Dobre praktyki i narzędzia open source w zwinnym wytwarzaniu oprogramowania

Paweł Lubomski, Paweł Pszczoliński, Michał Nowacki

Politechnika Gdańska, Centrum Usług Informatycznych, ul. Narutowicza 11/12, 80-233 Gdańsk
{lubomski, pszczola, michal.nowacki}@pg.edu.pl

Abstrakt

Zwinne wytwarzanie oprogramowania stało się bardzo popularne. Na podstawie projektu MOST Wiedzy przedstawiono procesy i narzędzia wspierające wytwarzanie oprogramowania na Politechnice Gdańskiej. Metodologia DDD (domain-driven design) wspierana przez repozytorium GIT oraz tzw. proces "cherry picking" powoduje, że proces wytwarzania oprogramowania jest elastyczny i efektywny. Dodatkowo wprowadzenie elementów CI (continuous integration), narzędzi do inspekcji kodu oraz procedur przeglądu samego kodu przez innych deweloperów podniosło jakość wytwarzanego oprogramowania. Przedstawiono również wykorzystanie wzorca „database migration" i testów automatycznych, które pozwoliły na zapewnienie łatwiejszego utrzymania oprogramowania oraz kontroli spójności konfiguracji różnych środowisk uruchomieniowych. Dzięki zastosowaniu wszystkich omówionych rozwiązań możliwe jest łatwe reagowanie na zmieniające się wymagania użytkowników, ich priorytetyzację oraz fluktuację zatrudnienia w zespole deweloperskim.

1. Wstęp

W dzisiejszych czasach sformułowanie 'czas to pieniądz' dotyka praktycznie każdego aspektu życia. Nie inaczej jest w procesie wytwarzania oprogramowania. Zespoły IT napotykają coraz większą presję by wytwarzać skomplikowane rozwiązania technologiczne tak szybko, jak to tylko możliwe [1]. Dostarczane rozwiązania muszą spełniać wymagania użytkowników, a czasem nawet je wyprzedzać/kreować. Oczywiście wszystko to powinno odbywać się zgodnie z założonym harmonogramem i budżetem projektu. Właśnie z takimi wymaganiami mierzy się zespół realizujący projekt MOST Wiedzy [2] współfinansowany z Europejskiego Funduszu Rozwoju Regionalnego w ramach Programu Operacyjnego Polska Cyfrowa na lata 2014-2020.

W fazie przygotowania projektu MOST Wiedzy, pojawił się problemem wyboru metodyki jego realizacji. Warto zaznaczyć, że specyfika przedsięwzięcia różni się od standardowych działań. Nie zdefiniowano szczegółowych wymagań dla nowopowstającego systemu. Zamiast tego wyznaczono cele, które powinny być osiągnięte. Zdefiniowano je następująco: zapewnienie obustronnego przepływu wiedzy pomiędzy nauką a biznesem, udostępnianie pełnych treści artykułów, wraz z popularyzacją idei Open Access [3] oraz udostępnianie społeczeństwu informacyjnemu informacji o zasobach naukowych uczelni. Jak widać cele są ambitne, ale wybór drogi, która doprowadzi do ich realizacji nie był zadaniem trywialnym. Zdecydowano się wybrać metodologię, która pozwoli na szybkie wprowadzanie zmian oraz iteracyjną realizację projektu. Dodatkowo, termin realizacji oraz budżet, są wartościami niepodlegającymi negocjacjom. Mając na uwadze powyższe uwarunkowania, zdecydowano się na implementację systemu z wykorzystaniem zwinnych metod wytwarza oprogramowania (ang. *Agile Software Development – ASD*).

Jako początek zwinnych metod, przyjmuje się ogłoszenia manifestu agile (Manifesto for Agile Software Development [4]). Powstał on jako odpowiedź na popularny w owym czasie kaskadowy model wytwarzania oprogramowania. Zwinne metodyki, w porównaniu ze znanym „wodospadem" (ang. w*aterfall*), kładą duży nacisk na częste dostarczanie działających fragmentów systemu oraz pozyskiwanie na ich podstawie opinii od użytkowników. Proces ten jest iteracyjny i ciągły (rys. 1).

