie używany i użytkownicy umieścili procedury instalacji i case study dla różnych środowisk.

To rozwiązanie oferuje przyjazny interfejs graficzny do konfiguracji projektów i kont użytkowników, śledzenia zmian, przeglądania kodu i historii projektów.

Co jest niezwykle ważne, społeczność użytkowników tej aplikacji publikuje dużą ilość różnych pluginów, które umożliwiają integrację z narzędziami do budowania aplikacji, testowymi, wyzwalacze, akcje uruchamiane po buildzie (*post-build*) itd. Integracja z Hudson CI może polegać na możliwości wykonywania komend danego narzędzia wprost z GUI lub interpretację raportów, często również interpretację graficzną trendu np.: trend pokrycia kodu testami.




**Rysunek 5.** *Algorytm działania CI*

Bardzo wygodnym elementem aplikacji jest automatyczne generowanie linków do ostatniego build'a, ostatniego build'a zakończonego sukcesem czy poszczególnych artefaktów. Linki do ostatnich wyników są rozsyłane przez system powiadamiania, który może obsługiwać e-mail, RSS, IRC, Jabber, Twitter itp.

Akcje uruchamiane po build'zie oferują tagowanie kodu w repozytorium, dzięki czemu łatwo można otworzyć to samo środowisko. Hudson potrafi także powiadamiać inne systemy o zadaniach składających się na build i na przykład w systemie Atlassian Jira, Issues rozpoznane na podstawie id umieszczonego w komentarzu zmiany są automatycznie komentowane z informacją, w którym numerze builda zmiana została zintegrowana.

W trakcie uruchomionego build'a można sprawdzać co się dzieje w konsoli systemowej.

## Jak to działa?

- Aplikacja kontrolująca CI sprawdza wyzwalacz, czyli na przykład czy upłynął odpowiedni czas do uruchomienia nowego procesu lub/i czy do repozytorium trafił nowy kod.
- CI ściąga najnowszą wersję kodu.
- CI uruchamia proces budowania aplikacji (build).

**Rysunek 6.** *Konfiguracja builda w Hudson CI*

- CI określa wynik build'a. Standardowo może to być Sukces (*Successful*), Porażka (Failed), Build niestabilny (Build unstable), Build naprawiony (Build fixed).
- Opcjonalnie CI wykonuje akcje uruchamiane po buildzie (*post-build*).

## Kroki w przykładowym skrypcie

- Przygotowanie środowiska i usunięcie pozostałości poprzeniego build'a
- Przetłumaczenie pliki gramatyki na klasy Java przy użyciu narzędzia ANTLR
- Kompilacja
- Uruchomienie testów jednostkowych za pomocą narzędzia DJUnit
- Zebranie danych o pokryciu kodu testami jednostkowymi
- Zebranie wyników testów jednostkowych
- Statyczne sprawdzenie stylu kodu za pomocą narzędzie checkstyle
- Wyliczenie złożoności cyklomatycznej McCabe'a dla wszystkich klas
- Wygenerowanie dokumentacji JavaDoc
- Zbudowanie paczki JAR
- Zbudowanie paczki WAR
- Instrumentalizacja kodu za pomocą narzędzia EMMA
- Zbudowanie paczki WAR z instrumentalizowanym kodem

**Rysunek 7.** *Lampy lava sygnalizują stan builda*

- Oznaczenie build'a jako gotowy do umieszczenia w środowisku testowym

Umieszczenie build'a w środowisku testowym i dalsze kroki należą do kolejnego projektu uruchamianego jako zależność.

## Budowanie zależności pomiędzy projektami

Hudson Ci pozwala nam na budowanie zależności pomiędzy projektami na kilka sposobów:

- Można użyć opcji Build after other proejcts are build w sekcji Build Triggers oraz Build other projects w sekcji Post-build Actions
- Można skonfigurować zależności w samej aplikacji za pomocą kroku Invoke Ant lub Invoke top-level Meaven target i skazanie odpowiedniego pliku w sekcji build

## Wskazówki

Pamiętaj, żeby dostarczać informacje zwrotną jak najszybciej. Przy dużej liczbie testów, gdy cały proces trwa więcej niż 30 minut dobrą praktyką jest rozdzielenie build'a na dwa, gdzie pierwszy jest smoke testem i przypadku napotkania problemu wysyła informacje. W przypadku sukcesu, ostatnim krokiem procesu jest uruchomienie drugiego builda z pełnym zestawem testów lub bardziej czasochłonnymi testami jak na przykład testy GUI z Selenium.

Zainstaluj Continuous Integration jako serwis. W przypadku automatycznego restartu wysłanego przez suport lub braku zasilania, serwer zostanie uruchomiony wraz ze startem systemu operacyjnego.

Ogranicz dostęp do konfiguracji i usuwania artefaktów. Nie chcesz, żeby system krytyczny dla zespołu był narażony na awarię na skutek pomyłki.

**Rysunek 8.** *Lampy w kształcie Misiów sygnalizują stan builda*

**Linki**

Hudson CI, *http://wiki.hudson-ci.org/display/HUDSON/Meet+Hudson*

Udokumentuj konfigurację na projektowym Wiki. Nie powinieneś być jedynym guru od CI. Może się zdarzyć że będziesz na wakacjach albo chory w domu, a zespół będzie potrzebował dodać/zmodyfikować krok lub utworzyć nowy projekt.

