
Helion
KOMPLETNY PRZEWODNIK
CIĄGŁE DOSTARCZANIE OPROGRAMOWANIA
EBERHARD WOLFF

Tytuł oryginału: A Practical Guide to Continuous Delivery

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-283-3784-8

Authorized translation from the English language edition, entitled: A PRACTICAL GUIDE TO CONTINUOUS DELIVERY; ISBN 0134691474; by Eberhard Wolff; published by Pearson Education, Inc, publishing as Addison-Wesley Professional.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/cidokp
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Rozdział 1

Ciągłe dostarczanie — co i jak

1.1. Wprowadzenie — czym jest ciągłe dostarczanie?

Niełatwo jest odpowiedzieć na to pytanie. Autorzy nazwy nie przedstawiają konkretnej definicji[1]. Martin Fowler w swoim omówieniu[2] ciągłego dostarczania koncentruje się na tym, że oprogramowanie można w dowolnym momencie udostępnić w środowisku produkcyjnym. Wymaga to automatyzacji procesów potrzebnych do instalacji oprogramowania i informacji zwrotnych dotyczących jakości oprogramowania. W Wikipedii[3] ciągłe dostarczanie zdefiniowane jest jako optymalizacja i automatyzacja procesu udostępniania oprogramowania.

W podsumowaniu: głównym celem ciągłego dostarczania jest analiza i optymalizacja procesu prowadzącego do udostępnienia oprogramowania. Proces ten zostaje płynnie zintegrowanym z rozwojem produktu.

1.2. Dlaczego udostępnianie oprogramowania jest tak skomplikowane?

Udostępnianie oprogramowania to wyzwanie. Prawdopodobnie każdemu działowi informatycznemu zdarzyło się pracować w weekend, aby udostępnić oprogramowanie w środowisku produkcyjnym. Takie sytuacje często kończą się tym, że oprogramowanie jest jakoś umieszczane w środowisku produkcyjnym, ponieważ od określonego momentu przywrócenie starszej wersji okazuje się jeszcze bardziej ryzykowne i trudniejsze niż podążanie naprzód. Jednak po zainstalowaniu wersji często następuje długa faza, kiedy oprogramowanie trzeba ustabilizować.

[1] Jez Humble, David Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley, 2010, ISBN 978-0-32160-191-9.

[2] Zob. *https://martinfowler.com/bliki/ContinuousDelivery.html*.

[3] Zob. *https://en.wikipedia.org/wiki/Continuous_delivery*.

1.2.1. Ciągła integracja daje nadzieję

Obecnie to właśnie udostępnianie oprogramowania w środowisku produkcyjnym stanowi wyzwanie. Nie tak dawno temu problemy zaczynały się znacznie wcześniej. Poszczególne zespoły pracowały nad modułami niezależnie od siebie, dlatego przed udostępnieniem kodu różne komponenty trzeba było zintegrować. Gdy moduły były łączone po raz pierwszy, system często nawet się nie kompilował. Nieraz zintegrowanie wszystkich zmian i doprowadzenie do udanej kompilacji zajmowało dni, a nawet tygodnie. Dopiero wtedy można było zacząć wdrażanie. Obecnie te problemy w większości zostały rozwiązane. Wszystkie zespoły pracują nad wspólną wersją kodu, która jest automatycznie stale integrowana, kompilowana i testowana. To podejście nazywane jest ciągłą integracją (ang. *continuous integration*). Infrastrukturę potrzebną do ciągłej integracji opisano w rozdziale 3. „Automatyzacja procesu budowania i ciągła integracja". Wyeliminowanie dawnych problemów związanych z tym etapem rodzi nadzieję na to, że pojawią się rozwiązania trudności z innych etapów przedprodukcyjnych.

1.2.2. Powolne i ryzykowne procesy

Procesy z dalszych etapów są często złożone i skomplikowane. Ponadto ręczne wykonywanie zadań sprawia, że procesy są bardzo żmudne i narażone na błędy. Jest tak w przypadku udostępniania kodu w środowisku produkcyjnym, ale też we wcześniejszych fazach, np. testów. Błędy często pojawiają się zwłaszcza w ręcznych procesach, które są wykonywane tylko kilka razy w roku, co dodatkowo pogarsza sytuację. To oczywiście zwiększa ryzyko niepowodzenia całej procedury udostępniania.

Z powodu wysokiego ryzyka błędów i złożoności wersje oprogramowania są rzadko udostępniane w środowisku produkcyjnym. Wynikający z tego brak praktyki dodatkowo wydłuża procesy. Trudno jest też je zoptymalizować.

1.2.3. Szybka praca jest możliwa

Zawsze możliwe jest szybkie udostępnienie wersji w środowisku produkcyjnym w sytuacji awaryjnej — na przykład gdy trzeba pilnie naprawić błąd. Jednak w takiej sytuacji pomijane są wszystkie testy, a tym samym wszystkie zabezpieczenia będące integralną częścią standardowego procesu. Jest to oczywiście ryzykowne. To, że standardowo takie testy są przeprowadzane, ma solidne uzasadnienie.

Dlatego normalna droga udostępniania kodu w środowisku produkcyjnym jest powolna i ryzykowna. W sytuacjach awaryjnych można ją przyspieszyć kosztem jeszcze większego wzrostu ryzyka.

1.3. Wartość ciągłego dostarczania

Celem jest optymalizacja udostępniania wersji w środowisku produkcyjnym z uwzględnieniem technik ciągłej integracji i motywacji związanej z tym podejściem.

Podstawowa zasada ciągłej integracji brzmi: „Jeśli coś boli, rób to częściej i uświadom sobie ten ból". Choć wygląda to na masochizm, w rzeczywistości jest sposobem rozwiązywania problemów. Zamiast unikać problemów z wersjami, udostępniając ich jak najmniej, procesy należy stosować tak często i wcześnie, jak jest to możliwe, by jak najszybciej je zoptymalizować

(z uwzględnieniem tempa pracy i niezawodności). Dlatego ciągłe dostarczanie zmusza firmę do zmian i przyjęcia nowego modelu pracy.