Rys. 1. Kluczowe etapy w zwinnych procesach wytwarzania oprogramowania

Pozwala, a wręcz nawet zachęca, do wprowadzania częstych zmian i udoskonaleń w tworzonym projekcie. Częsty kontakt z użytkownikiem i jego widoczny wpływ na kształt systemu pozwala zbudować zaufanie między zespołem wytwarzającym i odbiorcami. Wszystkie te cechy sprawiają, że realizacja projektów IT zgodnie z powyższą metodyką nie jest już nowością, lecz staje się normą. Według badań, aż dwie trzecie zespołów IT decyduje się na zwinne metodyki wytwarzania oprogramowania [5]. Jako główne zalety agile wskazywano większe zadowolenie końcowego odbiorcy z dostarczonego produktu, krótszy czas potrzebny do pierwszego uruchomienia produkcyjnego oraz niższe koszty wytworzenia.

2. Zwinne wytwarzanie oprogramowania w połączeniu z projektowaniem zorientowanym na użytkownika

Pragnąc osiągnąć cele postawione przed projektem, uznano, że jednym z priorytetów systemu powinien być nacisk na wysoką satysfakcję użytkownika końcowego. Dlatego też zdecydowano się wzbogacić proces wytwarzania oprogramowania o elementy projektowania zorientowanego na użytkownika (ang. *user-centred design – UCD*) [6]. Główne wytyczne UCD kładą nacisk na dużą współpracę z użytkownikami, zarówno w procesie projektowania, jak i wytwarzania oprogramowania. Analogicznie do zwinnych metodologii, UCD wymaga iteracyjnego podejścia do dostarczania końcowego produktu [7].

W początkowej fazie realizacji projektu eksperci UX (ang. *User eXperience*) [8] współpracowali z przedstawicielami kluczowych użytkowników powstającego systemu. Pozyskano w ten sposób dane, które pozwoliły na budowę profili trzech typowych grup odbiorców projektu. Przedstawiały one zbiory cech charakteryzujących przedsiębiorcę, naukowca oraz przedstawiciela społeczeństwa informacyjnego. Był to bardzo dobry punkt wyjścia do pracy nad projektem interfejsu użytkownika, który byłby dostosowany do stworzonych profili. Prototyp interfejsu został następnie poddany ocenie rzeczywistych użytkowników, wybranych przedstawicieli grup odbiorców w serii spotkań z ekspertami UX. Wyniki tych spotkań zostały poddane analizie i przełożone na konkretne wymagania wobec systemu. Po wprowadzeniu zmian do systemu ponownie przedstawiono system ocenie użytkowników.

Integracja projektowania zorientowanego na użytkownika oraz zwinnych procesów wytwarzania oprogramowania nie przysporzyła problemów zespołowi realizującemu projekt. Oba podejścia realizowane są w sposób iteracyjny, dodatkowo zachęcają zespoły IT do planowania częstych wydań i ewaluacji efektów oraz wprowadzania zmian wynikających z uwag użytkowników. Dodatkowo, połączenie tych dwóch metodologii pozwoliło zespołowi realizującemu projekt MOST Wiedzy na lepsze poznanie oczekiwań potencjalnych użytkowników wobec powstającego innowacyjnego systemu.

Dodatkowo wykorzystanie metodologii DDD (ang. *Domain-driven design*) [9] przekłada się także na implementację systemu, tak aby odpowiadał realnym zagadnieniem. Tworzenie obiektów, komponentów, usług oraz metod w metodologii DDD zakłada bardzo bliską współpracę użytkowników – ekspertów z projektantami na etapie modelowania rozwiązań. Doskonale wpisuje się to w proces oparty na ASD i UCD. Użytkownik musi być blisko całego procesu wytwarzania, aby realizowany produkt spełnił jego wymagania.

