ISSN 1507-3858
e-ISSN 2450-0003

Przemysław Bagiński, Krzysztof Labus, Jan Werewka
AGH Akademia Górniczo-Hutnicza w Krakowie
e-mails: baginski.przemyslaw@gmail.com; krzysztof.s.labus@gmail.com; werewka@agh.edu.pl

WPŁYW ŚWIADOMOŚCI DOTYCZĄCEJ DŁUGU TECHNICZNEGO I WIEDZY O NIM NA POWODZENIE PROJEKTÓW INFORMATYCZNYCH

IMPACT OF KNOWLEDGE AND AWARENESS OF TECHNICAL DEBT IN SOFTWARE DEVELOPING TEAMS ON IT PROJECTS SUCCESS

DOI: 10.15611/ie.2015.3.01
JEL Classification: L15

Streszczenie: Celem pracy jest analiza wpływu długu technicznego na osiąganie celów projektów informatycznych z perspektywy zespołów rozwijających oprogramowanie. W artykule skoncentrowano się na powodach narastania długu technicznego i trudnościach związanych z jego obsługą oraz spłacaniem. Zidentyfikowano najważniejszych interesariuszy, na których dług techniczny ma wpływ lub którzy wpływają na narastanie długu technicznego. Interesującym zagadnieniem jest spojrzenie na dług od środka, z perspektywy członków zespołów rozwijających oprogramowanie. W celu uzyskania potrzebnych informacji przeprowadzono ankietę skierowaną do członków zespołów rozwijających oprogramowanie. W ankiecie uzyskano odpowiedzi związane z napotykanymi rodzajami długu technicznego, nastawieniem, wiedzą oraz podejściem do długu technicznego. Wykazano, że brak wiedzy i świadomości w zespołach rozwijających oprogramowanie ma wpływ na narastanie długu technicznego. Opracowanie zawiera również propozycję rozwiązań bazujących na analizie SWOT, które mają na celu przeciwdziałanie niebezpiecznym sytuacjom wynikającym z długu technicznego.

Słowa kluczowe: dług techniczny, zarządzanie projektami informatycznymi, zespoły rozwijające oprogramowanie, metodyki zwinne, zarządzanie ryzykiem.

Summary: The aim of the study was to analyze the impact of technical debt on projects goals achievement, considered from the perspective of software developing teams. The investigations are addressed to technical debt accumulation, difficulties related to its service and repaying. The most important stakeholders are identified on which the dechnical debt has an impact or which influence the technical debt accumulation. An interesting issue was to look at the debt from the inside in order to obtain views of members of the software developing teams. In order to obtain all the necessary information, a survey is conducted referred to the members of the software developing teams. The responses obtained from the survey concerns

types of technical debt encountered, attitude, knowledge and approach to the debt. The article also shows that the lack of knowledge and awareness related to technical debt in teams affects the technical debt growth. The paper provides, based on SWOT analysis, solutions proposals that are designed to prevent dangerous situations resulting from technical debt.

Keywords: technical debt, IT project management, software developing teams, agile methodologies, risk management.

1. Wstęp

W wielu firmach informatycznych zauważono, że po początkowych sukcesach dostarczanego oprogramowania napotkano duże problemy z dostarczaniem nowych funkcjonalności, na które czekali klienci. Wraz ze zwiększeniem stopnia złożoności oprogramowania coraz trudniejsze było dochowanie terminów, chociaż przyrost funkcjonalności nie był wysoki. W pracach zajmujących się tymi zagadnieniami w pierwszej kolejności zwrócono uwagę na słabą jakość dostarczanego oprogramowania, polegającą na dużych trudnościach w jego dalszej rozbudowie. W roku 1992 W. Cunningham [Cunningham 1992] stwierdził: „Wdrożenie pierwszej wersji kodu przypomina zadłużanie się. Mała ilość długu przyspiesza proces deweloperski tak długo, jak długo jest on spłacany odpowiednio szybko (…) Ryzyko pojawia się, kiedy dług nie jest spłacany. Każda minuta spędzona na niezbyt poprawnym kodzie jest liczona jak odsetki od długu. Ilość długu wytworzona przez nieskonsolidowane implementacje może zablokować prace całych organizacji". Cytowany autor w swoim stwierdzeniu używa metafory długu finansowego, który jeśli jest mały, działa wspomagająco, pomagając przezwyciężyć pewne, chwilowe trudności oraz nadać impuls rozwojowi oprogramowania, jednak powyżej pewnego progu sytuacja diametralnie się zmienia. Dług staje się wtedy trudny bądź nawet niemożliwy do spłacenia, uniemożliwiając efektywną pracę, co powoduje konieczność jej przerwania i skupienia się wyłącznie na jego spłacie.

W literaturze anglojęzycznej ten rodzaj długu nazywany jest *technical debt*. W różnych opracowaniach polskich istnieje duże zamieszanie na skutek stosowania w sposób dowolny nazw „dług techniczny" i „dług technologiczny". Profesor Z. Łucki [Łucki 2015] postuluje: „nie mówmy »technologia« na technikę!" Cytowany autor stwierdza dalej: »"Słownik wyrazów obcych" podaje, że w języku greckim wyraz *techne* oznacza „sztukę, rzemiosło" i „ogół środków i czynności wchodzących w zakres działalności ludzkiej, związanej z wytwarzaniem dóbr materialnych"«. To samo źródło informuje, że termin „technologia" pochodzi od greckich słów *techne* i *logos* (zbiór, rozum i in.) i oznacza „metodę przetwarzania dóbr materialnych w dobra użyteczne; także: wiedzę o tym procesie". Autorzy niniejszego artykułu będą stosować nazwę „dług techniczny", gdyż rozważania dotyczą ogółu środków i czynności związanych z rozwojem oprogramowania, a nie metod lub wiedzy.

Początkowo pojęcie długu technicznego związane było jedynie z „niedoskonałościami" kodu, jednak wraz z upływem czasu zaczęło ono ewoluować o kolejne aspekty tworzenia oprogramowania. E. Yourdon w opracowaniu [*Czym jest dług technologiczny* 2014] określił dług techniczny jako pomijanie etapu analizy i projektowania, co prowadzi do ciągłego przeredagowywania kodu aplikacji wraz ze zmieniającymi się wymaganiami. Aktualnie jako dług techniczny traktowane są wszystkie braki oraz uchybienia w projekcie − te związane zarówno z kodem, projektowaniem, testowaniem, jak i z dokumentacją.

Dług techniczny jest czymś, z czym borykają się pracownicy we wszystkich projektach informatycznych, począwszy od osób zatrudnionych na stanowiskach menedżerskich, na programistach na najniższym szczeblu hierarchii pracowniczej skończywszy. Mimo że pojęcie długu technicznego zostało zdefiniowane już blisko ćwierć wieku temu, wiele osób nie zdaje sobie sprawy, że spotykają się z nim na co dzień. W związku z brakiem wiedzy na temat występowania długu brakuje również informacji na temat, w jaki sposób bezpiecznie nim zarządzać, jak go spłacać, czy zawsze spłata taka jest konieczna i celowa. Co najważniejsze, brak identyfikacji długu sprawia, że pracownicy nie są świadomi zagrożeń, jakie niesie on ze sobą, a często zagrożenia wynikające z długu technicznego są katastrofalne w skutkach.

W artykule zajęto się zagadnieniami identyfikacji i analizy wpływu długu technicznego na projekty informatyczne. Artykuł ma następującą budowę: we wstępie przedstawiono pojęcia związane z długiem technologicznym oraz dokonano przeglądu prac w tej dziedzinie. W następnym punkcie dokonano klasyfikacji przyczyn powstawania długu technicznego. Jednym z problemów związanych z powstawaniem i przyrostem długu technicznego jest brak jego świadomości oraz przyczyn powstawania. Zjawisko to jest częste w zespołach rozwojowych oraz wśród interesariuszy związanych z obszarem biznesowym. W tym celu przeprowadzono ankietę na temat świadomości związanej z długiem technicznym i czynników wpływających na jego przyrost. Kolejnym etapem było przygotowanie wstępnych propozycji przeciwdziałania opierających się na zasadach i atrybutach jakości.

2. Zagadnienie długu technicznego

Dług techniczny (*technical debt*) [*Technical debt* 2015] – jest metaforyczną koncepcją związaną z rozwojem oprogramowania (jego zarządzaniem, architekturą oraz kodowaniem), która odzwierciedla dodatkowe prace rozwojowe. Prace te powinny zostać wykonane tak, by projekt mógł być rozważany za wykonany zgodnie z kanonami dobrego rozwoju oprogramowania na płaszczyznach technicznej oraz zarządzania.

