helion.pl

Idź do

- Spis treści
- Przykładowy rozdział

Katalog książek

- Katalog online
- Zamów drukowany katalog

Twój koszyk

- Dodaj do koszyka

Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

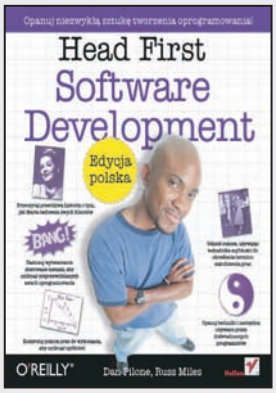
Czytelnia

- Fragmenty książek online

Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl
© Helion 1991-2008

Head First Software Development. Edycja polska



Autor: Dan Pilone, Russ Miles
Tłumaczenie: Tomasz Walczak
ISBN: 978-83-246-1547-6
Tytuł oryginału: Head First Software Development
Format: 200x230, stron: 472

Opanuj niezwykłą sztukę wytwarzania oprogramowania!

- W jaki sposób zadowolić klienta?
- Jak wygląda proces wytwarzania oprogramowania?
- Jakie pułapki czekają na Ciebie?

Proces wytwarzania oprogramowania – już sam opis sugeruje trudności. I rzeczywiście – jest to proces niezwykle złożony. Od samego początku trafiamy na kwestie takie, jak analiza potrzeb klienta i zebranie jego wymagań. Z każdym krokiem wszystko komplikuje się jeszcze bardziej... Konieczna jest implementacja poszczególnych wymagań klienta, testowanie tych rozwiązań, korekta znalezionych błędów. Na to wszystko nakłada się jeszcze konieczność tworzenia różnych wersji rozwiązań i zmienny nastrój klienta. Jak sobie z tym wszystkim poradzić? Jak bezboleśnie i skutecznie przejść przez cały ten proces?

Tylko bez obaw! Oto podręcznik, który dzięki innowacyjnym metodom przekazywania wiedzy sprawi, że szybko zrozumiesz proces wytwarzania oprogramowania i nauczysz się gładko podążać jego wyboistą ścieżką. Autorzy książki „Head First Software Development. Edycja polska" – Dan i Russ – pokażą Ci, w jaki sposób zadowolić klienta i zebrać od niego wymagania oraz określić jego potrzeby. Dowiesz się, jak zapanować nad poszczególnymi wersjami Twojego projektu. Nauczysz się prowadzić testy i usuwać błędy. Zdobędziesz informacje dotyczące wytwarzania oprogramowania sterowanego testami, a na koniec zobaczysz, jak taki proces wygląda w rzeczywistości. Wszystkie te informacje przedstawione zostały na licznych ilustracjach, co wydatnie ułatwia przyswajanie wiedzy, dodatkowo przekazanej przystępnym i pełnym humoru językiem. Po lekturze tego podręcznika nawet laik będzie w stanie zarządzać takim procesem!

- Zbieranie wymagań
- Planowanie projektu
- Kontrola wersji
- Wytwarzanie sterowane testami
- Testy integracyjne
- Usuwanie błędów

Tworzenie oprogramowania? Nic prostszego!!!

Spis treści (skrócony)

Spis treści (szczegółowy)

Wprowadzenie

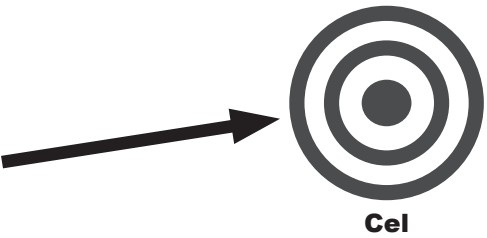
Twój mózg a rozwój oprogramowania. Próbujesz *nauczyć się* czegoś, jednak *mózg* próbuje powiedzieć, że to, czego się uczysz, *nie jest istotne*: „Lepiej umieścić w pamięci ważniejsze informacje, na przykład o dzikich zwierzętach, których należy unikać, i o tym, że wspinanie się po pionowych skałach to zły pomysł". Jak więc możesz przekonać mózg, że Twoje życie naprawdę zależy od nauczenia się rozwijania doskonałego oprogramowania?

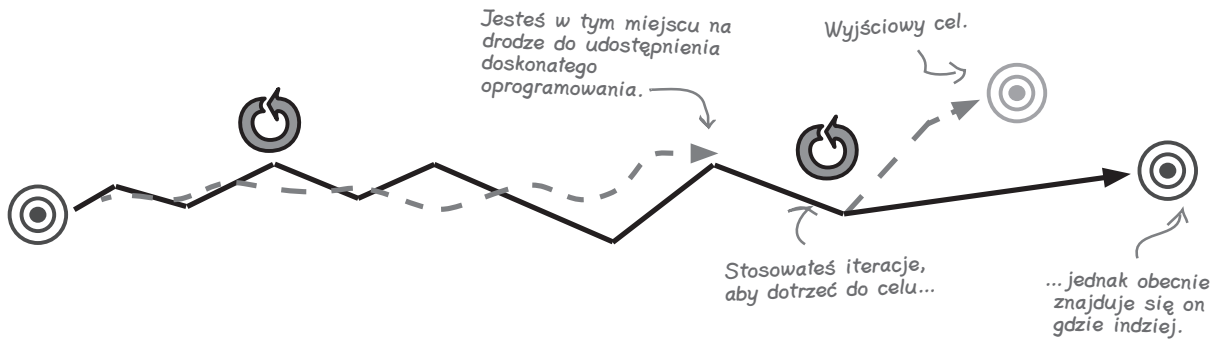
Doskonały rozwój oprogramowania

1 Zapewnianie zadowolenia klienta

Jeśli klient jest niezadowolony, wszyscy są nieszczęśliwi!

Rozwój wartościowego oprogramowania zawsze rozpoczyna się od genialnego pomysłu klienta. Twoim zadaniem — jako specjalisty od rozwoju oprogramowania — jest **realizacja takich pomysłów**. Jednak przekształcenie nieprecyzyjnej idei w działający kod, który ponadto **zadowala klienta**, nie jest proste. W tym rozdziale pokazujemy, jak sprostać wyzwaniom w obszarze rozwoju oprogramowania i udostępniać **potrzebne** programy **na czas** i **po ustalonych kosztach**. Pora złapać laptopy i rozpocząć naukę dostarczania doskonałego oprogramowania.





Zbieranie wymagań

2 Określanie potrzeb klientów

Nie zawsze można dostać to, czego się chce… jednak lepiej zapewnić to klientom!

Doskonały proces rozwoju oprogramowania prowadzi do udostępnienia tego, **czego chcą klienci**. W tym rozdziale opisujemy **komunikowanie się z klientami** w celu określenia ich **wymagań** wobec oprogramowania. Dowiesz się, w jaki sposób **opowieści użytkownika**, **burza mózgów** i **gra planistyczna** pomagają w zrozumieniu odbiorców. Dzięki temu do momentu zakończenia projektu zespół może uzyskać pewność, że stworzył to, czego chcą klienci, a nie tylko marną namiastkę oczekiwanego produktu.

Planowanie projektu

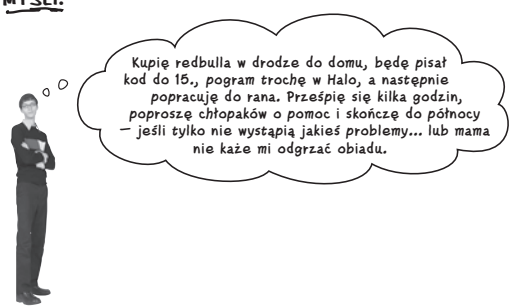
3 Planowanie z myślą o sukcesie

Każde wspaniałe oprogramowanie rozpoczyna się od świetnego planu.

W tym rozdziale pokazujemy, jak stworzyć dobry plan. Dowiesz się, jak współpracować z klientami w celu określenia **priorytetowych wymagań**. Zobaczysz, jak **planować iteracje**, które zespół może następnie realizować. Ponadto nauczysz się tworzyć realistyczny **plan rozwoju**, który zespół może rzetelnie **realizować** i **kontrolować**. Dzięki tym umiejętnościom będziesz umiał łatwo przejść od wymagań do wersji 1.0.

Oto, co programista **MÓWI**...

...a to, co tak naprawdę **MYŚLI**:

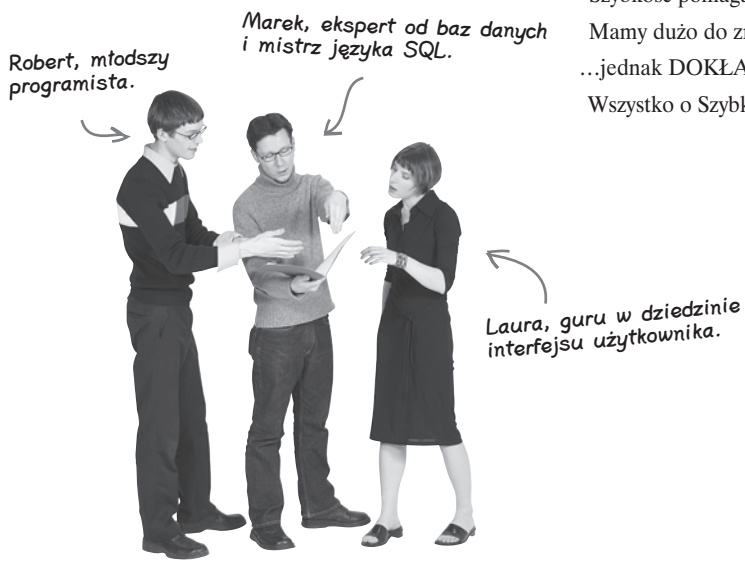


Opowieści użytkownika i zadania

4 Przystępowanie do prawdziwej pracy

Pora zabrać się do pracy. Opowieści użytkownika opisują, co programiści mają stworzyć. Następnie trzeba zakasać rękawy i **ustalić, co trzeba zrobić** w celu realizacji tych opowieści. W tym rozdziale dowiesz się, jak **podzielić opowieści użytkownika na zadania**, a także w jaki sposób **szacowanie czasu wykonania zadań** pomaga w kontrolowaniu projektu od początku do końca. Nauczysz się aktualizować tablice, realizować zadania będące w toku oraz **w pełni kończyć opowieści użytkowników**. Zobaczysz też, jak realizować nieuniknione **nieoczekiwane zadania** dodawane przez klientów i określać priorytety takich prac.

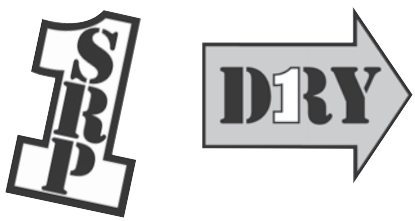
Pierwsze krótkie spotkanie robocze.

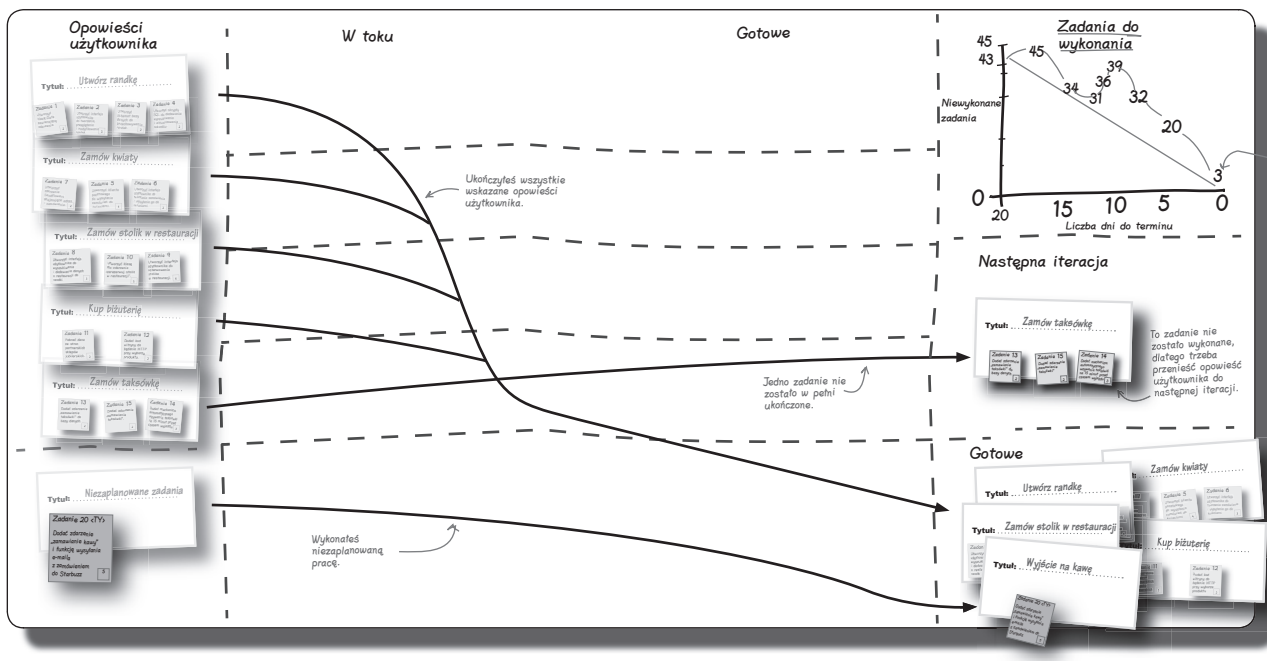


Wystarczająco dobry projekt

5 Tworzenie oprogramowania na podstawie doskonałych projektów

Dobry projekt pomaga realizować zadania. W poprzednim rozdziale sytuacja wyglądała niekorzystnie. **Zły projekt utrudniał wszystkim** pracę, a dodatkowych problemów przysparzały niezaplanowane zadania. W tym rozdziale pokazujemy, jak **refaktoryzować** projekt w celu **zwiększenia produktywności** zespołu. Dowiesz się, jak stosować **zasady dobrego projektowania**, a przy tym ostrożnie podchodzić do tworzenia ***„doskonałego projektu"***. Zobaczysz też, jak realizować **niezaplanowane zadania** w dokładnie taki sam sposób, jak wszystkie inne prace w projekcie, używając do tego dużej tablicy projektowej wiszącej na ścianie.