3. Elastyczny proces wytwórczy

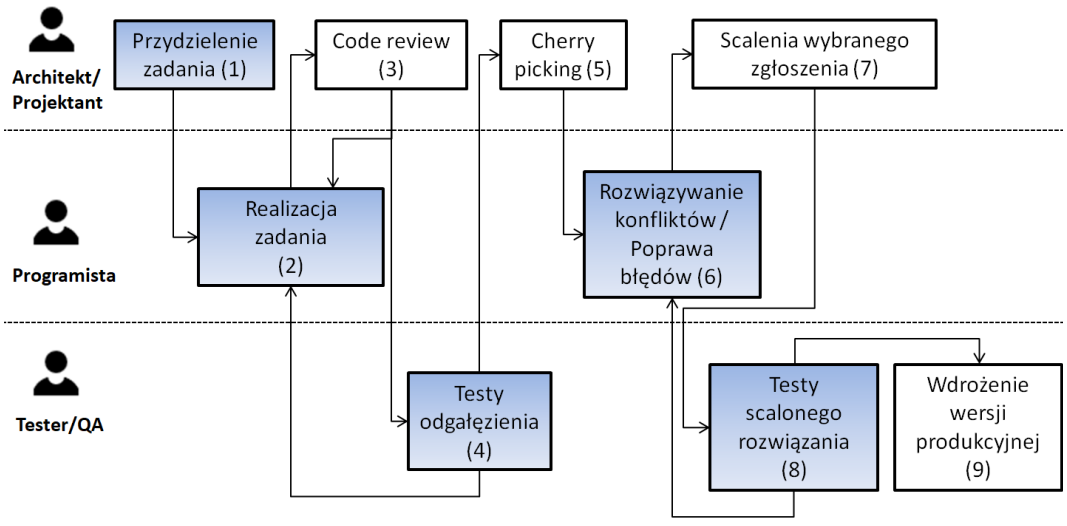
User eXperience jest bardzo ważnym, ale nie jedynym źródłem zadań do realizacji. Do innych należy zaliczyć:

- zgłoszenia zapotrzebowania na nowe funkcjonalności,
- zgłoszenia o poprawę już istniejących funkcjonalności,
- zgłoszenia o błędach w działaniu systemu,
- potrzeby poprawy wydajności działania systemu,
- potrzeby zmian technologicznych,
- wprowadzanie zabezpieczeń w działaniu systemu i dbanie o bezpieczeństwo danych.

Te wszystkie źródła generują bardzo wiele zadań do realizacji dla całego zespołu IT. Aby można było sprawnie i skutecznie je realizować należy zadbać, aby proces obsługi zgłoszeń był dobrze opisany, jasny dla wszystkich jego uczestników i poprawnie wdrożony w organizacji. Przy tworzeniu systemu MOST Wiedzy w procesie możemy wyróżnić 4 aktorów:

- projektanta/architekta – osobę odpowiedzialną za zarządzanie zadaniami i podejmowanie kluczowych decyzji dotyczących architektury systemu i strategii jego rozwoju,
- programistę – wytwórcę oprogramowania,
- testera – osobę odpowiedzialną za sprawdzenie wytworzonego rozwiązania i zagwarantowanie jakości tego rozwiązania,
- użytkownika – osobę, która będzie używać wytworzone rozwiązanie oraz znającą dokładanie procesy, które rozwiązanie to ma wspierać.

Wymienieni powyżej aktorzy uczestniczą w procesie w różnych rolach i na różnych etapach. Całość przedstawia rysunek 2. Etapy w których szczególnie ważny jest udział użytkownika i wykorzystanie User eXperience zaznaczono ciemniejszym wypełnieniem.



