Dydaktyka Informatyki 16(2021)
ISSN 2083-3156; e-ISSN 2543-9847
DOI: 10.15584/di.2021.16.12
http://www.di.univ.rzeszow.pl

Uniwersytet Rzeszowski

Andrzej CHMIELEWSKI[1], **Stanisław JARZĄBEK**[2]

[1] *ORCID: 0000-0002-9313-0685. Dr inż., Politechnika Białostocka, Wydział Informatyki, ul. Wiejska 45A, 15-351 Białystok; e-mail: a.chmielewski@pb.edu.pl*
[2] *ORCID: 0000-0002-7532-3985. Prof. nadzw. dr hab., Politechnika Białostocka, Wydział Informatyki, ul. Wiejska 45A, 15-351 Białystok; e-mail:s.jarzabek@pb.edu.pl*

PROJEKTOWO-ZORIENTOWANE NAUCZANIE ZASAD INŻYNIERII OPROGRAMOWANIA

TEACHING SOFTWARE ENGINEERING PRINCIPLES IN A PROJECT-ORIENTED COURSE SETTING

Słowa kluczowe: zasady inżynierii oprogramowania, projektowo-zorientowane nauczanie, projekt zespołowy, iteracyjny rozwój programu, interfejsy komponentów programowych.

Keywords: software engineering principles, project-based learning, team projects, iterative development, programming interfaces.

Streszczenie

Typowy model nauczania opiera się na wiedzy przekazywanej studentom na wykładach, ćwiczeniach/laboratoriach, oraz ewaluacji studentów w formie testów i egzaminu końcowego. W przypadku inżynierii oprogramowania taki model nauczania nie zawsze jest efektywny. Pomimo pomyślnego wyniku egzaminu, w doświadczeniu autorów, studenci często w niewystarczającym stopniu transferują informacje nabyte podczas zajęć w wiedzę roboczą pozwalającą im na wykorzystanie jej w praktyce programowania, a nawet w kolejnych kursach, w których należy zastosować zdobytą wiedzę w innym kontekście. Aby temu zaradzić, do programu zajęć często włącza się wykonanie także projektów programistycznych. Autorzy niniejszego artykułu są zdania, że aby uzyskać lepsze wyniki należy zwiększyć wagę projektów w dwóch aspektach. Po pierwsze, treści teoretyczne i formalne uczone w części wykładowej kursu powinny być ściśle powiązane z pracą projektową, zarówno tematycznie, jak i czasowo. Po drugie, zaliczenie końcowe powinno być oparte na ewaluacji pracy projektowej, a egzamin pisemny może pełnić rolę pomocniczą. Autorzy wyjaśniają metodologię nauczania projektowo-zorientowanego na przykładzie kursów uczonych na tych zasadach przez ostatnie 15 lat na Narodowym Uniwersytecie w Singapurze (NUS) i Politechnice Białostockiej (PB).

Abstract

Typically, our courses include teaching lectures, tutorials/labs, and student evaluation in interim tests and final exams. For courses in which students supposed to learn practical application

of software engineering principles, such a teaching model not always yields satisfactory results: Passing an exam does not guarantee that students can transfer absorbed knowledge into their programming practice, or even use it effectively in follow up courses that require students to apply that knowledge in a new context. To counter this problem, educators often include substantial programming projects into their courses. It is authors' opinion that to get better teaching outcomes, It is important to enhance the role of projects in software engineering courses in two aspects. Firstly, lecture material should be tightly integrated and synchronized with the project work. Secondly, course evaluation should be based on evaluation of the project work, with written tests and final exams playing a complementary role. In the paper, authors motivate and explain their methodology to teach a project-oriented course based on a 15-year experience of teaching such courseat the National University of Singapore and Bialystok University of Technology.