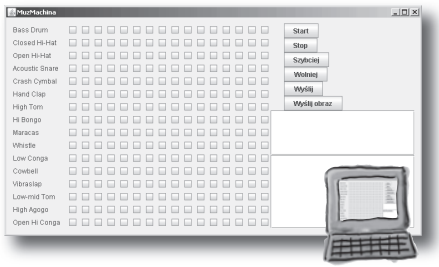


Kontrola wersji

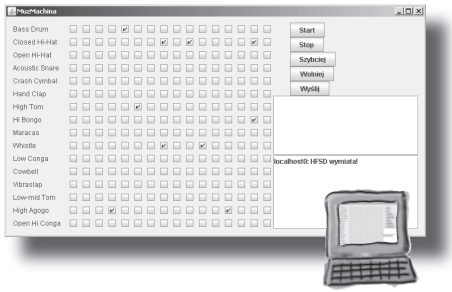
6 Programowanie defensywne

Przy rozwoju doskonałego oprogramowania na pierwszym miejscu trzeba stawiać bezpieczeństwo!

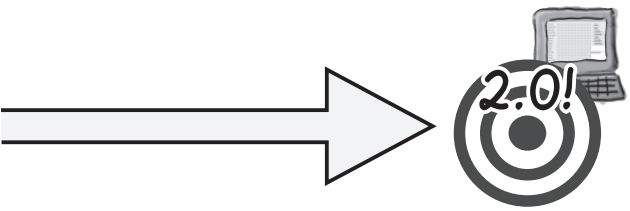
Tworzenie doskonałego oprogramowania nie jest proste... zwłaszcza kiedy trzeba sprawdzić, czy kod działa, a także **upewniać się, że wciąż funkcjonuje poprawnie**. Wystarczy literówka, zła decyzja współpracownika lub zepsuty dysk twardy, żeby całe rozwiązanie nagle przestało funkcjonować. Jednak dzięki **kontroli wersji** można upewnić się, że **kod jest zawsze bezpieczny** w repozytorium, a także **anulować błędy** oraz **poprawiać usterki** w nowych i starszych wersjach oprogramowania.



MuzMachina Pr1.0



MuzMachina Pr1.x

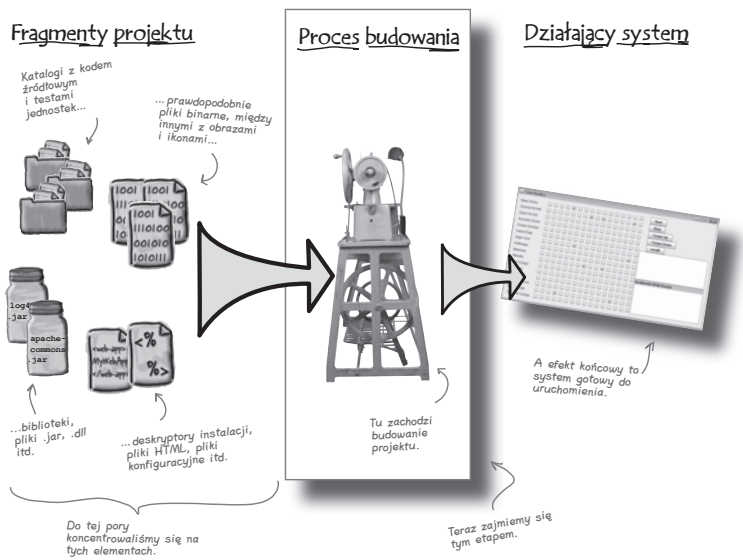


Kompilowanie kodu

6½ Wstaw element a w pole b...

Opłaca się przestrzegać instrukcji…
…zwłaszcza jeśli samemu się ją napisało.

Zarządzanie konfiguracją nie gwarantuje bezpieczeństwa kodu. Trzeba także uwzględnić **kompilację kodu** i tworzenie pakietów umożliwiających instalację programu. Ponadto trzeba zdecydować, która klasa powinna być klasą główną aplikacji i jak należy uruchomić taką klasę. W tym rozdziale dowiesz się, jak za pomocą **narzędzi do wspomagania kompilacji pisać własne instrukcje** do obsługi kodu źródłowego.

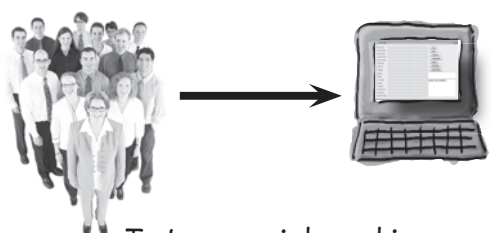


Testy i ciągła integracja

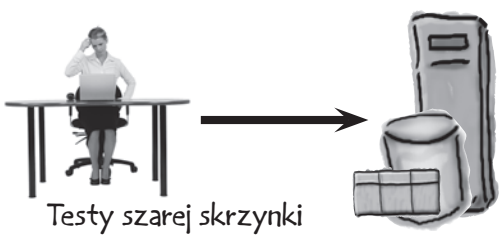
7 Pojawiają się problemy

Czasem nawet najlepszemu programiście zdarzy się naruszyć poprawność kompilacji.

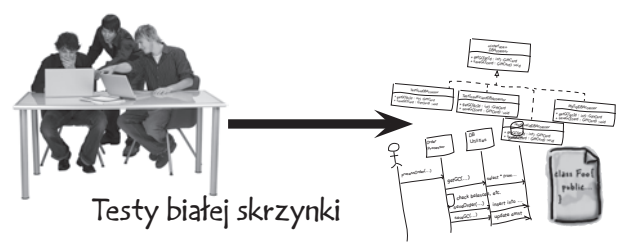
Przydarzyło się to z pewnością niemal każdemu. Jesteś pewien, że **Twój kod się kompiluje**, przetestowałeś go wielokrotnie na własnej maszynie, a następnie przesłałeś do repozytorium. Jednak gdzieś między maszyną, której używasz, a czarną skrzynką zwaną serwerem *ktoś* musiał zmodyfikować kod. Nieszczęśnik, który jako następny pobierze z repozytorium kod, będzie miał dużo pracy przy przywracaniu **działającej niegdyś jego wersji**. W tym rozdziale pokazujemy, jak przygotować **siatkę zabezpieczającą**, która zapewnia działanie kompilacji i **produktywność zespołu**.



Testy czarnej skrzynki



Testy szarej skrzynki

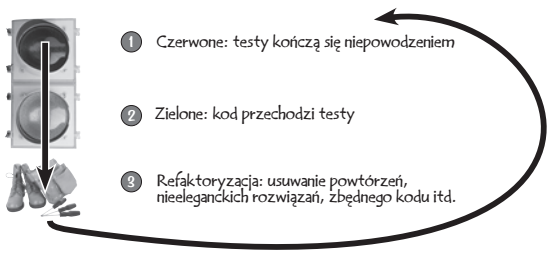


Testy białej skrzynki

Wytwarzanie sterowane testami

8 Zapewnianie poprawności kodu

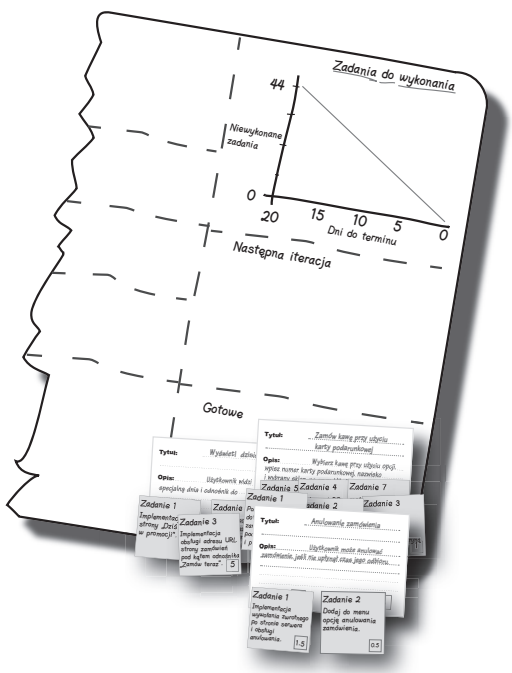
Czasem najważniejsze jest ustalenie oczekiwań. Dobry kod musi działać — każdy to wie. Skąd jednak **wiadomo, że kod działa**? Nawet przy zastosowaniu testów jednostkowych znaczne fragmenty kodu nie są sprawdzane. A gdyby potraktować testy jako **podstawowy element rozwoju oprogramowania**? Gdyby ***wszystkie*** zadania wykonywać z myślą o testach? W tym rozdziale wykorzystamy wiedzę na temat kontroli wersji, ciągłej integracji i testów automatycznych do przygotowania środowiska, które pozwala **bezpiecznie naprawiać usterki**, **refaktoryzować kod**, a nawet **ponownie implementować** fragmenty systemu.



Końcówka iteracji

9 Wszystkie elementy łączą się ze sobą...

Aplikacja jest już prawie gotowa! Zespół pracował ciężko i pozostało tylko dopracowanie rozwiązania. Zadania i opowieści użytkownika są już **zrealizowane**. Jak można najlepiej wykorzystać dodatkowy dzień? Kiedy przeprowadzić **testy z udziałem użytkowników**? Czy można wygospodarować czas na jeszcze jeden cykl **refaktoryzacji** lub **wprowadzenie zmian w projekcie**? Z pewnością w kodzie kryje się wiele **błędów** — kiedy należy je naprawić? Wszystkie te zadania to elementy **końcówki iteracji**, dlatego zacznijmy przybliżać się do końca.



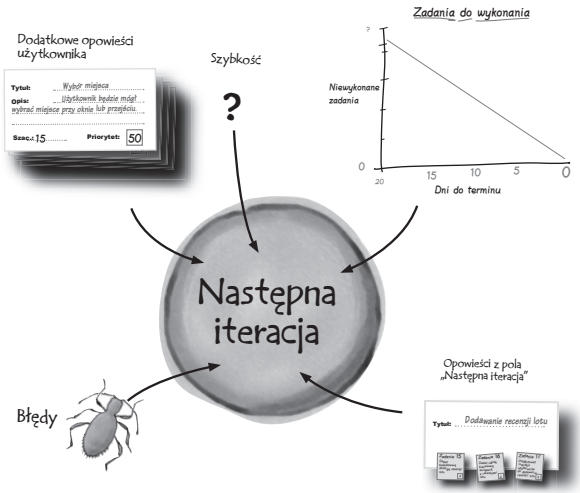
Następna iteracja

10 Jeśli nie jest zepsute... i tak lepiej to naprawić

Uważasz, że wszystko idzie dobrze?

Zachowaj czujność, bo może się to zmienić…

Iteracja poszła doskonale i na czas udostępniłeś działające oprogramowanie. Czas na następną iteracja? Bez problemu, prawda? Niestety, wprost przeciwnie. Rozwój oprogramowania wiąże się z ciągłymi **zmianami**, a dotyczy to także **przechodzenia do następnej iteracji**. W tym rozdziale pokazujemy, jak przygotować zespół do ***następnej*** iteracji. Trzeba **zmodyfikować tablicę** i **dostosować opowieści** oraz oczekiwania do tego, czego klienci żądają ***TERAZ***, a nie do ich wymagań sprzed miesiąca.

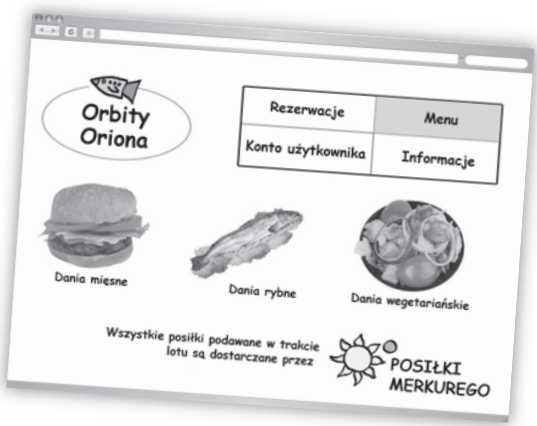


Błędy

11 Profesjonalne usuwanie błędów

Twój kod — Twoja odpowiedzialność. Twój błąd — Twoja reputacja!

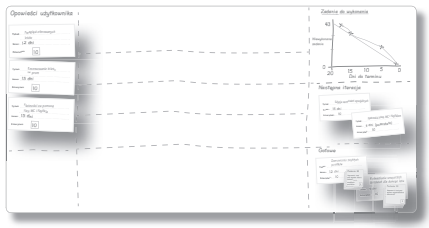
Kiedy pojawiają się problemy, to **Ty odpowiadasz** za ich rozwiązanie. **Błędy**, zarówno w kodzie napisanym przez Ciebie, jak i używanym przez rozwijane oprogramowanie, są **naturalnym zjawiskiem** w rozwoju oprogramowania. Sposób naprawiania usterek, podobnie jak wszystkie inne operacje, powinien pasować do całego procesu. Trzeba **przygotować tablicę**, **włączyć w prace klientów**, **wiarygodnie oszacować** czas potrzebny na wprowadzenie poprawek oraz zastosować **refaktoryzację** i **wstępną refaktoryzację** w celu naprawienia błędów oraz zapobieżenia występowaniu ich w przyszłości.



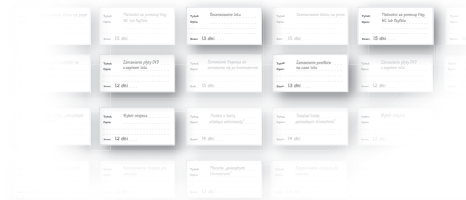
Rzeczywisty świat

12 Proces w praktyce

Dowiedziałeś się wielu rzeczy na temat rozwoju oprogramowania. Jednak zanim zaczniesz umieszczać diagramy zadań do wykonania w każdym pomieszczeniu, musisz nauczyć się uwzględniać specyficzne aspekty poszczególnych projektów. Istnieje wiele **podobnych rozwiązań** i **najlepszych praktyk**, które należy stosować w każdym projekcie, jednak zawsze pojawiają się też **wyjątkow**e elementy i trzeba się na to przygotować. Pora zobaczyć, jak zastosować zdobytą wiedzę w **konkretnym projekcie**, a także dowiedzieć się, gdzie szukać **dalszych informacji**.



