Helion

packt

# Pełnia możliwości DevOps, Git i GitHub

## Zastosowanie podejścia opartego na automatyzacji, współpracy i innowacji

YUKI HATTORI

Tytuł oryginału: DevOps Unleashed with Git and GitHub: Automate, collaborate, and innovate to enhance your DevOps workflow and development experience

Tłumaczenie: Robert Górczyński

ISBN: 978-83-289-1885-6



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

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

# Spis treści

# CZĘŚĆ 2. Zaawansowane funkcje GitHuba oraz podstawy potoku ciągłej integracji i ciągłego wdrażania

# DevOps i wrażenia programisty

Rozdział 1

W tym rozdziale omówię dwa ważne tematy: wprowadzenie do podejścia DevOps, które jest motywem przewodnim książki, oraz zagadnienie wrażeń programisty, czyli strategię istotną dla pomyślnego wprowadzenia i stosowania przez organizację podejścia DevOps. W tym kontekście zamieszczę również wprowadzenie do technologii Git i GitHub.

Ten rozdział ma pełnić funkcję przewodnika dla czytelników, którzy chcą poznać podejście DevOps, związane z nim narzędzia oraz praktyki stosowane podczas współpracy w zespole korzystającym z DevOps. Dzięki temu zdobędziesz podstawową wiedzę z zakresu tych koncepcji i praktyk oraz dowiesz się, jak mogą one usprawnić proces tworzenia oprogramowania.

Oto zagadnienia, które omawiam w tym rozdziale:

- DevOps — przyśpieszenie cyklu tworzenia oprogramowania poprzez zmniejszenie tarć;
- wrażenia programisty — strategia pozwalająca programiście osiągnąć doskonałość;
- Git — system, od którego rozpoczyna się współpraca nad kodem źródłowym;
- GitHub — platforma programistyczna wspierana przez sztuczną inteligencję.

## DevOps — przyśpieszenie cyklu tworzenia oprogramowania poprzez zmniejszenie tarć

Rozpocznę od omówienia podstaw podejścia DevOps.

W świecie technologii istnieje wiele pojęć. Czasami mogą być one abstrakcyjne i użytkownicy mają tendencję do odmiennego interpretowania określonych frameworków. DevOps nie jest tutaj wyjątkiem. Ponadto w odniesieniu do DevOps rozważanie tego pojęcia z perspektywy pojedynczego użytkownika, zespołu bądź całej organizacji może w niektórych sytuacjach wprowadzać niejasności co do tego, jak poszczególne komponenty ze sobą współdziałają.

Wyjaśnię podstawy podejścia DevOps, postaram się rozwiać wszelkie niejasności, a także przedstawię wybrane z typowych praktyk. Jednak zanim przejdę do wyjaśnienia, czym jest DevOps, chciałbym przybliżyć, jak tworzenie oprogramowania odbywało się w przeszłości.

## Kontekst dla podejścia DevOps

Dawniej oprogramowanie w zasadzie było czymś, co musiało być zainstalowane w komputerze. Zatem w przypadku zespołu zajmującego się tworzeniem oprogramowania potrzebni byli inżynierowie odpowiedzialni za jego opracowanie i przetestowanie. Skoro wspomniałem już o testowaniu, to warto wyjaśnić, że oznaczało ono proces znajdowania błędów w produkcie jeszcze przed jego wydaniem. Zastanów się nad tym przez chwilę, ponieważ mówimy o czasach, w których pojawiały się wczesne systemy gier wideo, takie jak Nintendo Entertainment System (NES). Gdy gra została umieszczona na nośniku, np. na kartridżu (ang. *cartridge*), w celu jej rozpowszechniania, oznaczało to brak możliwości jej uaktualniania bądź dodawania nowych funkcjonalności. Błędy w danym wydaniu gry pozostawały z użytkownikiem na zawsze. Pojawienie się serwera WWW (Apache HTTP Server) i przejście na komunikację poprzez sieć oznaczało powstanie nowej roli, operacje IT, w której to pracownicy zajmowali się zarządzaniem tym nowym paradygmatem.

W wielu przypadkach zespół operacji IT specjalizował się w przeprowadzaniu operacji informatycznych. Ich narzędzia, kultura pracy, a nawet cele były inne niż w przypadku zespołu programistycznego. Zadaniem zespołu programistycznego było implementowanie nowych funkcjonalności, podczas gdy priorytetem zespołu informatycznego było zapewnienie maksymalnej stabilności oprogramowania. Te cele okazywały się sprzeczne.

Co więcej, gdy w świecie podzielonym między te dwa zespoły przychodziło do wdrażania produktu, ta operacja mogła przypominać „przerzucanie" kodu źródłowego przez mur (zob. rysunek 1.1). Wówczas programiści przekazywali do działu operacji IT działający i gotowy do wdrożenia kod źródłowy. Jednak jeśli z jakiegokolwiek powodu zespół operacji IT otrzymywał coś niekompletnego i niefunkcjonującego w środowisku produkcyjnym, musiał się zmagać z wdrożeniem produktu i jego późniejszą obsługą techniczną.

**Rysunek 1.1. Inżynierowie „przerzucają" kod źródłowy przez mur**

W ostatnich latach dokonał się ogromny postęp technologiczny, którego wpływ znacznie wykroczył poza branżę tworzenia oprogramowania i objął praktycznie każdą gałąź przemysłu. Dlaczego? Dlatego, że w nowoczesnych technikach zarządzania biznesem

tworzenie oprogramowania odgrywa ważną rolę w strategiach organizacji, niezależnie od ich wielkości. Firmy objęte tą technologiczną falą trafiają do bardziej konkurencyjnego środowiska, w którym nacisk kładzie się na dostarczanie klientom produktów efektywnych, niezawodnych i oferujących najnowsze funkcjonalności.

Wprawdzie co kilka miesięcy są zauważalne znaczne zmiany warunków rynkowych i potrzeb klientów, ale wydawanie oprogramowania z taką częstotliwością okazuje się niewystarczające. Nie można pozwolić sobie na opóźnienie w wydawaniu oprogramowania spowodowane przez tarcia między wewnętrznymi silosami. W tym kontekście słowo „silos" oznacza odizolowany system, proces bądź dział, który nie jest zintegrowany lub skomunikowany z innymi, co może prowadzić do nieefektywności. Konieczne jest znalezienie sposobu na usprawnienie procesu od opracowania produktu po jego działanie. W tym miejscu do gry wchodzi zupełnie nowa metoda tworzenia oprogramowania, struktury organizacyjnej i kultury pracy.

DevOps to metoda, która narodziła się do obsługi wymienionego wcześniej scenariusza. Reprezentuje połączenie zadań związanych z opracowywaniem produktu i jego późniejszą obsługą. W tym podejściu zespoły odpowiedzialne za opracowywanie i obsługę produktu współpracują ze sobą i tworzą jednolity byt, aby w ten sposób zaszczepić wspólną kulturę pracy, dopracować procesy i wdrożyć narzędzia. To wszystko ma na celu zwiększenie tempa pojawiania się kolejnych wydań produktu, ogólną poprawę jego jakości, zebranie od użytkowników informacji o produkcie, a także zapewnienie klientom lepszej i sprawniejszej obsługi. To z kolei prowadzi do znacznie efektywniejszego funkcjonowania produktu na rynku.

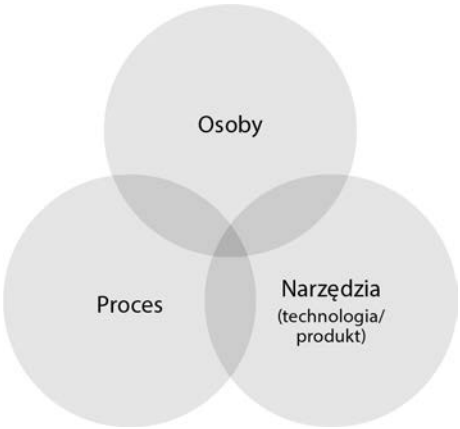
## Czym jest DevOps?

Określenie **DevOps** jest połączeniem angielskich słów **Development** (pol. *opracowywanie*) i **Operations** (pol. *operacje*).

Zatem w podejściu DevOps nacisk kładzie się na współpracę między zespołami zajmującymi się opracowaniem produktu i jego obsługą, prawda?

Poniekąd tak, przy czym rzeczywistość jest często znacznie bardziej skomplikowana. Wymienione zespoły zwykle są osobne, a każdy z nich ma własne priorytety i cele. Samo ich połączenie może przynajmniej do pewnego stopnia poprawić współpracę i wzajemne zrozumienie, przy czym często okazuje się to niewystarczające do efektywnego współdziałania. Podejście DevOps wykracza poza integrowanie ról bądź zespołów — oznacza także szersze przesunięcie w stosowanej przez organizację kulturze pracy. Ma to związek z dostosowywaniem priorytetów, usprawnianiem sposobu pracy oraz ostatecznie z przełamaniem barier uniemożliwiających efektywną komunikację i postęp.

Ujmując rzecz najprościej: DevOps to unia obejmująca osoby, proces i narzędzia (zob. rysunek 1.2), która pozwala na nieustanne dostarczanie wartości użytkownikom końcowym oraz stanowi poważną zmianę w kulturze pracy.




**Rysunek 1.2. DevOps to unia obejmująca osoby, procesy i narzędzia**

*„Rozumiem. DevOps to podejście w zasadzie dotyczące osób, procesów i narzędzi".*

Wielu czytelników będzie przekonanych, że już rozumie tę koncepcję, więc może przejść do implementacji zautomatyzowanego potoku dla DevOps. Jeszcze nie. Zwykłe zastąpienie wymienionych elementów nie gwarantuje osiągnięcia sukcesu w podejściu DevOps. Aby osiągnąć sukces, trzeba wiedzieć nieco więcej na temat tego, czym jest, a czym nie jest podejście DevOps oraz dlaczego w ogóle istnieje.

Definicja DevOps jest w istocie dość szeroka. Dlatego w szybkim tempie coraz większą popularność zyskuje koncepcja *Dev*Ops*, czyli rozbudowana wersja DevOps. Zaproponowano również wiele podejść pochodnych, np. DevSecOps i BizDevOps, co prowadzi do wielu perspektyw, z których można spoglądać na DevOps.

Niestety nie istnieje jedna, powszechnie akceptowana i precyzyjna definicja DevOps. Zatem dla różnych osób znaczenie określenia DevOps będzie odmienne. Nie należy się więc dziwić, że pojawiają się błędne koncepcje i niepoprawne interpretacje DevOps, co z kolei może prowadzić do nieporozumień. Gdy nadchodzi odpowiedni moment na rozmowę o DevOps, członkowie zespołu często nie chcą jej podejmować, pomimo ogromnej wagi takiej rozmowy (zob. rysunek 1.3).

**Rysunek 1.3. DevOps można porównać do znajdującego się w pomieszczeniu słonia, który często pozostaje niezauważony**

# Czym NIE jest DevOps?

Bardzo często podczas definiowania, czym jest DevOps, łatwiejsze może się okazać wyjaśnienie, czym DevOps *NIE* jest. Na początek wyjaśnię typowe błędne rozumienie, z którym często można się spotkać.

## DevOps to NIE tylko narzędzie, technologia bądź produkt

Na rynku istnieje sporo narzędzi, które można stosować w podejściu DevOps. Wielu dostawców usług chmury oraz narzędzi promuje je w mniej więcej taki sposób: *„Za pomocą tego narzędzia możesz pracować zgodnie z podejściem DevOps”* lub *„To narzędzie jest niezbędne dla zespołu stosującego podejście DevOps”*.

Gdy ludzie słyszą określenie „DevOps”, niektórym przychodzą na myśl technologie typu **Amazon Web Services** (AWS), Azure lub **Google Cloud Platform** (GCP) albo takie jak Docker i Kubernetes. Co więcej, ekosystem Kubernetes również obejmuje wiele komponentów, np. Istio, Flux, Helm, Envoy i Prometheus. Istnieje wiele platform, m.in. GitHub Actions, CircleCI i Jenkins, ułatwiających nieustanne, szybkie i częste przygotowywanie nowych wydań produktu. Świat narzędzi monitorowania jest równie podzielony. Jeżeli jeszcze nie kojarzysz tych nazw, poznanie i zrozumienie ich niuansów oraz zalet może wymagać nieco czasu.

Jeżeli jednak opanujesz pracę z nimi oraz będziesz stosować sprawdzone architektury i zapoznasz się z historiami przedstawiającymi osiągnięcie sukcesu za ich pomocą, nie oznacza to stosowania podejścia DevOps. Za narzędziem zawsze stoją osoby i procesy, a zmiana narzędzia nie wiąże się automatycznie z ich zmianą.