Wstęp: inspiracja i motywacja

Motywacją dla stworzenia projektowo-zorientowanego modelu uczenia opisanego w niniejszym artykule były liczne obserwacje i publikacje świadczące o niezadowalającej skuteczności uczenia zasad inżynierii oprogramowania w tradycyjnej formule[1]. Jednym ze źródłem takich obserwacji były zewnętrzne opinie pracodawców, sygnalizujących nieumiejętność stosowania w praktyce zasad programowania, które studenci powinni znać. Istotnie, studenci zasady znali, ale nie byli w stanie zastosować tej wiedzy w nowej sytuacji, np. pod presją chwili. Po drugie, jeszcze w trakcie trwania studiów, często okazywało się, że studenci nie umieli stosować metod doskonale im znanych z podstawowych kursów, w tych bardziej zaawansowanych. Czasem wynikało to jedynie ze zmiany kontekstu i, na przykład, należało wyuczony koncept zobaczyć w nieco innym świetle. W innym przypadku mogło to wynikać ze zmiany skali problemu, gdzie tę samą metodę, czy zasadę należało zastosować dla problemu większego i w zmienionej formie niż to robili studenci w kursie podstawowym.

Aby zilustrować problem, przytoczymy naukę zasady ukrywania informacji (ang. *information hiding*)[2] podczas definiowania interfejsów programowania aplikacji API (ang. *Abstract Program Interface*). Jest to fundamentalna zasada w dzisiejszym programowaniu umożliwiająca kontrolę złożoności dużych programów, wymianę danych pomiędzy systemami informatycznymi oraz ułatwiającą podział zadań w zespołach w czasie rozwoju oprogramowania. Dzięki niej

[1] D. Dzvonyar, L. Alperowitz, D. Henze, B. Bruegge, *Team Composition in Software Engineering Project Courses*, *2018 IEEE/ACM International Workshop on Software Engineering Education for Millennials (SEEM)*, Gothenburg, 2018, pp. 16–23; C. Ghezzi, M. Jazayeri, D. Mandrioli, *Fundamentals of Software Engineering*, Prentice Hall, 2002; D.O. Zmeev, O.A. Zmeev, *Project--Oriented Course of Software Engineering Based on Essence*, 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T), Munich, Germany, 2020, pp. 1–3.

[2] D. Parnas, *On the Criteria To Be Used in Decomposing Systems into Modules*, Communications of the ACM, Vol. 15, No. 12, December 1972, p. 1053–1058.

możemy modyfikować jeden moduł programowy bez konieczności śledzenia wpływu zmian na pozostałe składniki systemu. Już w początkowych kursach dotyczących języków obiektowych studenci zaznajamiają się z interfejsami klas programowych widocznymi (ang. *public*) oraz niewidocznymi (ang. *private*) dla reszty programu. Na dalszych etapach nauki mechanizm interfejsów stosuje się do komponentów wyższego poziomu, obejmujących wiele klas programowych. Po wyjaśnieniu koncepcji takiego ogólnego API, ze zdziwieniem stwierdzaliśmy, że studenci nie widzą związku pomiędzy nowym spojrzeniem na API a dobrze im znanymi interfejsami klasowymi.

Powyższe frustrujące obserwacje zainspirowały autorów do przeanalizowania przyczyn zjawiska i szukania bardziej efektywnego modelu praktycznej nauki stosowania zasad inżynierii programowania.

Pierwszym wyzwaniem z tym związanym jest skala problemów, nad jakimi studenci pracują. Krótkie zadania programowe można zwykle z powodzeniem rozwiązać bez ich stosowania. Jeśli mimo to wymagamy od studentów, aby je stosowali, nie zostawi to w ich doświadczeniu przekonującego śladu, że zasady te są rzeczywiście istotne i potrzebne. Pozostanie wrażenie, że stosuje się je jedynie po to, aby otrzymać dobrą ocenę za zadanie, a nie po to, aby efektywnie uporać się z danym praktycznym problemem programistycznym. Nawet po zdaniu egzaminu student nie wyniesie wiele na przyszłość.

W odpowiedzi na to wielu koordynatorów przedmiotów włącza projekt w program zajęć inżynierii programowania. Ma on dać studentom lepszą okazję do zastosowania zasad w bardziej realistycznej sytuacji. Podstawą rozliczenia kursu staje się projekt i egzamin sprawdzający wiedzę teoretyczną nabytą w czasie kursu.