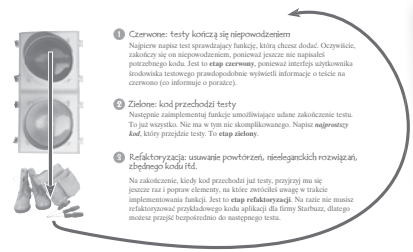
Duża tablica



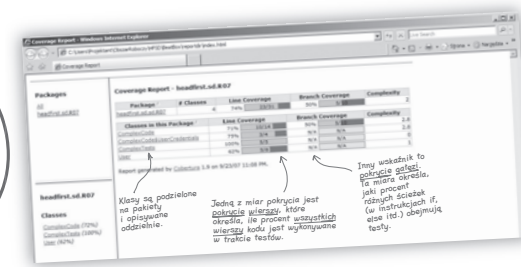
Opowieści użytkownika

Kontrola wersji

Ciągła integracja (CI)



Wytwarzanie sterowane testami (TDD)



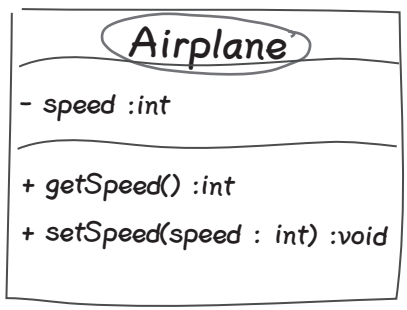
Pokrycie kodu testami

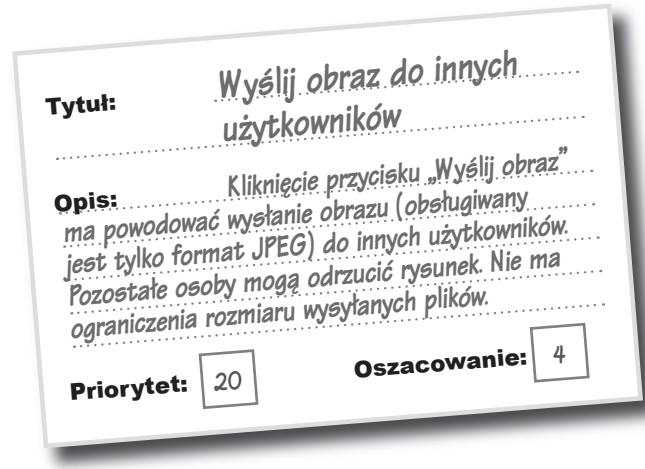
Dodatek A Pozostałości

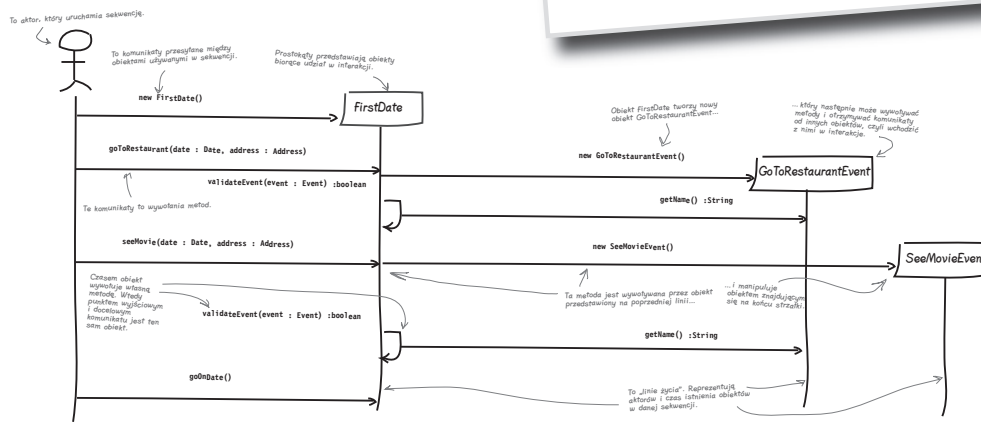
A Pięć najważniejszych tematów (których nie poruszyliśmy)

Czy miałeś kiedykolwiek wrażenie, że o czymś zapomniałeś? Wiemy, co to znaczy…

Kiedy się wydaje, że praca została ukończona, okazuje się, że pozostało jeszcze kilka zadań do wykonania. Nie możemy pominąć kilku dodatkowych zagadnień, których nie potrafiliśmy uwzględnić w pozostałych częściach książki (a przynajmniej nie bez wydłużania jej do tego stopnia, że do przewożenia tomu potrzebny byłby metalowy wózek na kółkach). Dlatego zobacz, z czym (jeszcze) się nie zapoznałeś.







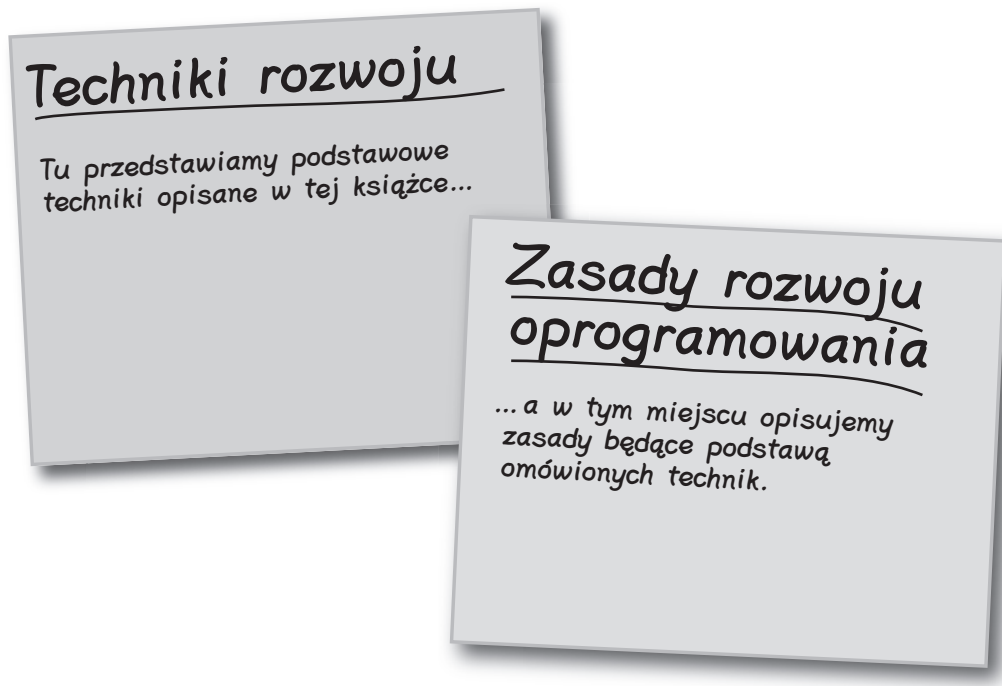
Dodatek B Techniki i zasady

B

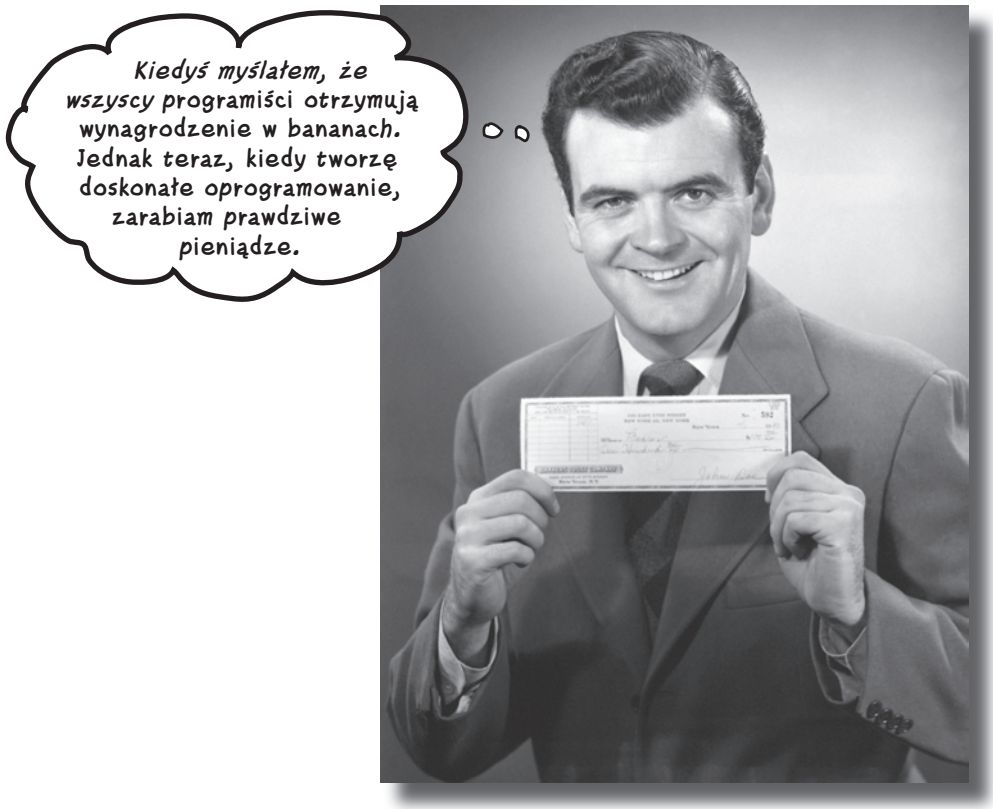
Narzędzia dla doświadczonych programistów

Czy kiedykolwiek chciałeś, aby cały zestaw doskonałych narzędzi i technik został opisany w jednym miejscu?

W tym dodatku przedstawiamy przegląd wszystkich omówionych w książce **technik** i **zasad** rozwoju oprogramowania. Przyjrzyj się im i sprawdź, czy **zapamiętałeś ich znaczenie**. Możliwe, że zechcesz **wyciąć te strony** i przyczepić je do dolnej części **dużej tablicy**, aby wszyscy mogli je przejrzeć w trakcie codziennych krótkich spotkań roboczych.



Zapewnianie zadowolenia klientów



Jeśli klient jest niezadowolony, *wszyscy* są nieszczęśliwi! Rozwój wartościowego oprogramowania zawsze rozpoczyna się od genialnego pomysłu klienta. Twoim zadaniem — jako specjalisty od rozwoju oprogramowania — jest **realizacja takich pomysłów**. Jednak przekształcenie nieprecyzyjnej idei w działający kod, który ponadto **zadowala klienta**, nie jest proste. W tym rozdziale pokazujemy, jak sprostać wyzwaniom w obszarze rozwoju oprogramowania i udostępniać **potrzebne** programy **na czas** i **po ustalonych kosztach**. Pora złapać laptopy i rozpocząć naukę dostarczania doskonałego oprogramowania.

Szlakami Macieja wchodzi do internetu

Maciej Wędrownik od lat oferuje znane na całym świecie wycieczki z przewodnikiem i sprzęt w swej tatrzańskiej chacie. Teraz chce zwiększyć sprzedaż, wykorzystując nowe technologie.

Maciek chce wprowadzić firmę do internetu.

Jestem najlepszym przewodnikiem po szlakach górskich na świecie. Jednak zbliża się ważna konferencja Drogi i bezdroża, na której chcę wszystkim pokazać następny etap ewolucji pieszych wędrówek, możliwy dzięki wykorzystaniu internetu.

Wersja witryny Szlakami Macieja zaproponowana przez Maćka

W większości projektów trzeba uwzględnić dwa główne zagadnienia

W trakcie rozmów klienci — obok omawiania swych pomysłów — zwracają zwykle uwagę na dwie podstawowe kwestie:

Ile będzie kosztować oprogramowanie?

Nie jest to zaskoczeniem. Większość klientów chce wiedzieć, ile będą musieli zapłacić za produkt. Jednak Maciej ma mnóstwo pieniędzy, dlatego cena programu nie stanowi dla niego problemu.

Zwykle pieniądze są ograniczeniem. Jednak Maciej ma ich mnóstwo i uważa, że poniesione nakłady wzbogacą go jeszcze bardziej.

