SYLABUS

DOTYCZY CYKLU KSZTAŁCENIA 2025/2026-2028/2029

ROK AKADEMICKI 2027/2028

1. PODSTAWOWE INFORMACJE O PRZEDMIOCIE

Nazwa przedmiotu	*Programowanie zespołowe*
Kod przedmiotu*	
Nazwa jednostki prowadzącej kierunek	*Instytut Informatyki, Wydział Nauk Ścisłych i Technicznych*
Nazwa jednostki realizującej przedmiot	*Instytut Informatyki, Wydział Nauk Ścisłych i Technicznych*
Kierunek studiów	*Informatyka i ekonometria*
Poziom studiów	*studia pierwszego stopnia*
Profil	*praktyczny*
Forma studiów	*studia stacjonarne*
Rok i semestr/y studiów	*rok III, sem. VI*
Rodzaj przedmiotu	*przedmiot kierunkowy*
Język wykładowy	*polski*
Koordynator	*dr inż. Przemysław Wiktor Pardel*
Imię i nazwisko osoby prowadzącej / osób prowadzących	*dr inż. Przemysław Wiktor Pardel*

1.1. Formy zajęć dydaktycznych, wymiar godzin i punktów ECTS

Semestr (nr)	Wykł.	Ćw.	Konw.	Lab.	Sem.	ZP	Prakt.	Inne (jakie?)	**Liczba pkt. ECTS**
6	15			15					2

1.2. Sposób realizacji zajęć

Zajęcia w formie tradycyjnej

1.3. Forma zaliczenia przedmiotu (z toku) (egzamin, zaliczenie z oceną, zaliczenie bez oceny)

Zaliczenie z oceną

2. WYMAGANIA WSTĘPNE

Znajomość zagadnień z przedmiotów: programowanie obiektowe, inżynieria oprogramowania, algorytmy i struktury danych, bazy danych

3. CELE, EFEKTY UCZENIA SIĘ, TREŚCI PROGRAMOWE I STOSOWANE METODY DYDAKTYCZNE

3.1. Cele przedmiotu

C1	Poznanie najnowszych narzędzi tworzenia oprogramowania (które mają być używane przez studentów na zajęciach laboratoryjnych).
C2	Nauka organizacji pracy zespołu informatycznego w procesie tworzenia, serwisowania i wdrażania oprogramowania
C3	Wykonanie przez studentów złożonego, praktycznego projektu informatycznego w grupie.

3.2. Efekty uczenia się dla przedmiotu

EK (efekt uczenia się)	Treść efektu uczenia się zdefiniowanego dla przedmiotu	Odniesienie do efektów kierunkowych
EK_01	Student ma wiedzę i umiejętności z języka programowania orientowanego obiektowo umożliwiającą realizację projektów.	K_W03, K_U01
EK_02	Student zna i potrafi użyć wybrane narzędzia zespołowego wytwarzania oprogramowania.	K_W03, K_U01, K_K01
EK_03	Student rozumie rolę dokumentacji technicznej zadania informatycznego i ma podstawową wiedzę na temat tworzenia dokumentacji technicznej tworzonego oprogramowania w wybranym narzędziu jej tworzenia.	K_W03, K_U13
EK_04	Student potrafi pracować w zespole nad wspólnym projektem.	K_U01, K_U09, K_U10, K_ U12, K_U15, K_K01

3.3. Treści programowe

A. Problematyka wykładu

1. Modele i metodyki wytwarzania oprogramowania: Agile, Scrum, Kanban, DevOps, Continuous Integration/Delivery.
2. Zarządzanie projektem programistycznym: role w zespole, planowanie sprintów, estymacja, zarządzanie backlogiem i ryzykiem.
3. Współpraca i komunikacja w zespole developerskim: code review, pair programming, narzędzia komunikacji (Slack, Teams, Discord, itp.), kultura feedbacku.
4. Systemy kontroli wersji i zarządzanie repozytorium: Git, GitHub/GitLab, strategie pracy z gałęziami (Git Flow, trunk-based development).
5. Jakość oprogramowania i testowanie: testy jednostkowe, integracyjne, automatyzacja testów, code coverage, CI pipelines.
6. Narzędzia wspomagające rozwój oprogramowania: IDE, systemy do zarządzania zadaniami (Jira, Trello), CI/CD (GitHub Actions, Jenkins), konteneryzacja (Docker).
7. Wdrażanie, utrzymanie i rozwój oprogramowania: deployment, monitoring, dokumentacja techniczna, DevOps w praktyce, feedback od użytkowników i iteracyjny rozwój.

B. Problematyka ćwiczeń , konwersatoriów, laboratoriów, zajęć praktycznych

1. Ustalenie 3–5-osobowych zespołów projektowych, wybór liderów oraz tematów projektów.
2. Praca z systemami kontroli wersji (Git, GitHub/GitLab) – tworzenie i utrzymanie repozytorium.
3. Organizacja projektu zespołowego w metodyce Scrum – backlog, sprinty, role, tablica zadań.
4. Określenie architektury systemu i podziału ról oraz zadań w zespole.
5. Tworzenie dokumentacji projektowej (README, wiki, backlog techniczny).
6. Konfiguracja środowiska programistycznego i współdzielenie ustawień w zespole.
7. Praca z gałęziami (branch), merge requestami i code review.
8. Projektowanie interfejsu użytkownika i/lub API systemu.
9. Implementacja modułów projektu zgodnie z przyjętą architekturą.
10. Tworzenie testów jednostkowych i integracyjnych.
11. Integracja modułów, rozwiązywanie konfliktów kodu, analiza jakości.
12. Przygotowanie wersji demonstracyjnej aplikacji i wdrożenie środowiska testowego.
13. Publiczna prezentacja projektów – pokaz działania, omówienie rozwiązań technicznych.
14. Retrospektywa projektu – analiza sukcesów, błędów i doświadczeń zespołu.
15. Zaliczenie projektów – ocena aplikacji, dokumentacji i pracy zespołowej.