Refaktoryzacja [*Code refactoring* 2015] (*refactoring*) to proces wprowadzania zmian w projekcie/programie, w wyniku których zasadniczo nie zmienia się funkcjonalność. Celem refaktoryzacji nie jest więc wytwarzanie nowej funkcjonalności, ale utrzymywanie odpowiedniej, wysokiej jakości organizacji systemu.

Dług techniczny przeważnie jest ściśle powiązany z etapem planowania prac projektowych, zwłaszcza w kontekście refaktoryzacji. Oznacza to, że restrukturyzacja istniejącego kodu (refaktoryzacja) jest wymagana w ramach procesu rozwoju. Innymi słowy, poprawki te nie są tylko wynikiem słabo napisanego kodu, ale również pojawiają się w oparciu na późniejszym zrozumieniu problemu lub dodatkowych wymaganiach wprowadzonych przez klienta. Częsta refaktoryzacja jest jednym ze skuteczniejszych sposobów rozwiązania tego problemu. Zważywszy na wspomniane fakty, już podczas planowania projektu warto się zastanowić nad refaktoryzacją kodu. Należy zwrócić szczególną uwagę na częstotliwość wykonywanych dotychczas poprawek oraz rozważyć, co jest głównym powodem małej liczby refaktoryzacji. Często przyczyną jest źle zaplanowany budżet oraz redukcja kosztów w postaci rezygnacji z refaktoryzacji. Kolejnym istotnym czynnikiem wpływającym negatywnie na jakość refaktoryzacji jest pośpiech, wynikający z presji czasu podczas realizacji funkcjonalności. Gdy oszacowanie pracochłonności konkretnego etapu projektu jest zaniżane, logiczne jest, że z czegoś należy zrezygnować. Dosyć powszechnie i niesłusznie stosowaną praktyką jest odstąpienie od refaktoryzacji.

W literaturze naukowej temat długu technicznego omawiany jest z różnych perspektyw. W pierwszej kolejności podejmowane są próby usystematyzowania pojęć. W artykule [Alves i in. 2014] przedstawiono propozycję porządkowania wiedzy na temat długu technicznego z zastosowaniem ontologii. W pracy między innymi dokonano identyfikacji różnych definicji i typów długu technicznego. W artykule [Li, Avgeriou, Liang 2015] przedstawiono systematyczne podejście w odwzorowaniu różnych rodzajów długu technicznego na sposoby zarządzania nimi.

Drugą klasą zagadnień jest opracowanie strategii postępowania z długiem technicznym. W [*Steve Garnett − Technical Debt*… 2013] rozpatrywano zagadnienia strategii i taktyk unikania i usuwania ryzyka długu technicznego. Zagadnienia długu technicznego dotyczące stosowanych praktyk w zwinnych metodykach rozwoju oprogramowania, przedstawione w [Holvitie, Leppanen, Hyrynsalmi 2014], bazowały na ankietach przeprowadzonych w przemyśle. Zwrócono uwagę na istotny wpływ architektury i struktury implementacyjnej na dług techniczny. Podobnie w pracy [Martini, Bosch 2015] zwrócono uwagę na niebezpieczeństwa wynikające z architektonicznego długu technicznego. Również w pracy [Li, Liang, Avgeriou 2015] rozpatrywano architektoniczny dług techniczny, wynikający z decyzji architektonicznych i zmian scenariuszy korzystania dla rozwijanego systemu.

W niektórych pracach podejmowano zagadnienia automatycznych metod wykrywania długu technicznego. Na przykład w [Da S Maldonado, Shihab 2015] dokonywano próby identyfikacji długu technicznego przez analizę kodu źródłowego oprogramowania. W opracowaniu [Zazworka i in. 2014] podano propozycję zastosowania czterech podejść do identyfikacji długu technicznego.

3. Identyfikacja przyczyn powstawania długu technicznego

Przyczyny powstawania długu technicznego są wielorakie, różnorodne jest również oddziaływanie długu na projekt. Tak jak w świecie finansów, tak i w obszarze technicznym mamy różne sposoby podejścia do długu. Ze względu na sposób powstawania dług techniczny możemy podzielić na trzy główne rodzaje [Rubin 2012]:

1. Dług techniczny mimowolny (lekkomyślny) − jest rodzajem długu technicznego, który spowodowany jest niewiedzą, brakiem doświadczenia czy też niedbałością.
2. Nieunikniony dług techniczny − jest to rodzaj długu technicznego, na którego przyrost nie mamy wpływu. Często związany jest on ze zmiennym rynkiem, zmianą wymagań klienta czy też narzędzi używanych do rozwoju naszego oprogramowania.
3. Strategiczny dług techniczny zaciągany świadomie dla uzyskania celów krótkoterminowych.

Przyjęło się, że uchybienia popełniane we wczesnych fazach projektu dają o sobie znać w kolejnych fazach, są one bardziej uciążliwe, a obsługa tego typu długu jest trudniejsza i kosztowniejsza. W dalszej części tekstu podano najważniejsze przyczyny powstawania i narastania długu technicznego, bazując na [*Technical debt* 2015].

1. Biznesowe

- Presja biznesu – zjawisko występuje wtedy, gdy produkt musi z różnych powodów zostać ukończony wcześniej od początkowo zamierzonego terminu bądź też rozwój oprogramowania z różnych powodów się opóźnia, a nie jest możliwe przesunięcie terminu jego wydania.

2. Użyteczności

- Problemy z wdrażaniem i integracją. Problem występuje szczególnie w przypadku nietypowych konfiguracji środowisk klienta, jak i integracji z produktami firm zewnętrznych. Z powodu mnogości różnorodnego oprogramowania na rynku, a także sposobów jego konfiguracji problem ten jest dość powszechny.

3. Organizacyjne

- Problemy komunikacyjne. Na te trudności mogą się składać niedomagania komunikacyjne między zespołem rozwijającym oprogramowanie a osobami odpowiedzialnymi za cele biznesowe, a także brak przepływu informacji wewnątrz zespołu. Należy też dodać, że zdarza się, iż jednostka rozwijająca oprogramowanie ma tak dużą wiedzę domenową, iż nie jest w stanie przekazać tej wiedzy innym w krótkim czasie. W ostateczności udziela informacji zdawkowo, co wywołuje konsekwencje w postaci niezadowolenia zainteresowanego, tj. pracownika ponownie borykającego się z już napotkanym i rozwiązanym wcześniej problemem.
- Brak współpracy w zespołach rozwojowych. Brak współpracy między programistami często prowadzi do błędnego wykorzystania już istniejącego kodu, przerabiania go w sposób nieprawidłowy bądź nawet do pisania kodu redundantnego.

4. Projektowe

- Problemy z architekturą. Błędy architektoniczne mają poważny wpływ na rozwój aplikacji, często uniemożliwiając rozbudowę oprogramowania bez czasochłonnych zmian poprzedzonych kolejną analizą.
- Niedostatecznie przeprowadzona analiza. Głównie dotyczy to analizy wymagań. Na przykład niewykonanie analizy wymagań niefunkcjonalnych wpływa na niewłaściwe decyzje związane z architekturą rozwijanych systemów.

5. Oprogramowania

- Równoległy rozwój wielu wersji oprogramowania. Rozwijanie wielu wersji w tym samym czasie prowadzi do licznych niespójności, którymi trudno zarządzać.
- Niedostateczna refaktoryzacja kodu. Brak nawyku bieżącego poprawiania drobnych niedoskonałości kodu sprawia, że wady zaczynają się nawarstwiać i to, co początkowo było drobnymi problemami, sprawia, że kod jest nieczytelny, mało optymalny, a kolejne modyfikacje stają się trudne do wprowadzenia i czasochłonne.
- Mało wartościowy kod (*code smell*). Kod, który nie zachowuje powszechnie przyjętych standardów, utrudnia jego konserwację, rozszerzanie oraz analizę przez innych programistów (np. [Fontana i in. 2015]).

6. Dokumentacyjne

- Problemy z dokumentacją kodu. Niedostatecznie udokumentowany kod sprawia wiele problemów podczas jego analizy. Analiza nieudokumentowanego kodu jest bardzo często czasochłonna i nie zawsze prowadzi do prawidłowego rozpoznania funkcjonalności oraz sposobu działania kodu. Analiza taka jest wykonywana nie tylko przez osoby, które po raz pierwszy mają do czynienia z takim kodem, ale też często przez jego autorów, którzy przez długi czas nie mieli z nim do czynienia.
- Problemy z dokumentacją aplikacji. Obsługa bardziej złożonych, specjalistycznych aplikacji, ze względu na ich budowę, często nie jest zbyt intuicyjna. Kluczowe znaczenie ma w tym przypadku dokumentacja użytkownika.