Ostatecznie podejście to nie jest zaskakujące. Każda firma informatyczna potrafi błyskawicznie udostępniać poprawki w środowisku produkcyjnym. W takim scenariuszu często przeprowadza się tylko część standardowych testów i sprawdzianów bezpieczeństwa. Jest to możliwe, ponieważ zmiany są niewielkie, dlatego oznaczają małe ryzyko. Widoczne staje się tu inne podejście minimalizowania ryzyka — zamiast próbować zabezpieczać się przed błędami za pomocą złożonych procesów i rzadkiego udostępniania, można częściej udostępniać w środowisku produkcyjnym niewielkie zmiany. To podejście jest identyczne ze strategią ciągłego dostarczania. W ciągłym dostarczaniu nawet drobne zmiany oprogramowania wprowadzone przez poszczególnych programistów i zespół są trwale integrowane. Różni się to od niezależnej pracy zespołów i programistów przez wiele dni i tygodni, po których dopiero na końcu integrowane są wszystkie zakumulowane modyfikacje. Ta ostatnia strategia często prowadzi do poważnych problemów. W niektórych sytuacjach są one tak duże, że oprogramowania w ogóle nie da się skompilować.

Jednak ciągłe dostarczanie to coś więcej niż „szybkie udostępnianie drobnych zmian". Podejście to jest oparte na innych wartościach, z których wynikają konkretne rozwiązania techniczne.

1.3.1. Regularność

Regularność oznacza, że procesy są wykonywane częściej. Wszystkie procesy potrzebne do udostępnienia oprogramowania w środowisku produkcyjnym należy wykonywać regularnie, a nie tylko w momencie udostępniania wersji. Na przykład konieczne jest budowanie środowisk testowych i przedprodukcyjnych (ang. *staging*). Środowiska testowe można wykorzystać do wykonywania testów akceptacyjnych i technicznych. Środowiska przedprodukcyjne mogą posłużyć do testowania i oceny funkcji z nowej wersji przez docelowego klienta. Proces generowania środowisk można przekształcić w standardowe zadanie wykonywane nie tylko w sytuacji, gdy trzeba utworzyć środowisko produkcyjne. Aby generować wiele środowisk bez dużego wysiłku, procesy trzeba zautomatyzować. Regularność prowadzi do automatyzacji. Podobne reguły dotyczą testów. Nie ma sensu odkładać niezbędnych testów do momentu przed udostępnieniem wersji. Zamiast tego należy je przeprowadzać regularnie. Także to podejście wymusza automatyzację w celu ograniczenia niezbędnego wysiłku. Regularność prowadzi też do wzrostu niezawodności. Często stosowane procesy można niezawodnie powtarzać i wykonywać.

1.3.2. Możliwość śledzenia i sprawdzalność zmian

Wszystkie zmiany w oprogramowaniu, które mają zostać wprowadzone w środowisku produkcyjnym i infrastrukturze potrzebnej dla danej wersji kodu, muszą być możliwe do prześledzenia. Potrzebna jest możliwość odtworzenia stanu oprogramowania i infrastruktury. To prowadzi do systemu kontroli wersji uwzględniającego nie tylko oprogramowanie, ale też potrzebne środowiska. W idealnych warunkach możliwe jest wygenerowanie każdego stanu oprogramowania wraz z potrzebnym do jego pracy środowiskiem i odpowiednią konfiguracją. Zapewnia to możliwość prześledzenia wszystkich zmian w oprogramowaniu i środowisku oraz umożliwia łatwe uzyskanie systemu do analizy błędów. Ponadto zmiany można dokumentować lub audytować.

Jednym z rozwiązań jest zapewnienie dostępu do środowisk produkcyjnego i przedprodukcyjnego tylko wybranym osobom. Ma to zapobiegać powstawaniu „szybkich poprawek", które

nie są udokumentowane i których śledzenie nie jest możliwe. Ponadto kwestie bezpieczeństwa i zabezpieczanie danych to argumenty na rzecz blokowania dostępu do środowiska produkcyjnego.

Gdy stosujesz ciągłe dostarczanie, interwencje w środowisko są możliwe tylko po zmianie skryptu instalacyjnego. Modyfikacje w skryptach są możliwe do śledzenia dzięki systemowi kontroli wersji. Twórcy skryptów nie mają dostępu do danych produkcyjnych, dlatego nie występuje problem z bezpieczeństwem danych.

1.3.3. Regresja

Aby zminimalizować ryzyko związane z udostępnianiem oprogramowania w środowisku produkcyjnym, oprogramowanie trzeba przetestować. Oczywiście poprzez użycie testów trzeba się upewnić, że nowe funkcje działają poprawnie. Wiele pracy wymaga unikanie regresji, czyli błędów powstałych w trakcie modyfikowania już przetestowanych fragmentów oprogramowania. Modyfikacja wymaga więc ponownego przeprowadzenia wszystkich testów, ponieważ zmiany w jednym miejscu systemu mogą spowodować błędy w innych. Dlatego konieczne są zautomatyzowane testy. W przeciwnym razie wykonywanie ich wymaga zbyt wiele wysiłku. Jeśli mimo testów w środowisku produkcyjnym wystąpi błąd, można go wykryć dzięki monitorowaniu. W idealnych warunkach możliwe jest proste zainstalowanie wolnej od błędów starszej wersji systemu produkcyjnego (wycofanie zmian, ang. *rollback*) lub szybkie wprowadzenie poprawki w środowisku produkcyjnym (zastąpienie nowszą wersją, ang. *roll forward*). Chodzi o to, by uzyskać system wczesnego ostrzegania, który dokonuje pomiarów na różnych etapach projektu (np. w czasie testów i w środowisku produkcyjnym), aby wykrywać regresję i umożliwiać jej wyeliminowanie.

