# Platformy wspomagające sprawdzanie wiedzy

Sprawdzanie wiedzy jest procesem żmudnym i czasochłonnym. Prowadzący zajęcia zamiast przeznaczyć czas na zajęcia dydaktyczne, albo rozwój osobisty, albo cokolwiek pożytecznego, musi sprawdzać setki prac, prac częstokroć niemal identycznych. Problem może wydać się błahy w skali tygodnia, jednak już w skali roku, dekady czy też całego życia, są to olbrzymie koszta. Niestety w dobie postępu, rozwoju i dążenia do ciągłego samodoskonalenia, czas jest dobrem deficytowym. Dobra deficytowe oczywiście należy oszczędzać, a z pomocą może przyjść możliwość informatyzacji procesu wykonywania testów i sprawdzania wiedzy.

Programów, czy też platform z wieloma funkcjonalnościami jest dostępnych na rynku naprawdę bardzo dużo. Od programów sprawdzających wiedzę z pewnego wąskiego wycinka wiedzy, np. programów dedykowanych do podręczników, czy też programów, które umożliwiają samodzielne tworzenie testów przez prowadzącego zajęcia, po platformy   
*e-learningowe*, które oferują możliwość przeprowadzenia całego cyklu edukacji z danej dziedziny, prowadzenie statystyk i inne czasem mniej lub bardziej przydatne funkcjonalności. Krytycy sprawdzania wiedzy przy użyciu komputera twierdzą, że może to generować sytuacje konfliktowe- np. łatwość ściągania z monitora kolegi. Można jednak temu zaradzić mieszając pytania, tak aby były one generowane u każdego użytkownika w sposób losowy, można tworzyć różne zestawy pytań, albo ustawiać czas, w którym się trzeba zmieścić, żeby udzielić odpowiedzi.

## Możliwości sprawdzania wiedzy z użyciem aplikacji komputerowych

W programy komputerowe, które pozwalają na sprawdzanie wiedzy warto zainwestować nie tylko ze względu na oszczędzane zasoby(czas, papier, miejsce), ale także ze względu na fakt, że taka forma przeprowadzania sprawdzianów jest często bardziej atrakcyjna dla młodych ludzi. Sprawdzanie wiedzy z użyciem komputera może sprawić, że uczeń będzie chętnie wiedzę nabywał, by następnie w potyczce z programem komputerowym nie dać za wygraną. Ponadto jest to po prostu dla ucznia przyjemniejsze, nie wzbudza tylu negatywnych emocji i niepotrzebnego stresu. Na rynku można nabyć programy, które pozwalają sprawdzać wiedzę tylko z pewnego zakresu, testy na prawo jazdy, programy dostarczane z podręcznikami. Mają zazwyczaj charakter testowy. Są też dostępne programy, które pozwalają definiować test samemu. W tych programach dostępne są opcje pozwalające wpisywać własne pytania, własne odpowiedzi, a także pozwalające wrzucać obrazki czy filmiki jako pytania i jako odpowiedzi.[Zastosowanie technik komputerowych do sprawdzania wiadomości i umiejętności uczniów.pdf] Ma to niebywałą wartość- kolorowe obrazki i filmiki pobudzają percepcję i pamięć mocniej niż zwykły tekst, dzięki temu uczniowie mogą z siebie wydobyć jeszcze więcej wiedzy. Programy te również mają charakter testowy, po wykonaniu testu sprawdzają odpowiedzi i generują wynik dla prowadzącego. Często takie programy dają również możliwość prowadzenia statystyk, które odzwierciedlają postęp ucznia, czy postęp całej grupy. Znane mi są takie programy jak: „Hot potatoes”, „Testy”, „Tester” czy „CATest”.[http://www.pcworld.pl/artykuly/358371/Programy.dla.nauczycieli.html] Mają bardzo podobne funkcje, różnią się szatą graficzną i zapewne różnią się z programistycznego punktu widzenia metodami realizacji funkcjonalności, niestety nie są *open source’owe*, więc nie mogę tego zweryfikować. Poniżej znajduje się *screenshot* z wybranego testu w programie „Testy”, przedstawia pytanie testowe jednokrotnego wyboru, które zawiera w sobie grafikę:



Rysunek Screen z programu "Testy"  
 Źródło: http://www.pcworld.pl/artykuly/358371/Programy.dla.nauczycieli.html

Bardzo popularną dziedziną, w której wiedza jest sprawdzana przy użyciu dedykowanego oprogramowania są testy na prawo jazdy. Tego typu oprogramowanie funkcjonuje zarówno przy egzaminach na prawo jazdy, jak i w ośrodkach szkolących kierowców. Zasadniczo oprogramowanie używane do egzaminowania nie różni się od tego dostarczanego do ośrodków szkolących, funkcjonalności są takie same i pytania testowe również. Różnica jest niewątpliwie w nakładach pieniężnych firm realizujących oprogramowanie dla ośrodków egzaminacyjnych w porównaniu do tych dedykowanych dla ośrodków szkolących. Oprogramowanie dla ośrodków egzaminacyjnych musi być całkowicie pozbawione jakichkolwiek błędów merytorycznych. Ewentualne błędy mogłyby wpłynąć na unieważnienie egzaminów, które zostały już przeprowadzone. Skutkiem byłyby olbrzymie koszty związane z ponownym przeprowadzeniem egzaminów, czy też z konserwacją oprogramowania. Zapewne oprogramowanie dla ośrodków egzaminacyjnych ma również większe nakłady na zapewnienie niezawodności samego oprogramowania, oraz infrastruktury systemu. Komputery, na których to oprogramowanie funkcjonuje muszą być odpowiednio skonfigurowane, piloty za pomocą których wybiera się poprawną odpowiedź, również muszą być niezawodne, a cała infrastruktura musi być wspierana przez UPS’y, które zachowają ciągłość trwających testów w razie awarii prądu.

Ciekawie problem sprawdzania wiedzy rozwiązują również programu do nauki języków obcych. Bardzo często po zrealizowaniu pewnego działu pozwalają sprawdzić, w jakim stopniu opanowaliśmy materiał. Generowane mogą być również statystyki, które pokazują, jakie elementy jeszcze musimy podszkolić. Pozwalają one nie tylko na testowe sprawdzanie wiedzy, ale także na uzupełnianie pewnych fragmentów zdania, na przestawianie szyku wyrazów, tak żeby utworzyć poprawne zdanie, a często też sprawdzają naszą fonetykę.

## Przegląd platform e-learningowych

Należy zacząć od zdefiniowania czym właściwie jest *e-learning* – wg *e-booka* „Poradnik dla projektujących kursy *e-learningowe*”, jest to proces dydaktyczny, który odbywa się w całości za pośrednictwem Internetu, a także proces sprawdzania wiedzy może się odbywać za pośrednictwem Internetu[Poradnik dla projektujących kursy e-learningowe]. Od kilku lat obserwuje się wyraźną ekspansję platform e-learningowych, które oprócz udostępniania materiałów dydaktycznych, czy przeprowadzania sprawdzianów, oferują bardzo szeroki wachlarz funkcjonalności, które można samemu adaptować.