Często można spotkać się z kaskadowym modelem programowania, nawet w przypadku korzystania z narzędzi zawierających w nazwie bądź opisie słowo „DevOps”. Pozostawanie w zgodzie z istniejącymi skomplikowanymi regułami organizacji poprzez zastosowanie w zautomatyzowanych rozwiązaniach wieloetapowego procesu zatwierdzania zmian, poprzez ograniczenie liczby wydań do jednego kwartalnie w celu uniknięcia zmian w księgowości bądź w procedurach bezpieczeństwa, a także sytuacja, w której zespół infrastruktury zarządza kontenerami utworzonymi przez zespół aplikacji, wykorzystując przy tym nowoczesny system koordynacji typu Kubernetes — to wszystko jedynie zwiększy problemy operacyjne i wprowadzi większą dezorientację, a nie przyniesie żadnej wartości biznesowej.

Ostatecznie osiągnięcie ważnych celów biznesowych i transformacji będzie wymagało zmian u członków zespołu, w organizacjach oraz w procesach. Zatem czy DevOps ma związek ze zmianami u poszczególnych członków zespołu i organizacji?

## DevOps to NIE tylko konkretne osoby, zespoły bądź role

Samo określenie DevOps stało się w pewnym sensie modnym słowem. Do zajmowania się zadaniami związanymi z DevOps firmy zatrudniają inżynierów, których stanowiska mają nazwy w stylu „inżynier DevOps” bądź „architekt DevOps”, a ich zespoły są określane mianem „zespołów DevOps”. Osoby, które uważają, że nie są w stanie samodzielnie wykonywać zadań charakterystycznych dla podejścia DevOps, mogą popełniać błąd polegający na zleceniu tego rodzaju obowiązków zewnętrznym partnerom.

Gdy spojrzysz na tę sytuację, możesz odnieść wrażenie, że DevOps odwołuje się do konkretnych osób, zespołów bądź ról, choć w rzeczywistości tak nie jest. W wielu przypadkach te określenia odwołują się po prostu do ról typu superinżynier infrastruktury, superprogramista lub superinżynier chmury.

Firmy działają w ten sposób, aby szybko reagować na zmieniające się wymagania biznesowe oraz móc dostarczać częste wydania produktu, co obecnie wiąże się z koniecznością zastosowania nowych technologii i automatyzacji. Co więcej, często muszą zapewnić możliwość obsługi skomplikowanych platform obejmujących komponenty zarówno istniejące, jak i nowe. To obejmuje użycie usługi GitHub Actions, której dokładniejsze omówienie znajdziesz w tej książce. Obszary zwykle wymagające ręcznej instalacji w dziedzinie infrastruktury, konfigurowania za pomocą aplikacji graficznych lub poleceń wydawanych w powłoce teraz wymagają umieszczenia ich w systemie kontroli wersji Git oraz konfigurowania i zarządzania za pomocą zautomatyzowanych rozwiązań.

Jednak w praktyce podejście DevOps zajmuje się jeszcze bardziej skomplikowanymi kwestiami. To nie jest jedynie świetne opanowanie tych narzędzi i technologii oraz umiejętność konfigurowania ról dla nich. W rzeczywistości konieczni są silni liderzy, którzy będą w stanie poprowadzić transformację w istniejących systemach i organizacjach. Bardzo często to obejmuje działania wykraczające poza szerokie określenie DevOps. Spotykałem się z wieloma osobami, którym naprawdę udało się w pełni zastosować podejście DevOps i osiągnąć znacznie większe cele niż jedynie zaimplementowanie *podejścia DevOps*.

DevOps to w zasadzie droga do przeobrażenia organizacji. W jej trakcie będzie używanych wiele skomplikowanych technologii i produktów. Tego nie można ograniczyć do przyjęcia kolejnych pracowników z nowymi umiejętnościami do istniejących zespołów albo do zmiany opisu zespołu lub stanowiska.

Zatem jeśli DevOps nie jest jedynie narzędziem oraz nie wiąże się wyłącznie z osobami i organizacjami, to czy istnieje proces określany mianem DevOps?

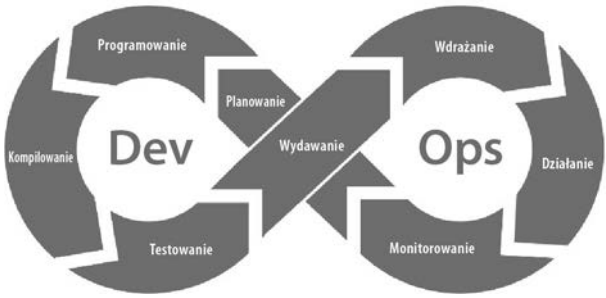
## DevOps NIE jest jedynie procesem

Zespół DevOps często stosuje iteracyjny model programowania, oparty na okresach wynoszących 2 – 3 tygodnie, powszechnie nazywanych **sprintami**, w trakcie których codziennie odbywają się krótkie spotkania. To prawda, że dzięki podejściu DevOps można stosować procesy zgodne z metodyką zwinną (Agile) oraz z najlepszymi praktykami Scrum. W branży tworzenia oprogramowania są to metody, w których poszczególne funkcjonalności są dzielone na mniejsze fragmenty, a praca nad nimi odbywa się za pomocą wielu krótkich cykli.

Zatem czy DevOps to rezultat ewolucji metod Agile lub Scrum, czy koncepcja je obejmująca? Odpowiedzią na oba te pytania jest jednocześnie i *tak*, i *nie*.

Zasięg podejścia DevOps jest szeroki i nie obejmuje jedynie koncentracji na tworzeniu oprogramowania, jak ma to miejsce w przypadku metodyk zwinnych. Jest rozszerzony jeszcze o wydawanie oprogramowania, zbieranie od użytkowników informacji na jego temat oraz wprowadzanie w nim usprawnień. Te reguły i filozofie mają zastosowanie poprzez tzw. **cykl życiowy tworzenia oprogramowania** (ang. *software development*

*life cycle*, SDLC). Zespół koncentruje się na całym cyklu produktu, a nie jedynie na opracowaniu nowych funkcjonalności lub zaprojektowaniu komponentów internetowych. Spójrz na rysunek 1.4.




**Rysunek 1.4. Osiem etapów cyklu życiowego tworzenia oprogramowania**

Jednak ostatecznym celem podejścia DevOps nie jest wprowadzanie nowego procesu.

W podejściu DevOps zastosowanie ma wiele praktyk. Wybrane praktyki omawiam dokładnie, począwszy od rozdziału 5. Jednak nie wszystkie te praktyki można w takim samym stopniu stosować dla każdego zespołu. Na przykład środowisko wymagające wdrożenia aplikacji w wielu serwerach, np. Microsoft, będzie inne niż w przypadku, gdy mała aplikacja jest wdrażana w egzemplarzu EC2. Praktyki możliwe do zastosowania będą różniły się także w zależności od wielkości bazy użytkowników: inne w przypadku miliona użytkowników, a inne dla bazy składającej się z zaledwie tysiąca użytkowników.

Oto co o metodyce Agile pisze na swoim blogu Andy Hunt:

> *Metodyki Agile zmuszają programistów do myślenia i, szczerze mówiąc, to może stanowić problem. Znacznie wygodniejsze jest po prostu stosowanie się do narzuconych reguł i stwierdzenie „robię to zgodnie z zasadami". To jest łatwe, pozwala uniknąć wyśmiania lub oskarżeń, a także zwolnienia. Wprawdzie można publicznie potępiać tego rodzaju wąski zbiór reguł, ale nie da się zaprzeczyć, że zapewnia on bezpieczeństwo i komfort. Jednak stosowanie metodyk zwinnych bądź zapewnienie efektywności nie polega na zapewnianiu komfortu.*

Ta idea ma zastosowanie również w przypadku podejścia DevOps. W celu częstego i płynnego udostępniania wydań produktu przygotowanych z myślą o użytkownikach, zaoferowania niezawodnej usługi i ostatecznie zapewnienia znacznego wpływu na działalność biznesową konieczne jest spojrzenie na problemy ze znacznie szerszej perspektywy niż jedynie narzędzia, osoby i procesy.

## DevOps to kultura pracy

DevOps to więcej niż tylko powiązanie zespołów programistycznego i operacyjnego, określonych ról, procesów, narzędzi, technologii bądź produktów. Często jednak istnieje tendencja do upraszczania DevOps poprzez sprowadzenie jedynie do ludzi, procesów i narzędzi.

Zatem czym tak naprawdę jest DevOps?

Warto zapoznać się ze stanowiskiem Patricka Deboisa, ważnego członka społeczności DevOps:

> *Moja obecna definicja Dev*Ops: wszystko, co robisz, aby pokonać tarcia spowodowane przez silosy. Wszystko inne to zwykła inżynieria.*

W obecnej erze technologii chmury zarówno firmy stosujące najnowsze rozwiązania technologiczne, jak i tradycyjne przedsiębiorstwa, a także startupy mogą łatwo uzyskiwać dostęp do tych samych środowisk. Nawet narzędzia oparte na sztucznej inteligencji są dostępne w rozsądnych cenach.

Ostatecznie czynnikiem utrudniającym rozwój zespołu i firmy są tarcia, które często pochodzą z silosów obejmujących osoby, procesy i narzędzia.

Sedno DevOps przedstawia zmianę kulturową i podejście opracowane w celu wyeliminowania tarć pojawiających się między silosami w organizacji. Wiąże się to ze zmianą nastawienia, zwyczajów i kultury pracy.

## Reguły DevOps

Jak dokładnie przedstawiają się filary kultury pracy w podejściu DevOps? Warto zagłębić się w podstawowe reguły stojące za tym podejściem, aby dzięki temu znacznie dokładniej zrozumieć, z czym tak naprawdę wiąże się ta kultura pracy.

### Koncentracja na kliencie

Każda akcja w poszczególnych procesach powinna mieć na celu dostarczenie wartości klientowi.

Koncentracja na kliencie podczas tworzenia oprogramowania i jego późniejszej obsługi technicznej pozwala na szybsze dostarczanie funkcjonalności oczekiwanych przez użytkowników. Krótki cykl otrzymywania informacji zwrotnych pomaga w lepszym dostosowaniu produktu do potrzeb użytkowników, zmniejsza ryzyko oraz maksymalizuje wartość najniższym możliwym kosztem. Dzięki skoncentrowaniu się na potrzebach klienta ostatecznym celem staje się efektywne rozwiązywanie rzeczywistych problemów. To ułatwia nadawanie priorytetu zadaniom, zarządzanie zbiorami zadań do wykonania, usprawnienie alokacji zasobów oraz zapewnienie, że operacje będą efektywne i zoptymalizowane pod względem kosztów. W efekcie otrzymujemy solidne podstawy dla długoterminowego sukcesu biznesowego poprzez lepsze spełnienie oczekiwań rynku i użytkowników.

W podejściu DevOps koncentracja na kliencie nie jest jedynie sloganem, to jest konieczność.

### Tworzenie z myślą o zakończeniu

Zrozumienie potrzeb klientów i rozwiązywanie rzeczywistych problemów powinno mieć pierwszeństwo przed działaniem opartym jedynie na założeniach. To zachęca do strategii holistycznej, w ramach której zespoły synchronizują zadania programistyczne i operacyjne, aby spełnić wymagania klienta. To obejmuje pełny cykl życiowy produktu, od jego powstania poprzez wdrożenie aż po nieustanną obsługę techniczną. W ten sposób mamy pewność, że ostatecznie produkt dostarczany użytkownikom końcowym będzie miał dla nich wartość.

Nieuwzględnienie od samego początku potrzeb użytkowników może prowadzić do powstania produktu, który jest nienaganny pod względem technicznym, a jednocześnie nie spełnia oczekiwań użytkowników ani nie ułatwia rozwiązanie problemów. Tę regułę można potraktować jako nieustanny sygnał nakazujący dostosowanie wszystkich działań technologicznych do celów biznesowych i oczekiwań użytkowników. To daje pewność, że ostatecznie otrzymany produkt będzie nie tylko funkcjonalny, ale również cenny i interesujący dla użytkowników.