7. Testowania

- Niewystarczające testy automatyczne. Testowanie automatyczne, wykonywane na bieżąco wraz z dostarczaniem kolejnych wersji kodu do repozytorium, umożliwia uzyskanie szybkich wyników dotyczących poprawności wprowadzonych zmian.
- Zbyt duża liczba testów ręcznych. Testy ręczne powinny być ograniczane do minimum, gdyż często zależą one od chwilowej dyspozycji i kompetencji testera.

8. Defektów

- Niepoprawiane błędy. Błędy, które nie są poprawiane, mogą prowadzić do narastania kolejnych nieprawidłowości podczas kodowania i testowania aplikacji.

9. Technologiczne

- Źle dobrana technologia. Wybór technologii dedykowanej konkretnemu rozwiązaniu jest istotnym czynnikiem wpływającym na sukces projektu. Niemniej

jednak w dobie coraz częściej pojawiających się konkurencyjnych rozwiązań, nierzadko tańszych, ciekawszych, zawierających dużo ulepszeń i gotowych komponentów decydujemy się sięgnąć po coś na tyle atrakcyjnego, że innowacyjność technologii przysłania nam potrzeby wynikające z projektu.

- Korzystanie z nieaktualnych narzędzi oraz oprogramowania. Zagadnienie to dotyczy narzędzi używanych do wytwarzania oprogramowania, których kolejne wersje dają nowe możliwości i ułatwienia, sprawiając, że praca staje się szybsza i łatwiejsza.

10. Braku wiedzy

- Brak doświadczenia związanego z określoną technologią. Brak doświadczenia sprawia, że zwiększa się podatność na popełnianie trywialnych błędów, które jednocześnie mogą być brzemienne w skutkach. Przykładem może być programista, który rozpoczął używanie nowego języka programowania. W początkowym okresie pracy z nową technologią może on nie być w pełni świadomy wszystkich jej atrybutów i różnic względem narzędzia, do którego był wcześniej przyzwyczajony.
- Brak świadomości długu technicznego. Brak świadomości długu technicznego powoduje, że działania osób uczestniczących w rozwoju oprogramowania powodują w sposób nieświadomy powstawanie długu technicznego. Zakłada się, że posiadanie świadomości na temat długu powodowałoby znaczne ograniczenie w jego powstawaniu.
- Brak wiedzy na temat ryzyka związanego z długiem technicznym. Różne rodzaje długu technicznego związane są z różną wielkością ryzyka projektowego. Zrozumienie ryzyka, które może wywołać konkretny dług techniczny, ma duże znaczenie dla osiągnięcia celu projektu.

Przedstawiona lista przyczyn powstawania i narastania długu technicznego może być rozbudowywana i uściślana. W artykule skoncentrowano się na dziesięciu czynnikach umożliwiających w sposób wiarygodny i efektywny przeprowadzenie ankiety i uzyskanie wartościowych wyników.

4. Konsekwencje zidentyfikowanych problemów długu technicznego

Podobnie jak w przypadku zwykłego długu finansowego, tak w odniesieniu do długu technicznego jego niespłacanie wiąże się z komplikacjami. O ile trudności takie są odpowiednio małe i dają się rozwiązać, zanim osiągnięty zostanie ostateczny, nieprzekraczalny termin wykonania produktu, sytuacja taka nie wywołuje większych perturbacji i można ją uznać za zupełnie normalną w toku produkcji. Czasami jednak gwałtowny, niekontrolowany przyrost długu sprawia, że projekty informatyczne kończą się fiaskiem. Braki w zarządzaniu oraz obsłudze długu technicznego sprawiają, że tak naprawdę nie jesteśmy świadomi jego wielkości. Należy bowiem

zaznaczyć, że dług techniczny przyrasta w sposób nieliniowy. Powyżej pewnego progu, który w literaturze został określony mianem punktu przegięcia [Rubin 2012], obsługa długu jest nierealna. Uwarunkowania sprawiają, że jakikolwiek postęp nie jest możliwy bez naprawy zastałej sytuacji, a jest to proces pochłaniający ogromne zasoby czasowe i finansowe. Jest kilka głównych przyczyn niepowodzeń projektów spowodowanych przez dług techniczny:

- Przekroczenie dostępnego budżetu. Wraz ze wzrostem długu technicznego skomplikowanie kodu rośnie, co sprawia, że wprowadzanie nawet najmniejszych zmian wymaga o wiele więcej pracy. Większa pracochłonność sprawia, że koszty rozwoju i naprawy produktu rosną. W celu zaoszczędzenia zawieszone zostają implementacje dodatkowych funkcjonalności oraz poprawa błędów. Ta sytuacja powoduje, że produkt staje się mało atrakcyjny, więc nie jest chętnie kupowany, a co za tym idzie − wpływy z jego sprzedaży maleją. Taki rozwój wypadków często sprawia, że budżet projektu zostaje przekroczony jeszcze przed jego ukończeniem, co może spowodować jego przerwanie. Dodatkowo proces wytwórczy i późniejsze utrzymanie całej infrastruktury (wsparcie, konserwacja kodu) może okazać się kosztowniejsze niż poniesione przychody.
- Czas zakończenia projektu. W związku z narastaniem długu technicznego duże trudności sprawia oszacowanie czasu potrzebnego na ukończenie projektu. Dług sprawia, że wytwarzanie oprogramowania się przedłuża, a przerwy między kolejnymi wydaniami produktu wzrastają. W obszarze IT szybkość wydania kolejnych wersji oprogramowania unowocześnionych o nowe funkcjonalności świadczy o konkurencyjności produktu. Opóźnienie wydania produktów innowacyjnych nawet o kilka tygodni może sprawić, iż przestaniemy być innowacyjni, a klienci wybiorą produkt konkurencji.
- Jakość oprogramowania. Skomplikowanie kodu oraz jego mała przewidywalność, spowodowane długiem technicznym, sprawiają, że znalezienie, a następnie poprawa błędów stają się trudne i czasochłonne. Z reguły w takich sytuacjach, w bezpośrednim okresie przed wydaniem produktu, rezygnuje się z ich poprawy, nie tylko z powodu braku czasu i środków, ale również z obawy przed wprowadzeniem dodatkowych usterek do kodu. Takie działania mogą prowadzić do tego, że wydany produkt staje się mało przewidywalny i trudny bądź niemożliwy w obsłudze. Duża liczba błędów świadczy o jakości produktu, a to znowu wpływa na liczbę sprzedanych licencji, a więc, co za tym idzie − na zyski. Dług techniczny, który związany jest z warstwą projektową czy też warstwą technologii, sprawia, że produkt może działać znacznie poniżej optymalnej wydajności. W przypadku rynków, w których konkurencja jest bardzo duża, aspekt wydajności produktu czy też nawet mało znaczących dodatkowych funkcjonalności może świadczyć o przewadze produktu względem konkurentów.
- Kolejne wydania produktu. Kolejne wydania zawierają zwykle poprawione błędy starszych publikacji oraz nowe funkcjonalności. Często takiemu planowaniu towarzyszy założenie, że projekt zacznie być rentowny dopiero wraz z wy-

daniem określonej wersji. Niespłacony i niezarządzany dług techniczny może sprawić, że wydanie kolejnej wersji nie będzie możliwe z powodu trudności w implementacji poprawek czy nowych funkcjonalności. Konieczne wtedy są działania naprawcze, polegające na poprawie kodu bądź architektury. Działanie takie jednak generuje kolejne koszty dla projektu, który nie przyniósł jeszcze żadnych zysków, a więc generowane są realne straty finansowe.

- Wizerunek firmy. Wszystkie wymienione wcześniej problemy nawet w sytuacji, gdy same w sobie nie są powodem niepowodzenia projektu informatycznego, powodują narastanie frustracji. Niezadowolenie narasta zarówno wśród klientów, jak i pracowników odpowiedzialnych za rozwój produktu. Po stronie programistów, projektantów czy też testerów narasta niezadowolenie spowodowane monotonią pracy wynikłą z ciągłego borykania się z przeszkodami. Wszystkie działania ukierunkowane są na poprawianie niedoskonałości kodu, nie są podejmowane żadne prace postępowe. Aktywność w takim trybie jest mało rozwijająca, nudna i męcząca. Należy podkreślić, że w sektorze IT jednym z głównych czynników motywacyjnych jest ciekawa praca, pozwalająca na doskonalenie umiejętności. Pracownicy uczestniczący w projektach przeciążonych długiem technicznym bardzo często uciekają z nich, poszukując nowych miejsc zatrudnienia. Fakt ten przysparza kolejnych kłopotów, wiążących się z rekrutacją nowych pracobiorców, ich przeszkoleniem i wdrożeniem, a przede wszystkim z odpływem doświadczonych pracowników. Także klienci, niezadowoleni z powodu wielu niedoróbek, opóźnień, niskiej jakości oprogramowania, tracą zaufanie do produktu. Czasami cała firma postrzegana jest przez ten pryzmat i traci na rzetelności.