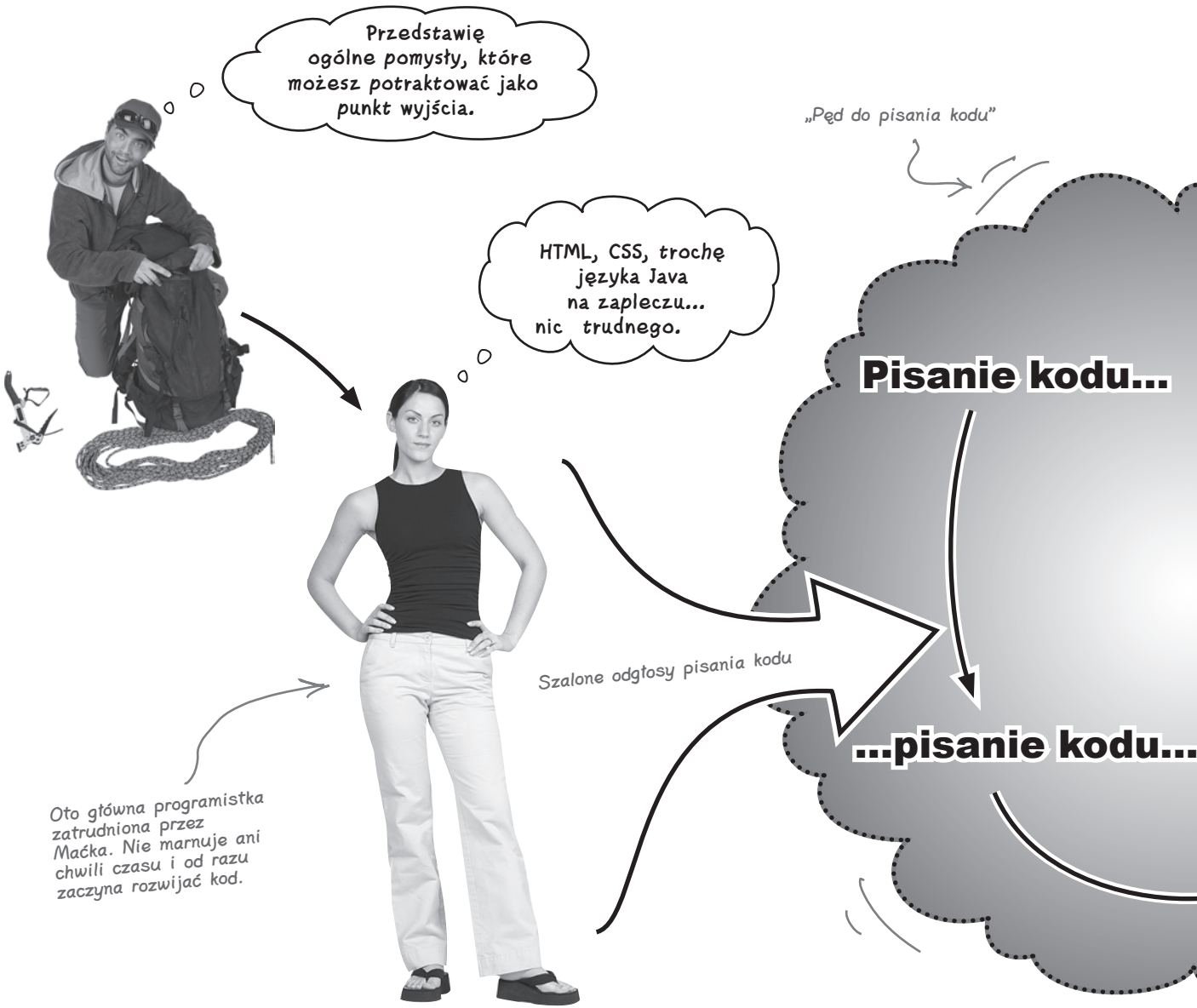
Ile czasu zajmie rozwój produktu?

Następnym poważnym ograniczeniem jest czas. Klienci prawie nigdy nie mówią: „Poświęćcie na rozwój tyle czasu, ile potrzebujecie". Zazwyczaj odbiorca chce, aby zespół przygotował oprogramowanie na konkretne wydarzenie lub do ustalonego dnia.

Maciej chce, aby witryna była gotowa za trzy miesiące, na ważną konferencję Drogi i bezdroża.

Rozwój oprogramowania metodą wielkiego wybuchu

Ponieważ do zakończenia pracy pozostał tylko miesiąc, nie wolno marnować czasu. Główny programista zatrudniony przez Maćka zabiera się do pracy



Przenieśmy się w czasie — dwa tygodnie później

Główna programistka zaangażowana przez Maćka pokonała wszystkie przeszkody na drodze do utworzenia witryny Szlakami Macieja i wykorzystała wszystkie swe umiejętności programistyczne do przygotowania produktu zgodnego — jej zdaniem — z oczekiwaniami klienta.

Wielki wybuch: po ciężkiej pracy słychać głośne BUM! i nagle pojawia się coś wielkiego oraz skomplikowanego.

To podejście jest też nazywane „zaćmieniem", ponieważ programista kontaktuje się z klientem na początku projektu, a następnie znika do momentu ukończenia oprogramowania.

Rozwój oprogramowania metodą wielkiego wybuchu kończy się zwykle WIELKIMI PROBLEMAMI

Choć programiści włożyli w projekt dużo pracy, Maciek jeszcze nie widział ukończonego (miejmy nadzieję) oprogramowania. Zobaczmy, co myśli o przygotowanej witrynie.

Co to jest? Witryna nie wygląda tak, jak tego oczekiwałem. Czy nie możecie poświęcić trochę więcej czasu na jej prawidłowe zaprojektowanie? Ta wersja wygląda tak, jakbyście w ogóle nie wiedzieli, na czym mi zależy.

Jeśli klient jest niezadowolony, oznacza to, że dostarczyłeś nieodpowiednie oprogramowanie.

Rozwój oprogramowania metodą wielkiego wybuchu wymaga wiele pracy, przy czym zespół nie pokazuje klientom programu do momentu jego ukończenia. Minusem tego podejścia jest to, że możesz **sądzić**, iż rozwijasz produkt zgodnie z oczekiwaniami odbiorcy, lecz nie otrzymujesz żadnych informacji zwrotnych do momentu, kiedy **uważasz**, że ukończyłeś pracę.

Najważniejsze jest tu to, że myślisz, iż ukończyłeś produkt, podczas gdy może być inaczej.

Niezależnie od tego, jak wysoko TY SAM oceniasz oprogramowanie, to klient musi być zadowolony. Dlatego jeśli odbiorcy nie podoba się przygotowany produkt, nie należy tracić czasu na przekonywanie klienta do zalet oprogramowania. Trzeba po prostu przygotować się do wprowadzenia zmian.

Jak jednak odkryć, czego klient naprawdę oczekuje? Nie zawsze jest to proste…

Zaostrz ołówek

Czy potrafisz zauważyć, w którym miejscu popełniono błąd? Maciek wymienił trzy przedstawione poniżej operacje, których wykonywanie miała umożliwiać witryna. Twoim zadaniem jest wybór opcji najbardziej zbliżonej do oczekiwań klienta. Przy trzeciej operacji sam musisz ustalić, co Maciek miał na myśli. Powodzenia!

1. Maciek powiedział: „Klient powinien mieć możliwość wyszukiwania szlaków".

- ☐ Klient powinien widzieć mapę świata i móc wpisać adres, aby znaleźć szlaki w pobliżu określonej lokalizacji.
- ☐ Klient powinien mieć możliwość przewijania listy miejscowości turystycznych i wyszukiwania szlaków, które prowadzą do tych punktów i wychodzą z nich.
- ☐ Klient powinien móc wyszukiwać szlaki w obszarze określonego kodu pocztowego i o podanym stopniu trudności.

2. Maciek powiedział: „Klient powinien mieć możliwość zamówienia wyposażenia".

- ☐ Klient powinien móc przejrzeć dostępne u Maćka wyposażenie, a następnie zamówić produkty znajdujące się w magazynie.
- ☐ Klient powinien móc zamówić dowolne potrzebne wyposażenie, jednak jeśli produkt jest niedostępny, obsługa zamówienia może potrwać dłużej, ponieważ Maciek musi dopiero sprowadzić odpowiedni sprzęt.

3. Maciek powiedział: „Klient powinien móc zarezerwować wycieczkę".

Napisz, jak oprogramowanie powinno działać TWOIM zdaniem.

.....

.....

.....

Nie jesteś pewien, co Maciek miał na myśli? Nie przejmuj się. Postaraj się zgadnąć jak najtrafniej.

Problemy z wyborem opcji są czymś całkowicie normalnym. Postaraj się jak najlepiej wykonać to zadanie, a w dalszej części rozdziału dowiesz się, jak ustalić znaczenie słów klienta.

Zaostrz ołówek

Rozwiązanie

Czy potrafisz ustalić, gdzie popełniono błąd? Twoim zadaniem był wybór opcji najbardziej zbliżonych do znaczenia poszczególnych wypowiedzi Maćka. Przy trzecim stwierdzeniu musiałeś sam ustalić jego sens.

?

Wielki znak zapytania? To ma być odpowiedź? Jak mam przygotować doskonałe oprogramowanie, jeśli nawet nie wiem, czego oczekuje klient?

Jeśli nie jesteś pewien, czego oczekuje klient, a nawet wtedy, kiedy sądzisz, że to wiesz, zawsze powinieneś zapytać o zdanie odbiorcę

Przy ustalaniu wymagań to klient jest najważniejszą osobą. Jednak naprawdę rzadko zdarza się, że odbiorca już na początku prac na projektem potrafi ***dokładnie*** wyrazić swe oczekiwania.

W momencie, kiedy próbujesz zrozumieć oczekiwania klienta, czasem nawet on sam nie zna jeszcze wszystkich odpowiedzi, a co dopiero programista! Jeśli znikniesz w pośpiechu i od razu zaczniesz pisać kod, możesz znać tylko połowę wymagań… lub jeszcze mniej.

Rozwój oprogramowania nie powinien opierać się na zgadywaniu. Musisz się upewnić, że rozwijasz doskonałe oprogramowanie nawet wtedy, kiedy na początku prac potrzeby nie są jeszcze w pełni znane. Dlatego powinieneś ***zapytać*** klienta o znaczenie jego słów. ***Zapytać*** o dalsze szczegóły. ***Zapytać*** o możliwe sposoby realizacji jego wielkich pomysłów.

Rozwój oprogramowania NIE polega na zgadywaniu. Musisz włączyć klienta w prace, aby upewnić się, że zmierzasz we właściwym kierunku.

Doskonały rozwój oprogramowania polega na...

Wymieniliśmy kilka warunków, które musi spełniać udane oprogramowanie. Trzeba uwzględnić wielkie pomysły klientów, ich pieniądze oraz harmonogram. Jeśli chcesz stale rozwijać doskonałe oprogramowanie, musisz spełnić wszystkie te warunki.

Doskonały rozwój oprogramowania prowadzi do dostarczenia...

To, czego potrzebuje klient. Są to wymagania wobec oprogramowania. Więcej o wymaganiach piszemy w następnym rozdziale.

Tego co potrzebne,

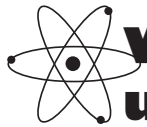
Uzgodniony z klientem czas ukończenia oprogramowania.

na czas

i

po ustalonych kosztach

Należy obciążać klienta tylko uzgodnionymi kosztami.



Wytęż umysł

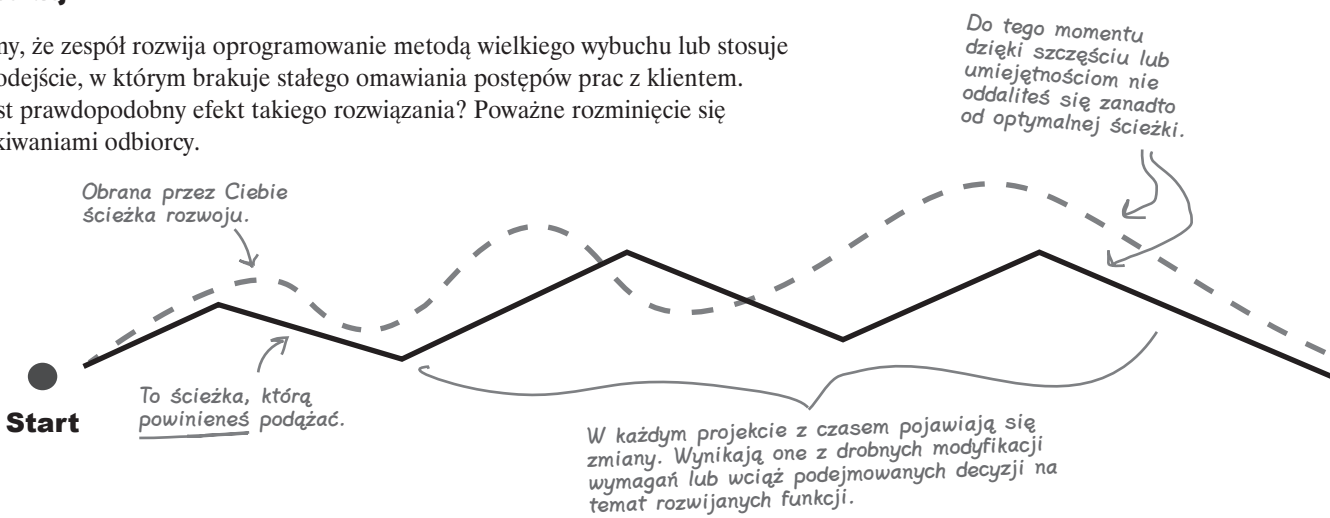
Czy potrafisz przypomnieć sobie trzy projekty rozwoju oprogramowania, w których naruszono przynajmniej jedną z tych reguł?

Dochodzenie do celu dzięki ITERACJOM

Kluczem do doskonałego rozwoju oprogramowania są **iteracje**. Wiesz już, że nie można ignorować klienta w trakcie pisania programu. Iteracje umożliwiają zadanie na każdym etapie rozwoju pytania: „Jak mi idzie?”. Poniżej ilustrujemy przebieg dwóch projektów. W jednym z nich zastosowano iteracje, w drugim — nie.