W doświadczeniu autorów, niezbędnym elementem powodzenia jest ścisłe zsynchronizowanie pracy projektowej z wykładami i ewaluacją kursu[3]. Wykłady powinny być zsynchronizowane z projektem zarówno czasowo, jak i w treści.

W pozostałej części artykułu opiszemy taki model nauczania zasad inżynierii programowania oraz doświadczenia z 15 lat uczenia i rozwijania formuły kursu.

Model powyższego kursu w początkowych latach jego uczenia na studiach podstawowych NUS został opisany w: S. Jarząbek, *Teaching Advanced…* Od tego czasu uczyliśmy go dla studentów drugiego stopnia na PB, znacznie wzbogacając formułę kursu i wprowadzając narzędzia umożliwiające lepszą ocenę stopnia, w jakim studenci opanowali praktykę stosowania zasad inżynierii programowania.

---

[3] S. Jarząbek, *Teaching Advanced Software Design in Team-Based Project Course*, 26th IEEE-CS Conf. on Software Engineering Education and Training (CSEET), San Francisco, May 2013, p. 35–44; P. Robillard, *Teaching Software Engineering through a Project-Oriented Course*, Proc. Conf. on Software Engineering Education, CSEE'96, 1996, p. 85–94.

Przegląd kursu

Celem kursu jest przekazanie studentom wiedzy dotyczącej praktycznego stosowania pryncypiów projektowania inżynieryjnego[4] oraz testowania systemów programowych. Studenci uczą się projektowania architektur oraz API, rozpatrywania alternatywnych rozwiązań i uzasadniania wybranych decyzji projektowych, przeprowadzania testów czarnej i białej skrzynki, weryfikacji specyfikacji wymagań, narzędzi do automatycznego testowania, tworzenia planu testów, raportowania wyników z testowania oraz planowania i realizacji iteracyjnego procesu rozwoju programów. W kontekście takich technicznych realiów studenci nabywają umiejętności pracy w zespole, skutecznego komunikowania się – na piśmie i w dyskusji – z innymi członkami zespołu projektowego.

W różnych okresach i z nieco inną treścią, kurs, który tu opiszemy, realizowany był zarówno na studiach pierwszego, jak i drugiego stopnia (i ich odpowiednikach na NUS) na kierunku informatyka, w wydaniu skondensowanym jednosemestralnym lub dwusemestralnym, z różnym wymiarem godzinowym wykładów i form praktycznych. Nieodmiennie jednak trzonem kursu pozostawał duży – jak na warunki uniwersyteckie – projekt, który studenci wykonują przez cały czas trwania kursu w zespołach projektowych liczących 4–6 studentów. Wykłady (4 godziny tygodniowo) odbywają się jedynie w pierwszej części kursu. Wyjaśniają problem, nad jakim studenci pracują w projekcie i metody inżynierskie, jakie powinny być zastosowane. Treść wykładu oscyluje wokół przykładów związanych z projektem. Przez pierwsze cztery tygodnie kursu studenci również odbywają ćwiczenia, w trakcie których oswajają się z problemem projektowym i metodami, jakie mają stosować w jego realizacji. Po tym okresie studenci mają już niezbędne przygotowanie do systematycznej pracy nad projektem. Ćwiczenia zostają zastąpione zajęciami projektowymi, w trakcie których studenci dyskutują z instruktorem konkretne problemy napotkane w ich pracy.

Ponadto, studenci muszą zaliczyć dwa testy, w trzecim i piątym tygodniu kursu. Sprawdzają one znajomość wymagań systemu, nad którym studenci będą pracowali oraz metodami inżynierskimi, które będą stosowali. Celem testów jest upewnienie się, że wszyscy studenci będą w stanie czynić wartościowe wkłady do prac zespołu. Studenci korzystają z podręcznika napisanego specjalnie dla naszego kursu, opisującego problem projektowy (opisany w rozdziale 3) i metody inżynierskie obowiązujące w projekcie[5].