5. Interesariusze długu technicznego

Przez pojęcie interesariusza rozumiemy osobę, która jest pod wpływem określonego przedsięwzięcia bądź ma na nie wpływ. Pierwsze przemyślenia na temat tego, kto najbardziej doświadcza problemów związanych z długiem technicznym, każą nam brać pod uwagę programistów i klientów, jednak problem jest o wiele bardziej złożony. Zidentyfikowano najważniejszych interesariuszy, na których dług techniczny ma wpływ lub którzy wpływają na narastanie długu technicznego:

Klienci. Klienci stanowią najbardziej obszerną grupę osób, która ma bardzo duże oddziaływanie na projekt. To oni są uwikłani w dług techniczny przez wiele czynników, takich jak: opóźnienie wydania produktu, duża liczba błędów czy możliwość szybkiego reagowania producenta na sugestie przez nich zgłoszone. Fakt ten decyduje o tym, że należy dbać o tę grupę osób w sposób szczególny i w jak najwydajniejszy sposób niwelować wszelkie wpływy długu technicznego [*Managing Technical Debt* 2015].

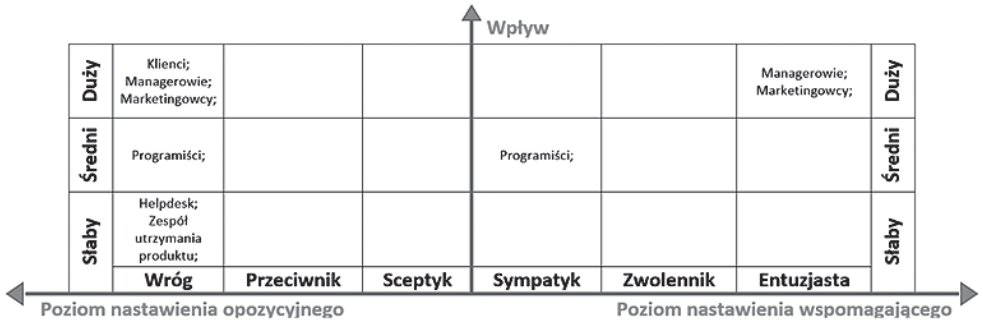
Programiści. Powszechna niska jakość kodu, spowodowana długiem technicznym, wywołuje w pewnym momencie zmęczenie zmaganiem się z niedoskonałościami

kodu i pociąga za sobą spadek wydajności pracy. Pracownicy czują się przeciążeni, a praca nie jest dla nich w pełni satysfakcjonująca. Naturalnym postępowaniem w takim przypadku jest rezygnacja z pracy i ucieczka do innego pracodawcy. Szkoda jest podwójna: tracimy doświadczonego pracownika, a na jego miejsce musimy zatrudnić nową osobę oraz wdrożyć ją do projektu, co zajmuje czas i zasoby [*Managing Technical Debt* 2015].

Zespoły utrzymania i wsparcia użytkowników oprogramowania. Możemy tutaj wyróżnić linię wsparcia technicznego. W produkcie, który został wydany z licznymi błędami, *helpdesk* jest szczególnie obciążony. Sytuacja taka może mieć konsekwencje długofalowe. Ogólna zależność jest taka, że im produkt ma więcej błędów, tym potrzebujemy większej liczby osób dostarczających wsparcie. Ta sama zasada dotyczy osób, które zajmują się utrzymaniem aplikacji. Dodatkowo, w przypadku wystąpienia błędów krytycznych, powstaje duża presja czasu na ich poprawę, a planowane konserwacje czy obsługa mniejszych, ale równie ważnych problemów (wydawanie łatek) zostaje odłożona w czasie [*Managing Technical Debt* 2015].

Kadra kierownicza. Menedżerowie oraz osoby odpowiedzialne za marketing to kolejne „ofiary" pojawienia się długu technicznego. Niedotrzymanie zobowiązań sprawia duże problemy pod względem wizerunkowym nie tylko danego produktu, ale często też całej firmy. Raz utracone zaufanie bardzo trudno odzyskać, szczególnie gdy problemy z dostarczeniem czy też jakością produktu powodują uszczerbek na finansach klienta. Produkt, który nie sprzedaje się zgodnie z wcześniejszymi przewidywaniami czy też wręcz przynosi straty, może nie tylko być kulą u nogi, ale również prowadzić do upadku firmy.

Już ta krótka lista interesariuszy wskazuje, że w problematykę długu technicznego zostaje uwikłanych o wiele więcej osób, niż to mogłoby wydawać się na pierwszy rzut oka. Ich zaangażowanie niepotrzebnie zużywa zasoby finansowe i czasowe.



Rys. 1. Diagram wpływu i nastawienia interesariuszy wobec długu technicznego

Źródło: opracowanie własne.

Ważnym zagadnieniem jest nastawienie interesariuszy wobec długu technicznego. Rysunek 1 przedstawia dualizm poparcia projektu w zależności od wpływu długu na

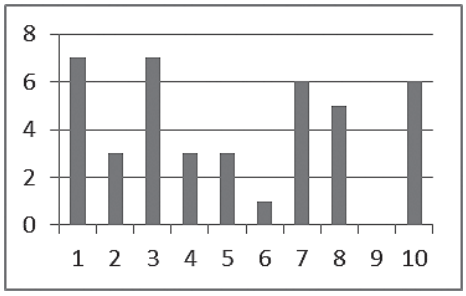
projekt. W zależności od wielkości długu można być jednocześnie zwolennikiem lub przeciwnikiem projektu.

6. Świadomość długu technicznego w zespole

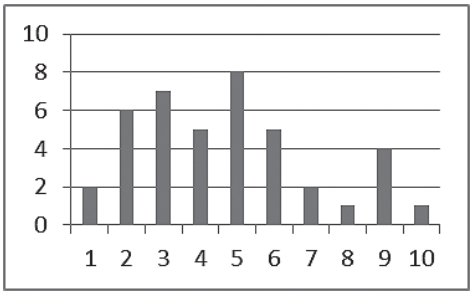
Zespoły programistyczne często dopuszczają się świadomych uchybień, zaciągając dług techniczny. Zdarza się, że implementują one funkcjonalności w taki sposób, jakby miały pozostać rozwiązaniami prototypowymi, wzorując się przy tym na rozwiązaniach, o których doświadczone osoby mogą kolokwialnie powiedzieć: „jakoś działające". Warto zauważyć, że dopiero gdy konieczna jest poprawa funkcjonalności, pojawia się pytanie, czy nie łatwiej byłoby podejść do tematu rzetelniej, na początku implementacji, może już na etapie analizy. Ostatecznie niewielu z nas potrafi uczyć się na cudzych błędach, a tzw. dobre praktyki stosowane są dopiero, gdy zetkniemy się z problemem osobiście lub gdy dostrzeżemy autorytet u chociażby jednego ze współpracowników. Zaufanie i właśnie autorytet lub, innymi słowy, dojrzałość zespołu wpływają na to, czy jego członkowie całościowo będą dbać o pryncypia, które sami ustalają podczas napotykanych problemów. Wypada zaznaczyć, że czas nadłożony na konsultację rozwiązań, analizę, poprawki czy tworzenie dokumentacji nie jest czasem straconym, wręcz odwrotnie. Jest to czas, który umożliwia zmniejszenie długu technicznego w późniejszych fazach rozwoju projektu.

Aby zobrazować istnienie problemu pobocznego, jakim jest brak wiedzy na temat długu technicznego, w sieci Internet została utworzona i następnie opublikowana ankieta przeprowadzona wśród 41 respondentów (3 kobiety, 38 mężczyzn). Dzięki niej otrzymano rezultaty przedstawiające zróżnicowane stanowiska osób z branży IT. Pracownicy zostali poproszeni o przedstawienie swego stanowiska na temat wiedzy i podejścia do długu technicznego. Warto zaznaczyć, że główną rolę w ankiecie odgrywają programiści, stanowią oni bowiem aż 64,1% respondentów, następnie są to analitycy, stanowiący 20,5% badanych osób, a także projektanci, stanowiący 5,1% ankietowanych, pozostali uczestnicy stanowili 10,3% ogółu. Średni okres zatrudnienia respondentów przekraczał 4 lata.