### Autonomiczne i wielozadaniowe zespoły

Warunkiem koniecznym do osiągnięcia zapewniającej sukces implementacji DevOps jest posiadanie strategii niezbędnej do przygotowania autonomicznych i wielozadaniowych zespołów. W przeciwieństwie do tradycyjnych środowisk inżynieryjnych, w których specjaliści z dziedzin programowania, operacji i **zapewnienia jakości** (ang. *quality assurance*, QA) pracują w oddzielnych działach, w przypadku DevOps następuje wyeliminowanie granic między nimi. Znaczenie kluczowe ma tutaj nie tylko przygotowanie zróżnicowanych zespołów, ale również zapewnienie im autonomii — możliwości nadzorowania produktu bądź funkcjonalności od jej powstania aż po dostarczenie.

Dlaczego to ma znaczenie krytyczne? Odpowiedź kryje się w zwinności i efektywności. Gdy zespół posiada pewien zakres umiejętności, od tworzenia kodu źródłowego, poprzez testowanie i wdrażanie, po projektowanie (a nawet podstawową znajomość reguł biznesowych), wówczas przyśpiesza się proces podejmowania decyzji. To pozwala zastąpić powolny i uciążliwy łańcuch zarządzania istniejący w systemach hierarchicznych kulturą szybkich i skutecznych działań.

Wyeliminowanie organizacyjnych wąskich gardeł nie tylko przyśpiesza prace, ale również wspiera kulturę własności i odpowiedzialności. Zespół nie musi czekać, aż dział zewnętrzny bądź znajdujący się wyżej w hierarchii będzie podejmował decyzje, zespół ma bowiem umiejętności i upoważnienie do samodzielnego radzenia sobie z trudnościami.

Poprzez wyeliminowanie barier, które często prowadzą do tarć w organizacji, autonomiczne i wielozadaniowe zespoły stają się podstawowym czynnikiem zapewniającym płynność działania w podejściu DevOps. Efektem jest usprawniony proces, który ułatwia szybkie reagowanie na zmiany oraz promuje kulturę współpracy i odpowiedzialności. To nie jest jedynie cecha, którą dobrze jest mieć w nowoczesnej inżynierii, lecz podstawowa strategia dla każdej organizacji, która myśli o pomyślnej implementacji podejścia DevOps.

### Nieustanne usprawnianie

Nieustanne usprawnianie stanowi podstawę podejścia DevOps. Oferuje przy tym korzyści zarówno techniczne, jak i kulturowe, które okazują się niezbędne w nowoczesnych środowiskach tworzenia oprogramowania i przeprowadzania związanych z nim operacji. Pod względem technicznym pozwala zapewnić takie cechy jak niezawodność, adaptacyjność i efektywność procesu dostarczania oprogramowania, co jest możliwe dzięki nieustannej analizie wskaźników wydajności działania i poziomu użycia, przeprowadzanej przez zautomatyzowane rozwiązania. Dzięki tym cechom produkt końcowy nie tylko jest niezawodny, ale również ma swój wkład w optymalizację zasobu i szybsze dostarczanie funkcjonalności. Pod względem kulturowym nieustanne usprawnianie sprzyja

współpracy, gwarantuje odpowiedzialność, zachęca do uczenia się i ostatecznie pomaga w łamaniu barier organizacyjnych. Przekłada się to na lepsze wrażenia programisty.

Waga nieustannego usprawniania staje się jeszcze bardziej widoczna, gdy jest analizowana przez pryzmat pętli informacji zwrotnych, która działa jako układ nerwowy cyklu życiowego DevOps. Te pętle umożliwiają monitorowanie w czasie rzeczywistym oraz dostarczają wskaźniki pozwalające reagować i bezpośrednio wpływające na innowacje związane z nieustannym usprawnianiem. Dzięki regularnej ocenie systemu wydajności działania, zaangażowania użytkownika oraz innych **kluczowych wskaźników wydajności** (ang. *key performance indicators*, KPI) organizacje mogą szybko dostosowywać strategie i zapewnić długoterminowy sukces. Tego rodzaju dynamiczne i oparte na danych podejście ma ważne znaczenie w obecnym szybko zmieniającym się świecie technologii. Z tego powodu nieustanna integracja to nie tylko najlepsza praktyka, ale również podstawowe wymaganie, które trzeba spełnić, aby sprostać konkurencji.

### Automatyzacja

W tradycyjnym modelu tworzenia oprogramowania zadania związane z programowaniem i zadania dotyczące operacji często są wykonywane przez dwa oddzielne zespoły. Programiści koncentrują się na tworzeniu kodu źródłowego i budowaniu aplikacji, podczas gdy dział operacji zajmuje się wdrażaniem i późniejszą obsługą techniczną. Skutkiem takiego rozdziału kompetencji są często opóźnienia, nieefektywność i tarcia występujące między tymi dwoma zespołami.

**Ciągła integracja/ciągłe wdrażanie** (ang. *continuous integration/continuous delivery*, CI/CD) to system, który pojawił się jako most pozwalający połączyć te dwa światy. Dzięki ciągłej integracji (CI) zmiany wprowadzane w kodzie przez wielu programistów są często łączone we współdzielonym repozytorium, w którym zautomatyzowane testy mają na celu wykrywanie błędów i niespójności najwcześniej jak to możliwe. To zachęca do większej współpracy poprzez ułatwianie identyfikowania problemów na wczesnych etapach cyklu pracy. Z kolei ciągłe wdrażanie (CD) gwarantuje, że kod w każdej chwili jest gotowy do wdrożenia. To eliminuje długie *okresy zamrożenia*, w których nie można dodawać nowych funkcjonalności, ponieważ przygotowywane jest nowe wydanie produktu.

Ręcznie wykonywane procesy nie tylko są podatne na błędy, ale również prowadzą do ogromnego spadku szybkości i efektywności działania, czyli do problemów, które ma rozwiązywać DevOps. Zanim upowszechniła się automatyzacja, administratorzy ręcznie konfigurowali serwery, co było procesem żmudnym i podatnym na błędy. Ponadto programiści często mieli trudności z odtworzeniem środowiska operacji na potrzeby testów, to zaś notorycznie prowadziło do syndromu „u mnie działa".

W tym kontekście automatyzacja nie jest luksusem, lecz koniecznością. Skrypty zautomatyzowane odpowiadają za wykonywanie różnych zadań, od testowania kodu po jego wdrażanie, gwarantując przy tym maksymalną standaryzację procesu. W efekcie następuje wyeliminowanie wielu błędów i opóźnień nieuchronnie związanych z ręcznym wykonywaniem zadań.

# Osiąganie doskonałości w stosowaniu praktyk DevOps

Dotychczas wyjaśniłem podstawową koncepcję DevOps. Ta koncepcja tak naprawdę nieustannie ewoluuje. Niektóre koncepcje są przedstawiane jako *co powinno być uwzględnione od samego początku* albo *co zostało pierwotnie uwzględnione, a nie było omówione*. Mogą być one nazywane inaczej niż DevOps.

Warto zapoznać się z wybranymi spośród najważniejszych koncepcji w tym kontekście. Ponadto dobrze jest je włączyć do kultury pracy DevOps.

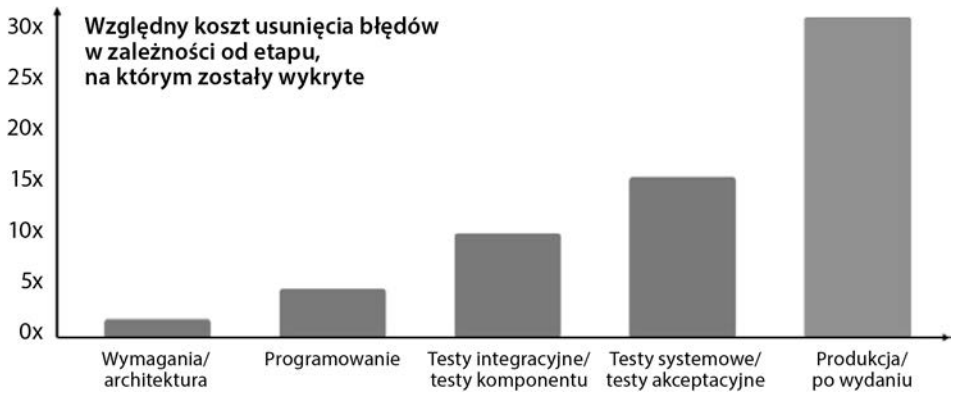
## DevSecOps

DevSecOps to podejście, w którym elementy zapewnienia bezpieczeństwa zostają zintegrowane w postaci frameworka DevOps. W tej metodzie już od samego początku pracy zapewnienie bezpieczeństwa jest uznawane za aspekt o znaczeniu krytycznym. Celem jest powstanie oprogramowania, które będzie zarówno efektywne, jak i bezpieczne, a ponadto zostanie dostarczone na czas.

Tradycyjnie za zapewnienie bezpieczeństwa odpowiadał oddzielny, wyspecjalizowany zespół, który przystępował do pracy na ostatnich etapach cyklu programistycznego. W wielu przypadkach luki w zabezpieczeniach były odkrywane już po zakończeniu pracy przez programistów, co prowadziło do opóźnień w wydaniu oprogramowania. Dzięki przyjęciu podejścia polegającego na przesunięciu w lewo w kierunku zapewnienia bezpieczeństwa można w znacznym stopniu zmniejszyć koszt wprowadzania poprawek. W kontekście DevSecOps **przesunięcie w lewo** odnosi się do praktyki zajmowania się aspektami bezpieczeństwa na wcześniejszych etapach cyklu życiowego tworzenia oprogramowania. Idealnie będą to fazy projektowania i programowania. Zajęcie się kwestiami bezpieczeństwa już od samego początku oznacza, że zespół jest w stanie znacznie efektywniej wyszukiwać i eliminować luki w zabezpieczeniach. To przekłada się na niższy koszt i mniejsze ryzyko w porównaniu do wprowadzania poprawek na późniejszych etapach pracy. Podejście przesunięcia w lewo kładzie nacisk na bezpieczeństwo proaktywne zamiast reaktywne, gwarantując, że bezpieczeństwo aplikacji w naturalny sposób wynika z ich projektu. Koszty usunięcia problemów związanych z zapewnieniem bezpieczeństwa, które pojawiają się na etapie działań operacyjnych, mogą być niewspółmiernie wysokie. Co więcej, w przypadku koncentrowania się na późniejszych etapach pojawia się ryzyko dla danych klienta, a ponadto zagrożona może być reputacja firmy.

Zgodnie z badaniami prowadzonymi przez **National Institute of Standards and Technology** (**NIST**) wyeliminowanie błędów na etapie produkcji może być 30 razy kosztowniejsze (zob. rysunek 1.5). Ten koszt może być nawet 60-krotnie większy w przypadku usterek związanych z zapewnieniem bezpieczeństwa.

Cechą wyróżniającą DevSecOps jest koncepcja traktowania bezpieczeństwa nie jako punktu końcowego, ale jako **nieustannego stanu**. Na przykład w przeszłości powszechne było przeprowadzanie statycznej analizy kodu źródłowego bądź sprawdzanie kolejnych punktów na liście bezpieczeństwa, co odbywało się podczas wydawania aplikacji lub w trakcie wdrożeń technicznych. Jednak cykle wydań oprogramowania uległy skróceniu,

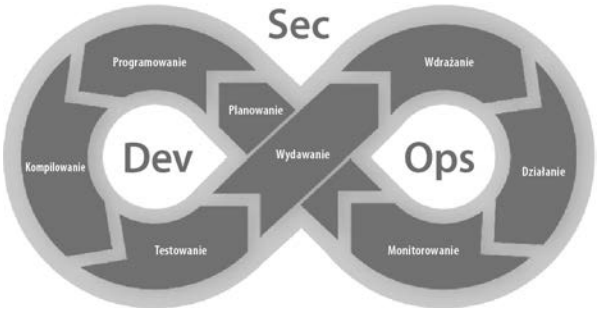



**Rysunek 1.5. Wykres przedstawiający względny koszt usunięcia błędów w zależności od etapu, na którym zostały wykryte (źródło: NIST)**

a technologia rozwija się znacznie szybciej, więc ręczne sprawdzenie wszystkich aspektów bezpieczeństwa za każdym razem stało się niepraktyczne. DevSecOps pozwala rozwiązać ten problem poprzez zastosowanie zautomatyzowanych narzędzi w celu nieustannego zapewniania bezpieczeństwa. To pozwala zareagować szybko w przypadku wykrycia nowych luk w zabezpieczeniach.

Pod tym względem DevSecOps można zdefiniować jako proces integracji bezpieczeństwa i narzędzi z procesem DevOps. W ten sposób powstaje kultura, w której programiści również uwzględniają kwestie bezpieczeństwa, traktują bezpieczeństwo jako stan, a nie artefakt w konkretnej chwili. To prowadzi do powstania stanu, w którym procesy, środowiska i dane zawsze zostają zachowane, aby bezpieczeństwo i innowacje mogły być ze sobą zgodne.