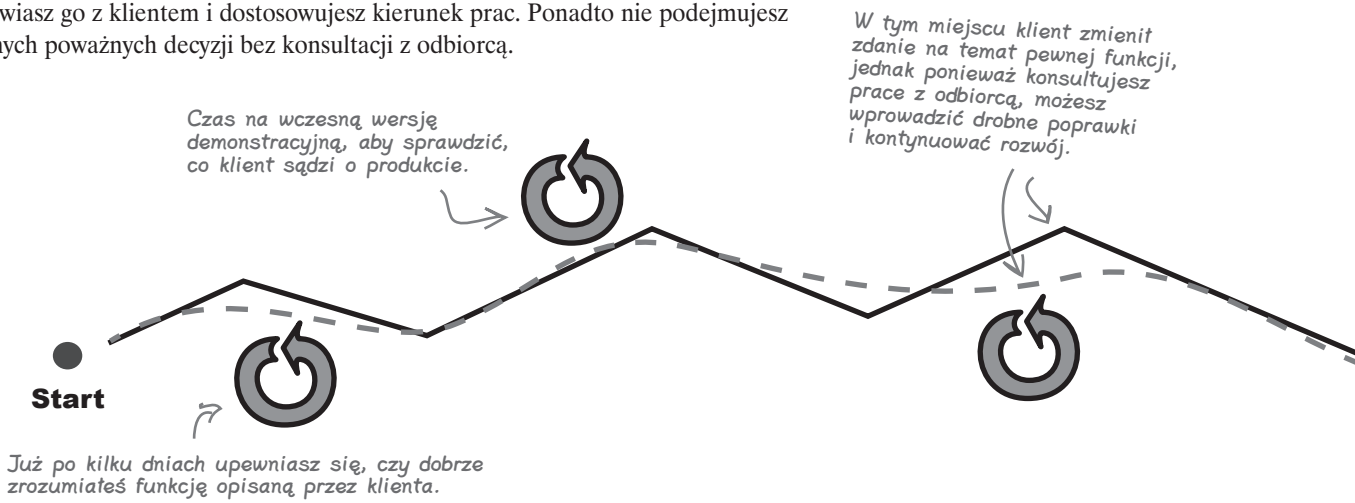
Bez iteracji

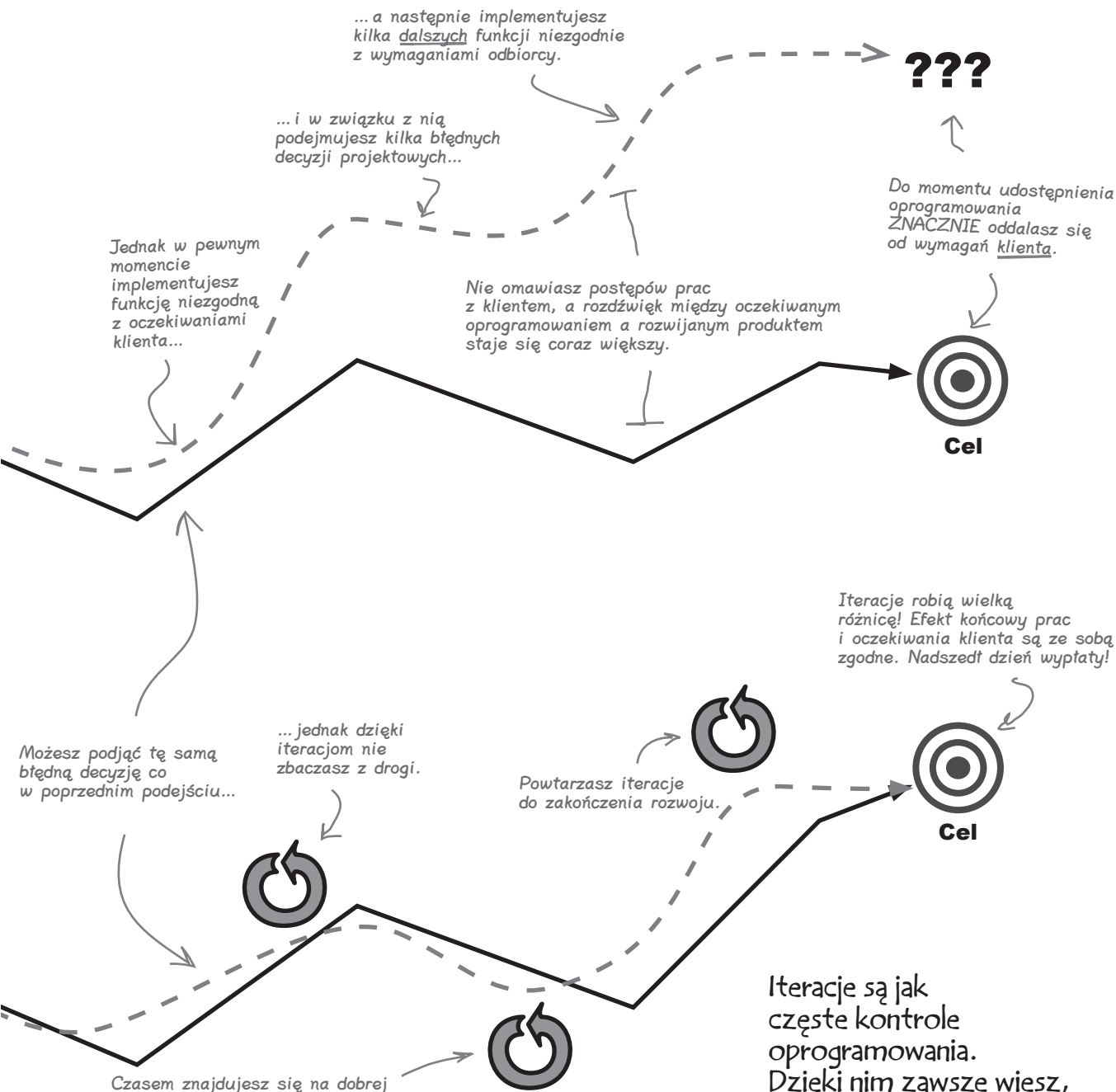
Załóżmy, że zespół rozwija oprogramowanie metodą wielkiego wybuchu lub stosuje inne podejście, w którym brakuje stałego omawiania postępów prac z klientem. Jaki jest prawdopodobny efekt takiego rozwiązania? Poważne rozminięcie się z oczekiwaniami odbiorcy.



Z iteracjami

W tym podejściu po ukończeniu każdego większego fragmentu oprogramowania omawiasz go z klientem i dostosowujesz kierunek prac. Ponadto nie podejmujesz żadnych poważnych decyzji bez konsultacji z odbiorcą.





Iteracje są jak częste kontrole oprogramowania. Dzięki nim zawsze wiesz, jak postępują prace.

Nie ma niemądrych pytań

P: Co zrobić, jeśli już na samym początku projektu jestem pewien, jakie są oczekiwania klienta? Czy mimo to powinienem stosować iteracje?

O: Oczywiście. Iteracje i informacje zwrotne od klienta są szczególnie istotne, jeśli sądzisz, że znasz wszystkie wymagania. Czasem wydaje się, że rozwój prostego fragmentu programu nie wymaga zastanowienia, jednak ZAWSZE warto porozmawiać na ten temat z odbiorcą. Nawet jeśli klient powie, że świetnie sobie radzisz, a wszystkie wymagania są znane od początku, iteracje pozwalają upewnić się, że prace toczą się w dobrym kierunku. Warto też pamiętać, że odbiorca może w każdym momencie zmienić zdanie.

P: Cały rozwój zajmie tylko dwa miesiące. Czy warto stosować iteracje w tak krótkim projekcie?

O: Tak, iteracje są przydatne także w bardzo krótkich projektach. Dwa miesiące to całe 60 dni, w ciągu których zespół może zboczyć z optymalnej drogi rozwoju lub mylnie zrozumieć wymagania klienta. Iteracje pozwalają wykryć wszelkie potencjalne problemy, zanim ujawnią się w projekcie i — co ważniejsze — zanim skompromitujesz się przed klientem.

Niezależnie od wielkości zespołu i czasu trwania projektu iteracje są ZAWSZE jednym z kluczy do rozwoju doskonałego oprogramowania.

P: Czy nie lepiej poświęcić więcej czasu na ścisłe określenie wymagań i ustalenie, czego naprawdę oczekuje klient, niż pozwalać odbiorcom na zmianę zdania w trakcie trwania prac?

O: Na pozór może się tak wydawać, jednak jest to prosta droga do katastrofy. W dawnych, trudnych czasach programiści poświęcali na początku projektu wiele czasu na upewnienie się, że zebrali wszystkie wymagania klienta. Bez ich ustalenia nie myśleli nawet o pisaniu kodu lub podejmowaniu decyzji projektowych.

Niestety, to podejście nie przyniosło sukcesów. Nawet jeśli już na początku prac sądzisz, że w pełni rozumiesz oczekiwania klienta, sam odbiorca często ich nie rozumie. Dlatego nie tylko programiści muszą ustalić potrzeby klientów, ale też oni sami.

Potrzebna jest metoda, która pomoże zespołowi i klientom lepiej zrozumieć oprogramowanie wraz z jego rozwojem. Nie jest to możliwe przy stosowaniu strategii wielkiego wybuchu, w której wymagania są ustalane z góry, a programiści oczekują, że od momentu rozpoczęcia prac potrzeby nie zmienią się.

P: Kto powinien brać udział w iteracjach?

O: Wszystkie osoby, które oceniają, czy oprogramowanie jest zgodne z wymaganiami, a także pracownicy odpowiedzialni za spełnianie tych wymagań. Zwykle jest to przynajmniej klient, Ty sam i inni programiści pracujący nad projektem.

P: Mój zespół składa się tylko ze mnie. Czy mimo to muszę stosować iteracje?

O: Dobre pytanie. Tak, nawet wtedy powinieneś stosować iteracje (czy zaczynasz zauważać regułę?). Nawet jeśli pracujesz sam, powodzenie projektu zawsze zależy od przynajmniej dwóch osób: klienta i Ciebie. Trzeba uwzględnić co najmniej dwa punkty widzenia, aby upewnić się, że oprogramowanie jest rozwijane prawidłowo, dlatego iteracje są pomocne także w najmniejszym możliwym zespole.

P: Na jak wczesnym etapie projektu należy zacząć stosowanie iteracji?

O: Wtedy, kiedy gotowy jest działający fragment oprogramowania, który można omówić z klientem. Jako praktyczną regułę proponujemy stosować iteracje obejmujące około 20 dni pracy (czyli miesiąc kalendarzowy), jednak, oczywiście, można przeprowadzać je częściej. Zespoły nierzadko stosują iteracje jedno- lub dwutygodniowe. Jeśli drugiego dnia pracy nie jesteś pewien, co klient miał na myśli, powinieneś do niego zadzwonić. Czekanie do następnej iteracji lub zgadywanie oczekiwań odbiorcy nie są najlepszymi rozwiązaniami.

P: Co się stanie, kiedy klient stwierdzi, że rozwijane oprogramowanie jest niezgodne z wymaganiami? Co powinienem wtedy zrobić?

O: Doskonałe pytanie! Kiedy zdarzy się najgorsze i znacznie zejdziesz z optymalnej drogi rozwoju, musisz powrócić na nią w ciągu kilku następnych iteracji. W dalszej części książki pokazujemy, jak to zrobić, a jeśli chcesz dowiedzieć się tego od razu, zajrzyj do rozdziału 4.

W porządku, rozumiem, iteracje są ważne. Jednak stwierdziliście, że powinnam przeprowadzać iterację za każdym razem, kiedy utworzę działające oprogramowanie, czyli mniej więcej co 30 dni kalendarzowych lub co 20 dni roboczych. Co zrobić, jeśli po miesiącu pracy nie przygotuję działającego programu? Co mam pokazać klientowi?

20 dni roboczych to tylko wskazówka. W konkretnym projekcie możesz wydłużyć lub skrócić ten czas.

Iteracja prowadzi do utworzenia działającego oprogramowania

W dawnym podejściu do rozwoju oprogramowania metodą wielkiego wybuchu programiści zwykle nie mieli gotowego oprogramowania do momentu ukończenia projektu, a jest to najgorszy moment na zorientowanie się, że prace potoczyły się w złym kierunku.

Przy stosowaniu iteracji zespół na każdym etapie sprawdza, czy znajduje się na dobrej drodze. Wymaga to zapewnienia kompilacji oprogramowania niemal od pierwszego dnia pracy (a jeszcze lepiej od pierwszej godziny, jeśli jest to możliwe). Trzeba unikać dłuższych okresów, kiedy kod nie działa lub się nie kompiluje, nawet jeśli obejmuje tylko nieliczne funkcje.

Ciągłą kompilację i testy opisujemy w rozdziałach 6. i 7.

Sprawna kompilacja znacznie zwiększa wydajność zespołu, ponieważ dzięki niej programiści nie muszą poświęcać czasu na poprawianie kodu napisanego przez inne osoby przed przystąpieniem do wykonywania swych zadań.

Następnie należy pokazać klientom przygotowane funkcje. Czasem jest ich niewiele, jednak nawet wtedy klient może potwierdzić ich przydatność.

To bardzo prosty fragment witryny Szlakami Macieja. Obejmuje tylko nawigację, jednak warto poprosić Maćka o jej ocenę.

Zamiast tworzyć od razu całą witrynę, należy podzielić pracę na mniejsze fragmenty. Następnie można prezentować klientowi poszczególne zbiory funkcji.

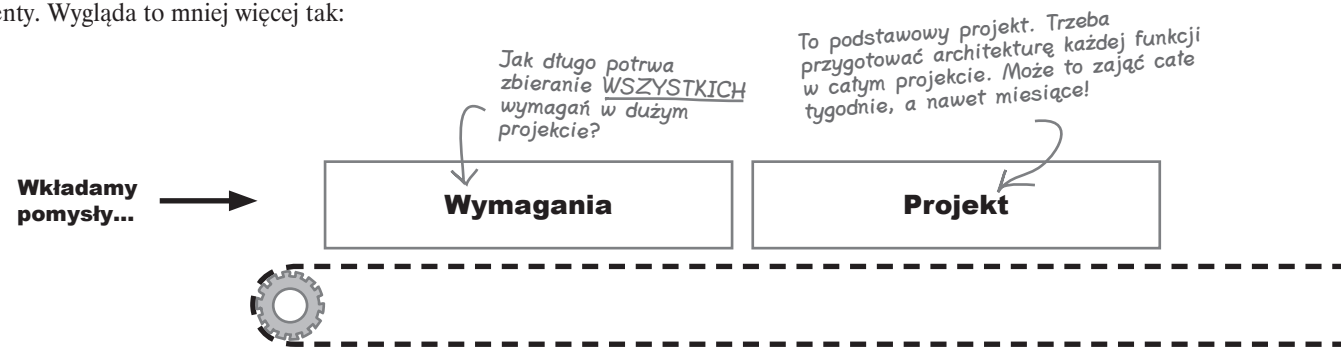
Maciek zapoznał się z działającym oprogramowaniem i przedstawił kilka ważnych sugestii, które zespół może natychmiast uwzględnić.