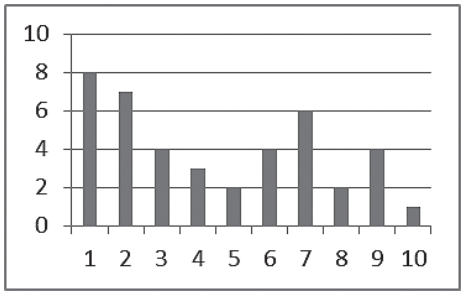
Pierwsze pytanie miało następujące brzmienie: „Jaki typ długu technicznego sprawia największe problemy w twoim otoczeniu? Posortuj rodzaje długu od sprawiających największe problemy (waga 1) do tych, które powodują najmniej problemów (waga 10)". Podane zostały następujące typy długów: oprogramowania, defektów, dokumentacyjny, testowania, użyteczności, biznesowy, techniczny, organizacyjny, projektowania, braku wiedzy. Wyniki ankiet przedstawiono na rys. 2-10. Podsumowanie wyników przedstawia rys. 11. Spośród wyszczególnionych typów długu ankietowani wyznaczyli dług użyteczności jako najbardziej istotny. Dług ten dotyczy braku analizy funkcjonalności, niedostatecznej znajomości konfiguracji środowiska klienta oraz problemów związanych z wdrażaniem oprogramowania. Dodatkowo, na podstawie załączonego diagramu, można stwierdzić, że najmniej istotnym długiem jest dług dokumentacyjny, dotyczący słabo udokumentowanego kodu (kłopot ze zrozumieniem



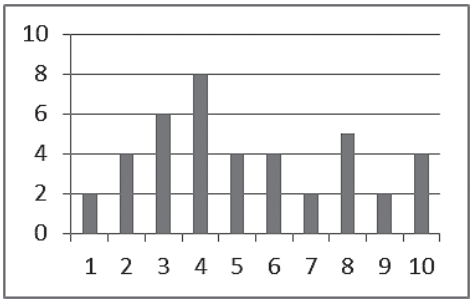
Rys. 2. Dług oprogramowania



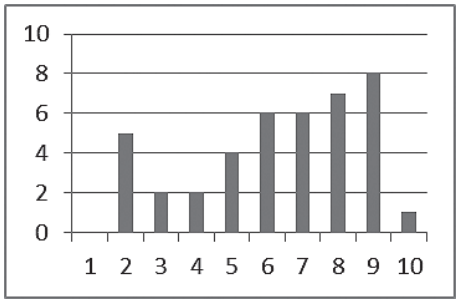
Rys. 3. Dług defektów



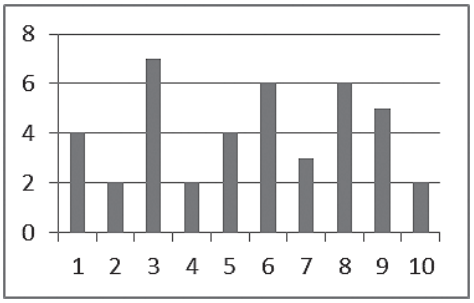
Rys. 4. Dług dokumentacyjny



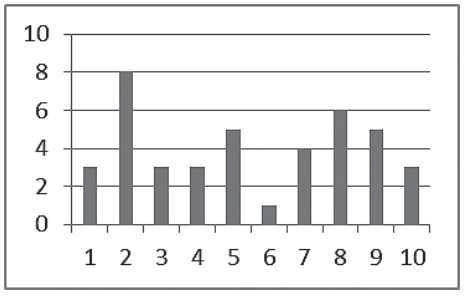
Rys. 5. Dług testowania



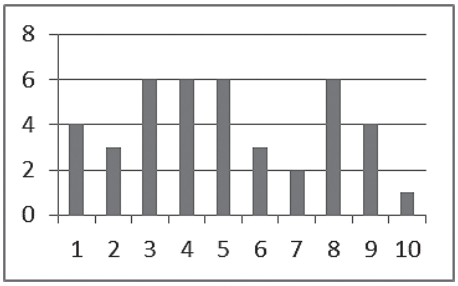
Rys. 6. Dług użyteczności



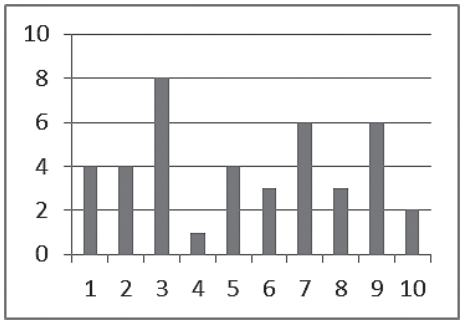
Rys. 7. Dług biznesowy



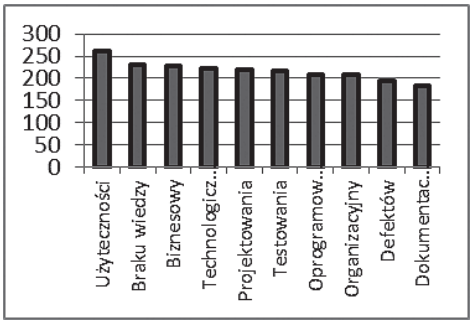
Rys. 8. Dług technologiczny



Rys. 9. Dług organizacyjny



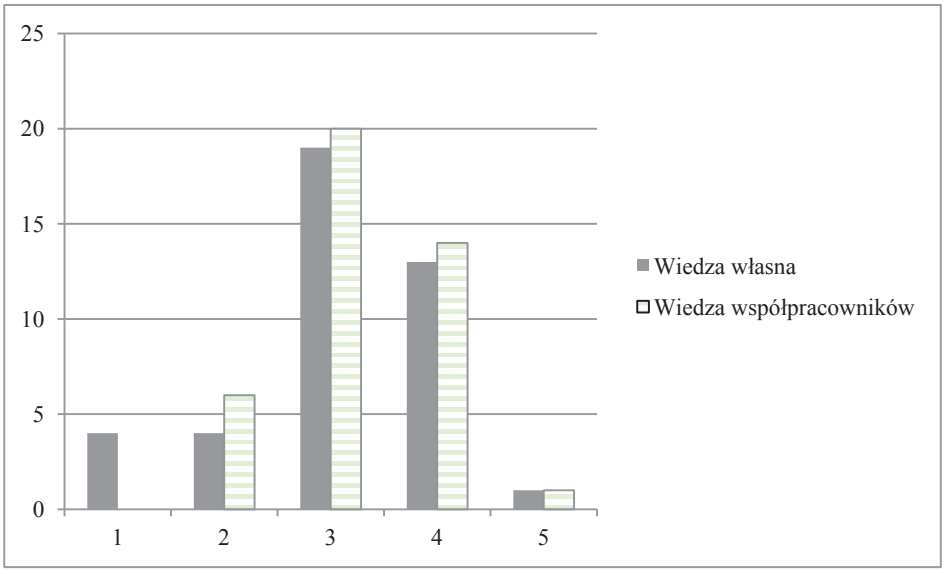
Rys. 10. Dług projektowania



Rys. 11. Sumaryczne dane ankiety

Źródło: opracowanie własne na podstawie zebranych wyników ankiet.

napisanego fragmentu oprogramowania). Prawdopodobnie wytypowanie tej kategorii jako mało istotnej wynika z coraz częściej stosowanego przez programistów przekonania, że kod jest kodem samodokumentującym się.



Rys. 12. Ocena respondentów dotycząca poziomu wiedzy o długu technicznym: własnej i współpracowników

Źródło: opracowanie własne na podstawie zebranych wyników ankiet.

Kolejne dwa pytania miały następujące brzmienie: „Jak określiłbyś poziom swojej wiedzy na temat długu technicznego?", „Jak określiłbyś poziom wiedzy twoich współpracowników na temat długu technicznego?". Uczestnicy ankiety swoją wiedzę

oceniali na średnim poziomie, podobne wyniki uzyskano na analogiczne pytanie dotyczące wiedzy współpracowników (rys. 12). Zestawiając powyższe fakty, można stwierdzić, że ankietowani nie wiedzą, jak ocenić swoją wiedzę. Z czego to może wynikać? Niejednokrotnie, aby móc ocenić obiektywnie stan jakiejkolwiek rzeczy, potrzebujemy wzorca lub punktu odniesienia.

Na pytanie: „Czy uważasz, że kwestie długu technicznego są pomijane w twoim otoczeniu?", blisko połowa ankietowanych stwierdziła, iż kwestie długu technicznego są pomijane. Z kolei w odpowiedzi na pytanie: „Czy twoi przełożeni kładą wystarczający nacisk na kwestie długu technicznego?", respondenci byli bardziej zgodni (59% ankietowanych), stwierdzając, że przełożeni nie kładą wystarczającego nacisku na kwestie długu technicznego. Uzyskany wynik pozwala zwrócić uwagę na zbyt małe zainteresowanie kierownictwa badanym tematem. Dodatkowym tego potwierdzeniem są odpowiedzi respondentów na pytania otwarte dotyczące przyczyny długu technicznego oraz perspektyw jego zmniejszenia.