DevSecOps ma znaczenie krytyczne szczególnie dla firm wykorzystujących **oprogramowanie otwartoźródłowe** (ang. *open source software*, OSS). Obecnie większość firm aktywnie korzysta z jakiejś formy oprogramowania otwartoźródłowego, które może zawierać jeszcze nie odkryte luki w zabezpieczeniach. Dzięki wykorzystaniu reguł DevSecOps oraz cotygodniowemu sprawdzaniu ich zastosowania w przyjętych sposobach pracy istnieje możliwość wykrycia tych luk w zabezpieczeniach na wczesnym etapie oraz szybkiego wprowadzenia niezbędnych zmian. Spójrz na rysunek 1.6.




**Rysunek 1.6. DevSecOps kładzie nacisk na bezpieczeństwo w cyklu życiowym DevOps**

Ogólnie rzecz biorąc, celem podejścia DevSecOps jest zadbanie o bezpieczeństwo w trakcie cyklu życiowego od programowania po operacje, co pozwala tworzyć znacznie bezpieczniejsze i efektywniejsze oprogramowanie. Takie podejście zintegrowane umożliwia współistnienie biznesu i bezpieczeństwa.

## Infrastruktura jako kod

**Infrastruktura jako kod** (ang. *infrastructure as code*, IaC) to metoda, w której to kod jest używany do przygotowywania systemów infrastruktury i zarządzania nim, m.in. konfiguracji sieciowej i ustawień serwera. Tego rodzaju zadania są automatyzowane za pomocą wyspecjalizowanego oprogramowania do zarządzania konfiguracją. Zadania takie jak przygotowanie serwera i konfiguracja sieci tradycyjnie były przeprowadzane ręcznie przez człowieka, na podstawie określonej dokumentacji. To wiązało się z wieloma problemami, m.in. wysokim stopniem skomplikowania i czasochłonną naturą tych zadań, wysokim ryzykiem błędu ludzkiego oraz potencjalnymi niespójnościami między formalną dokumentacją i rzeczywistym środowiskiem. Te problemy stawały się szczególnie dotkliwe w ogromnych systemach i prowadziły do tego, że ręczne zarządzanie stawało się niemożliwe.

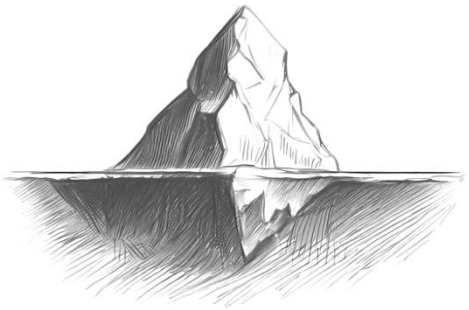
Pojawienie się infrastruktury jako kodu pomogło w ogromnym stopniu ograniczyć skalę problemu. W środowisku IaC stan infrastruktury jest definiowany w kodzie, który następnie jest wykonywany przez narzędzia przeznaczone do zarządzania konfiguracją. To eliminuje potrzebę ręcznego wykonywania żmudnej pracy oraz gwarantuje wysoki poziom reprodukcji i spójności. Co więcej, ponieważ te narzędzia obsługują zautomatyzowane rozwiązania w zakresie tworzenia i działania infrastruktury, proces staje się znacznie efektywniejszy i może być zautomatyzowany, co z kolei pozwala otrzymać dużo bardziej niezawodny i skalowalny system zarządzania.

## Obserwowalność

Obserwowalność ma istotne znaczenie w zarządzaniu nowoczesnymi systemami oprogramowania, zwłaszcza w przypadku podejścia DevOps, w którym dąży się do wyeliminowania pojawiających się tarć oraz usprawnienia komunikacji między zespołami w organizacji. Obserwowalność jest często postrzegana jako ewoluująca forma monitorowania, przy czym każde z tych zadań ma odmienne przeznaczenie.

W przypadku tradycyjnego monitorowania nacisk kładzie się na obserwację wcześniej określonych elementów systemu. W zasadzie monitorowanie bazuje na regułach. Obejmuje zdefiniowanie wskaźników i testów wydajności, których niespełnienie spowoduje wygenerowanie powiadomienia. Monitorowanie wiąże się z szukaniem znanych problemów — jest to podejście, w którym zadaje się pytanie typu „czy mój system działa zgodnie z oczekiwaniami?”. Jednak zasięg monitorowania jest zwykle ograniczony do obserwacji miejsc, w których pojawiają się problemy. Nie ma związku z tym, co się zdarzyło. W efekcie monitorowanie może pokazywać jedynie wierzchołek góry lodowej (zob. rysunek 1.7). Otrzymanie pełnego obrazu sytuacji może wymagać nieco czasu.

Z kolei obserwowalność to znacznie dokładniejsze podejście. Nie tylko wykorzystuje monitorowanie, ale również idzie o krok dalej i oferuje pełny zbiór informacji o ogólnej kondycji systemu. W takim modelu mamy mniej przyglądania się pod kątem określonych



**Rysunek 1.7. Obszary widoczne podczas monitorowania to zaledwie wierzchołek góry lodowej**

problemów, a więcej dokładniejszego zrozumienia, co tak naprawdę dzieje się w systemie. Obserwowalność zachęca do spojrzenia „pod maskę" i pozwala zadawać pytania typu „jaki jest bieżący stan systemu i dlaczego taki jest?".

W przeciwieństwie do monitorowania obserwowalność bazuje na wskaźnikach, dziennikach zdarzeń i śladach, znanych jako dane telemetryczne, aby zapewniać wszechstronny i spójny obraz wydajności działania systemu. W świecie podejścia DevOps monitorowanie to nie jest po prostu odizolowane zadanie dla zespołu operacji, lecz wspólna odpowiedzialność obejmująca członków zespołu zarówno operacji, jak i programistycznego. Gdy systemy stają się coraz bardziej złożone, zwłaszcza w związku z pojawieniem się technologii natywnej chmury i mikrousług, gwałtownie rośnie również znaczenie obserwowalności. Oferuje ona bardziej holistyczny sposób na zrozumienie, jak poszczególne komponenty systemu oddziałują na siebie, ułatwia wychwytywanie wąskich gardeł, pomaga w debugowaniu problemów oraz umożliwia optymalizację wydajności działania (zob. rysunek 1.8).


**Rysunek 1.8. Trzy główne filary obserwowalności to śledzenie, wskaźniki i dzienniki zdarzeń**

Narzędzia tradycyjnego monitorowania często były tworzone w celu spełnienia wymagań i potrzeb określonych struktur organizacyjnych, co zwykle prowadziło do powstania sfragmentowanych rozwiązań. Jednak środowiska natywnej chmury wymagają bardziej zintegrowanego podejścia. Narzędzia obserwowalności zostały opracowane z myślą o zapewnieniu wspomnianej integracji i oferują ujednolicony podgląd stanu systemu w różnych środowiskach. Takie wszechstronne podejście pomaga zespołom DevOps w usprawnieniu procesów, złagodzeniu ryzyka oraz znacznie efektywniejszym uczestniczeniu w realizacji celów organizacji.

## Następne wyzwanie

W tym momencie rozumiesz już podstawy DevOps. W tym podrozdziale wyjaśniłem, czym ono jest i czym nie jest, a także jak wygląda kultura pracy w przypadku stosowania podejścia DevOps. Omówiłem również obszary tworzące DevOps, m.in. DevSecOps, IaC i obserwowalność.

Załóżmy więc, że udało Ci się osiągnąć sukces w podejściu DevOps albo znajdujesz się na najlepszej drodze do jego osiągnięcia. Jakie będzie następne wyzwanie? Mam tutaj na myśli pytanie w stylu „jak DevOps może nadal być stosowany z powodzeniem i jak może się rozwijać razem z organizacją?".

Może się zdarzyć coś, co można przedstawić następująco: „Mamy doskonały zespół DevOps. Nie jest idealnie, ale kultura pracy uległa zmianie oraz w dużo większym stopniu współpracujemy ze sobą". Gdy już sądzisz, że zmierzasz do doskonałości, wówczas masowo zaczynają odchodzić inżynierowie, którzy się zawodowo rozwinęli. Szukają lepszego środowiska pracy.

W wielu przypadkach najbardziej doświadczeni inżynierowie zespołu posiadający największą wiedzę będą w tle wykonywali najbardziej skomplikowane operacje. Najcenniejsi inżynierowie być może będą musieli wykonywać te drobne i trywialne zadania, nawet jeśli nie chcą tego robić. Prawda jest taka, że większość organizacji nie posiada środowiska, w którym programiści mogliby pracować wydajnie i z przyjemnością. Przyczyną tego stanu rzeczy mogą być narzędzia bądź ogromna liczba mało istotnych zadań i zbyt duża liczba narzędzi.

Wprawdzie to są tylko przykłady, ale w celu zachowania najlepszych cech podejścia DevOps i umożliwienia jego dalszego rozwoju organizacja musi na dłuższą metę brać pod uwagę programistów. Niezależnie od tego, czy jesteś zwykłym pracownikiem, czy menedżerem, odpowiadasz za zagwarantowanie, że członkowie zespołu będą czuć się dobrze w pracy, co pozwoli zespołowi osiągnąć najlepsze możliwe efekty. Jeżeli Ty i Twoi współpracownicy zapewnicie dobre wrażenia podczas pracy, inni członkowie zespołu również tak zrobią.

Warto powrócić do pierwszego z wcześniej zadanych pytań: jak DevOps może nadal być stosowany z powodzeniem i jak może się rozwijać razem z organizacją?

Jedną z odpowiedzi jest zapewnienie programistom środowiska pracy, które podniesie poziom ich zadowolenia, czyli zagwarantuje jak najlepsze wrażenia programistów.

# Wrażenia programisty — strategia sprzyjająca osiągnięciu doskonałości

**Wrażenia programisty** to pojęcie, które wkracza poza zwykłe wrażenia odbierane przez użytkowników i obejmuje także kwestie produktywności i zadowolenia programistów z ich pracy w zespole DevOps. To również rodzaj strategii organizacyjnej, która ma na celu powodzenie podejścia DevOps. Jeżeli inżynierowie są zadowoleni, to nie ma tarć w organizacji, a jeśli komunikacja odbywa się płynnie, podejście DevOps realizowane pomyślnie. W takim kontekście GitHub to platforma pomagająca w maksymalizacji wrażeń programistów, Git zaś jest narzędziem przyczyniającym się do osiągnięcia wyznaczonego celu. Spójrz na rysunek 1.9.


**Rysunek 1.9. Wrażenia programisty to strategia o kluczowym znaczeniu dla osiągnięcia sukcesu w podejściu DevOps**

Gdy spojrzysz na poszczególne narzędzia w świecie DevOps, mogą się one wydawać małe. Jednak patrząc z perspektywy osiągnięcia sukcesu DevOps i jego kontynuowania, połączenie między wrażeniami programisty oraz sposobami użycia technologii Git i GitHub zaczyna nabierać znaczenia. Jest to płaszczyzna, na której odbywa się ważna komunikacja między programistami.

## Wrażenia programisty to strategia

Wrażenia programisty w zasadzie wiążą się z utworzeniem środowiska, w którym programiści mogą najlepiej wykonywać swoją pracę. W kontekście DevOps podejście to można zinterpretować jako strategię tworzenia, rozwijania i utrzymywania najlepszej kultury pracy podczas stosowania podejścia DevOps w organizacji, które tak naprawdę opiera się właśnie na kulturze pracy. Jeżeli zespół programistyczny jest niezadowolony, nie należy spodziewać się niesamowitej kultury pracy. Bez takiej kultury pracy podejście DevOps nie rozkwitnie.

Według firmy GitHub koncepcję wrażeń programisty można przedstawić za pomocą równania pokazanego na rysunku 1.10. To równanie pojawiło się w artykule *Developer experience: What is it and why should you care?* (*https://github.blog/2023-06-08-developer-↪experience-what-is-it-and-why-should-you-care/*).