Wygląda nieźle. Czy można dodać zaokrąglone narożniki? Ponadto wolałbym etykietę „Napisz do nas" zamiast „Kontakt". Ostatnia rzecz: czy można dodać opcję „Stan zamówienia"?

Każda iteracja to miniprojekt

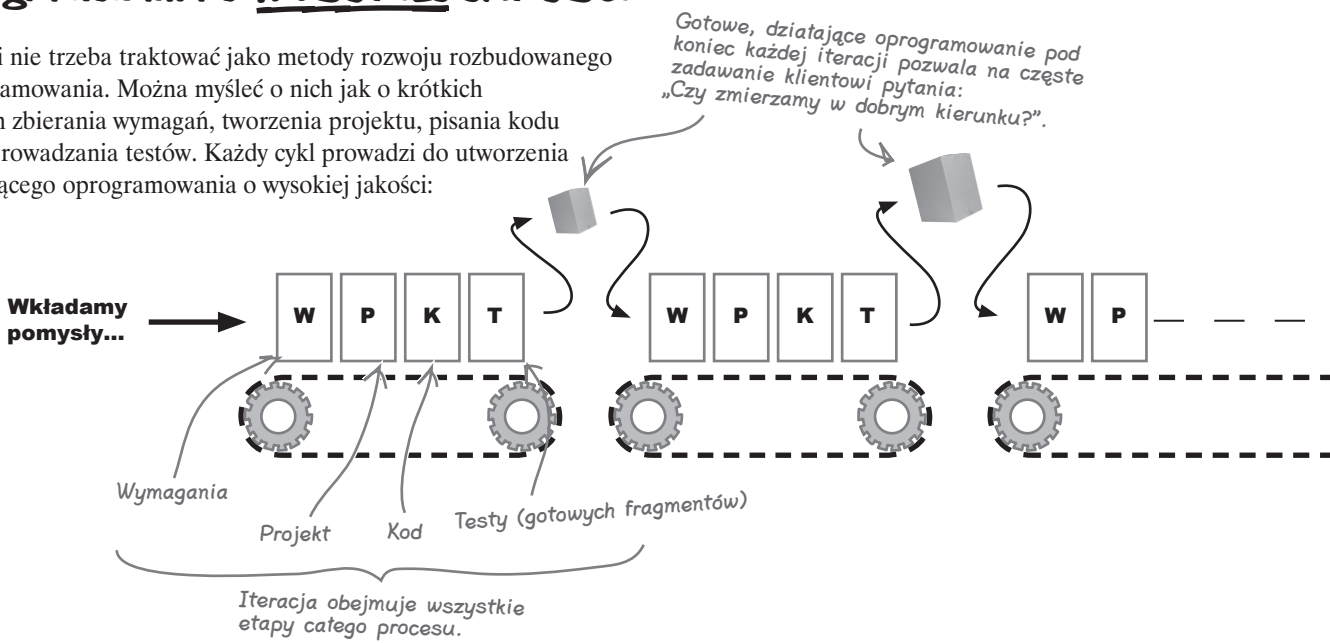
Przy stosowaniu iteracji należy **w każdej z nich** wykonywać takie same zadania, jak w całym projekcie. Poszczególne iteracje są miniprojektami i obejmują zbieranie wymagań, projektowanie, pisanie kodu, przeprowadzanie testów itd. Dlatego zespół nie przedstawia klientom byle czego, ale starannie przygotowane fragmenty gotowego oprogramowania.

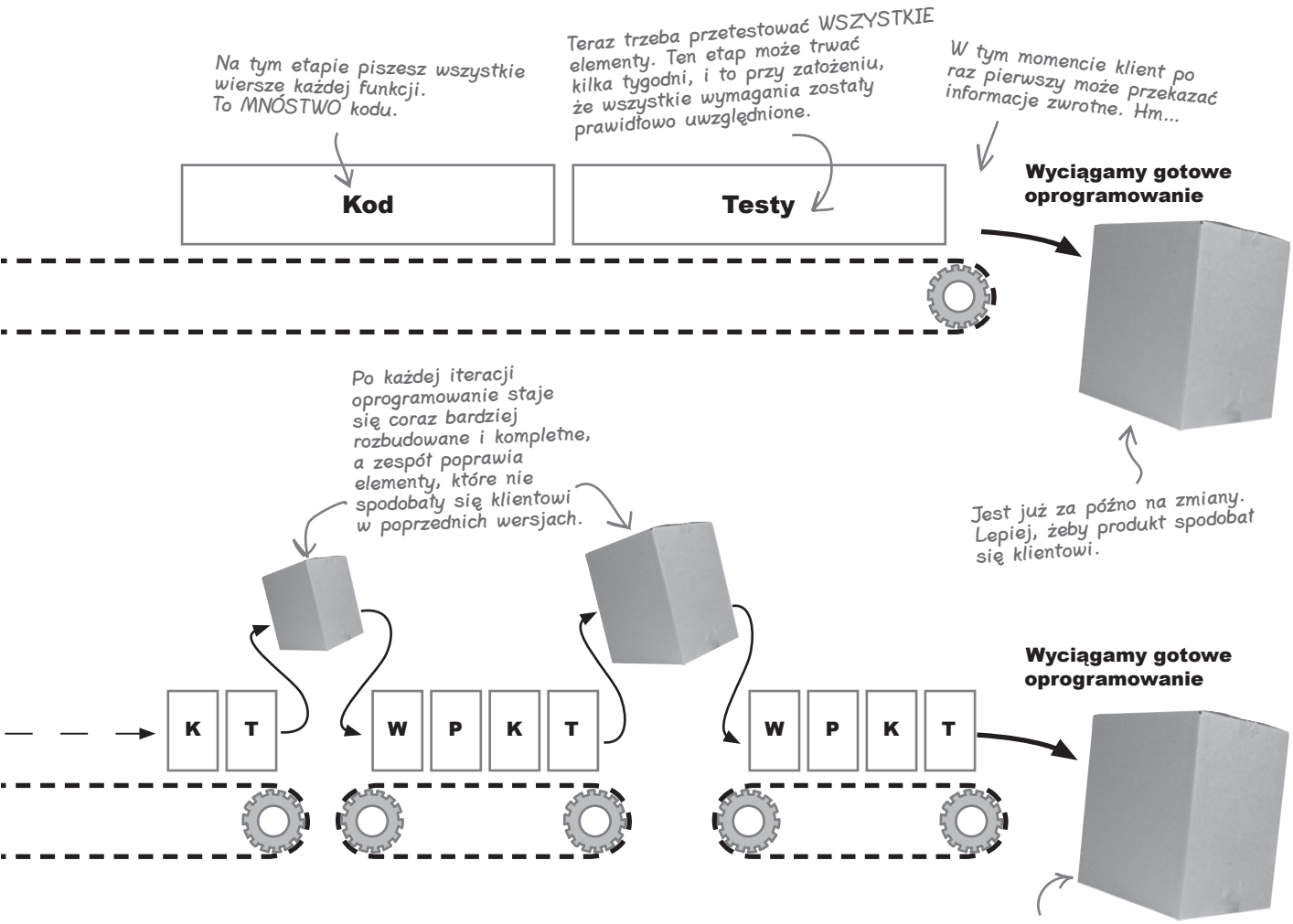
Pomyśl o rozwoju typowych aplikacji. Najpierw trzeba zebrać wymagania (czyli ustalić oczekiwania klienta), przygotować projekt całego oprogramowania, przez długi czas pisać kod, a następnie przetestować wszystkie elementy. Wygląda to mniej więcej tak:



Każda iteracja prowadzi do rozwoju oprogramowania o WYSOKIEJ JAKOŚCI

Iteracji nie trzeba traktować jako metody rozwoju rozbudowanego oprogramowania. Można myśleć o nich jak o krótkich cyklach zbierania wymagań, tworzenia projektu, pisania kodu i przeprowadzania testów. Każdy cykl prowadzi do utworzenia działającego oprogramowania o wysokiej jakości:




Na tym etapie piszesz wszystkie wiersze każdej funkcji. To MNÓSTWO kodu.
Teraz trzeba przetestować WSZYSTKIE elementy. Ten etap może trwać kilka tygodni, i to przy założeniu, że wszystkie wymagania zostały prawidłowo uwzględnione.
W tym momencie klient po raz pierwszy może przekazać informacje zwrotne. Hm...
Kod
Testy
Wyciągamy gotowe oprogramowanie
Jest już za późno na zmiany. Lepiej, żeby produkt spodobał się klientowi.
Po każdej iteracji oprogramowanie staje się coraz bardziej rozbudowane i kompletne, a zespół poprawia elementy, które nie spodobały się klientowi w poprzednich wersjach.
K
T
W
P
K
T
W
P
K
T
Wyciągamy gotowe oprogramowanie
Zespół sprawdza oprogramowanie pod koniec każdej iteracji, dlatego jest dużo bardziej prawdopodobne, że będzie odpowiadało ono wymaganiom klienta.

Ćwiczenia

Pora zastosować iteracje przy rozwoju witryny Szlakami Macieja. Przy każdej funkcji, której zażądał Maciek, zapisaliśmy szacunkowy czas potrzebny na jej przygotowanie. Następnie ustaliliśmy, jak ważne dla Maćka są poszczególne elementy, i przypisaliśmy im priorytety (10 to priorytet najwyższy, a 50 — najniższy). Umieść karteczki z funkcjami na osi czasu projektu i dodaj w odpowiednich miejscach iteracje.

Każda karteczka odpowiada jednej funkcji wymaganej przez Maćka.

Logowanie
2 dni
Priorytet klienta: 30

Porównywanie szlaków
1 dzień
Priorytet klienta: 50

Kupowanie sprzętu
15 dni
Priorytet klienta: 10

Ocena znaczenia danej funkcji dla Maćka. „10" oznacza, że dany element jest niezbędny.

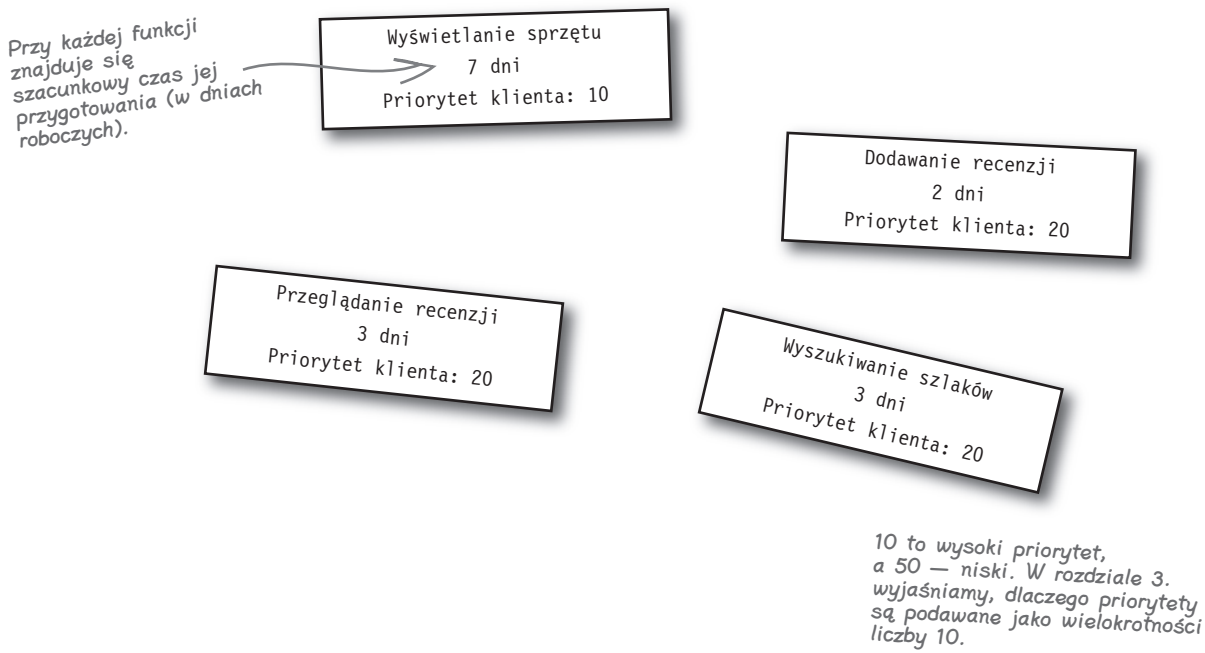
Pierwszą funkcję i iterację dodaliśmy już za Ciebie.

Przeglądanie szlaków
10 dni
Priorytet klienta: 10

Pierwsza iteracja

Jeśli to możliwe, iteracje powinny trwać około 20 dni roboczych. Warto pamiętać, że praca jest planowana w miesiącach kalendarzowych, a po odjęciu weekendów w każdej iteracji pozostaje co najwyżej 20 dni roboczych.