1.4. Korzyści płynące z ciągłego dostarczania

Ciągłe dostarczanie zapewnia wiele korzyści. W zależności od scenariusza różne korzyści mogą mieć inną wagę i wpływać na sposób stosowania ciągłego dostarczania.

1.4.1. Ciągłe dostarczanie w celu przyspieszenia udostępniania

Ciągłe dostarczanie skraca czas potrzebny do udostępnienia zmian w środowisku produkcyjnym. Zapewnia to istotne korzyści biznesowe, ponieważ znacznie łatwiej jest reagować na zmiany na rynku.

Jednak zalety wykraczają poza szybsze udostępnianie. Nowe podejścia takie jak Lean Startup[4] zalecają strategię, która pozwala odnieść jeszcze większe korzyści ze wzrostu szybkości prac. W modelu Lean Startup celem jest udostępnianie produktów na rynku i ocenianie ich szans, przy czym inwestowany wysiłek ma być jak najmniejszy. Podobnie jak w eksperymentach naukowych z góry definiowane jest, jak oceniany będzie sukces produktu. Następnie eksperyment jest przeprowadzany, po czym następuje ocena sukcesu (lub porażki).

[4] Eric Ries, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*, Crown Business, 2010, ISBN 978-0-67092-160-7; wyd. polskie: *Metoda Lean Startup. Wykorzystaj innowacyjne narzędzia i stwórz firmę, która zdobędzie rynek*, Onepress, 2012.

1.4.2. Przykład

Przyjrzyj się teraz konkretnemu przykładowi. W sklepie internetowym utworzona ma zostać nowa funkcja — dostarczanie zamówień w określonym dniu. W ramach pierwszego eksperymentu można zacząć reklamować nową funkcję. Liczba kliknięć odsyłacza z reklamy może posłużyć do oceny sukcesu tego eksperymentu. Na tym etapie oprogramowanie nie zostało jeszcze zbudowane — funkcja nie jest jeszcze zaimplementowana. Jeśli eksperyment nie daje obiecujących wyników, funkcja najwyraźniej nie daje korzyści i można — nie marnując dużo energii — priorytetowo potraktować inne mechanizmy.

1.4.3. Implementowanie funkcji i udostępnianie jej w środowisku produkcyjnym

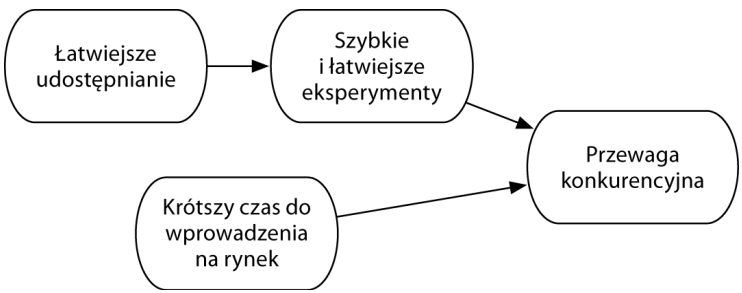
Jeśli eksperyment zakończył się powodzeniem, funkcja jest implementowana i udostępniana w środowisku produkcyjnym. Nawet ten krok można przeprowadzić jako eksperyment. Wskaźniki mogą pomóc w ocenie powodzenia funkcji. Możesz na przykład mierzyć liczbę zamówień o ustalonej dacie dostarczenia.

1.4.4. Przejście do następnej funkcji

Analizy wskaźników pokazują, że liczba zamówień jest wystarczająco duża. Co ciekawe, większość z nich nie jest kierowana do nabywcy, ale do osób trzecich. Z dodatkowych pomiarów wynika, że zamawiane towary to najwyraźniej prezenty urodzinowe. Na podstawie tych informacji można rozbudować funkcję, np. o kalendarz z urodzinami i polecane prezenty. Wymaga to oczywiście zaprojektowania dodatkowych funkcji, zaimplementowania ich, udostępnienia w środowisku produkcyjnym i oceny ich sukcesu. Inna możliwość to ocena potencjalnego sukcesu rynkowego takich funkcji bez ich implementowania. Można posłużyć się do tego reklamami, wywiadami z klientami, ankietami i innymi technikami.

1.4.5. Ciągłe dostarczanie zapewnia przewagę konkurencyjną

Ciągłe dostarczanie umożliwia szybsze wprowadzanie niezbędnych zmian w oprogramowaniu w środowisku produkcyjnym. Pozwala to firmie na szybsze testowanie różnych pomysłów i rozwijanie modelu biznesowego. To zapewnia przewagę konkurencyjną. Ponieważ można sprawdzić więcej pomysłów, łatwiej jest ustalić te właściwe — i to nie na podstawie subiektywnych szacunków prawdopodobieństwa sukcesu rynkowego, ale według obiektywnie zmierzonych danych (rysunek 1.1).



Rysunek 1.1. *Argumenty na rzecz stosowania ciągłego dostarczania w startupie*

1.4.6. Scenariusz bez ciągłego dostarczania

Bez ciągłego dostarczania funkcję dostaw w określonym dniu trzeba zaplanować i umieścić w środowisku produkcyjnym w czasie udostępniania następnej wersji — prawdopodobnie za kilka miesięcy. Do tego czasu dział marketingu zapewne nie odważy się reklamować funkcji, ponieważ długi czas do jej udostępnienia sprawia, że takie reklamy są bezcelowe. Gdyby funkcja nie okazała się sukcesem, jej zaimplementowanie wymagałoby poniesienia wysokich kosztów bez zapewniania korzyści. W tym tradycyjnym podejściu ocena powodzenia nowej funkcji też oczywiście jest możliwa, jednak informacje są uzyskiwane znacznie wolniej. Efekty dalszych prac, na przykład funkcje ułatwiające zakupy prezentów urodzinowych, trafiają na rynek dużo później, ponieważ wymagają ponownego umieszczenia oprogramowania w środowisku produkcyjnym i przejścia drugi raz przez czasochłonny proces udostępniania. Ponadto wątpliwe jest, czy w tym podejściu powodzenie funkcji da się przeanalizować wystarczająco szczegółowo, aby odkryć potencjał związany z dodatkowymi mechanizmami wspomagającymi zakupy prezentów urodzinowych.