**Rysunek 1.10. Równanie określające wrażenia programisty**

Poszczególne elementy tego równania można wyjaśnić w następujący sposób:

- **Produktywność programisty**. Oznacza *efektywność* i *szybkość*. Innymi słowy odzwierciedla, jak efektywnie i jak szybko programista jest w stanie wykonywać zadania.
- **Wpływ programisty**. Obejmuje *implementowanie zmian w kodzie źródłowym* oraz *przejście od pomysłu do produktu*, czyli udział programisty w przekuwaniu pomysłów w rzeczywiste produkty lub usługi.
- **Satysfakcja programisty**. Wskazuje na osiągnięcie *dużego wpływu przy niewielkich tarciach w środowisku pracy*, a także obejmuje *sposób pracy* i *narzędzia*. W praktyce ten element wskazuje, na ile zadowolony jest programista ze swojej pracy, z jaką płynnością się ona odbywa oraz jak efektywne są narzędzia używane przez programistę.

Ponadto wszystkie te elementy są wzmacniane przez **współpracę**. Im lepsza współpraca i komunikacja w zespole, tym większe poziomy produktywności, wpływu i satysfakcji programistów. W kolejnych podpunktach nieco dokładniej wyjaśnię sposób działania tych elementów w podejściu DevOps.

## Produktywność programisty

Zwiększenie produktywności programisty okazuje się sporym wyzwaniem. Podejście DevOps działa jako katalizator usprawniający organizacyjną nieefektywność oraz poprawia ją w zespołach programistycznych.

Obecnie na programistach spoczywa duża odpowiedzialność: tworzenie kodu źródłowego, przeglądanie kodu, planowanie architektury oraz czasami wdrażanie infrastruktury. Niektóre z tych zadań są wykonywane na poziomie organizacji bądź zespołu, podczas gdy inne na poziomie poszczególnych osób. DevOps zdecydowanie podkreśla wagę współpracy, przy czym znaczenie krytyczne ma również skoncentrowanie się na poprawie wydajności działania poszczególnych programistów — to wymaga ostrożnego monitorowania. Wrażenia programisty obejmują również te aspekty.

Wiele gałęzi gospodarki przeżywa obecnie trudności spowodowane brakiem wykształconych inżynierów. Zastanów się nad następującą kwestią: gdyby w organizacji zatrudniającej 100 programistów każdemu z nich udało się zaoszczędzić 10 minut dziennie, dałoby to niemalże 17 dodatkowych godzin. To jakby do organizacji dołączyło dwóch dodatkowych programistów.

Jednak sprawdzenie poziomu wydajności poszczególnych inżynierów będzie rodziło trudności. Wprawdzie pewne wskaźniki ilościowe, takie jak na przykład liczba utworzonych wierszy kodu lub liczba przeprowadzonych operacji przekazywania kodu źródłowego do repozytorium, mogą wydawać się niezawodnymi danymi, ale bardzo często nie są one w stanie pokazać pełnego obrazu sytuacji. Czasami jeden dobrze przemyślany wiersz kodu okaże się cenniejszy niż dziesięć wierszy utworzonych w pośpiechu. Zadania typu opracowywanie i projektowanie architektury z reguły są niemierzalne, co jeszcze bardziej utrudnia ocenę. Opieranie się wyłączenie na tych wąskich wskaźnikach może prowadzić do nieświadomego kultywowania kultury mikrozarządzania. Gdy wskaźniki staną się klasyfikującymi testami wydajności, to pojawia się niebezpieczeństwo promowania sposobu działania, w którym nacisk kładzie się na robienie wrażenia zamiast na tworzenie rzeczywistej wartości. Do tematu wskaźników jeszcze powrócę w rozdziale 6.

W kontekście produktywności programisty jedną z perspektyw będzie używanie sztucznej inteligencji w różnych zadaniach programistycznych. Zastosowanie rozwiązań z zakresu sztucznej inteligencji w pracy programistów oznacza nową erę. Narzędzia takie jak GitHub Copilot działają jako wspomagane przez sztuczną inteligencję asystenty podczas tworzenia kodu źródłowego, pomagają w procesie tworzenia tego kodu, a tym samym prowadzą do skrócenia czasu pracy i zmniejszenia jej kosztu. Od chwili udostępnienia technologii ChatGPT w listopadzie 2022 roku wkład sztucznej inteligencji w proces tworzenia oprogramowania zwiększa się.

## Wpływ programisty

W świecie DevOps często koncentrujemy się na tym, jaki wpływ ma zespół na klientów oraz ogólnie na działalność biznesową. Wskaźniki w tym obszarze często rozpoczynają się od produktu bądź klienta (np. pomiar częstotliwości wdrożeń). Dla porównania w przypadku wrażeń programisty koncentracja dotyczy programisty, a nacisk kładzie się na wpływ, jaki będzie on miał zarówno na bazę kodu, jak i na pomysły, które ostatecznie zostaną przekute na produkty.

Wobec tego jaka jest różnica między produktywnością programisty a jego wpływem? W przypadku produktywności miarą sukcesu może być wykonywanie więcej pracy w krótszym czasie. Natomiast bardzo trudno jest określić, jak mierzyć wpływ i co należy mierzyć w tym zakresie. Ponadto różne zespoły mogą mieć odmienne definicje wpływu.

Najłatwiejsze będzie zidentyfikowanie czegoś, co można wyrazić ilościowo i potraktować jako wkład w ostateczną wartość produktu. Na przykład jeśli zespół opracowuje nową funkcjonalność, miarą wpływu może być liczba korzystających z niej użytkowników bądź wygenerowany przez nią zysk. Innymi przykładami mogą być liczba pobrań, liczba żądań, liczba tzw. **umów o gwarantowanym poziomie świadczenia usług** (ang. *service level agreement*, SLA) itd.

Uważam, że inżynier będzie zadowolony, gdy zostanie wypracowana dobra kultura pracy DevOps. Jednak ktoś mógłby sobie pomyśleć: „Ostatecznie to wszystko i tak sprowadza się do pieniędzy, czyż nie?". Faktycznie, pomiar wpływu programisty to kwestia sprowadzająca się do pieniędzy.

Jeżeli wpływ inżyniera nie zostanie zmierzony poprawnie, a jego wynagrodzenie nie będzie odpowiadało jego wartości rynkowej, wówczas satysfakcja takiego inżyniera z wykonywanej pracy będzie spadała. Dalszy scenariusz nie jest trudny do przewidzenia. Wiele organizacji staje się coraz bardziej skomplikowanych, a wpływ inżyniera staje się pośredni. Jednak nawet obecnie niewiele firm jest w stanie zmierzyć poziom takiego wpływu.

Ostateczne jest to ściśle powiązane z zasadą podejścia DevOps. Przecież w celu zmierzenia wpływu programisty trzeba poznać klienta i jego stan w odniesieniu do produktu, aby opracować produkt z nastawieniem „z myślą o kliencie".

Istnieje wiele wskaźników mierzących wpływ programisty. Najłatwiejszy z nich to czas potrzebny programiście na przeprowadzenie operacji tzw. prośby o scalenie kodu (ang. *pull request*) i czasu, jaki upływa od chwili otrzymania informacji zwrotnej od klienta do chwili jej faktycznego wykorzystania podczas pracy nad kodem.

## Satysfakcja programisty

Co najważniejsze, produktywność programisty i jej skutki w tym kontekście nie mają służyć menedżerom do oceniania produktywności poszczególnych programistów, lecz raczej mają pomagać programistom w przyśpieszeniu ich rozwoju w bardziej pozytywny sposób.

Organizacje powinny zadbać, by ich inżynierowie byli zadowoleni z pracy i dostarczali produkt wartościowy dla klientów. W tym celu organizacja powinna zapewniać jak najlepsze środowisko pracy, procedury i narzędzia.

Podczas gdy na szerszym poziomie organizacji, w kontekście ogólnej optymalizacji, konieczne jest przygotowanie środowiska do zwiększenia produktywności wszystkich programistów, to na węższym poziomie organizacji konieczne jest opracowanie strategii rozwoju wspomagających zwiększanie produktywności programisty zgodnie z jego zespołem, rolą i doświadczeniem. To będzie oznaczało skracanie czasu pracy nad projektem dzięki wprowadzeniu praktyk DevOps, które wyeliminują wąskie gardła w zespołach, a jednocześnie dostarczanie poszczególnym programistom narzędzi niezbędnych do optymalnego działania.

## Współpraca

Gdy chodzi o wzmocnienie wrażeń programisty i pomyślne stosowanie przez niego podejście DevOps, znaczenie kluczowe ma współpraca. Nie sprowadza się ona jedynie do integrowania zespołów programistycznego i infrastruktury oraz współdzielenia pewnych obowiązków. Mam tutaj na myśli przygotowanie środowiska, w którym programiści mogą świetnie się rozwijać, współpracować nad produktem oraz mieć poczucie własności i zaangażowania w projekt. Idealnym rozwiązaniem byłby stan przejrzystej współpracy, w ramach której organizacyjne bariery nie będą jedynie zredukowane, ale całkowicie

wyeliminowane. To swobodny przepływ pomysłów, najlepszych praktyk i konstruktywnej krytyki, prowadzący do powstania środowiska pracy, które będzie wzmacniało wrażenia programisty.

Na przykład uważa się, że podejście DevOps to nie tylko narzędzia i praktyki, ale również kultura pracy, filozofia i koncepcja, których celem jest osiągnięcie sukcesu biznesowego i organizacyjnego. W tym kontekście technologie Git i GitHub są często marginalizowane do *zaledwie narzędzi* lub *kanałów komunikacji*. Czasami taki punkt widzenia jest poprawny, choć często bywa krótkowzroczny. Przyjęcie takiej perspektywy może ograniczyć Cię tylko do pojedynczych aspektów, np. jak używać usługi GitHub Actions na potrzeby tworzenia potoku ciągłej integracji/ciągłego wdrażania (CI/CD) albo jak używać systemu kontroli wersji Git i strategii gałęzi Git, gdy chcesz poznać technologie Git i GitHub na potrzeby zastosowania podejścia DevOps.

Organizacja bądź zespół zwykle składa się z różnych osób, m.in. z programistów. Komunikacja może stać się wyjątkowo skomplikowana, zwłaszcza gdy w zespole lub w organizacji mamy do czynienia z rotacją programistów. Sukces w tym dialogu jest bez wątpienia kluczem prowadzącym do usprawnienia produktywności zespołu lub jego członków. Git i GitHub to nie tylko narzędzia, ale także platformy do współdzielenia kodu źródłowego, otrzymywania informacji zwrotnych, przeprowadzania integracji oraz rozwoju projektu. Prawidłowa komunikacja w tego rodzaju środowiskach ma znaczenie krytyczne dla osiągnięcia sukcesu w podejściu DevOps.

Jednym z najpotężniejszych paradygmatów ułatwiających taką współpracę jest podejście InnerSource. Zapożyczając z modelu współpracy znanego ze świata oprogramowania otwartoźródłowego, InnerSource wzmacnia programistów w organizacji, aby wspomagali zespoły pracujące nad innymi projektami, którymi oni oficjalnie nie muszą się zajmować. Podobnie jak ma to miejsce w przypadku tworzenia projektów otwartoźródłowych, podejście InnerSource zachęca do otwartego dialogu, współdzielenia się kodem, wspólnego rozwiązywania problemów, co w praktyce doprowadzi do zmniejszenia barier często będących plagą w organizacjach.

Model współpracy znany z projektów oprogramowania otwartoźródłowego służy jako potwierdzenie tego, co można osiągnąć w przypadku usunięcia barier. Dzięki publicznemu udostępnieniu kodu źródłowego projekty oprogramowania otwartoźródłowego trafiają do globalnej społeczności programistów, z których każdy wkłada do projektu swoją unikatową perspektywę oraz doświadczenie. Istnieje dwustronna relacja między rozwojem jednostki i wspólnym osiąganiem postępu w pracy. To rodzaj symbiozy, do której osiągnięcia powinny wewnętrznie dążyć zespoły stosujące podejście DevOps. Taki otwarty model niesie ze sobą przejrzystą dokumentację, możliwość analizowania kodu źródłowego przez innych programistów oraz demokratyczne podejście do rozwiązywania problemów, co w efekcie prowadzi do usprawnienia wrażeń programisty.

Przejrzystość okazuje się kluczem w takiej współpracy. Gdy każdy, od początkującego programisty aż po lidera zespołu, może uczestniczyć w dyskusjach, przeglądać bazę kodu źródłowego oraz mieć wpływ na podejmowanie decyzji, wówczas bariery między rolami i działami zaczynają upadać. Trzeba w tym miejscu dodać, że brak barier w naturalny sposób pozwala na szybsze wychwytywanie problemów, znacznie efektywniejsze ich rozwiązywanie oraz powstawanie ujednoliconej wizji produktu.