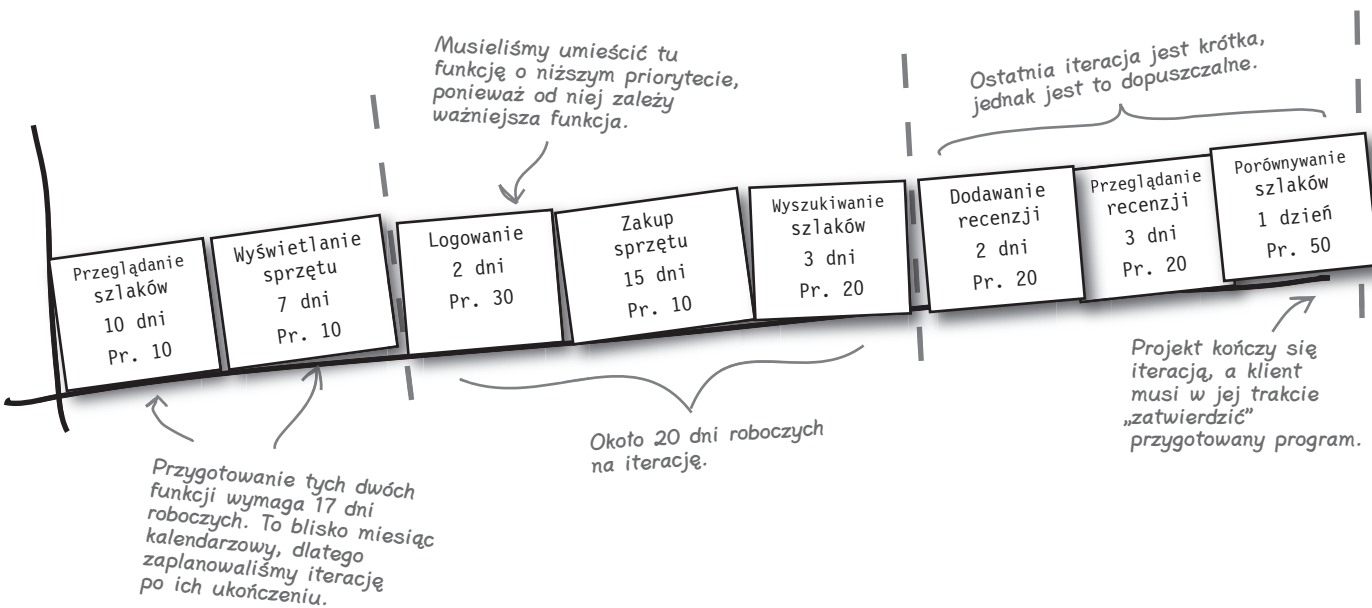
I jeszcze jedna rzecz: Maciek nie chce, żeby klienci mogli kupować sprzęt bez wcześniejszego zalogowania się do witryny. Należy uwzględnić to w planach.



Nie zapomnij dodać odpowiedniej liczby iteracji.

Rozwiązania ćwiczeń

Twoim zadaniem było utworzenie planu iteracji na potrzeby rozwoju witryny Szlakami Macieja. Powinieneś utworzyć listę podobną do poniższej:



Jest to prawdopodobnie jedyny plan, który uwzględnia priorytety klienta, obejmuje iteracje o odpowiedniej długości i prowadzi do ukończenia projektu. Jeśli zaproponowałeś inne rozwiązanie, zastanów się, dlaczego dokonałeś innych wyborów.

Długość iteracji należy dostosować do tempa KONKRETNEGO projektu

Iteracje pomagają zachować prawidłowy kierunek prac, dlatego możesz uznać, że powinny trwać krócej lub dłużej niż 30 dni kalendarzowych. Może się wydawać, że to dużo czasu, jednak po odjęciu weekendów oznacza to około 20 dni roboczych w trakcie iteracji. Jeśli nie jesteś pewien, jaka długość iteracji jest odpowiednia, zacznij od 30 dni kalendarzowych, a następnie dostosuj tę liczbę do prowadzonego projektu.

Najważniejsze jest przeprowadzanie iteracji wystarczająco często, aby wykryć odstępstwa od oczekiwań klienta, jednak nie nazbyt często, aby nie poświęcać zbyt dużo czasu na przygotowania do zakończenia iteracji. Zaprezentowanie odbiorcy przygotowanych funkcji i wprowadzenie poprawek wymaga pracy, dlatego trzeba pamiętać o uwzględnieniu tych zadań przy określaniu długości iteracji.

Nie ma niemądrych pytań

P: Ostatnia funkcja zaplanowana w iteracji spowoduje wydłużenie czasu do ponad miesiąca. Co powinienem zrobić?

O: Zastanów się nad przeniesieniem tej funkcji do następnej iteracji. Możesz przemieszczać zadania między 20-dniowymi iteracjami do momentu uzyskania pewności, że zdołasz zakończyć prace w zaplanowanym czasie. Wydłużanie iteracji grozi zejściem z optymalnej ścieżki rozwoju.

P: Porządkowanie zadań według priorytetów określonych przez klienta jest korzystne, co jednak zrobić, jeśli niektóre funkcje są niezbędne do realizacji innych?

O: Jeśli dana funkcja zależy od innej, warto je połączyć i upewnić się, że zostaną zrealizowane w tej samej iteracji. Dozwolone jest przy tym umieszczanie funkcji o niższych priorytetach przed ważniejszymi zadaniami, jeśli wymaga tego ukończenie tych ostatnich.

Takie rozwiązanie zastosowaliśmy w ostatnim ćwiczeniu, w którym funkcja „Logowanie" miała niższy priorytet, jednak trzeba było ją zaplanować przed implementacją funkcji „Zakup sprzętu".

P: Czy po zatrudnieniu dodatkowych osób można zwiększyć liczbę zadań wykonywanych w każdej iteracji?

O: Tak, jednak trzeba zachować przy tym ostrożność. Dodanie drugiej osoby nie skraca czasu przygotowania funkcji o połowę. W rozdziale 2., przy omawianiu szybkości pracy, opisujemy, jak uwzględnić koszty ogólne związane z dużymi zespołami.

P: Co się stanie, jeśli wystąpią zmiany wymagające zmodyfikowania planów?

O: Zmiany są — niestety — stałym elementem rozwoju oprogramowania i trzeba je uwzględniać w każdym procesie. Na szczęście iteracje pozwalają sprawnie radzić sobie z nowymi wymaganiami. Przewróć stronę i przekonaj się, co mamy na myśli.

Klient BĘDZIE zmieniał zdanie

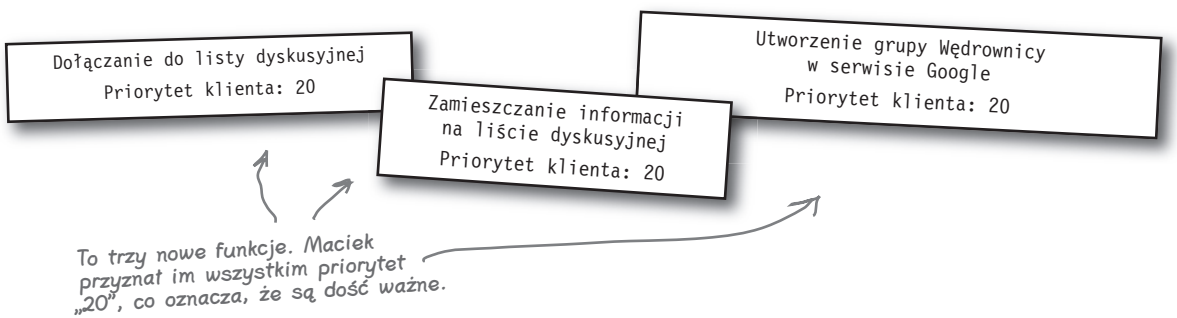
Maciek zatwierdził plan, a iteracja numer 1 została ukończona. Od pewnego czasu pracujesz już nad drugą iteracją, wszystko idzie zgodnie z planem, kiedy nagle dzwoni Maciek…

Witryna zaczyna wyglądać świetnie, jednak po ostatniej iteracji przyszło mi do głowy kilka pomysłów. Uważam, że naprawdę ważne jest, aby na witrynie Szlakami Macieja znajdowała się lista dyskusyjna, co pozwoli klientom komunikować się ze sobą.

Pamiętaj, jeśli Twoje aplikacje nie będą spełniać oczekiwań odbiorców, nie zajdziesz daleko w branży rozwoju oprogramowania.

Wprowadzenie poprawek to Twoje zadanie

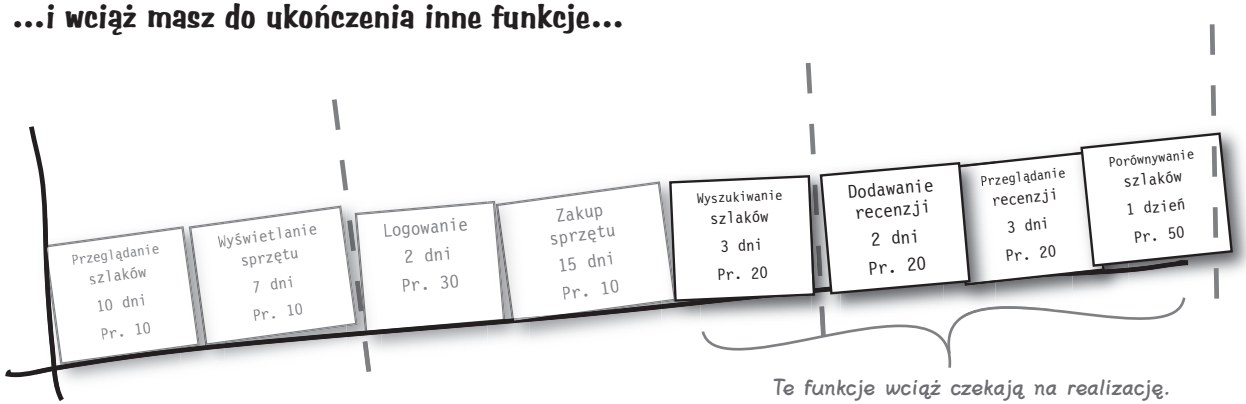
Nowy pomysł Maćka oznacza, że trzeba dodać trzy nowe funkcje, z których każda ma wysoki priorytet. Ponadto nie wiadomo jeszcze, ile czasu zajmie ich implementacja. Jednak musisz znaleźć sposób na uwzględnienie ich w planach.



Musisz poradzić sobie z POWAŻNYMI problemami…

Prace są już mocno zaawansowane...

...i wciąż masz do ukończenia inne funkcje...



...a termin się nie zmienił.

Pamiętasz termin z 39. strony? Pozostał taki sam, choć zmieniły się oczekiwania Maćka.

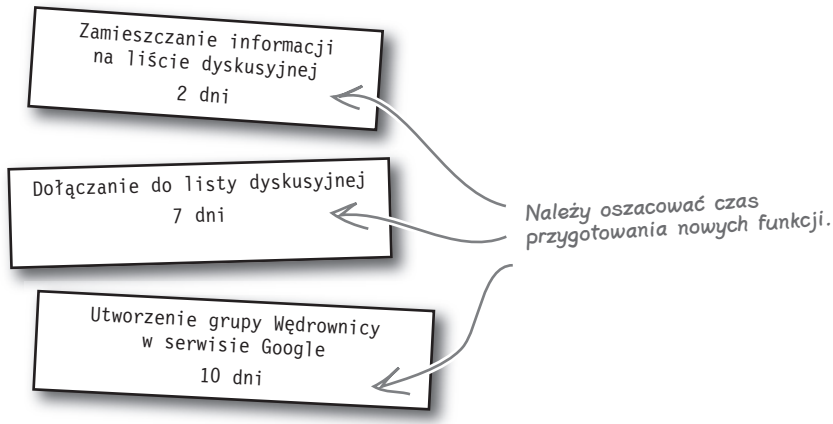
Do konferencji Drogi i bezdroża pozostał nieco ponad miesiąc.

Iteracje automatycznie uwzględniają zmiany (w pewnym sensie)

Plan iteracji jest już podzielony na krótkie cykle i pozwala ukończyć wiele poszczególnych funkcji. Musisz wykonać następujące operacje:

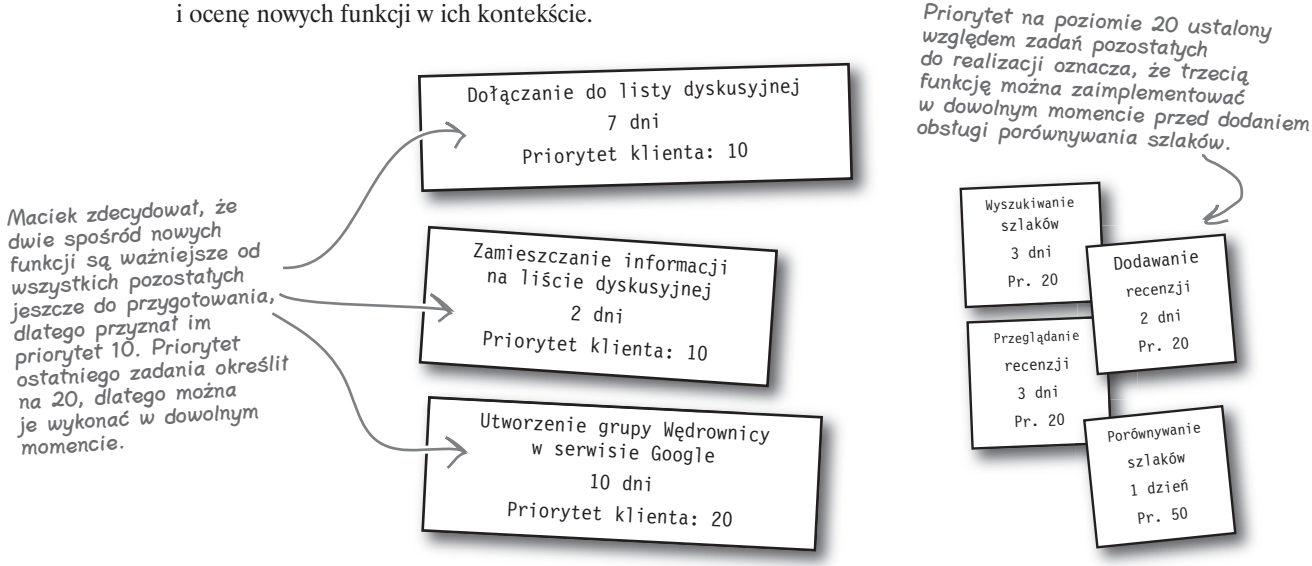
❶ Oszacować czas wykonania nowych funkcji

Najpierw musisz oszacować, jak długo potrwa ukończenie każdej nowej funkcji. W dalszych rozdziałach szczegółowo opisujemy proces szacowania, a na razie przyjmijmy wartości przedstawione poniżej:



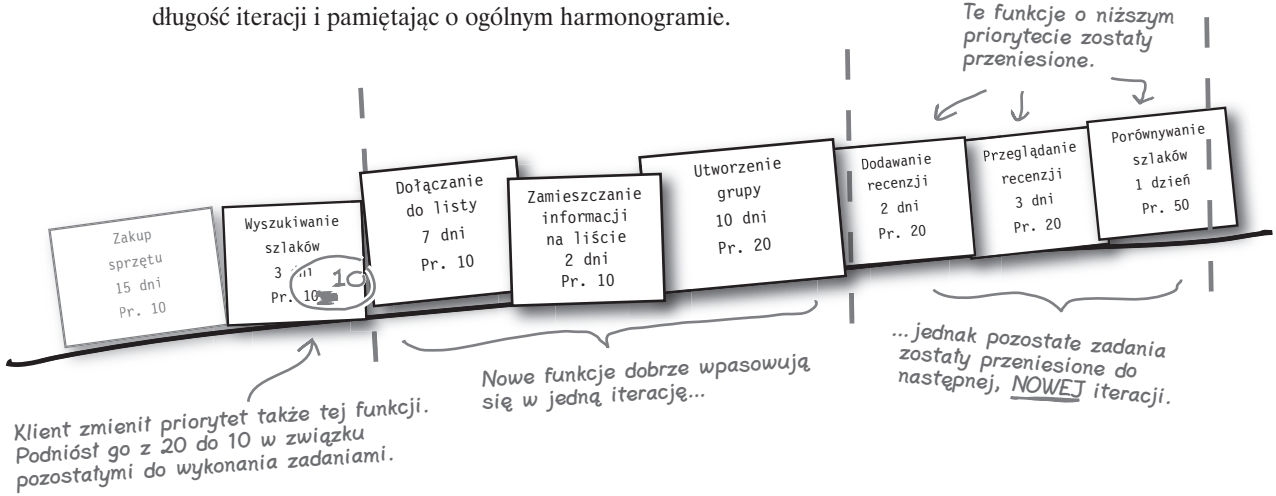
❷ Poprosić klienta o określenie priorytetów nowych funkcji

Maciek przyznał już wszystkim funkcjom priorytet 20, prawda? Jednak trzeba poprosić go o przyjrzenie się pozostałym niezrealizowanym jeszcze zadaniom i ocenę nowych funkcji w ich kontekście.