1.4.7. Ciągłe dostarczanie i Lean Startup

Dzięki ciągłemu dostarczaniu cykle optymalizacji są znacznie krótsze, ponieważ każdą funkcję można umieścić w środowisku produkcyjnym w niemal dowolnym czasie, co pozwala na stosowanie podejść takich jak Lean Startup. Wpływa to na pracę działów biznesowych, które muszą szybciej wymyślać nowe funkcje i nie muszą koncentrować się na długoterminowych planach, ponieważ mogą błyskawicznie reagować na wyniki obecnie przeprowadzanych eksperymentów. Jest to łatwe przede wszystkim w startupach, jednak omawiane podejście można zastosować także w tradycyjnych firmach. Podejście Lean Startup ma, niestety, mylącą nazwę. Dotyczy ono wprowadzania produktów na rynek za pomocą serii eksperymentów, co jest oczywiście możliwe nie tylko w startupach, ale i w zwykłych firmach. To podejście można też stosować do produktów, które są sprzedawane w tradycyjny sposób (np. na nośnikach takich jak płyty CD), wymagają złożonych procedur instalacji lub są częścią innego produktu (np. maszyny). Wtedy instalację oprogramowania trzeba uprościć, a najlepiej zautomatyzować. Oprócz określenia grupy klientów trzeba ustalić, kto chciałby przetestować nowe wersje oprogramowania, i przekazać informacje na ich temat. Osoby takie to klasyczni testerzy wersji beta i zaawansowani użytkownicy.

1.4.8. Wpływ na proces rozwoju produktu

Ciągłe dostarczanie wpływa na proces rozwoju oprogramowania. Gdy firma chce udostępniać w środowisku produkcyjnym pojedyncze funkcje, proces musi to umożliwiać. W niektórych procesach iteracje trwają tydzień lub kilka tygodni. Po zakończeniu każdej iteracji w środowisku produkcyjnym udostępniana jest nowa wersja z kilkoma nowymi funkcjami. Nie jest to idealne rozwiązanie dla ciągłego dostarczania, ponieważ funkcje nie mogą pojedynczo przepływać przez potok. Taki model utrudnia też stosowanie podejścia Lean Startup. Gdy kilka funkcji jest udostępnianych jednocześnie, nie jest oczywiste, która zmiana wpływa na uzyskane wyniki. Załóżmy, że opcja dostawy w określonym dniu jest udostępniana równolegle ze zmianą kosztów dostawy. Nie da się wtedy stwierdzić, która z tych zmian wywarła większy wpływ na wzrost liczby sprzedanych produktów.

Dlatego procesy takie jak Scrum, programowanie ekstremalne (ang. *extreme programming* — XP) i oczywiście model kaskadowy są przeszkodą, ponieważ zawsze oznaczają udostępnianie w środowisku produkcyjnym kilku funkcji. Natomiast Kanban[5] polega na etapowym wprowadzaniu jednej funkcji do środowiska produkcyjnego i jest w pełni zgodny z ciągłym dostarczaniem. Inne procesy można oczywiście dostosować, aby umożliwić dostarczanie pojedynczych funkcji. Oznacza to jednak modyfikację procesów, przez co nie są one stosowane we wzorcowy sposób. Inna możliwość to dezaktywacja na początku dodatkowych funkcji. Pozwala to udostępnić razem zbiór funkcji z jednej wersji w środowisku produkcyjnym, a przy tym mierzyć skutki wprowadzenia każdej z nich z osobna.

Ciągłe dostarczanie oznacza też, że w zespołach pracują osoby pełniące różne role. Oprócz ról związanych z rozwojem produktu i eksploatacją mogą to być także role biznesowe dotyczące na przykład marketingu. Dzięki zmniejszeniu w ten sposób przeszkód w organizacji można jeszcze szybciej projektować eksperymenty na podstawie informacji zwrotnych dotyczących aspektów biznesowych.

Wypróbuj i poeksperymentuj

- Poszukaj informacji na temat podejść Lean Startup i Kanban. Skąd pochodzi Kanban?

Wybierz znany Ci projekt lub znajomą funkcję:

- Jak wyglądałaby minimalna wersja produktu? Powinna ona dać wyobrażenie o szansach rynkowych planowanego kompletnego produktu.
- Czy można ocenić produkt bez gotowego oprogramowania? Czy możliwe jest na przykład zareklamowanie go? Czy możliwe są wywiady z potencjalnymi użytkownikami?
- Jak można zmierzyć sukces funkcji? Czy ma ona wpływ na sprzedaż, liczbę kliknięć lub inne mierzalne aspekty?
- Ile czasu działy marketingu i sprzedaży mają na zaplanowanie produktu lub funkcji? Na ile jest to zgodne z podejściem Lean Startup?

1.4.9. Minimalizowanie ryzyka za pomocą ciągłego dostarczania

Stosowanie ciągłego dostarczania w sposób opisany w poprzednim punkcie jest związane z określonym modelem biznesowym. Jednak tradycyjne firmy często wymagają długoterminowych planów. Wtedy nie można stosować podejścia takiego jak Lean Startup. Ponadto w wielu firmach czas do wprowadzenia produktu na rynek jest stosunkowo mało istotny. Nie we wszystkich branżach ten aspekt ma kluczowe znaczenie. Może się to oczywiście zmienić, gdy firmy zaczną mierzyć się z konkurencją, która potrafi wejść na rynek z wykorzystaniem podejścia Lean Startup.