# Elementy wzmacniające podejście DevOps i wrażenia programisty

Istnieją pewne koncepcje, które mogą pomóc zmaksymalizować sukces w stosowaniu podejścia DevOps. Są to zastosowanie zdecentralizowanego modelu współpracy oraz **platforma jako produkt** (ang. *platform as a product*, PaaP). Tak, InnerSource i platforma inżynierii. One poprawią wrażenia programisty.

## InnerSource — zdecentralizowany model współpracy

Podejście InnerSource czerpie inspirację ze świata oprogramowania otwartoźródłowego i zostało opracowane w celu stosowania w ramach organizacji praktyk pochodzących z projektów otwartoźródłowych. Umożliwia zespołom wielozadaniowym współpracę i dzielenie się doświadczeniem technicznym, a co ważniejsze ma na celu opracowanie znacznie bardziej przejrzystej i globalnej kultury. Takie podejście jest ściśle dopasowane do filozofii DevOps eliminowania organizacyjnych barier oraz promowania kultury współpracy i przejrzystości. InnerSource ma również na celu doprowadzenie do kulturowej zmiany, ruchu w praktyce wzmacniającego wrażenia programisty.

### Definicja podejścia InnerSource

InnerSource to zasadniczo podejście, w którym współpraca nad projektami z dziedziny oprogramowania jest prowadzona w ramach jednej organizacji. Ta koncepcja została spopularyzowana przez firmy takie jak PayPal, aby wyeliminować bariery i granice, które często istnieją w ogromnych organizacjach. Podejście InnerSource zachęca do stosowania kultury otwartości, w której to każdy może mieć wkład w tworzenie dowolnego projektu, a procesy i podejmowanie decyzji są przejrzyste.

### Cztery filary podejścia InnerSource

Warto dokładniej poznać cztery podstawowe reguły podejścia InnerSource, a mianowicie *otwartość*, *przejrzystość*, *priorytetowe mentorstwo* i *dobrowolny udział w tworzeniu kodu źródłowego*.

- **Otwartość**. Eliminuje bariery wejścia dla każdego inżyniera w organizacji. Czytelna dokumentacja, zbliżona do plików *README.md* i *CONTRIBUTING.md* umieszczanych w repozytoriach projektów otwartoźródłowych, powoduje, że z projektami łatwo się zapoznać. Ten poziom otwartości poprawia wrażenia programisty poprzez zmniejszenie tarć, a także pozwala inżynierom na łatwą zmianę kontekstu bądź zespołu. Nawet w przypadku ogromnej liczby mikrousług w środowisku DevOps pomocna będzie taka kultura w zakresie dokumentacji, aby ułatwić współpracę w zespole.
- **Przejrzystość**. Gdy w podejściu InnerSource używam określenia „przejrzystość", mam na myśli jawność podczas podejmowania decyzji dotyczących projektu. Na przykład rozmowy dotyczące problemów i analizy żądań aktualizacji odbywają się z zachowaniem przejrzystości, są udokumentowane i łatwo dostępne. To zapewnia dokładne informacje co do tego, dlaczego określone decyzje zostały podjęte, kto je podjął, a także związany z tym kontekst.

Przejrzystość nie tylko zwiększa jakość projektu, ale również znacząco wpływa na poprawę wrażeń programisty poprzez wywołanie poczucia własności i zaangażowania w projekt.

- **Mentorstwo jako priorytet**. W InnerSource mentorstwo nie jest zajęciem dodatkowym, lecz priorytetem. Rola opiekuna repozytorium w podejściu InnerSource jest również określana jako **zaufana osoba przekazująca kod do repozytorium**, biorąc pod uwagę różnice w czynnościach związanych z ograniczeniami wewnętrznymi.

  Rola ta ma tutaj znaczenie krytyczne. Pełniąca ją osoba nie tylko bierze udział w tworzeniu kodu źródłowego, ale również staje się adwokatem projektu, koncentruje się na jakości i trwałości kodu, a także pomaga nowym współpracownikom. Doskonale zna repozytorium i organizację oraz działa jako most między nimi. Nastawieniem takiej osoby jest otwartość i poświęcenie, aby pomagać innym. Ta rola ma ważne znaczenie dla zachowania wysokiego poziomu wrażeń programistów poprzez zagwarantowanie odpowiedniej jakości pracy pozostałych współpracowników oraz sprzyjanie kulturze nieustannego uczenia się.

- **Dobrowolny udział w tworzeniu kodu źródłowego**. Podstawowym założeniem tej reguły jest podkreślenie dobrowolności współpracy, co prowadzi do wypracowania kultury, w której inżynierowie mają poczucie własności i odpowiedzialności za dany projekt. Jest to podejście oddolne, prowadzące do powstania bardziej organicznego i sprzyjającego współpracy środowiska. Tego rodzaju kultura staje się samowystarczalna, ponieważ inżynierowie angażują się dobrowolnie oraz szanują kulturę zarówno projektu, jak i ogólnie organizacji. Takie środowisko znacznie poprawia wrażenia programisty, a także oferuje możliwości w zakresie rozwoju osobistego i zawodowego. Można je uznać za odzwierciedlenie istniejącej w podejściu DevOps kultury współwłasności, w której wkład poszczególnych osób ma na celu usprawnianie systemu jako całości, a nie tylko pojedynczych komponentów.

### Uzupełniająca się relacja między podejściami InnerSource i DevOps

Dzięki łączeniu praktyk stosowanych w podejściach InnerSource i DevOps organizacje mają możliwość tworzenia holistycznego środowiska sprzyjającego rozwojowi uwzględniającemu zarówno klienta, jak i programistę. InnerSource oferuje narzędzia pozwalające poprawiać wrażenia programisty poprzez skoncentrowanie się na aspektach kulturowych — otwartości, przejrzystości, mentorstwie i dobrowolnym udziale — dzięki którym codzienna praca inżynierów staje się bardziej angażująca, znacząca i satysfakcjonująca.

## Platforma inżynierii

Dostępnych jest wiele usług chmury, zwłaszcza oferowanych przez największych publicznych dostawców tego rodzaju usług. Szczególnie w przypadku fundacji **CNCF** (ang. *cloud native computing foundation*) ekosystem zbudowany wokół Kubernetesa został znacznie rozbudowany i obecnie obejmuje ponad 1000 projektów. Taka mnogość narzędzi zwiększa obciążenie poznawcze zespołów programistycznych.

Platforma inżynierii to nowe podejście w branży, koncentrujące się na racjonalizacji wrażeń programistów oraz na efektywności operacyjnej. Celem jest zaimplementowanie narzędzi wielokrotnego użycia i funkcji samoobsługowych, automatyzacja operacji infrastruktury, a tym samym poprawa wrażeń programistów i ich produktywności. W praktyce klientami dla zespołów platformy są zespoły programistyczne i istnieje szczególny nacisk na spełnienie ich potrzeb.

Zasadniczo platforma inżynierii jest związana z opracowaniem tzw. **wewnętrznych platform programistów** (ang. *internal developer platforms*, IDPs) i zarządzaniem nimi. Służą one do integracji różnych narzędzi, usług oraz zautomatyzowanych sposobów działania, aby zapewnić programistom możliwości w zakresie samodzielnej obsługi. W praktyce zapewniają one programistom *złotą ścieżkę*, pomagając w przejściu od programowania do wdrożenia bez konieczności zagłębiania się w złożoność infrastruktury.

Zespoły platformy często dostarczają portale programistyczne, takie jak Backstage w przypadku Spotify, wzmacniając tym samym ideę PaaP. Najważniejsze nastawienie wiąże się z uznawaniem platformy nie tylko za narzędzie, ale także za produkt przeznaczony dla wewnętrznych programistów — klientów.

Zespoły platformy mają wiele ról, od tworzenia wewnętrznych platform programistów po opracowywanie wewnętrznych umów o gwarantowanym poziomie świadczenia usług. Monitorują wskaźniki wydajności działania zespołu, a także nadzorują proces bezpiecznego i efektywnego dostarczania produktów. Istotnym elementem zestawu narzędziowego zespołów platformy jest infrastruktura jako kod, połączona z niezawodnymi potokami ciągłej integracji i ciągłego wdrażania. Tego rodzaju rozwiązania działają jako centralny układ nerwowego na potrzeby tworzenia i wdrażania kodu źródłowego, automatyzując wszystko, od przygotowania infrastruktury aż po kompilację, testowanie i przekazywanie kodu do różnych środowisk. To pozwala zespołom platformy skoncentrować się na cenniejszych zadaniach i wartości dla klienta — nie są przytłaczane przez operacje wykonywane ręcznie i podatne na błędy.

W gruncie rzeczy platforma inżynierii wzmacnia DevOps, poprawia wrażenia programisty oraz dostosowuje aspekty operacyjne podczas dostarczania oprogramowania. Takie podejście ma na celu wyeliminowanie typowych barier istniejących w ogromnych organizacjach, a także przyśpieszenie drogi od kodu do klienta. W trakcie wykonywania tego zadania platforma inżynierii wypełnia ważną lukę w nowoczesnym świecie podejścia DevOps, nie tylko zapewniając operacyjną efektywność, ale również sprzyjając kulturze współpracy i wspólnej odpowiedzialności.

# Git — system, od którego rozpoczyna się współpraca nad kodem źródłowym

Zarządzanie zmianami kodu źródłowego to trudne zadanie. Członkowie różnych zespołów nieustannie dodają nowy kod do ujednoliconej bazy kodu, który w każdej chwili musi pozostać w stanie funkcjonującym.

Git zmienił sposób, w jaki współpracują ze sobą zespoły tworzące oprogramowanie. Dzięki temu, że jest to system rozproszony i umożliwia tworzenie gałęzi, Git stał się cennym zasobem w zbiorze narzędziowym podejścia DevOps. Wspomniana **gałąź** (ang. *branch*) to w zasadzie oddzielny tor pracy, jakby wszechświat równoległy, w którym można pracować nad nową funkcjonalnością bądź usunięciem błędu, bez wpływania na projekt główny. To pozwala zespołom skoncentrować się jednocześnie na wielu aspektach, co w efekcie prowadzi do tego, że proces programistyczny jest znacznie efektywniejszy i łatwiej się adaptuje do nagłych zmian i krótkich cykli wydań, co jest kluczowym wymogiem w środowisku DevOps.

Git to znacznie więcej niż narzędzie przeznaczone do śledzenia zmian w kodzie — to katalizator zmiany w organizacjach. Dzięki zapewnieniu niezawodnego sposobu współpracy Git promuje lepszą komunikację między zespołami. Jest to zgodne z podstawowym celem podejścia DevOps, jakim jest wyeliminowanie barier w organizacjach, a tym samym usunięcie tarć, aby proces tworzenia i wdrażania oprogramowania stał się bardziej spójny i płynny.

## Świat bez systemu kontroli wersji

Wyobraź sobie świat, w którym nie istnieje system kontroli wersji. Pozbawieni dostępu do systemowego narzędzia śledzenia historii zmian programiści borykaliby się z zarządzaniem zmianami w plikach. Każda zmiana musiałaby zostać udokumentowana ręcznie, co doprowadziłoby do bezkresnego morza komentarzy zaśmiecających rzeczywiste przeznaczenie kodu źródłowego. Wówczas szybkie dostarczanie oprogramowania, jak ma to miejsce w przypadku podejścia DevOps, byłoby praktycznie niemożliwe. Brak systemu kontroli wersji prowadziłby również do tarć we współpracy nad kodem. Ponadto konflikty między plikami stanowiłyby kolejną dużą trudność podczas pracy. Wyobraź sobie następującą sytuację: pracujesz nad pewną funkcjonalnością i nagle się orientujesz, że ktoś nadpisał Twój plik wprowadzonymi przez siebie zmianami — tego rodzaju konflikty mogłyby doprowadzić do zatrzymania pracy nad projektem. Bez systemu kontroli wersji trzeba byłoby stosować także dziwne konwencje nazewnicze typu *v1*, *v2*, *backup-foobar-20230930.py* itd. w celu zachowania kopii zapasowej starszych wersji plików.

## Historia systemu Git

Utworzony w 2005 roku przez Linusa Torvaldsa system Git jest najpopularniejszym na świecie **rozproszonym systemem kontroli wersji** (ang. *distributed version control system*, DVCS).