Rys. 2. Zwinny proces wytwarzania oprogramowania w projekcie MOST Wiedzy

Pierwszym etapem (1) jest zarządzanie zgłoszeniami. Zgłoszone zadania należy przeanalizować pod kątem zasadności jego realizacji oraz, gdy to koniecznie, dopisać niezbędne szczegóły. Na tym etapie niezwykle ważna jest „zwinna" współpraca projektanta rozwiązania z użytkownikiem. Następnie zadanie musi otrzymać priorytet wykonania oraz zakładany termin realizacji. Tak opisane zadanie może trafić do realizacji do programisty. Sprawdzonym narzędziem wspierającym zarządzanie zgłoszeniami jest Trac [10]. Umożliwia on pełną konfigurację cyklu życia oraz właściwości zgłoszenia. Możliwe jest dowolne grupowanie zgłoszeń względem odpowiednich wydań czy kamieni milowych. Bardzo przydatnym dodatkiem jest funkcjonalność Wiki, dzięki której można szybko budować bazę wiedzy, zbierając wszelkie ustalenia, standardy, procedury w jednym miejscu w czytelnej formie.

Na etapie realizacji zgłoszenia (2) musi zostać stworzone odgałęzienie w repozytorium kodu na potrzeby wprowadzania zmian. Jako repozytorium kody najlepiej sprawdza się Git [11]. Jest to rozproszony system kontroli wersji w pełni wspierający model odgałęzień. Narzędzie to jest bardzo wydajne, umożliwia pełną kontrolę nad wprowadzanymi zmianami oraz wspiera funkcjonalności opisane na kolejnych etapach procesu, takie jak przegląd kodu czy wybór zgłoszeń do scalenia. W danym odgałęzieniu realizowane jest zaprojektowane rozwiązanie. Podczas implementacji wszelkie wątpliwości co do realizacji powinny być konsultowane z projektantem i/lub użytkownikiem. Zaimplementowane rozwiązanie powinno zawierać oprócz właściwych zmian w kodzie także, jeśli to konieczne, migracje bazy danych, testy jednostkowe oraz automatyczne testy UI. Migracje baz danych zgodne z wzorcem projektowym „database migration" [12] powinny zawierać implementację procedur wprowadzających niezbędne modyfikacje struktury danych bądź samych danych wymaganych przy wdrażaniu danej zmiany. Migracje powinny być atomowe, niezależne od siebie i jeżeli to możliwe, to odwracalne. Doskonałym narzędziem wspierającym implementacje migracji jest Flywaydb [13]. Jest to zestaw bibliotek, dzięki którym można w łatwy sposób tworzyć migracje spełniające wszystkie wymagane kryteria. Taki pakiet zadań pozwala na zapewnienie wysokiej powtarzalności i jakości wytwarzania.

Kolejny etap procesu (3) to przegląd i inspekcja kodu. Jest to etap, w którym może brać udział kilku programistów. Sprawdzenie wykonywane jest według z góry ustalonego scenariusza, natomiast inspekcja (statyczna analiza kodu) jest robiona automatycznie według określonych reguł. Do statycznej analizy kody wykorzystywane są dwa narzędzia. Na lokalnych maszynach programistów używany jest FindBugs [14] natomiast kod który trafia do głównych gałęzi dodatkowo podlega centralnej analizie przy użyciu narzędzia Sonar Analyzer [15]. Proces ten może być cykliczny i trwać tak długo, aż wszystkie sprawdzenia i inspekcje zostaną poprawnie rozwiązane. Dzięki temu etapowi jakość kodu jest na zakładanym poziomie i zagwarantowane są spójne standardy wytwarzania.