Następne pytania dotyczyły przeprowadzania refaktoryzacji i sposobu wpływania na zmniejszenie długu technicznego w firmie. Respondenci w większości wyrazili opinię, że nie mają czasu na wykonywanie refaktoryzacji, twierdząc, że szybkość wykonywania projektów jest ważniejsza niż jakość wytwarzanego produktu. Pojawiły się również sugestie odnośnie do małych spotkań wewnątrz zespołu celem uniknięcia prostych błędów popełnianych przez osoby ze stosunkowo małym stażem pracy. Wśród poruszonych tematów pojawiły się kwestie braku znajomości technologii oraz niewykorzystywania narzędzi usprawniających refaktoryzację, jak Style-Cop, Git czy Mercurial. Niejednokrotnie osoby, które korzystają z wymienionych narzędzi, nie zdają sobie sprawy z ich szerszego zastosowania.

Podstawową kwestią w walce z długiem technicznym jest uświadomienie, że dług taki istnieje w każdym projekcie. Związane to jest z ciągłą zmianą wymagań, a co za tym idzie − z utrzymaniem spójnej architektury produktu. To, że nie jesteśmy świadomi problemów z długiem, świadczy tylko o tym, że albo nie natrafiliśmy jeszcze na ten typ problemu, albo po prostu nie jesteśmy w stanie poprawnie go zidentyfikować. Jednak tak naprawdę to nie jest szczególnie trudne, wystarczy bowiem uświadomić sobie, że takie podstawowe trudności, jak: brak wiedzy dziedzinowej z danego zakresu, trudności w implementacji nowych funkcjonalności spowodowane wewnętrzną strukturą aplikacji, korzystanie z przestarzałych bibliotek czy też duża liczba błędów, to symptomy zaciągniętego długu technicznego.

7. Dług techniczny w aspekcie ryzyka

Na dług techniczny możemy spojrzeć przez pryzmat zarządzania ryzykiem. W zależności od procesu rozwoju oprogramowania możemy mieć do czynienia z innego typu ryzykiem. Przykładem może być analiza ryzyka w przypadku stosowania oprogramowania typu *Open Source* [Kieruzel 2014], której wynikiem były oceny modelu ryzyka produkcji oprogramowania i w projektach informatycznych. Zacią-

ganie długu technicznego przypomina zaciąganie kredytu w bankach, dlatego warto uwzględnić rozwiązania sektora bankowego. W pracy [Siarka 2015] nakreślono spójną koncepcję polityki ryzyka kredytowego, wykazując, że efektywność systemu zarządzania ryzykiem kredytowym jest zależna od sposobu organizacji wewnętrznych procesów. Podobne zależności można odwzorować w procesach zarządzania projektami informatycznymi [Werewka, Lewicka, Zakrzewska-Bielawska 2012] z zastosowaniem metodyki PMBOK. W metodyce PMBOK [PMI 2013] wyróżniamy sześć procesów zarządzania ryzykiem; są nimi: planowanie zarządzania ryzykiem, identyfikacja ryzyka, jakościowa analiza ryzyka, ilościowa analiza ryzyka, planowanie odpowiedzi na ryzyko oraz monitorowanie i kontrola ryzyka. Podejście to rozpatrzymy pod kątem długu technicznego.

Podstawowym procesem PMBOK jest proces identyfikacji ryzyka. Aby zacząć spłacać dług techniczny, musimy najpierw go zidentyfikować. Problemy związane z długiem technicznym zazwyczaj ujawniają się same. W celu ich identyfikacji bardzo często wystarczy spotkanie z inżynierami odpowiedzialnymi za rozwój aplikacji. Inną grupą osób, które z łatwością mogą wskazać dług, są testerzy − zarówno podczas wykonywania testów automatycznych i ich analizy, jak i wykonywania testowania ręcznego wykrywane są błędy, które są miernikiem długu technicznego. Innym sposobem analizy i identyfikacji długu są metody automatyczne. Pamiętać jednak należy o tym, że, mimo że są to metody szybsze i dają wyniki, zanim jeszcze pojawią się pierwsze symptomy problemów, ich rezultaty zawsze powinny zostać sprawdzone przez osoby, które mają doświadczenie w rozwoju oprogramowania. Narzędzia do automatycznej analizy długu technicznego sprawdzają kod pod kątem kilku rodzajów uchybień. Przykładem może być np. analiza kodu pod względem jego pokrycia testami automatycznymi. Uznaje się, że pokrycie testami poniżej 75% kodu może prowadzić do poważnych problemów. Innymi metrykami mogą być m.in. złożoność cyklomatyczna kodu, cykliczne zależności, prawidłowe rozmiary klas i metod czy automatyczne sprawdzane dobrych praktyk kodowania. Narzędzia z opisanej grupy to m.in. Structure101 czy Sonar. Należy również podkreślić, że zagadnieniem automatycznego rozpoznawania długu technicznego zajmuje się wiele środowisk naukowych organizujących liczne konferencje oraz udostępniających liczne artykuły naukowe.

Proces planowania odpowiedzi na ryzyko zawiera działania osłabiające zagrożenia dla celów projektu. Można zaproponować różne strategie reakcji na ryzyko długu. Jedną z podstawowych jest zapobieganie powstawaniu ryzyka długu technicznego. W szczególności należy się wystrzegać zaciągania długu technicznego mimowolnego. Oprócz tego należy się postarać o wprowadzanie dobrych praktyk programistycznych związanych z samym kodowaniem, rozważyć możliwość zastosowania pisania kodu, dla którego wcześniej stworzone zostały testy jednostkowe (*Test-Driven Development* − TDD). Innym mechanizmem pomagającym w prewencji długu technicznego może być wykonanie analizy statycznej za pomocą inspekcji kodu, tzw. *code review*. Technika ta polega na tym, że zanim kod zostanie umieszczony w repozytorium, jest on przeglądany oraz oceniany przez współpracowników. Praktyką, której zawsze należy

przestrzegać, jest sumienne wykonywanie implementacji danej części funkcjonalnej. Nigdy nie wolno porzucać nieukończonego kodu. Związany z tym jest również jak najlepszy opis tego, co powinna zawierać kompletnie ukończona funkcjonalność. W przypadku porzucenia rozwoju określonej funkcjonalności należy wycofać z repozytorium cały jej kod. Nawet takie problemy, jak różne przyzwyczajenia w formatowaniu kodu czy też niedbałość w tej czynności, mogą zostać rozwiązane za pomocą narzędzi autoformatujących kod przed umieszczeniem go w repozytorium. Chociaż problem ten wydaje się trywialny, częste, niestaranne formatowanie może prowadzić do pomyłek czy też szybszego zmęczenia podczas jego analizy.

Ponieważ dług nie jest sam w sobie czyś pozytywnym, należy w marę możliwości stosować strategię unikania ryzyka. Istnieje wiele różnych rodzajów długów, które należy spłacać: od długu publicznego, przez długi finansowe oraz wiele innych, na długu wdzięczności kończąc. Istotny jest fakt, że przed każdym zadłużeniem można się zabezpieczyć w stopniu uzależnionym od naszych potrzeb. Tak samo jest z długiem technicznym − możemy go niwelować. Niestety decydenci często nie myślą perspektywicznie i nie decydują się na poprawę tego stanu, brak poprawy może być widziany bowiem jako „zysk". Wówczas chwilowy „zysk" powiększa dług techniczny, a skomasowane defekty oprogramowania lub zmiana w strukturze czy zachowaniu firmy są dużo bardziej kosztochłonne, niż mogły być na etapie rozpoznania problemu. Dodatkowo, aby zniwelować dług techniczny w organizacji, należy skupić się na takich elementach, jak: rozpropagowanie wiedzy na temat istoty długu technicznego, stworzenie środowiska pracy umożliwiającego przeprowadzanie refaktoryzacji, zadbanie o płynny przepływ informacji i o dobre relacje wewnątrz zespołu. Wymienione zasady są bardzo ogólne, lecz w znacznym stopniu powinny pomóc odpowiedzieć na pytanie, jak bardzo dbamy o to, by nasze zadłużenie nie było duże.