Być może się zastanawiasz, dlaczego Git w ogóle powstał. Krótka odpowiedź brzmi: z konieczności. Linus Torvalds potrzebował **systemu kontroli wersji** (ang. *version control system*, VCS), który byłby w stanie świetnie wykonywać kilka zadań. Po pierwsze, musiał być szybki i efektywny, aby programiści mogli pracować bez przestojów. Po drugie, musiał pozwalać wielu programistom pracować nad tym samym projektem tak, aby nie wchodzili sobie w drogę. Po trzecie, musiał bez problemu obsługiwać ogromne bazy kodu. Ponadto Torvalds chciał otrzymać system, który byłby w stanie niezawodnie przechowywać historię całego projektu. Oczekiwał również elastyczności w zakresie nieliniowego podejścia do

programowania oraz systemu zapewniającego efektywne zarządzanie wieloma gałęziami projektu i operacjami scalania kodu źródłowego. W odpowiedzi na te potrzeby został opracowany system kontroli wersji Git, który okazał się prosty i jednocześnie niewiarygodnie wręcz potężny w zakresie możliwości.

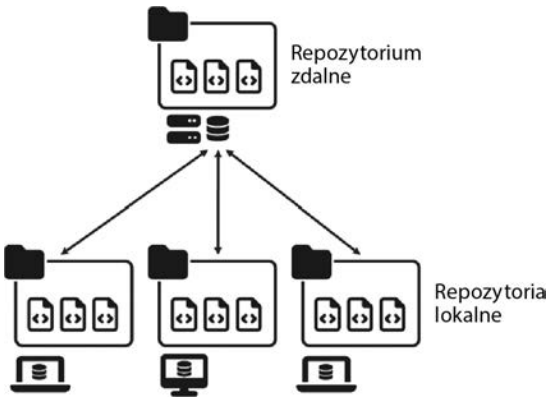
## Czym jest VCS?

Czym więc jest system kontroli wersji (VCS)? Jest to system, który służy do monitorowania modyfikacji wprowadzanych w kodzie na przestrzeni cyklu życiowego tworzenia oprogramowania. W projektach rozwijanych przez wielu programistów niezwykle istotne jest śledzenie tego, kto i kiedy coś zmienił — a także czy spowodowało to powstanie błędów. System kontroli wersji efektywnie koordynuje taki proces śledzenia.

Technologie VCS można podzielić na dwie podstawowe kategorie:

- **Scentralizowane**. W takim modelu mamy jedno zdalne repozytorium przechowujące dane projektu i dostępne dla wszystkich członków zespołu. Przykładami takich rozwiązań są SVN i CVS.
- **Rozproszone**. W modelu rozproszonym, którego przykładem jest Git, każdy programista pracuje z lokalną kopią repozytorium, wprowadza w niej zmiany, a następnie synchronizuje je ze zdalnym repozytorium centralnym.

Na rysunku 1.11 pokazałem, jak pliki są rozpowszechniane i przekazywane w przypadku używania systemu kontroli wersji Git. System ten umożliwia posiadanie wielu repozytoriów zdalnych i elastycznej rozproszonej struktury programowania. Jednak najczęściej jest tworzone jedno repozytorium zdalne z wykorzystaniem platformy programistycznej takiej jak GitHub, a następnie programiści mogą współpracować z repozytoriami zdalnymi i lokalnymi.




**Rysunek 1.11. System kontroli wersji Git działa w modelu rozproszonym, pozwalając wielu osobom jednocześnie pracować nad projektem**

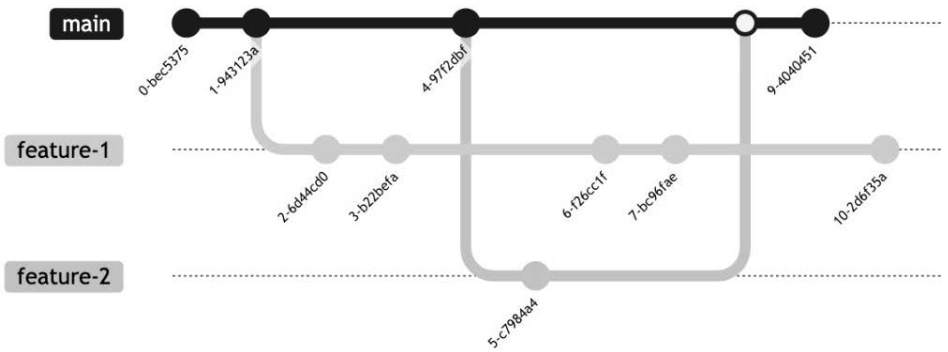
Ogromną zaletą modelu rozproszonego, takiego jak zastosowany w systemie Git, jest autonomia, jaką cieszą się programiści. Tego rodzaju zdecentralizowane podejście pozwala na znacznie sprawniejszą pracę oraz zmniejsza ryzyko powstawania konfliktów między kodem źródłowym tworzonym przez różnych programistów.

## Bezpieczeństwo i spójność

W przypadku systemu Git zapewnienie spójności kodu źródłowego ma absolutnie najwyższy priorytet. W tym systemie każda operacja przekazywania kodu źródłowego do repozytorium ma przypisaną unikatową wartość skrótu (ang. *hash*). Ta wartość jest generowana na podstawie przekazywanego kodu i jego metadanych. Dlatego jeśli kod zostanie zmodyfikowany, wartość skrótu również ulegnie zmianie, co wręcz uniemożliwia majstrowanie przy historii operacji. Operacje wpływające na gałęzie lub tagi, np. scalanie lub przywracanie kodu źródłowego, również będą częścią historii zmian.

## Przystosowalność

Git jest systemem charakteryzującym się wysokim poziomem przystosowalności i można go zastosować w wielu różnych sposobach pracy. Niezależnie od tego, czy masz do czynienia z niewielkim projektem, czy aplikacją na poziomie korporacyjnym, Git potrafi się dostosować na wiele sposobów w zakresie śledzenia zmian oraz ułatwić współpracę członkom zespołu (zob. rysunek 1.12).




**Rysunek 1.12. Git zarządza kodem za pomocą gałęzi, operacji przekazywania kodu źródłowego do repozytorium oraz efektywnych operacji scalania kodu źródłowego**

## Git jest niezastąpiony w nowoczesnym środowisku tworzenia oprogramowania

Przed pojawieniem się Gita dominowały scentralizowane systemy kontroli wersji, takie jak SVN i CVS. Jednak Git spowodował wyraźne przesunięcie w stronę rozproszonych systemów kontroli wersji, widoczną zarówno w przypadku indywidualnych programistów, jak i w organizacjach.

Wraz z pojawieniem się technologii chmury granice między aplikacjami i infrastrukturą zaczęły się zacierać. Git nie jest więc już przeznaczony jedynie dla programistów tworzących kod źródłowy, ale również dla tych, którzy pracują nad infrastrukturą.

W świecie DevOps, w którym bezproblemowa komunikacja i współpraca mają ogromne znaczenie, Git działa jako rdzeń wrażeń programisty. Wspomaga nie tylko zarządzanie kodem źródłowym, ale również strategie pracy, co ma na celu poprawę współpracy.

We współczesnym świecie tworzenia oprogramowania system kontroli wersji nie jest komponentem opcjonalnym, lecz wymaganym.

Dokładne omówienie systemu Git znajdziesz w rozdziałach 2. i 3.

# GitHub — platforma programistyczna wspierana przez sztuczną inteligencję

GitHub to wspierana przez sztuczną inteligencję platforma programistyczna przeznaczona do tworzenia, skalowania i dostarczania bezpiecznego oprogramowania.

GitHub często postrzega się jako jedynie usługę repozytorium bądź jako usługę repozytorium z funkcjonalnością potoku ciągłej integracji i ciągłego wdrażania. Jednak obecnie GitHub to również platforma zapewniająca obsługę całego cyklu życiowego tworzenia oprogramowania, od pisania kodu źródłowego aplikacji po jej kompilację i wydanie. GitHub to także podstawowa platforma przeznaczona do automatyzacji i współpracy w ramach podejścia DevOps. Z perspektywy wrażeń użytkownika to hub dla inicjatyw podejścia InnerSource oraz miejsce, w którym znajduje się kod niezbędny dla platformy inżynierii. Co więcej, GitHub jest wspomagany przez sztuczną inteligencję. Niezależnie od tego, czy tworzysz kod, czy go analizujesz, sztuczna inteligencja GitHuba będzie Cię wspierać w różnych scenariuszach.

Możliwości GitHuba można podzielić na pięć głównych kategorii: **wparcie przez sztuczną inteligencję**, **współpraca**, **produktywność**, **bezpieczeństwo** i **skala**. W kolejnych punktach dokładniej je omówię.

## Wsparcie przez sztuczną inteligencję

Jesteśmy świadkami trwającej transformacji związanej z rozwojem **generatywnej sztucznej inteligencji** (ang. *generative artificial intelligence*) wspomaganej przez **duże modele językowe** (ang. *large language models*, LLM). Ta zmiana wpływa również na inżynierów i pozwala programistom korzystać z pomocy sztucznej inteligencji podczas tworzenia kodu źródłowego. Sztuczna inteligencja potrafi nie tylko czerpać z istniejącego kodu, ale jeszcze przekształcać polecenia języka naturalnego bezpośrednio na kod, a nawet przedstawiać za pomocą tego języka objaśnienia dla kodu. Na początku rewolucji związanej ze sztuczną inteligencją owe funkcjonalności były ograniczone do prostych operacji generowania kodu źródłowego w edytorach. Jednak GitHub jako platforma idzie o krok dalej poza samo generowanie kodu i oferuje wszechstronną obsługę w trakcie całego cyklu życiowego tworzenia oprogramowania.

### GitHub Copilot — sztuczna inteligencja jako partner programistów

GitHub Copilot to jedna z usług GitHuba wykorzystujących sztuczną inteligencję. Wyniki ankiety przeprowadzonej w 2023 roku przez Stack Overflow wskazują, że programiści ją uwielbiają. Jak sztuczna inteligencja poprawia produktywność? Nie ulega wątpliwości, że programiści są w stanie z jej pomocą tworzyć kod szybciej i dokładniej. Jednak zadanie

inżyniera nie ogranicza się do tworzenia kodu. Na przykład wyszukiwanie czy dokumentowanie to również istotne elementy. Inżynierowie często zajmują się różnymi zadaniami — przechodzą z edytora kodu źródłowego do przeglądarki WWW albo do Slacka. Sztuczna inteligencja minimalizuje tę potrzebę wielozadaniowości, co pozwala programistom zachować płynność pracy i koncentrację.

### Wszechstronność sztucznej inteligencji

Sztuczna inteligencja pomaga nie tylko w pracy nad kodem źródłowym. Wspomaga także konfigurowanie potoków ciągłej integracji i ciągłego wdrażania oraz implementację plików YAML na podstawie określonych formatów. Implementacje poleceń powłoki to również łatwe zadania dla narzędzi wykorzystujących sztuczną inteligencję. Okazują się one niewiarygodnie wręcz użyteczne nie tylko dla programistów aplikacji, ale również dla inżynierów platformy oraz przyczyniają się do zwiększenia produktywności zarówno poszczególnych pracowników, jak i całego zespołu.

Ewolucja usług sztucznej inteligencji przebiega niezwykle szybko, a najnowsze informacje błyskawicznie stają się nieaktualne. Dlatego w tej książce nie będę się zagłębiać w konkretne usługi i funkcjonalności GitHub Copilot. Zamiast tego w rozdziale 8. znajdziesz ogólne wskazówki dotyczące technik tworzenia oprogramowania z użyciem technologii LLM.

## Współpraca

GitHub to potężna platforma opracowana w celu zapewnienia zespołom programistycznym doskonałych wrażeń podczas współpracy.

Szybsze wdrożenie się do pracy nowych członków zespołu niezwykle ułatwiają funkcjonalności **GitHub Projects**, **GitHub Issues i GitHub code search**. GitHub Projects pozwala na wizualne układanie zadań i śledzenie ich postępu za pomocą metod Kanban, Roadmap i Table. Natomiast GitHub Issues służy do wskazywania określonych problemów lub błędów. Dzięki oznaczaniu etykietami poszczególnych problemów nowi członkowie mogą szybko zrozumieć, gdzie zaczął się problem. Taka przejrzystość sprzyja współpracy. Z kolei GitHub code search umożliwia szybkie pobieranie poprzednich projektów, analiz i kodu, a także płynne wprowadzenie nowych członków do istniejącej bazy wiedzy i historii projektu.

Dla zapewnienia jakości kodu źródłowego niezwykle użyteczne okazują się funkcjonalności **prośby o scalenie kodu** (ang. *pull requests*) i **kolejek scalania kodu** (ang. *merge queues*). W trakcie operacji prośby o scalenie kodu zmiany wprowadzone w kodzie są przeglądane. Ten proces jest bardziej przejrzysty i efektywny dzięki zintegrowanemu widokowi komentarzy analizy kodu i samego kodu. Z kolei wykorzystanie kolejek scalania kodu gwarantuje, że sprawdzone zmiany są efektywnie i bezpiecznie złączone ze środowiskiem produkcyjnym.