Gdy kod zostanie sprawdzony oraz automatyczne testy zostaną wykonane należy przekazać rozwiązanie do testera aby ten sprawdził, czy zaimplementowane rozwiązanie odpowiada temu co zostało zgłoszone w zadaniu i czego oczekuje zgłaszający. Na tym etapie (4) tester uruchamia kod zawarty w odgałęzieniu na dedykowanym środowisku. Do budowania obrazów i konteneryzacji wykorzystywane są Ansible [16] i Docker [17][18]. Dzięki temu wyeliminowany został problem różnych ustawień na maszynach programistów i testerów. Wykorzystanie kontenerów Dockera daje gwarancję spójności na wszystkich maszynach członków zespołu, którzy uruchamiają środowisko do realizacji i testowania danego zgłoszenia. Tester sprawdza wszystkie wytyczne zawarte w zgłoszeniu i w razie wątpliwości konsultuje się na bieżąco z użytkownikiem. Ten etap również może być cyklicznie ponawiany, gdyż tester może przekazać zgłoszenie ponownie do programisty, aby ten wprowadził poprawki. Etap kończy się, gdy tester uzna, że zgłoszenie zostało zrealizowane poprawnie i oznaczy je jako zrealizowane, ale nie wdrożone.

Następnie (5) projektant na bieżąco przegląda zrealizowane zgłoszenia i na podstawie priorytetów i terminów realizacji decyduje, które z nich powinny zostać wdrożone do wersji produkcyjnej systemu. Wybierane zadania (ang. *cherry-picking*) muszą zostać scalone do gałęzi produkcyjnej. Doskonałym wsparciem do obsługi gałęzi i ich scalania jest GitLab [19]. Jest to rozbudowana platforma współpracująca z repozytorium kodu opartym na Git. W opisywanym procesie

wykorzystywana jest głównie do scalania wybranych zmian do odpowiednich gałęzi oraz zarządzania ewentualnymi konfliktami w kodzie. Aby scalić zmiany, wszystkie konflikty w kodzie muszą zostać rozwiązane przez programistę (6), najlepiej twórcę rozwiązania. Gdy kod jest gotowy do scalenia projektant, przy pomocy narzędzia GitLab, nanosi zmiany do odpowiedniej gałęzi produkcyjnej.

Aby mieć pewność, że wszystko zostało poprawnie scalone musi to zostać zweryfikowane (7). Ciągła integracja (ang. *continuous integration – CI*) zapewnia kompilację kodu, wykonanie testów automatycznych i publikację odpowiedniej wersji na środowisku testowym. Za CI w procesie odpowiada narzędzie Jenkins [20]. Pozwala ono na automatyzację procesu wydawniczego, współpracując z repozytoriom kodu i umożliwia wykonanie wszystkich zaplanowanych etapów niezbędnych do skutecznej publikacji nowej wersji oprogramowania na wybranych serwerach. Tester sprawdza czy scalone rozwiązanie jest zgodne z wytycznymi z zadania (8). Etap ten może być cykliczny, gdy okaże się, że na etapie scalania coś zostało niepoprawnie zrealizowane, programista konsultując się z projektantem, wprowadza niezbędne poprawki. Na tym etapie także może być potrzebny udział użytkownika w celu ostatecznej akceptacji wprowadzonego rozwiązania. Po tym etapie zadanie zostaje oznaczone jako gotowe do wdrożenia wraz z najbliższym wydaniem wersji produkcyjnej.

Wdrożenie wersji (9) produkcyjnej powinno być automatyczne, zainicjowane przez testera pełniącego funkcję menedżera wdrożeń. Po wdrożeniu wersji, na środowisku produkcyjnym tester sprawdza czy wszystkie zgłoszenia zostały opublikowane i oznacza zadanie jako zamknięte – zrealizowane, co kończy proces. Bardzo ważne jest, aby zgodnie z założeniami UX użytkownik systemu, którego dotyczy zadanie był blisko jego realizacji, a wszelkie uwagi wynikające z takiej współpracy były opisywane w zadaniu. Dzięki temu opisany proces pozwala na szybkie i skuteczne realizowanie zadań, bez potrzeby czasochłonnej analizy i dużej ilości dokumentacji przy jednoczesnym zachowaniu bardzo wysokiej jakości wytwarzania.