Iteracyjny proces rozwoju projektu bazuje na hybrydowym modelu z przeważającymi elementami podejścia zwinnego (ang. *agile*), ale ze sporą dozą

[4] C. Ghezzi, M. Jazayeri, D. Mandrioli, *Fundamentals of Software Engineering*, Prentice Hall, 2002.

[5] S. Jarząbek, *Software Engineering Project*, Pearson Education Asia Pte Ltd, 2012.

wstępnej analizy wymagań i projektowania architektury systemu. Projekt obejmuje cztery iteracje, przy czym pierwsze trzy są obligatoryjne dla wszystkich studentów, a czwarta zawiera rozszerzenia wymagań, za których implementacje studenci mogą uzyskać wyższą ocenę.

Projekt, jaki studenci dostarczają na koniec semestru obejmuje program (ok. 10,000 linii kodu w C++) i dokumentację architektury systemu, ze szczegółowym wykazem decyzji projektowych.

Kluczowa jest ewaluacja kursu, gdyż określa ona cel, do którego mają zdążać studenci. Na zakończenie prezentowane są projekty, przy czym każdy z członków zespołu jest egzaminowany osobno ze swoich osiągnięć projektowych i materiału teoretycznego, jaki kurs obejmuje. Jakość programów jest podstawowym kryterium oceny pracy zespołu, ponieważ świadczy ona o tym na ile opanowali zasady inżynierii programowania nauczane w kursie. Specyfika problemu programowego (rozdział 3) i rozmiar kodu powodują, że bardzo trudno jest osiągnąć wysoką jakość programu nie stosując tych zasad. Programy studentów testujemy przy pomocy dedykowanego do kursu narzędzia testowania regresywnego zwanego TRAcker, w oparciu o bibliotekę zawierającą ponad 500 testów. Automatyczna analiza wyników testów TRAcker'a pozwala nam skutecznie zaobserwować, które funkcjonalności pracują zgodnie z wymaganiami, a które zawierają błędy, i ocenić ich wagę.

Motywację dla stworzenia powyższego modelu nauczania zasad inżynierii programowania autorzy czerpali z pozytywnych doświadczeń z problemowo-zorientowanego uczenia[6], nowatorskiego kształcenia studentów medycyny na Uniwersytecie McMaster w oparciu na pracy w małych zespołach i wcześniejszych próbach wplatania projektu w cykl nauczania inżynierii oprogramowania.

Modelowy problem – Statyczny Analizator Programów (SAP)

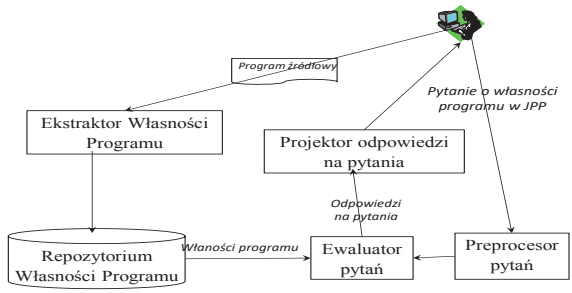
Zagadnienie, nad którym studenci pracują w projekcie, zdefiniowaliśmy specjalnie na potrzeby kursu. Z uwagi na praktyczny charakter tych zajęć, zadbaliśmy o to, aby rozwiązywany problem nie tylko odpowiadał metodyce kursu, ale również był istotny z punktu widzenia praktyki programowania.

W czasie zajęć studenci rozwijają narzędzie zwane Statycznym Analizatorem Programów (SAP)[7] (ogólny schemat jego działania przedstawia rys. 1). Pomaga on programistom zrozumieć program w czasie jego utrzymywania.

---

[6] P. Robillard, *Teaching Software Engineering through a Project-Oriented Course*, Proc. Conf. on Software Engineering Education, CSEE'96, 1996, s. 85–94.

[7] S. Jarząbek, *Design of Flexible Static Program Analyzers with PQL*, IEEE Transactions on Software Engineering, March 1998, p. 197–215.