Nie każdy dług techniczny można natychmiast spłacić. Dlatego ważnym zagadnieniem jest monitorowanie i kontrola ryzyka. Dług, o którym wiemy, powinien być w jakiś sposób odnotowany, by mógł być on spłacony w późniejszym terminie i nie zaskoczył nas w najmniej oczekiwanym momencie. Na rynku istnieją narzędzia do śledzenia długu technicznego, jednak operacja ta może odbywać się również dzięki wykorzystaniu kartki papieru i długopisu. Podstawową grupą narzędzi do śledzenia części długu w postaci błędów są tzw. bugtrackery, czyli oprogramowanie do zapisywania oraz obsługi wszystkich błędów napotkanych w aplikacji. Oprogramowanie takie umożliwia zapis w jednym miejscu informacji o błędach, zarządzanie podziałem pracy i procesem naprawy błędu, statusem błędu oraz odczytem danych historycznych związanych z ogółem prac i informacji o błędzie. Najbardziej popularnymi narzędziami tego typu są m.in. Bugzilla, Jira czy Mantis. Oprócz tego można stworzyć osobny rejestr, którego jedynym zadaniem będzie śledzenie długu technicznego. W opisie poszczególnych składowych należy zawrzeć informację o rodzaju i umiejscowieniu długu, jego wpływie na produkt, szacowanej złożoności działań związanych z jego naprawą oraz o priorytecie jego naprawy. W przypadku zwinnych technik wytwarzania oprogramowania informacje takie, zamiast umieszczania ich w osobnym rejestrze,

można zawrzeć w tzw. rejestrze produktu. Należy tutaj nadmienić, że dobrą praktyką jest takie estymowanie zadań, by ich czas był zwiększony o około 5%. Powinno się zaplanować bufor czasowy przeznaczany na refaktoryzację kodu, który pozwala na spłacanie ewentualnego napotkanego długu technicznego.

Jedną ze strategii postępowania wobec ryzyka jest godzenie się na ryzyko i reakcja na wystąpienie ryzyka. Każdy dług finansowy powinien zostać spłacony, jednak w przypadku długu technicznego sprawa wygląda nieco inaczej. Zanim przystąpimy do jego spłaty, należy się zastanowić, czy w określonych warunkach jest ona konieczna. Bardzo często działania takie są jedynie stratą czasu i marnowaniem zasobów. Sytuacja taka ma miejsce w kilku przypadkach; są one następujące:

- Dług jest długiem strategicznym zaciągniętym w celu przyspieszenia prac czy też poprawy sytuacji budżetowej. W tej sytuacji spłacanie takiego długu niweczy plany jego zaciągnięcia.
- Produkt nie będzie już więcej rozwijany (co oznacza koniec życia produktu).
- Produkt zaprojektowany jest dla krótkiego czasu życia, a wartością kluczową jest szybkość dostarczenia go na rynek.
- Produkt został stworzony jedynie w celu wysondowania rynkowych preferencji użytkowników i nie będzie dalej rozwijany.
- Spłata długu jest kosztowniejsza (czasowo, budżetowo) niż dalsze rozwijanie produktu wraz z borykaniem się z problemami z niego wynikającymi.
- Dług można zniwelować przez jego konwersję – przykładem może być zamiana naszego kodu pełniącego określoną funkcjonalność na bibliotekę zewnętrzną, która − co prawda − nie rozwiązuje naszych problemów, jednak minimalizuje je w akceptowalny sposób.

Jeśli jednak okaże się, że nie zaszła żadna z powyższych przesłanek, należy rozpocząć spłacanie długu.

Podstawową czynnością pomagającą w regulowaniu długu jest wprowadzenie bufora dodatkowego czasu. Czas taki dodawany jest do każdego zadania, by w okresie jego trwania, przez refaktoryzację kodu, czyli takie jego przeorganizowanie i przeprojektowanie, by spełniał on nadal te same funkcje, ale był bardziej czytelny, niezawodny i łatwy w utrzymaniu oraz modyfikacji. Jeśli okaże się, że bufor jest zbyt mały, należy zgłosić problem wraz z jego dokładną estymacją oraz opisem do stworzonego wcześniej rejestru długu. Rejestr taki powinien być cały czas w polu uwagi osoby odpowiedzialnej za zarządzanie projektem, a spłacanie zawartego w nim długu powinno odbywać się w sposób cykliczny.

W przypadku metodyk zwinnych powinna zostać zawiązana nić porozumienia między kierownikiem wykonawczym projektu a osobą odpowiedzialną za definiowanie rejestru produktu. Menedżerowi należy uświadomić, jak ważna jest spłata długu. Idealna byłaby sytuacja, gdyby każda iteracja rozwoju oprogramowania miała nieznaczną część czasu poświęconą tylko i wyłącznie na spłatę istniejącego długu. Sposób wykorzystania tego czasu powinien należeć tylko i wyłącznie do decyzji członków zespołu. Przed wyborem rodzaju długu, jakim należy się zająć, bardzo

ważna jest priorytetyzacja problemów. W celu refaktoryzacji czy też wprowadzenia innego sposobu ich rozwiązania należy wybierać najpierw te, które mają największe oddziaływanie na produkt. Taki sposób działania sprawia, że największa liczba problemów zostaje rozwiązana na samym początku.

8. Strategie wykorzystania długu technologicznego

Dług technologiczny może być doskonałym narzędziem do uzyskania celów projektowych, które w normalnych warunkach byłyby celami nieosiągalnymi, bardzo kosztownymi, wymagającymi więcej zasobów bądź też nierealnymi do osiągnięcia w akceptowalnym czasie. W celu kalkulacji, czy zaciąganie długu technologicznego ma jakikolwiek sens oraz przede wszystkim czy jest ono w miarę bezpieczne, należy wykonać wstępne analizy. W poniższym przykładzie wykorzystana zostanie metoda analizy SWOT [SWOT TOWS 2015]. Tabele 1 i 2 zawierają szablon postępowania wobec długu technicznego dla firm rozwijających oprogramowanie. Tabela 1 zawiera ogólną propozycję szablonu dla silnych i słabych stron organizacji rozwijającej oprogramowanie oraz szans i zagrożeń płynących z otoczenia.

Tabela 1. Analiza SWOT dla wykorzystania długu technicznego

Mocne strony (S)	Słabe strony (W)
S1. Szybsze wydawanie produktu. S2. Dotrzymywanie terminów. S3. Szybsze rozpoczęcie zwrotu wydatków. S4. Opiniowanie produktu (*feedback*) przez potencjalnych klientów przy oszczędności na rozwoju lub nie w pełni wyspecyfikowanych funkcjonalności (projekty startupowe).	W1. Duża liczba błędów, a więc i poprawek. W2. Konieczność częstszej refaktoryzacji. W3. Brak spójności architekturalnej. W4. Problemy z estymacją (budżet, pracochłonność, czas). W5. Trudności w rozwoju i pielęgnacji kodu.
Szanse (O)	**Zagrożenia (T)**
O1. Wypuszczenie innowacyjnego produktu. O2. Wyprzedzenie konkurencji. O3. Umiejętne manipulowanie budżetem (relacja: koszty–przychód). O4. Możliwość rozwoju produktu z jego minimalnym budżetem początkowym (*crowdfunding*, startupy) – nawet do ponad 90% jego wartości. O5. Rozwiązanie problemów dotychczas trudnych bądź niemożliwych do rozwiązania (wykorzystanie nowych, mało znanych bądź ustabilizowanych technologii).	T1. Obciążenie linii wsparcia. T2. Przekroczenie budżetu. T3. Przekroczenie czasu realizacji. T4. Konieczność zaniechania implementacji części funkcjonalności (brak pokrycia zobowiązań pionu kierowniczego i marketingowego). T5. Utrata zaufania do produktu bądź firmy. T6. Spadek atrakcyjności produktu (przychodów). T7. Odejścia pracowników.

Źródło: opracowanie własne.

Natomiast tabela 2 zawiera propozycję ogólnych strategii postępowania wobec długu technicznego. Konkretna firma informatyczna, analizując swoje postępowanie wobec długu technicznego, powinna w pierwszym kroku dokonać adaptacji tab. 1, biorąc pod uwagę własne uwarunkowania. W kroku drugim należy przeprowadzić analizę SWOT. Na podstawie analizy należy wybrać odpowiednią strategię postępowania.