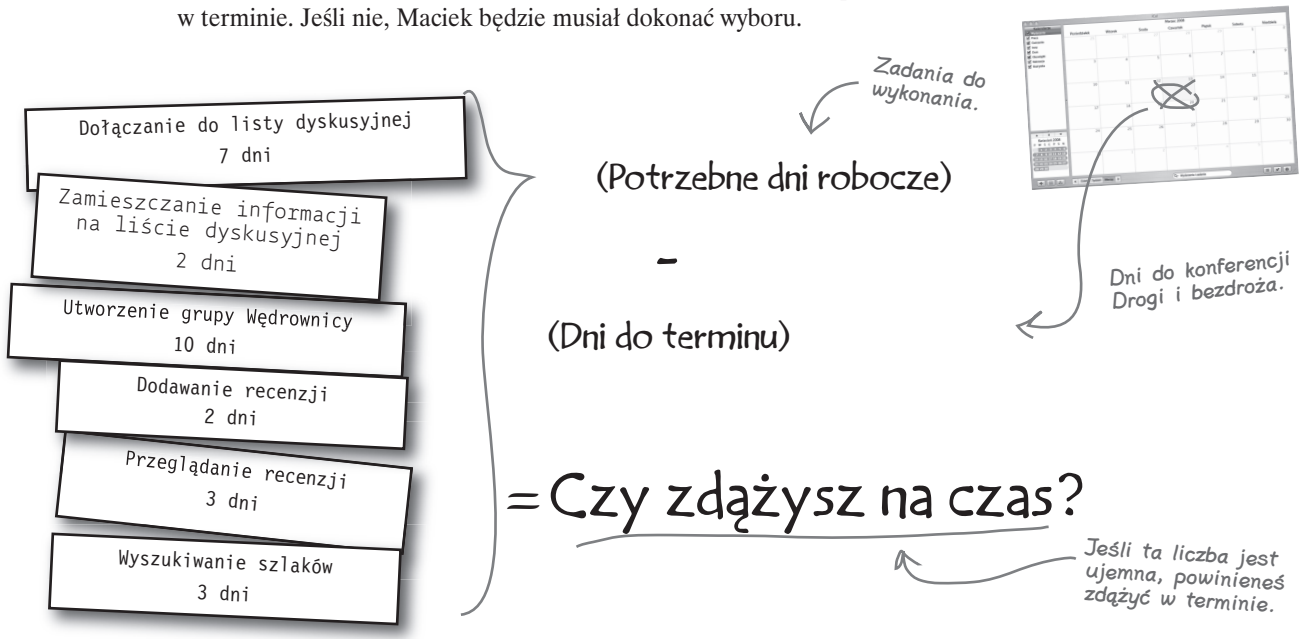
❸ Zmodyfikować plan iteracji

Kolejność funkcji została ustalona na podstawie priorytetów, a poszczególne zadania są niezależne od siebie. Teraz należy zmodyfikować plan, zachowując długość iteracji i pamiętając o ogólnym harmonogramie.



❹ Sprawdzić termin ukończenia projektu

Pamiętasz o konferencji Drogi i bezdroża? Musisz sprawdzić, czy pozostałe do wykonania zadania — łącznie z nowymi funkcjami — uda Ci się przygotować w terminie. Jeśli nie, Maciek będzie musiał dokonać wyboru.



Chcecie zaskoczyć mnie rozbudowanym, wymyślnym procesem rozwoju, prawda? Uważacie, że jeśli zastosuję RUP, Quick, DRUM lub inną metodę, nagle zacznę tworzyć doskonałe oprogramowanie?

Proces to po prostu sekwencja etapów

Proces, przede wszystkim w branży rozwoju oprogramowania, nie cieszy się najlepszą opinią. A przecież jest to po prostu sekwencja kroków wykonywanych w celu ukończenia zadania, którym w tym przypadku jest tworzenie aplikacji. Dlatego kiedy opisujemy iteracje, priorytety i szacowanie, tak naprawdę omawiamy proces rozwoju oprogramowania.

Proces to niekoniecznie formalny zestaw reguł dotyczących diagramów, dokumentów i testów (choć gorąco zachęcamy do przeprowadzania testów!). Często jedynie określa, co i kiedy należy robić. Do jego opisu nie jest potrzebny wymyślny akronim — proces ma po prostu działać.

Nie jest dla nas ważne, jakiego procesu używasz, o ile obejmuje on elementy, które na końcu cyklu rozwoju prowadzą do powstania oprogramowania o wysokiej jakości.

Wygląda na to, że iteracje można zastosować w dowolnym procesie, prawda?

Odpowiedni dla CIEBIE proces tworzenia oprogramowania to ten, który umożliwi TOBIE rozwój doskonałego oprogramowania na czas i po ustalonych kosztach.

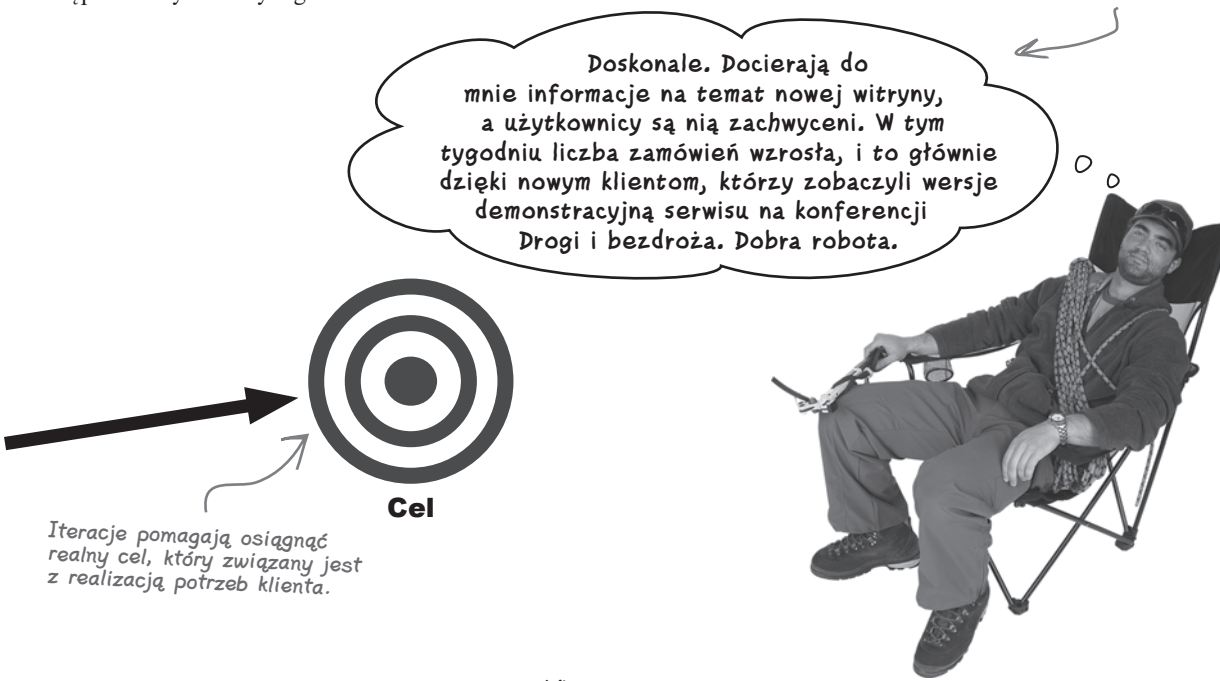
Iteracja to coś więcej niż proces

Niezależnie od etapów, które obejmuje używany proces, iteracje są najlepszym rozwiązaniem. Jest to technika, którą można zastosować w ***dowolnym*** procesie. Iteracje zwiększają szanse udostępnienia potrzebnego oprogramowania na czas i po ustalonych kosztach. Niezależnie od stosowanego procesu iteracje powinny być jego podstawowym elementem.

Oprogramowanie jest gotowe dopiero w momencie jego UDOSTĘPNIENIA

Dodałeś nowe funkcje, a obecnie z pomocą zespołu kończysz projekt na czas i po ustalonych kosztach. Na każdym etapie pracy otrzymywałeś pod koniec iteracji informacje zwrotne od klienta, a ***następnie*** uwzględniałeś je i nowe funkcje w kolejnych cyklach rozwoju. Teraz możesz udostępnić oprogramowanie, a następnie otrzymasz wynagrodzenie.

Maciek nie pisze o oprogramowaniu uruchamianym na komputerze programisty. Dla klienta ważna jest aplikacja działająca w świecie rzeczywistym.



Nie ma niemądrych pytań

P: Co się stanie, jeśli klient przedstawi nowe wymagania, a zespół nie będzie mógł wykonać wszystkich dodatkowych zadań w trakcie bieżącej iteracji?

O: Wtedy należy zwrócić uwagę na priorytety. Klient musi ustalić, co naprawdę wymaga realizacji w trakcie danej iteracji. Zadania, których nie uda się ukończyć, trzeba przełożyć do następnego cyklu rozwoju. W kilku następnych rozdziałach omawiamy iteracje bardziej szczegółowo.

P: Co zrobić, jeśli nie ma następnej iteracji? Jak postąpić, kiedy zespół jest w trakcie ostatniej iteracji, a klient nagle zażąda dodania funkcji o najwyższym priorytecie?

O: Jeśli informacje o niezbędnej funkcji pojawiają się zbyt późno i nie można zrealizować jej w ostatniej iteracji, trzeba przede wszystkim wyjaśnić klientowi, dlaczego nie można jej przygotować. Należy postępować uczciwie i pokazać odbiorcy plan iteracji oraz wytłumaczyć, dlaczego — przy dostępnych zasobach — dodatkowe zadanie utrudnia udostępnienie potrzebnego produktu w ustalonym terminie.

Najlepszym rozwiązaniem — jeśli klient wyrazi na nie zgodę — jest wydłużenie terminu i umieszczenie nowych wymagań w nowej iteracji. Można też zatrudnić dodatkowych programistów lub wydłużyć godziny pracy, jednak lepiej wystrzegać się takich metod. Dołączenie nowych osób i praca w nadgodzinach często zwiększają koszty projektu i bardzo rzadko prowadzą do oczekiwanego wzrostu wydajności (zobacz rozdział 3.).

Narzędzia do Twojej programistycznej skrzynki narzędziowej

Rozwój oprogramowania polega na tworzeniu i udostępnianiu programów, których klienci potrzebują. W tym rozdziale poznałeś kilka technik, które pomogą Ci odkryć myśli klienta i uzyskać wymagania odpowiadające potrzebom odbiorcy. Pełną listę narzędzi opisanych w książce znajdziesz w dodatku B.

Techniki rozwoju

Iteracje pomagają zachować prawidłowy kierunek prac.

Szczegółowo planuj iteracje i zachowuj ich stałą długość, kiedy (nie jeśli) zajdą zmiany.

Na każdym etapie projektu iteracje prowadzą do utworzenia działającego oprogramowania i uzyskania informacji zwrotnych od klienta.

To kilka kluczowych technik przedstawionych w tym rozdziale...

...a to zasady, na których oparte są powyższe techniki.

Zasady rozwoju

Udostępniaj potrzebne oprogramowanie.

Dostarczaj programy na czas.

Udostępniaj oprogramowanie po ustalonych kosztach.

KLUCZOWE ZAGADNIENIA

- **Informacje zwrotne** uzyskiwane w każdej **iteracji** to najlepsze narzędzie do zagwarantowania zgodności oprogramowania z potrzebami klienta.
- Iteracja to **miniaturowa** wersja kompletnego projektu.
- Udane oprogramowanie nie jest rozwijane w próżni. Niezbędne są **ciągłe informacje zwrotne** od klientów, a można je uzyskać dzięki iteracjom.
- Prawidłowy rozwój oprogramowania prowadzi do udostępniania doskonałych programów **na czas** i **po ustalonych kosztach**.
- Zawsze lepiej jest dostarczyć **wybrane** funkcje i zapewnić ich **doskonałe działanie**, niż udostępnić wszystkie mechanizmy, które jednak nie działają prawidłowo.
- Dobrzy programiści rozwijają oprogramowanie; **programiści doskonali** udostępniają je.