Rys. 1. Schemat działania Statycznego Analizatora Programów (SAP)

Źródło: S. Jarząbek, *Design of Flexible Static Program Analyzers with PQL*, IEEE Transactions on Software Engineering, March 1998, p. 197–215.

SAP analizuje wstępnie program źródłowy i tworzy opis własności programu w postaci relacji pomiędzy elementarnymi jednostkami programu takimi jak: procedury, funkcje, metody, klasy, zmienne, czy instrukcje programu. Przykładami relacji są: relacja wywołania pomiędzy procedurami oraz relacja modyfikacji, czy też użycia, pomiędzy procedurami i zmiennymi. Programista formułuje pytania o interesujące go własności programu źródłowego w półformalnym Języku Pytań Programowych (JPP). SAP odpowiada na pytania odwołując się do repozytorium własności programu.

W poniższych przykładach zapytań znajdujemy odwołania do relacji opisujących własności programu, takich jak: *Calls* – wywołania procedur, *Modifies* – zmiennych modyfikowanych w procedurach, *Next* – przepływ kontroli w programie, i *Affects* – przepływ danych.

P1. **Select** *q* **suchthat** *Calls* (“P”, *q*),

P2. **Select** *v* **such that** *Modifies* (“P”, *v*),

P3. **Select** BOOLEAN **such that** *Next* (20, 100),

P4. **Select** *s* **suchthat** *Affects*(10, *s*),

P5. **Select** *q* **such that** *Calls* (*“P”*, *q*) **and** *Modifies* (*q*, “*x”*)

Architektura SAP daje szerokie pole dla stosowania zasad inżynierskich, w szczególności tworzenia i używania interfejsów komponentów programowych API.

Iteracyjny proces rozwoju programu

Analiza problemu SAP i projekt architektury programu (którego centralnym punktem jest definiowanie interfejsów API) zajmuje pierwsze cztery tygodnie

kursu. W tym czasie studenci prototypują swój program, a systematyczna praca nad implementacją zaczyna się w piątym tygodniu (pierwsza iteracja). Zakres każdej iteracji zostawiamy do decyzji studentów, choć dajemy im wskazówki mające na celu uniknięcie błędów, które mogą zbytnio opóźnić prace nad projektem.

Natura problemu SAP zachęca do horyzontalnego cięcia systemu tak, że każda iteracja dotyka, w rozmaitym stopniu w różnych iteracjach, każdego z głównych podsystemów SAP pokazanych na Rys. 1. Na końcu każdej iteracji studenci otrzymują funkcjonalny mini system, który testują regresywnie przy pomocy TRAckera pod kątem wymagań SAP.

Architektura SAP wraz z interfejsami API, opracowana w początkowych tygodniach kursu, daje gwarancję, że studenci bez zbytnich trudności przechodzą z jednej iteracji do następnej, w razie potrzeby poprawiając decyzje projektowe (np. w zakresie wyboru struktur danych w Repozytorium Własności Programu).

Język implementacji

Preferowanym językiem implementacji projektu jest C++. SAP jest programem systemowym, który wymaga dużej efektywności i skalowalności (program źródłowy, jaki SAP analizuje, może składać się z setek tysięcy rozkazów). Takie programy najczęściej tworzy się w warunkach przemysłowych przy użyciu C lub C++. Z uwagi na bogate API zaprojektowane dla Repozytorium Własności Programu, C++ pasuje lepiej do problemu niż C, ze względu na konstrukcje wspomagające tworzenie kolejnych interfejsów.

Dodatkowym argumentem jest potrzeba ekspozycji studentów do języków oferujących pełny asortyment zaawansowanych konstrukcji programowych, jakim jest C++. Studenci najczęściej wiedzą o istnieniu pełnego wachlarza konstrukcji programowych z wcześniejszych zajęć przeglądowych dotyczących języków programowania. Jednakże, w naszym kursie studenci mają okazję stosować te konstrukcje w kontekście problemu programowego realistycznych rozmiarów. Jest więc szansa, że wiedzę tak nabytą studenci wyniosą na stałe i będą stosowali ją w pracy zawodowej.