Z perspektywy kultury korporacyjnej funkcjonalności takie jak prośby o scalenie kodu, GitHub Issues, GitHub Discussions i wewnętrzne repozytoria GitHub Enterprise w praktyce prowadzą do *braku barier w podejściu InnerSource* stosowanym w organizacji. Wykorzystanie tych funkcjonalności oznacza wyższy poziom przejrzystości w komunikacji między zespołami, prowadzi do znacznie bardziej otwartej kultury tworzenia oprogramowania

oraz wspiera podejście oddolne. Zapobiega również wyważaniu otwartych drzwi. Zastosowanie podejścia InnerSource minimalizuje nakładanie się wysiłków oraz ogólnie poprawia **satysfakcję pracownika**.

## Produktywność

GitHub oferuje różne narzędzia i funkcjonalności umożliwiające poprawę produktywności.

Przede wszystkim GitHub zawiera funkcjonalności przeznaczone do automatyzacji oraz zwiększenia efektywności podejścia DevOps. Dzięki użyciu usługi GitHub Actions można automatyzować proces na potrzeby potoku ciągłej integracji i ciągłego wdrażania. To pozwala na szybkie wprowadzanie stabilnych zmian w produkcie przy jednoczesnym zachowaniu jakości kodu źródłowego. GitHub Actions obejmuje samodzielnie hostingowane komponenty wykonawcze oraz ogromne komponenty wykonawcze, co pozwala dostosować się do środowisk ograniczonych oraz konkretnych wymagań sprzętowych. Tego rodzaju elastyczność umożliwia szybkie wprowadzanie zmian na rynek, a tym samym skrócenie **czasu do pojawienia się produktu na rynku** (ang. *time to market*, TTM).

Ponadto funkcjonalność automatyzacji w GitHub Projects poprawia efektywność pracy poprzez automatyzację serii zadań. Na przykład można zautomatyzować sposób radzenia sobie z problemami albo sposób nadawania etykiet zgłaszanym problemom. Oczywiście GitHub Actions pozwala również stosować złożone rozwiązania w zakresie automatyzacji. GitHub Codespaces oferuje dostępne online środowisko programistyczne, co pozwala tworzyć kod źródłowy dosłownie z każdego miejsca. Dzięki temu zespoły programistyczne mogą efektywnie współpracować zdalnie. To znacznie skraca czas i minimalizuje ilość zasobów wymaganych do przygotowania środowiska pracy dla nowych członków. Łatwiejsze staje się używanie menedżera `npm` i narzędzia zarządzania pakietami GitHub Packages. Układając zależności, możesz efektywnie współdzielić kod i wielokrotnie go używać, jeszcze bardziej skracając cykl życiowy tworzenia oprogramowania.

GitHub oferuje również funkcjonalność GitHub Copilot, czyli wirtualnego programistę wspomaganego przez sztuczną inteligencję. Zapewnia ona wysoko wydajną funkcję uzupełniania kodu wspomaganą przez duże modele językowe. Pomaga to programistom w znacznie efektywniejszym tworzeniu kodu źródłowego. Ponadto poprawia jego jakość oraz w ogromnym stopniu zwiększa produktywność programisty.

Wymienione funkcjonalności zmniejszają obciążenie programistów, którzy tworzą rzeczywistą wartość produktu. Dzięki temu mogą oni skoncentrować się na generowaniu rzeczywistej wartości i tym samym przyczynić do **wzrostu przychodów**.

GitHub może usprawnić proces tworzenia oprogramowania oraz — dzięki poprawie jakości kodu źródłowego i skróceniu cyklu wydań — przyczynić się do szybszego wprowadzania na rynek produktów spełniających oczekiwania klientów, a tym samym do **wzrostu zadowolenia klienta**.

## Bezpieczeństwo

Wprawdzie GitHub to platforma powszechnie używana podczas tworzenia oprogramowania, ale oferuje ona również zaawansowane możliwości w zakresie zapewnienia bezpieczeństwa. Wraz z coraz szybciej zwiększającą się wagą wysokiego poziomu bezpieczeństwa aplikacji ta funkcjonalność GitHuba zaczyna nabierać większego znaczenia.

Na przykład funkcjonalność **Advanced Security** automatycznie wykrywa luki w zabezpieczeniach kodu źródłowego, ułatwiając firmom i programistom ograniczenie ryzyka poprzez wcześniejsze wyeliminowanie tych luk.

Platforma umożliwia również łatwe zarządzanie problemami z zakresu bezpieczeństwa projektu. W szczególności panel **Security Overview** ułatwia przeglądanie wszystkich ostrzeżeń i ustawień związanych z zapewnieniem bezpieczeństwa, centralizując proces zarządzania.

Ponadto funkcjonalność **secret scanning** automatycznie skanuje kod źródłowy pod kątem wszelkich kluczy tajnych użytkownika bądź kluczy API przypadkowo przekazanych do repozytorium GitHub, a następnie o nich informuje. W przypadku obsługiwanych platform GitHub idzie o krok dalej i automatycznie unieważnia klucze, które pomyłkowo zostały przekazane do publicznie dostępnych repozytoriów. Potrafi również sprawdzić poprawność aktywnych kluczy. Samo wykrywanie kluczy tajnych użytkownika nie spełnia rozbudowanych wymagań narzucanych przez DevSecOps. Takie podejście nie tylko już na wczesnych etapach proaktywnie wyszukuje potencjalne problemy, ale również w ogromnym stopnia ogranicza związane z nimi ryzyko. Poza tym funkcjonalność **push protection** stanowi mechanizm, który chroni przed przypadkowym przekazaniem do repozytorium GitHub informacji wrażliwych.

Dzięki zdefiniowaniu polityki bezpieczeństwa oraz wymuszeniu uwzględniania założeń funkcjonalności **Advanced Security** w repozytoriach programiści mogą znacznie łatwiej przygotowywać bezpieczne środowiska tworzenia oprogramowania. To owocuje znacznie spójniejszym zbiorem środków bezpieczeństwa w organizacji.

Warto w tym miejscu wspomnieć o łańcuchu dostaw. Obecnie praktycznie każda firma w takiej bądź innej postaci korzysta z technologii otwartoźródłowej. Wiele projektów zawiera setki zależności otwartoźródłowych, które mogą powodować ryzyko. Co się stanie w przypadku odkrycia luki w zabezpieczeniach w jednej z takich zależności? W tym miejscu do gry wchodzi Dependabot. Automatycznie wykrywa on luki w zabezpieczeniach zależności oraz proponuje uaktualnienia, pomagając w zmniejszeniu ryzyka związanego z zapewnieniem bezpieczeństwa łańcucha dostaw.

GitHub zajmuje się wieloma aspektami na obszarze zapewnienia bezpieczeństwa. Programistom i firmom oferuje użyteczne narzędzia do tworzenia i zarządzania bezpieczniejszym kodem źródłowym w znacznie krótszym przedziale czasu.

## Skala

W celu skalowania biznesu istotne jest posiadanie dostępu do skalowanej platformy, takiej jak GitHub. Wiarygodność, dostęp globalny i nieustanna innowacja to niezbędne czynniki. Dzięki *większej liczbie używających go programistów niż którejkolwiek z pozostałych platform* (ponad 100 milionów użytkowników) GitHub można uznać za platformę zapewniającą ogromną wiarygodność i jakość.

Jako **dom dla oprogramowania otwartoźródłowego** GitHub ma globalną rozpoznawalność. Programiści i zespoły mogą otwarcie dzielić się swoim kodem źródłowym oraz współpracować z programistami i zespołami z całego świata. Otwartość GitHuba okazała się kluczem zapewniającym dostęp do różnych rynków, inżynierów i produktów.

Co więcej, dzięki GitHubowi użytkownicy mogą nie tylko samodzielnie tworzyć oprogramowanie, ale również korzystać z innych projektów i otwartego kodu źródłowego, by tworzyć nowe pomysły i rozwiązania. To potwierdza, że GitHub służy do przechowywania kodu źródłowego, a zarazem stanowi katalizator dla innowacji i współpracy na skalę globalną.

W rozdziałach od 4. do 8. przedstawiam wszystkie funkcjonalności, których możesz potrzebować do realizacji projektów.

# Podsumowanie

W tym rozdziale rozpoczęła się nasza przygoda, podczas której zobaczysz, w jaki sposób podejście DevOps rewolucjonizuje branżę tworzenia oprogramowania. Wyjaśniłem wagę wrażeń programisty jako strategii pozwalającej zapewnić jak najlepsze środowisko pracy w zespole DevOps. Przybliżyłem również nieco Gita i GitHuba, które stanowią podstawę współpracy w podejściu DevOps.

Wszystko jest ze sobą połączone i istnieją ku temu powody. Potrzeba stosowania podejścia DevOps i korzystania z Gita i GitHuba, waga kultury związanej z osobami, procesami i narzędziami w organizacji — wszystkie te elementy są ze sobą powiązane. Razem tworzą strategię wrażeń programisty, pozwalając programistom w zespołach stosujących DevOps wykonywać pracę w sposób zapewniający produktywność, wpływ i satysfakcję. To, co zrobisz, ostatecznie powróci do Ciebie oraz wzmocni Cię jako programistę.

Podejście DevOps powinno być opracowane w sposób pozwalający na popełnianie błędów w trakcie krótkich cykli tworzenia oprogramowania. Każda pomyłka jest krokiem na drodze do opanowania podejścia DevOps, poprawia Twoje umiejętności, a także stanowi cenny wkład w funkcjonowanie zespołu i organizacji.

Warto więc wykorzystać tę nowo zdobytą wiedzę i przejść do zastosowań praktycznych. W następnym rozdziale poznasz więc w podstawowy sposób pracy z Gitem, czyli systemem kontroli wersji, który stanowi filar kultury pracy w podejściu DevOps oraz współpracy między członkami zespołu.

# Dalsza lektura

- *The Secret Formula to Improve Developer Experience Revealed!*: *https://www.youtube.com/watch?v=mRqoVlhtVzA*
- *DevEx: What Actually Drives Productivity*: *https://queue.acm.org/detail.cfm?id=3595878*
- *Getting Started with GitHub for Startups*: *https://www.youtube.com/watch?v=K5zhNxnrVW8*

# Doskonałość w działaniu — odkryj potencjał DevOps z Git i GitHub!

Nowoczesne zespoły programistyczne łączą podejście DevOps z potokami ciągłej integracji i ciągłego wdrażania. Zasady DevOps i możliwości technologii Git i GitHub pozwalają na radykalne usprawnienie pracy, a także na poprawę współpracy zespołów i wspieranie innowacji. W efekcie zespoły o wiele lepiej sobie radzą z ciągłym usprawnianiem produktów.

Dzięki tej książce dowiesz się, jak korzystać z możliwości platformy GitHub w trakcie transformacji przepływu pracy DevOps. Rozpoczniesz od podstaw technologii Git i od zrozumienia podejścia DevOps, zapoznasz się również z kwestią wrażeń odbieranych przez programistę. W kolejnych rozdziałach znajdziesz informacje o udostępnionych na platformie GitHub funkcjach automatyzacji i współpracy. Nauczysz się też używać funkcjonalności GitHub Copilot do zwiększenia produktywności. Ponadto dowiesz się, jak wyeliminować lukę DevOps, zachować jakość kodu i zaimplementować niezawodne środki bezpieczeństwa. Liczne ćwiczenia pomogą Ci w praktycznym poprawianiu wrażeń programisty, optymalizacji pracy zespołowej i wspieraniu innowacyjności. Szybko się przekonasz, jaki potencjał drzemie w podejściu DevOps!

W książce:

- podstawy technologii Git i GitHub
- DevOps jako siła napędowa automatyzacji
- potoki ciągłej integracji i ciągłego wdrażania (CI/CD)
- użycie usługi GitHub Actions
- pomiary tempa programowania i usprawnienie tego procesu
- GitHub Copilot i poprawa wrażeń programisty

**Yuki Hattori** jest architektem w firmie GitHub. Wcześniej był architektem rozwiązań chmurowych w Microsofcie. Jest zagorzałym orędownikiem ruchu open source w korporacjach. Aktywnie wspiera stosowanie podejścia InnerSource, dzięki któremu zespoły w ramach organizacji chętniej stawiają na innowacyjność.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-1885-6
9 788328 918856
Cena: 69,00 zł
packt