W wielu scenariuszach czas do wejścia na rynek nie jest motywacją do zastosowania ciągłego dostarczania. Jednak techniki z tego podejścia i tak mogą okazać się przydatne, ponieważ zapewnia ono wiele dodatkowych korzyści.

[5] David J. Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business*, Blue Hole Press, 2010, ISBN 978-0-98452-140-1.

- Ręczne procesy udostępniania wymagają wiele wysiłku. Nierzadko zdarza się, że cały dział informatyczny musi przez cały weekend pracować nad udostępnieniem kodu. Ponadto po zakończeniu tego procesu często trzeba wykonać wiele prac dodatkowych.
- Ponadto ryzyko jest wysokie. Udostępnianie oprogramowania wymaga wielu ręcznych modyfikacji, co może łatwo prowadzić do pomyłek. Jeśli błędy nie zostaną szybko wykryte i poprawione, mogą mieć daleko idące konsekwencje dla firmy.

Koszty tego ponoszą pracownicy działów informatycznych — programiści i administratorzy systemów, którzy pracują w weekendy i nocami, by udostępnić kod w środowisku produkcyjnym i poprawić błędy. Oprócz pracy w nadgodzinach są też poddani dużemu stresowi z powodu wysokiego ryzyka, którego nie należy lekceważyć. Na przykład firma Knight Capital straciła 440 milionów dolarów z powodu nieudanego udostępniania oprogramowania[6]. Wskutek tego firma stała się niewypłacalna. Takie scenariusze prowadzą do wielu pytań[7] — zwłaszcza o to, dlaczego problem wystąpił, dlaczego szybko go nie wykryto i jak można zapobiec podobnym zdarzeniom w innych środowiskach.

Rozwiązaniem takich problemów może być ciągłe dostarczanie. Podstawowymi aspektami tego podejścia są wyższa niezawodność i jakość procesu udostępniania. Dzięki temu programiści i administratorzy systemu mogą dosłownie spać spokojnie. Korzyści te wynikają z różnych czynników.

- Dzięki wyższemu poziomowi automatyzacji procesów udostępniania ich efekty łatwo jest powtórzyć. Dlatego gdy oprogramowanie zostanie wdrożone i przetestowane w środowisku testowym lub przedprodukcyjnym, dokładnie ten sam efekt można uzyskać w środowisku produkcyjnym, ponieważ jest ono identyczne z pozostałymi. Pozwala to wyeliminować wiele źródeł błędów, co prowadzi do zmniejszenia ryzyka.
- Ponadto testowanie oprogramowania staje się znacznie łatwiejsze, ponieważ testy są w dużym stopniu zautomatyzowane. To dodatkowo poprawia jakość, ponieważ testy można wykonywać częściej.
- Gdy częstotliwość wdrażania rośnie, ryzyko spada, ponieważ przy każdym wdrożeniu liczba zmian wprowadzanych w środowisku produkcyjnym jest mniejsza. Mniej zmian oznacza mniejsze ryzyko pojawienia się błędów.

Sytuacja jest w pewien sposób paradoksalna — tradycyjne działy informatyczne starają się udostępniać wersje w środowisku produkcyjnym jak najrzadziej, ponieważ proces ten jest wysoce ryzykowny. W trakcie każdego udostępniania mogą pojawić się usterki o potencjalnie katastrofalnych skutkach. Mniej wersji powinno więc skutkować ograniczeniem problemów.

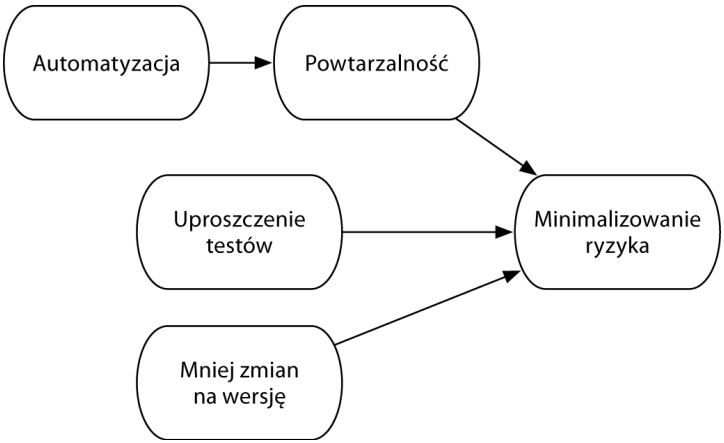
Jednak ciągłe dostarczanie zachęca do częstego udostępniania. W tym przypadku w każdej wersji udostępniana jest mniejsza liczba zmian, co zmniejsza prawdopodobieństwo wystąpienia błędów. Wymogiem wstępnym w tej strategii są zautomatyzowane i niezawodne procesy. W przeciwnym razie często udostępnianie prowadzi do przeciążenia informatyków wykonujących ręczne procesy, a także zwiększa ryzyko, ponieważ takie procesy są w wyższym stopniu narażone na błędy. Zamiast dążyć do rzadkiego udostępniania, należy zautomatyzować procesy, by ograniczyć ryzyko. Dodatkową korzyścią jest tu oczywiście to, że przy częstym udostępnianiu poszczególne wersje obejmują mniej zmian, dzięki czemu ryzyko wystąpienia błędów też jest mniejsze.

---

[6] Zob. *https://www.sec.gov/litigation/admin/2013/34-70694.pdf*.

[7] Zob. *http://www.kitchensoap.com/2013/10/29/counterfactuals-knight-capital/*.