W początkowym okresie oferowania kursu studenci tworzyli projekt w języku Java. Był to jednosemestralny kurs na trzecim roku studiów w NUS z podwójnym kredytem (odpowiednikiem ECTS). Studenci nie mieli wcześniejszej ekspozycji do C ani C++, a pierwszym ich językiem programowania był Scheme. Zachodziła więc obawa, że studenci nie będą w stanie opanować języka C++ i wykonać zadania projektowego w tak krótkim czasie. Po pięciu latach

uczenia kursu nastąpiły zmiany w programie studiów na wczesnych latach, które zachęciły nas do sprawdzenia, jak studenci poradzą sobie z językiem C++. Dodatkowym elementem zachęty do podjęcia tego kroku były konsultacje z firmami pełniącymi role naszych doradców przemysłowych. Eksperyment udał się i to nadspodziewanie dobrze. Pozytywnym zaskoczeniem była poprawa jakości programów napisanych C++ w porównaniu z tymi implementowanymi w języku Java.

Uwagi końcowe

Pomimo że studenci implementują podobny system, w kolejnych latach, w których oferujemy kurs, nie spotykamy się z plagiaryzmem. Po pierwsze, sprawdzamy podobieństwo kodu w bieżących i poprzednich projektach przy pomocy detektora klonów programowych zwanego Clone Miner[8] (ang. *clone detector*). Clone Miner znajduje nie tylko identyczne fragmenty i struktury kodu, ale też takie, które zostały zmodyfikowane. Po drugie, w cotygodniowych konsultacjach prowadzący mają bliski kontakt z zespołami, dyskutując rozwiązania koncepcyjne i programowe. „Zapożyczenie" większych komponentów rozwiązania SAP byłoby bez trudu zauważone w czasie konsultacji, natomiast mniejszych komponentów rozwiązania „zapożyczać" się nie opłaca, ponieważ więcej pracy by zajęła ich integracja z resztą projektu niż napisanie tych komponentów na nowo. Po trzecie, na końcu projektu testujemy programy przy pomocy narzędzia regresywnego testowania z bogatym asortymentem metod analizy wyników testów. „Zapożyczone" rozwiązania wykazałyby się znacznym podobieństwem wyników testowania i można by je łatwo wykryć.

Bibliografia

Basit H.A., Jarząbek S., *Data Mining Approach for Detecting Higher-level Clones in Software*, IEEE Trans. on Soft. Eng., July/August 2009, Vol. 35, No. 4, Published online January 2009.

Dzvonyar D., Alperowitz L., Henze D., Bruegge B., *Team Composition in Software Engineering Project Courses, 2018 IEEE/ACM International Workshop on Software Engineering Education for Millennials (SEEM)*, Gothenburg 2018.

Ghezzi C., Jazayeri M., Mandrioli D., *Fundamentals of Software Engineering*, Prentice Hall, 2002.

Jarząbek S., *Teaching Advanced Software Design in Team-Based Project Course*, 26th IEEE-CS Conf. on Software Engineering Education and Training (CSEET), San Francisco, May 2013.

[8] D.O. Zmeev, O.A. Zmeev, *Project-Oriented Course of Software Engineering Based on Essence*, 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T), Munich, Germany, 2020, p. 1–3.

Jarząbek S., *Software Engineering Project*, Pearson Education Asia Pte Ltd, 2012.
Jarząbek S., *Design of Flexible Static Program Analyzers with PQL*, IEEE Transactions on Software Engineering, March 1998.
Robillard P., *Teaching Software Engineering through a Project-Oriented Course*, Proc. Conf. on Software Engineering Education, CSEE'96, 1996.
Parnas D., *On the Criteria To Be Used in Decomposing Systems into Modules*, Communications of the ACM, Vol. 15, No. 12, December 1972.
Zmeev D.O., Zmeev O.A., *Project-Oriented Course of Software Engineering Based on Essence*, 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T), Munich, Germany, 2020.