Tabela 2. Strategie postępowania wobec długu technicznego

Strategia agresywna (przewaga SO)	Strategia konserwatywna (przewaga WO)
• Rozwój aplikacji skoncentrowany na innowacyjności, implementacji nowych funkcjonalności oraz czasie dostarczenia produktu. • Brak dogłębnej analizy potrzeby zaciągania długu. Śmiałe zaciąganie długu mimowolnego oraz strategicznego. • Znikome zarządzanie długiem. • Spłata długu tylko w przypadku pojawienia się problemów. • Rozwinięty dział utrzymania i wsparcia. • Duża rotacja nowych technologii.	• Wystrzeganie się długu technicznego w jakiejkolwiek postaci. • Skłanianie się do renegocjacji kontraktów (termin realizacji, zbiór funkcjonalności), by dostarczać produkt bez konieczności zaciągania długu. • Duży nacisk na prewencję długu (szkolenia, TDD, *code review* itd.). • Długofalowe planowanie zarządzania i konsekwentnej spłaty długu.
Strategia konkurencyjna (przewaga ST)	Strategia defensywna (przewaga WT)
• Dogłębna analiza poprzedzająca zaciągnięcie długu strategicznego. • Unikanie długu mimowolnego. • Rozwinięte planowanie spłaty długu. • Priorytetyzacja dostarczenia produktu (czas, funkcjonalność) nad zaciąganiem długu.	• Preferowanie wydań częstszych, ale o mniejszym przyroście funkcjonalności (łatwiejsza kontrola długu). • Wyznaczenie nieprzekraczalnego poziomu długu (lokalny „punkt przegięcia"). • Jak najszybsza spłata długu (brak planu długoterminowego).

Źródło: opracowanie własne.

Przy stosowaniu strategii agresywnej należy pamiętać, że dług może być na tyle duży, że utrudnienia z nim związane sprawiają, iż dalszy rozwój projektu jest bardzo utrudniony bądź wręcz niemożliwy, a jedynym sposobem, aby kontynuować pracę jest wykonanie prac związanych tylko ze spłatą długu. Idealnym rozwiązaniem jest spłacanie długu podczas prac rozwojowych produktu. Spłata długu w ekstremalnych sytuacjach może przyjąć o wiele większe rozmiary. Wydawane są wtedy tzw. *cleanup-release* − produkty, które z punktu widzenia klienta niczym nie różnią się od poprzednich, jednak struktura wewnętrzna kodu została w takich przypadkach oczyszczona i przeprojektowana, a większość długu technicznego została zniwelowana. Czynności takie nie świadczą najlepiej o sposobie zarządzania projektem, z drugiej jednak strony są one wykonywane tylko w odniesieniu do produktów przynoszących naprawdę duże zyski, które warto utrzymywać na rynku i nadal rozwijać.

9. Podsumowanie

Podsumowując, warto jeszcze raz podkreślić, że koszty długu technicznego są tym bardziej dalekosiężne w skutkach, im dłużej mamy do czynienia z powielaniem błędów, brakiem przepływu informacji, ze złymi analizami. Niestety, problem ten nie jest łatwy do rozpoznania z dnia na dzień. Niejednokrotnie musimy się zmierzyć z poważnymi problemami i konsekwencjami takiego postępowania po czasie. Ostatecznie konieczne jest wyciąganie wniosków z niepowodzeń, a także należy pamiętać o zabezpieczaniu się przed podobnymi sytuacjami w celu ich uniknięcia. Podczas spłaty długu technicznego straty ponosi firma, kierownictwo, lecz również współpracownicy. Z powodu słabej organizacji kierowania rozwojem oprogramowania niejednokrotnie cierpi zespół, co często jest niewspółmiernym kosztem w stosunku do czasu, jaki można było poświęcić na organizację pracy (chociażby przez zwiększenie przepływu informacji, refaktoryzację lub zatwierdzanie kodu).

Rezultaty ankiety i obserwacje otoczenia pozwalają stwierdzić, że świadomość dotycząca występowania długu technicznego jest niska. W artykule wykazano, że kwestia długu technicznego dotyczy każdego uczestnika projektu, niezależnie od funkcji pełnionej przez niego w strukturze firmy.

Dług techniczny jest czymś, z czym osoby związane z projektami informatycznymi mają do czynienia na co dzień. Należy być świadomym faktu, iż choć nie zawsze dług można szybko zidentyfikować, zawsze jest on obecny, a zniwelowanie go do zera nie jest możliwe. Na uwadze powinno się mieć konieczność unikania zaciągania długu w nieodpowiedzialny sposób. Dług już zaciągnięty należy spłacać w sposób cykliczny, by nie dopuścić do sytuacji, w której uniemożliwia lub dezorganizuje on pracę. Należy również pamiętać o tym, że mimo wszystkich niebezpieczeństw, jakie niesie ze sobą zaciąganie długu technicznego, w sprawnych rękach jest on doskonałym narzędziem rozwoju oprogramowania.

Literatura

Alves N.S.R., Ribeiro L.F., Caires V., MendesT.S., Spinola R.O., 2014, *Towards an ontology of terms on technical debt, 2*014 Sixth International Workshop on Managing Technical Debt (MTD), s. 1-7, http://doi.org/10.1109/MTD.2014.9.

Code refactoring, hasło [w:] *Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Code_refactoring&oldid=682022427 (21.09.2015).

Cunningham W., 1992, *The WyCash Portfolio Management System*, http://c2.com/doc/oopsla92.html (6.01.2016).

Czym jest dług technologiczny? Jarosław Żeliński IT-Consulting, 2014, http://it-consulting.pl/autoinstalator/wordpress/2014/05/12/czym-jest-dlug-technologiczny/.

Da S Maldonado E., Shihab E., 2015, *Detecting and quantifying different types of self-admitted technical debt*, 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), s. 9-15, http://doi.org/10.1109/MTD.2015.7332619.

Fontana F.A., Ferme V., Zanoni M., Roveda R., 2015, *Towards a prioritization of code debt: A code smell intensity index,* 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), s. 16-24, http://doi.org/10.1109/MTD.2015.7332620.

Holvitie J., Leppanen V., Hyrynsalmi S., 2014, *Technical debt and the effect of agile software development practices on it − an industry practitioner survey*, 2014 Sixth International Workshop on Managing Technical Debt (MTD), s. 35-42, http://doi.org/10.1109/MTD.2014.8.

Kieruzel M., 2014, *Metoda oceny ryzyka realizacji oprogramowania do wspomagania działalności przedsiębiorstwa na przykładzie oprogramowania typu open source*, Informatyka Ekonomiczna, 1, http://doi.org/10.15611/ie.2014.1.20.

Li Z., Avgeriou P., Liang P., 2015, *A systematic mapping study on technical debt and its management*, Journal of Systems and Software, 101, s. 193-220, http://doi.org/10.1016/j.jss.2014.12.027.

Li Z., Liang P., Avgeriou P., 2015, *Architectural technical debt identification based on architecture decisions and change scenarios*, 2015 12th Working IEEE/IFIP Conference on Software Architecture (WICSA), s. 65-74, http://doi.org/10.1109/WICSA.2015.19.

Łucki Z., 2015, *Proszę... nie mówmy „technologia" na technikę*!, http://www.uci.agh.edu.pl/bip/63/11_63.htm (25.08.2015).

Managing Technical Debt, 2015, http://www.infoq.com/articles/managing-technical-debt (15.08.2015).

Martini A., Bosch J., 2015, *The danger of architectural technical debt: contagious debt and vicious circles*, 2015 12th Working IEEE/IFIP Conference on Software Architecture (WICSA), s. 1-10, http://doi.org/10.1109/WICSA.2015.31.

PMI, 2013, *A Guide to the Project Management Body of Knowledge: PMBOK(R) Guide* (5 edition), Newtown Square, Project Management Institute, Pennsylvania.

Rubin K.S., 2012, *Essential Scrum: A Practical Guide to the Most Popular Agile Process* (1 edition), Upper Saddle River, Addison-Wesley Professional.

Siarka P., 2015, *System informacyjny banku – integracja procesów zarządzania ryzykiem kredytowym*, Informatyka Ekonomiczna, 1, http://doi.org/10.15611/ie.2015.1.05.

Steve Garnett − Technical Debt: Strategies & Tactics for Avoiding & Removing It., 2013, http://blogs.ripple-rock.com/SteveGarnett/2013/03/05/TechnicalDebtStrategiesTacticsForAvoidingRemovingIt.aspx (15.08.2015).

SWOT TOWS, 2015, http://swottows.com/ (6.01.2016).

Technical debt, 2015, hasło [w:] *Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Technical_debt&oldid=694276930 (8.12.2015).

Werewka J., Lewicka D., Zakrzewska-Bielawska A.F., 2012, *Zarządzanie projektami w przedsiębiorstwie informatycznym: Metodologia i strategia zarządzania,* t. 1, Wydawnictwa AGH, Kraków.

Zazworka N., Izurieta C., Wong S., Cai Y., Seaman C., Shull F. i in., 2014, *Comparing four approaches for technical debt identification*, Software Quality Journal, 22(3), s. 403-426.