Jak w każdym systemie z wieloma użytkownikami pozakładaj użytkownikom osobne konta, żeby śledzić zmiany i szybo sprawdzić kto jest za nie odpowiedzialny. Jest to bardzo przydatna zasada kiedy nagle coś przestaje działać.

Dla maszyny wykonującej testy i dla CI dobrze jest mieć zawsze włączony monitor bez wygaszacza, żeby móc sprawdzić aktualny stan i wychwycić ewentualne błędy np.: okno aktualizacji albo nie zamknięta przeglądarka wprowadzająca skrypty w błąd.

Ustaw archiwizowanie artefaktów. W przypadku defektu w aplikacji albo pomyłki kogoś z zespołu ściągniesz starszą wersję zbudowanej aplikacji za pomocą kilku kliknięć. Każdy tester może tez w prosty sposób ściągnąć zbudowaną aplikację na swoja maszynę i przetestować lokalnie.

Ustaw przechowywanie max 50 ostatnich build'ów. W przeciwnym razie przy częstych build'ach miejsce na dysku szybko się skończy.

Pamiętaj o tworzeniu kopii zapasowej środowiska.

Nie używaj żadnych kroków, które wymagają ręcznej ingerencji w proces.

Obserwuj trend pokrycia kodu testami wraz z rosnącą bazą kodu. Jeżeli spada albo trend ilości uruchamianych testów jest stały, oznacza to, że do repozytorium jest zgłaszany kod bez testów.

Skonfiguruj ustawienia maszyny wirtualnej Java, żeby dać Hudsonowi odpowiednią ilość pamięci.

Raczej używaj archiwizacji niż usuwania projektów. Często okazuje się, że te dane mogą być jeszcze potrzebne.

Ustal harmonogram okresowych prac pielęgnacyjnych takich jak czyszczenie logów, usuwanie starych build'ów i aktualizacja środowiska do nowej wersji.

Wszelkie zmiany w konfiguracji czy skrypcie do budowania aplikacji dobrze przetestuj lokalnie. Fałszywe powiadomienia o porażce szybko prowadzą do utraty zaufania zespołu i zaprzestania reagowania na kolejne powiadomienia.

Jeżeli w zestawie testów są testy, które kończą się porażką i z pewnych ważnych względów nie można ich teraz naprawić, przenieś te testy do osobnej grupy, która nie powoduje porażki całego build'a, ale nadal pokazuje porażki testów. Zwykłe wykomentowywanie testów jest złą praktyką i prowadzi do zapominania o nich.

Możesz użyć niestandardowej metody powiadamiania o statusie buildów jak na przykład lampy lava Lamps czy lampki z misiami.

## O AUTORZE

***W ciągu siedmiu lat pracy w IT, Krystian zdobył wszechstronne doświadczenie w całym cyklu wytwarzania oprogramowania. Pracował jako programista, wdrożeniowiec, support, kontakt klienta, Scrum Master, tester i Test Manager, dzięki czemu patrzy na oprogramowanie oraz proces jego wytwarzania z kilku perspektyw i znajduje wspólny język zarówno z biznesem, jak i IT.***

***Przełomowy w jego karierze okazał się pierwszy międzynarodowy projekt w Szwecji, gdzie odkrył talent do testowania. Wykonywanie już zaplanowanych czynności przestało mu wystarczać, postanowił dowiedzieć się więcej i zdobył certyfikat ISEB Certified Tester Foundation Level (CTFL), zaczął czytać fachową prasę i książki. Po trzech latach od rozpoczęcia tej przygody Krystian pracował jako Test Manager dla jednej z największych firm telekomunikacyjnych. Zdobyte doświadczenie i posiadana wiedza pozwoliły na otrzymanie kolejnego certyfikatu, ISTQB Certified Tester Advanced Level - Test Manager w 2010 roku.***

***Spotkanie z framework'iem Scrum i metodyką Agile w 2006 roku zaowocowało zainteresowaniem tematem Quality Assurance – Zapewnianiem Jakości i prowadzeniem zespołu w wymagającym środowisku „zwinnych" technik. Dwa lata później zdobył certyfikat Certified ScrumMaster, a w 2010 Certified Scrum Professional. W swojej praktyce Krystian brał udział we wprowadzaniu Scrum'a do organizacji oraz ulepszaniu istniejącej implementacji zwiększając tym samym zadowolenie udziałowców oraz podnosząc morale zespołu. Posiadane umiejętności komunikacyjne i zaufanie kierownictwa sprawiły, że powierzono mu także nadzorowanie współpracy z kontraktorami i dostawcami zewnętrznymi.***

***Obecnie Krystian pracuje dla największego na świecie wydawnictwa naukowego i jest odpowiedzialny za tworzenie i zarządzanie procesem testowania aplikacji webowych, promowanie jakości oprogramowania oraz - jako Scrum Master - prowadzenie zespołu w dostarczaniu nowych funkcjonalności zgodnie z framework'iem Scrum. Krystian Kaczor jest także właścicielem firmy QAgile świadczącej usługi konsultingowe i szkolenia.***

***Kontakt z autorem krystian@kaczor.info.***