3.4. Metody dydaktyczne

Wykład z prezentacją multimedialną, dyskusja

Laboratorium: wykonywanie ćwiczeń tablicowych i programistycznych, dyskusja,

4. METODY I KRYTERIA OCENY

4.1. Sposoby weryfikacji efektów uczenia się

Symbol efektu	Metody oceny efektów uczenia się (np.: kolokwium, egzamin ustny, egzamin pisemny, projekt, sprawozdanie, obserwacja w trakcie zajęć	Forma zajęć
EK_ 01	kolokwium zaliczeniowe / obserwacja w trakcie zajęć / projekt / test	wykład, laboratorium
EK_ 02	kolokwium zaliczeniowe / obserwacja w trakcie zajęć / projekt / test	wykład, laboratorium
EK_ 03	kolokwium zaliczeniowe / obserwacja w trakcie zajęć / projekt / test	wykład, laboratorium
EK_ 04	obserwacja w trakcie zajęć / projekt	wykład, laboratorium

4.2. Warunki zaliczenia przedmiotu (kryteria oceniania)

1. Zaliczenie przedmiotu odbywa się na podstawie dwóch składowych:
   a. zaliczenia wykładu,
   b. oceny z laboratorium.
2. Zaliczenie laboratorium obejmuje:
   a. projekt grupowy,
   b. publiczną prezentację projektu grupowego,
   c. kolokwium zaliczeniowe
3. Projekt grupowy polega na opracowaniu rozwiązania konkretnego problemu poprzez przygotowanie i wdrożenie aplikacji oraz publiczną prezentację wyników projektu.
4. Student zalicza laboratorium, jeżeli:
   a. uzyska ocenę co najmniej 3.0 z projektu,
   b. ocenę co najmniej 3.0 z kolokwium zaliczeniowego.
5. Student zalicza wykład, jeśli:
   a. otrzymał z laboratorium ocenę co najmniej 3.0.
   b. napisał test na min. 50% punktów

Ocenę dst. z kolokwium/projektu student uzyskuje w przypadku uzyskania minimum połowy możliwych do zdobycia punktów. Uzyskane punkty odpowiadają skali: do 50% - ocena 2.0; od 51% do 69% - ocena 3.0; od 70% do 79% - ocena 3.5; od 80% do 87% - ocena 4.0; od 88% do 94% - ocena 4.5; od 95% do 100% - ocena 5.0.
Taka forma weryfikacji pozwala pełni ocenić stopień osiągnięcia przez studenta efektów uczenia się.

5. CAŁKOWITY NAKŁAD PRACY STUDENTA POTRZEBNY DO OSIĄGNIĘCIA ZAŁOŻONYCH EFEKTÓW W GODZINACH ORAZ PUNKTACH ECTS

Forma aktywności	Średnia liczba godzin na zrealizowanie aktywności
Godziny kontaktowe wynikające z harmonogramu studiów	30
Inne z udziałem nauczyciela akademickiego (udział w konsultacjach, egzaminie)	2
Godziny niekontaktowe – praca własna studenta (przygotowanie do zajęć, egzaminu, napisanie referatu itp.)	20
SUMA GODZIN	52
SUMARYCZNA LICZBA PUNKTÓW ECTS	2

6. PRAKTYKI ZAWODOWE W RAMACH PRZEDMIOTU

wymiar godzinowy	----
zasady i formy odbywania praktyk	----

7. LITERATURA

Literatura podstawowa

1. S. McConnell, Code Complete. A Practical Handbook of Software Construction, Microsoft Press, 2nd ed., 2004.
2. R. C. Martin, Clean Code. A Handbook of Agile Software Craftsmanship, Prentice Hall, 2008.
3. K. Schwaber, J. Sutherland, Scrum Guide. Przewodnik po Scrumie, najnowsza edycja online (dostępna po polsku i angielsku na scrumguides.org).
4. M. Fowler, Refactoring. Improving the Design of Existing Code, Addison-Wesley, 2nd ed., 2018.
5. M. Białas, Inżynieria oprogramowania. Aspekty praktyczne, PWN, Warszawa 2020.

Literatura uzupełniająca

1. A. Cockburn, Agile Software Development: The Cooperative Game, Addison-Wesley, 2nd ed., 2006.
2. R. C. Martin, Clean Architecture. A Craftsman's Guide to Software Structure and Design, Prentice Hall, 2017.
3. J. Hunt, Advanced Guide to Git, Apress, 2020.
4. J. Humble, D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley, 2010.
5. R. Pressman, B. Maxim, Inżynieria oprogramowania. Podejście praktyczne, PWN, Warszawa 2022 (tłum. Software Engineering: A Practitioner's Approach).
6. The Scrum Guide, Ken Schwaber, Jeff Sutherland. Dostęp: https://scrumguides.org
   Oficjalny przewodnik po metodyce Scrum – dostępny w języku polskim i angielskim. Fundamentalne źródło wiedzy o pracy zespołowej w projektach IT.
7. Pro Git (Scott Chacon, Ben Straub). Wydanie online: https://git-scm.com/book/en/v2
8. Zbiór krótkich kursów o tworzeniu dokumentacji technicznej i komunikacji w zespołach developerskich. Bardzo przydatny przy nauce pisania README i wiki projektowych.
9. Google for Developers – Software Engineering Best Practices. Dostęp: https://developers.google.com/tech-writing

Akceptacja Kierownika Jednostki lub osoby upoważnionej