Tak więc motywacja do stosowania ciągłego dostarczania (rysunek 1.2) jest wyraźnie odmienna od tej związanej z podejściem Lean Startup. Ważna jest niezawodność i wyższa jakość techniczna procesu udostępniania, a nie czas do wejścia na rynek. Ponadto korzyści odczuwają działy informatyczne, a nie tylko jednostki biznesowe.



Rysunek 1.2. *Przyczyny stosowania ciągłego dostarczania w firmie*

Ponieważ korzyści są odmienne, można pogodzić się z innymi kompromisami. Na przykład warto zainwestować w potok ciągłego dostarczania nawet wtedy, jeśli nie obejmuje on całej drogi do środowiska produkcyjnego (czyli gdy środowisko produkcyjne i tak trzeba budować ręcznie). W końcu środowisko produkcyjne wystarczy zbudować raz dla każdej wersji, natomiast dla różnych testów potrzebnych jest wiele środowisk. Jeśli jednak główną motywacją do wprowadzenia ciągłego dostarczania jest czas do wejścia na rynek, niezbędne jest, by potok obejmował też środowisko produkcyjne.

Wypróbuj i poeksperymentuj

Zastanów się nad projektem, nad którym pracujesz:

- Gdzie w trakcie instalacji zwykle występują problemy?
- Czy można je rozwiązać za pomocą automatyzacji?
- Gdzie obecne podejścia można uprościć, aby umożliwić automatyzację i optymalizację? Oceń potrzebne wysiłki i oczekiwane korzyści.
- Jak obecnie budowane są systemy produkcyjne i testowe? Czy odpowiada za to ten sam zespół? Czy możliwe jest zastosowanie automatyzacji do obu obszarów, czy tylko do jednego z nich?
- Dla których systemów automatyzacja byłaby przydatna? Jak często budowane są te systemy?

1.4.10. Szybsze informacje zwrotne i podejście Lean

Gdy programista modyfikuje kod, otrzymuje informacje zwrotne dzięki własnym testom, testom integracyjnym, testom wydajności i ostatecznie ze środowiska produkcyjnego. Jeśli zmiany są wprowadzane w środowisku produkcyjnym tylko raz na kwartał, to od zmodyfikowania kodu do uzyskania informacji ze środowiska produkcyjnego może minąć kilka miesięcy. To samo dotyczy testów akceptacyjnych i wydajności. Jeśli błąd zostanie wykryty po takim czasie, programista będzie musiał przypomnieć sobie, co implementował miesiące temu i w czym może tkwić problem.

Ciągłe dostarczanie sprawia, że cykl otrzymywania informacji zwrotnych jest znacznie krótszy. Za każdym razem, gdy kod przepływa przez potok, programista i cały jego zespół otrzymują informacje zwrotne. Po każdej zmianie można uruchamiać zautomatyzowane testy akceptacyjne i przepustowości. To pozwala programiście i zespołowi rozwoju produktu szybciej wykrywać i poprawiać błędy. Czas do uzyskania informacji zwrotnych można dodatkowo skrócić, stosując szybkie testy (na przykład testy jednostkowe) i najpierw przeprowadzając testy ogólne, a dopiero później szczegółowe. To od początku gwarantuje, że wszystkie funkcje działają przynajmniej w łatwych scenariuszach (tak zwanych „ścieżka optymistycznych"). Pozwala to łatwiej i szybciej wykrywać proste usterki. Ponadto testy, o których z doświadczenia wiadomo, że częściej kończą się niepowodzeniem, należy przeprowadzać na początku.

Ciągłe dostarczanie jest zgodne z podejściem Lean. W tym podejściu wszystko, za co klient nie płaci, jest uznawane za marnotrawstwo. Każda zmiana w kodzie jest marnotrawstwem, chyba że zostanie wprowadzona w środowisku produkcyjnym, ponieważ tylko wtedy klient będzie gotowy zapłacić za modyfikacje. Ponadto w ciągłym dostarczaniu stosowane są krótsze cykle w celu szybszego uzyskania informacji zwrotnych. Jest to następna koncepcja z podejścia Lean.

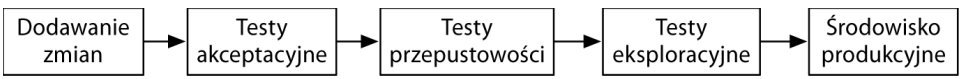
Wypróbuj i poeksperymentuj

Przyjrzyj się projektowi, nad którym obecnie pracujesz.

- Ile czasu mija od zmodyfikowania kodu do:
  - uzyskania informacji zwrotnych z serwera ciągłej integracji?
  - uzyskania informacji zwrotnych z testów akceptacyjnych?
  - uzyskania informacji zwrotnych z testów wydajności lub przepustowości?
  - udostępnienia zmian w środowisku produkcyjnym?

1.5. Procesy generowania i struktura potoku ciągłego dostarczania

Ciągłe dostarczanie to rozwinięcie ciągłej integracji o dodatkowe etapy. Ich przegląd przedstawia rysunek 1.3.