4. Podsumowanie

Powyższy proces został wdrożony w Centrum Usług Informatycznych Politechniki Gdańskiej w ramach realizacji projektu MOST Wiedzy. Dzięki wykorzystaniu zwinnego procesu wytwarzania oprogramowania możliwe jest:

- szybkie reagowanie na incydenty i błędy,
- szybkie wprowadzanie zmian,
- zapewnienie wysokiej jakości rozwiązań,
- uniknięcie powstawania zbędnej dokumentacji,
- łatwe wdrażanie nowych członków zespołu,
- kontrolowanie przestrzegania dobrych praktyk i standardów,
- panowanie nad częstymi zmianami kodu i scalanie tylko przetestowanych funkcjonalności,
- dostosowanie systemu do potrzeb użytkowników, a nie odwrotnie.

Warunkiem koniecznym do realizacji procesu jest ciągła współpraca z klientem. Na Politechnice Gdańskiej przy tworzeniu platformy MOST Wiedzy wybrana grupa użytkowników uczestniczy w procesie tworzenia systemu od pierwszego etapu jego wytwarzania. Dzięki temu proces wytwarzania może sprawnie funkcjonować czego efektem jest regularne wdrażanie nowych wersji jako odpowiedzi na zapotrzebowania użytkowników.

Bibliografia:

[1] M. A. Cusumano and D. B. Yoffie, "Software development on Internet time," *Computer*, vol. 32, no. 10, pp. 60–69, 1999.

[2] "Multidyscyplinarny Otwarty System Transferu Wiedzy – MOST Wiedzy." [Online]. Available: http://mostwiedzy.pl/. [Accessed: 10-Feb-2017].

[3] P. Suber, *Open Access*. The MIT Press, 2013.

[4] "Manifesto for Agile Software Development." [Online]. Available: https://www.agilealliance.org/agile101/the-agile-manifesto/. [Accessed: 23-Aug-2017].

[5] "Survey: Is agile the new norm?" [Online]. Available: https://techbeacon.com/survey-agile-new-norm. [Accessed: 23-Aug-2017].

[6] S. P. Cycles, B. Y. G. Pearson, and S. Pearsall, "Becoming Agile," vol. 4, no. 4, pp. 4–6, 2005.

[7] D. Fox, J. Sillito, and F. Maurer, "Agile Methods and User-Centered Design: How These Two Methodologies are Being Successfully Integrated in Industry," in *Agile 2008 Conference*, 2008, pp. 63–72.

[8] W. Albert and T. Tullis, *Measuring the User Experience 2nd Edition*. Morgan Kaufmann, 2013.

[9] E. Evans, *Domain-driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2004.

[10] "Trac Open Source Project." [Online]. Available: Trac Open Source Project. [Accessed: 24-Aug-2017].

[11] "git." [Online]. Available: https://git-scm.com/. [Accessed: 24-Aug-2017].

[12] T. Laszewski and P. Nauduri, "Database Schema and Data Migration," in *Migrating to the Cloud*, Elsevier, 2012, pp. 93–130.

[13] "Flywaydb." [Online]. Available: https://flywaydb.org/. [Accessed: 24-Aug-2017].

[14] "FindBugs." [Online]. Available: http://findbugs.sourceforge.net/. [Accessed: 24-Aug-2017].

[15] "sonarqube." [Online]. Available: https://www.sonarqube.org/. [Accessed: 24-Aug-2017].

[16] "Ansible." [Online]. Available: https://github.com/ansible. [Accessed: 24-Aug-2017].

[17] "Docker." [Online]. Available: https://www.docker.com/. [Accessed: 24-Aug-2017].

[18] D. Merkel, "Docker: lightweight Linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[19] "GitLab." [Online]. Available: https://about.gitlab.com/. [Accessed: 24-Aug-2017].

[20] "Jenkins. Build great things at any scale." [Online]. Available: https://jenkins.io/. [Accessed: 24-Aug-2017].

Artykuł powstał w ramach projektu "Multidyscyplinarny Otwarty System Transferu Wiedzy – MOST Wiedzy" – umowa nr POPC.02.03.01-00-0014/16-00.