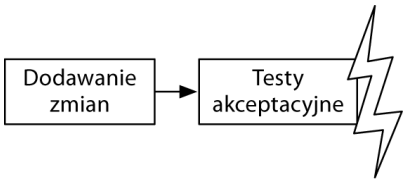
Rysunek 1.3. *Etapy potoku ciągłego dostarczania*

W tym podrozdziale przedstawiona jest struktura środowiska ciągłego dostarczania. Jest ono zgodne z opisem Humble'a i współpracowników (zob. przypis 1) i obejmuje następujące etapy:

- Faza dodawania zmian (ang. *commit*) obejmuje czynności obsługiwane zwykle przez infrastrukturę ciągłej integracji, takie jak proces budowania, testy jednostkowe i statyczna analiza kodu. Szczegółowe omówienie tej części potoku zawiera rozdział 3.
- Następnym krokiem są testy akceptacyjne, będące tematem rozdziału 4. „Testy akceptacyjne". Ściśle rzecz biorąc, omawiane są w nim testy automatyczne — albo interakcje z graficznym interfejsem użytkownika są automatyzowane w celu przetestowania systemu, albo wymogi są opisywane w języku naturalnym w sposób umożliwiający wykorzystanie ich jako testów automatycznych. Od tego etapu (jeśli nie wcześniej) konieczne jest generowanie środowisk, w których aplikacje mogą działać. Dlatego w rozdziale 2. „Zapewnianie infrastruktury" opisane jest generowanie takich środowisk.
- Testy przepustowości (rozdział 5. „Testy wydajności") gwarantują, że oprogramowanie poradzi sobie z oczekiwanym obciążeniem. W tym celu należy posłużyć się testami automatycznymi, które jednoznacznie określają, czy oprogramowanie jest wystarczająco szybkie. Ważna jest tu nie tylko szybkość, ale też skalowalność. Dlatego test można przeprowadzić także w środowisku, które nie jest identyczne ze środowiskiem produkcyjnym. Jednak musi ono zapewniać wyniki zgodne z działaniem oprogramowania w środowisku produkcyjnym. W zależności od konkretnej sytuacji i wymogów niezwiązanych z pracą oprogramowania (a na przykład związanych z bezpieczeństwem) czasem możliwe jest zautomatyzowanie takich testów.
- W trakcie testów eksploracyjnych (rozdział 6. „Testy eksploracyjne") aplikacja jest sprawdzana, ale nie według ścisłego planu. Zamiast tego eksperci testują aplikację, koncentrując się na nowych funkcjach i nieoczekiwanym ich działaniu. Dlatego nawet ciągłe dostarczanie nie wymaga automatyzacji wszystkich testów. Dzięki dużej liczbie automatycznych testów można znaleźć czas na testy eksploracyjne, ponieważ pracownicy nie muszą już ręcznie przeprowadzać rutynowych testów.
- Wdrażanie kodu w środowisku produkcyjnym (rozdział 7. „Wdrażanie — udostępnianie w środowisku produkcyjnym") polega jedynie na zainstalowaniu aplikacji w innym środowisku, dlatego jest stosunkowo mało ryzykowne. Istnieją różne podejścia pozwalające dodatkowo zminimalizować zagrożenia związane z umieszczaniem kodu w środowisku produkcyjnym.
- W trakcie eksploatacji aplikacji też pojawiają się wyzwania — zwłaszcza w obszarze monitorowania i obserwacji plików dziennika. Te wyzwania są opisane w rozdziale 8. „Eksploatacja".

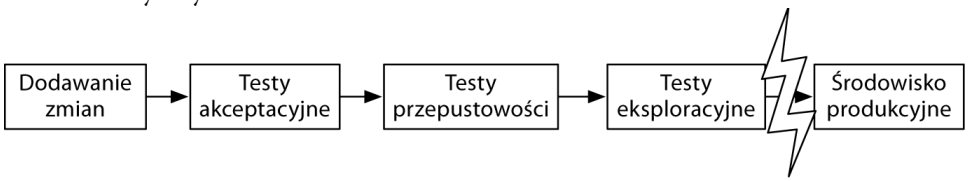
Wersje są przenoszone między poszczególnymi etapami. Możliwe jest, że wersja z powodzeniem przejdzie etap testów akceptacyjnych, ale okaże się za mało wydajna w fazie testów przepustowości. Wtedy nie należy przenosić wersji do następnych etapów, takich jak testy eksploracyjne lub środowisko produkcyjne. W tym modelu trzeba udowodnić, że oprogramowanie spełnia coraz wyższe wymogi. Dopiero po tym można udostępnić je w środowisku produkcyjnym.

Załóżmy, że w logice oprogramowania znajduje się błąd. Najpóźniej wykryty zostanie on w trakcie testów akceptacyjnych, ponieważ sprawdzają one, czy aplikacja jest poprawnie zaimplementowana. W konsekwencji potok zostaje zablokowany (rysunek 1.4). Na tym etapie dodatkowe testy nie są już potrzebne.



Rysunek 1.4. *Potok ciągłego dostarczania zostaje zablokowany na etapie testów akceptacyjnych*

Programiści później naprawiają błąd i budują oprogramowanie od nowa. Tym razem przechodzi ono także testy akceptacyjne. Jednak w nowej funkcji nadal występuje błąd, który nie zostaje wykryty w automatycznych testach akceptacyjnych. Można go znaleźć tylko w trakcie testów eksploracyjnych. Dlatego tym razem potok zostaje zablokowany na poziomie testów eksploracyjnych i oprogramowanie znów nie jest udostępniane w środowisku produkcyjnym (rysunek 1.5). Dzięki temu testerzy nie marnują czasu na oprogramowanie, jeśli nie spełnia ono wymogów z zakresu obsługi obciążenia lub zawiera błędy, które można wykryć za pomocą testów automatycznych.



Rysunek 1.5. *Potok ciągłego dostarczania zablokowany na etapie testów eksploracyjnych*

Teoretycznie możliwe jest równoległe przetwarzanie w potoku kilku wersji kodu. Oczywiście zachodzi warunek, aby potok na to pozwalał. Jeśli testy są wykonywane w niezmiennym środowisku, równoległe przetwarzanie wersji jest niewykonalne, ponieważ gdy środowisko jest zajęte jednym testem, nie może jednocześnie sprawdzać drugiej wersji.

Jednak w ciągłym dostarczaniu równoległe przetwarzanie wersji zdarza się bardzo rzadko. Należy utworzyć dokładnie jeden stan wersji, która jest następnie przenoszona w potoku. Co najwyżej modyfikacje oprogramowania można wykonywać z taką szybkością, by nowa wersja była umieszczana w potoku przed opuszczeniem go przez poprzednią wersję. Możliwe są wyjątki związane z pilnymi poprawkami, jednak jednym z celów ciągłego dostarczania jest traktowanie wszystkich wersji tak samo.

1.5.1. Przykład

W książce używana jest przykładowa aplikacja z procesem rejestracji klienta zainspirowanym przykładem opisującym Wielki Sklep Internetowy (zob. podrozdział P.2). Logika aplikacji jest celowo bardzo prosta. Rejestrowane są: imię, nazwisko i adres e-mail klienta. Aplikacja sprawdza poprawność rejestrowanych danych. Adres e-mail musi mieć poprawną składnię, a jeden adres może posłużyć do zarejestrowania tylko jednej osoby. Ponadto zarejestrowane osoby można wyszukiwać na podstawie adresów e-mail i usuwać.

Ponieważ ta aplikacja nie jest złożona, można ją stosunkowo łatwo zrozumieć. Dzięki temu czytelnicy mogą skoncentrować się na różnych aspektach ciągłego dostarczania przedstawianych za pomocą przykładowej aplikacji.

Aplikacja jest zaimplementowana z użyciem Javy i platformy Spring Boot. Dzięki temu można uruchamiać aplikację (w tym interfejs sieciowy) bez instalowania serwera WWW lub serwera aplikacji. Ułatwia to testy, ponieważ nie trzeba instalować infrastruktury. Jednak w razie potrzeby aplikację można też uruchamiać na serwerze aplikacji lub serwerze WWW takim jak Apache Tomcat. Dane są przechowywane w bazie HSQLDB. Jest to przechowywana w pamięci baza działająca w procesie Javy. Także to rozwiązanie zmniejsza techniczną złożoność aplikacji.

Przykładowy kod źródłowy możesz pobrać ze strony *http://github.com/ewolff/user-registration-V2*. Ważna uwaga: przykładowy kod obejmuje usługi, które działają z uprawnieniami administratora i są dostępne z poziomu internetu. W środowisku produkcyjnym jest to nieakceptowalne z powodu problemów z bezpieczeństwem. Jednak ten przykładowy kod jest przeznaczony tylko do eksperymentów. W tym kontekście wygodna struktura przykładów jest bardzo przydatna.

1.6. Wnioski

Udostępnianie oprogramowania w środowisku produkcyjnym to powolny i ryzykowny proces. Zoptymalizowanie go pozwala zwiększyć skuteczność i wydajność rozwoju oprogramowania. Dlatego ciągłe dostarczanie może być jednym z najlepszych sposobów na usprawnienie projektów budowania oprogramowania.

Ciągłe dostarczanie ma skutkować powstawaniem stabilnych i powtarzalnych procesów dostarczania oprogramowania (w zakresie integrowania wszystkich zmian podobne efekty zapewnia ciągła integracja). Ciągłe dostarczanie jest świetnym sposobem na skrócenie czasu do wprowadzenia produktu na rynek, a dodatkowo ma do zaoferowania o wiele więcej. Podejście to minimalizuje ryzyko w projektach rozwoju oprogramowania, ponieważ gwarantuje, że aplikację można wdrożyć i uruchomić w środowisku produkcyjnym. Tak więc korzyści można odnieść w każdym projekcie — nawet w mało konkurencyjnej branży, gdzie czas do wejścia na rynek nie ma dużego znaczenia.

Skorowidz

CIĄGŁE DOSTARCZANIE OPROGRAMOWANIA
KLUCZEM DO OPTYMALIZACJI PRACY Z KODEM

Pojęcie ciągłego dostarczania wywodzi się z metodyk zwinnego wytwarzania oprogramowania, a polega na sprawnej analizie i optymalizacji procesu prowadzącego do udostępnienia oprogramowania oraz na zintegrowaniu go z rozwojem produktu. Dzięki usprawnieniu tych prac proces udostępniania oprogramowania automatyzuje się, staje się bardziej powtarzalny i o wiele mniej ryzykowny, a to spotyka się z aprobatą klientów.

Niniejsza książka jest wartościowym przewodnikiem dla zespołów projektowych. Docenią ją zwłaszcza programiści i menedżerowie pracujący według zasad DevOps. Przedstawiono tu podstawowe procesy, wymagania, korzyści i konsekwencje techniczne. Pokazano, w jaki sposób należy implementować potoki i nimi zarządzać. Dzięki lekturze tej książki płynne integrowanie ciągłego dostarczania z architekturą oprogramowania i pracą działów informatycznych stanie się o wiele łatwiejsze. Opisano tu również przykładowe projekty, które stanowią punkt wyjścia do samodzielnych eksperymentów, a nawet do tworzenia własnych potoków ciągłego dostarczania.

EBERHARD WOLFF od przeszło 15 lat zajmuje się architekturą oprogramowania i doradztwem w obszarze styku biznesu i technologii. Wygłaszał prelekcje i przemówienia na międzynarodowych konferencjach, był członkiem komisji programowych wielu sympozjów. Napisał ponad 100 artykułów i książek. Koncentruje się na nowoczesnych architekturach, często obejmujących chmurę, ciągłe dostarczanie, DevOps, mikrousługi i bazy typu NoSQL.

Najważniejsze zagadnienia przedstawione w książce:

- ciągłe dostarczanie: co to jest i jakie problemy rozwiązuje
- automatyzacja tworzenia oprogramowania
- testy: akceptacyjne, wydajności i eksploracyjne
- wdrażanie metodyki ciągłego dostarczania w organizacji
- wpływ ciągłego dostarczania na architekturę aplikacji

ISBN 978-83-283-3784-8
9 788328 337848
Cena: 49,00 zł

Addison-Wesley
ALWAYS LEARNING
PEARSON