Platforma **Moodle**, która jest oprogramowaniem *open source* oferowana na otwartej licencji GNU GPL, pozwala każdemu dopisać swoje własne funkcjonalności przy użyciu języka PHP, a także samemu podłączyć bazę danych MySql lub PostgreSQL i nią zarządzać. Stwarza to olbrzymie możliwości rozwoju tej platformy. Moodle jest dostępna u każdego, kto posiada zainstalowaną przeglądarkę internetową i Internet, aczkolwiek można ją także uruchomić na lokalnym serwerze, bez dostępu do Internetu. Platforma Moodle pozwala na tworzenie kursu, wewnątrz kursu istnieje możliwość stworzenia grup, do których można wysyłać materiały dydaktyczne w każdej postaci(prezentacje multimedialne, pliki PDF, pliki tekstowe, filmiki instruktażowe i wiele innych). Moodle oferuje również czat „On live” pomiędzy użytkownikami i prowadzącym kurs, a także wiadomości prywatne. Udostępniane są również konta dostępowe do poczty elektronicznej. Moodle oprócz wysyłania materiałów dydaktycznych umożliwia także prowadzenie telekonferencji w czasie rzeczywistym. Możliwe jest też po prostu przeprowadzanie lekcji on-line, lub indywidualne konsultacje z prowadzącym kurs. Jest to potężne narzędzie, które pozwala tworzyć kursy dla użytkowników rozproszonych po całym świecie.

Przeprowadzanie sprawdzania wiedzy również jest mocno rozbudowane. Oprócz prostych testów, czy pytań otwartych, możliwe jest również odpytywanie przy użyciu samej fonii, lub wizji i fonii razem. Udostępniana może być też wirtualna tablica, na której uczeń może rozwiązywać zadania, np. z fizyki.

Ważne są również cechy takie jak:

* Ciekawy *layout*, przyciągający uwagę i pobudzający myślenie,
* Możliwość korzystania z kursów innych(jeśli twórca kursu wyrazi na to zgodę),
* Możliwość zrzeszania się ludzi, poznawanie innych, ciekawych ludzi, często z zagranicy,
* Forum, na którym możemy rozwiązywać nasze problemy dotyczące konfiguracji środowiska, czy też otrzymać pomoc dotyczącą własnych funkcjonalności, które dopisujemy do systemu

Kolejną platformą bardzo powszechną jest **Piazza**. Oferuje możliwości bardzo podobne do Moodle, ale jest dedykowana dla uczelni wyższych. Zdaje się, że twórcy Piazzy mocno postawili na ochronę praw autorskich materiałów, które trafiają na kursy na Piazzie. Nie ma tam możliwości podglądu, co się dzieje na danym kursie, jeśli prowadzący kursu nas nie autoryzuje. Piazza oferuje również kastomizację środowiska. Piazza daje także możliwość użycia języka LaTex do definiowania dokumentów tekstowych. Udostępnia szeroki *support* dla klientów- forum, na którym możemy zadawać pytania dotyczące problemów konfiguracyjnych, zgłaszanie *ticketów*[[1]](#footnote-2), jeśli coś nie działa jak należy, czy też serię *videotutoriali* pokazujących, jak skonfigurować najważniejsze funkcje platformy.

Bardzo znaną platformą e-learningową dedykowaną głównie do uczelni wyższych jest **Blackboard**. Blackboard wyróżnia się między innymi tym, że gotowe kursy można tam kupować i realizować dla swoich studentów. Blackboard udostępnia także inne usługi, np. swój własny CMS, usługi związane z prowadzeniem statystyk i analiz(nie tylko związanych   
z edukacją), czy też systemy wspomagające zarządzanie kampusem akademickim. Usługą bardzo rozpowszechnioną w Stanach Zjednoczonych jest usługa „Security Managament”. Jest to zintegrowany, skalowalny system zarządzający bezpieczeństwem, który pozwala autoryzować wszystkich wchodzących na teren kampusu akademickiego, może także wysłać do wszystkich użytkowników powiadomienia, jeśli coś niepokojącego ma miejsce na terenie kampusu.

Nie można nie docenić tak olbrzymich możliwości platform *e-learningowych*, które przecież cały czas się rozrastają, powstają nowe moduły, rozszerzenia, zaczynają być *targetowane* do coraz nowych grup. Autorzy „*Poradnika dla projektujących kursy   
e-learningowe*” dostrzegają następujące zalety tej formy prowadzenia zajęć dydaktycznych:

* Elastyczność i mobilność,
* Brak ograniczeń terytorialnych,
* Możliwość lepszego i bardziej efektywnego zarządzania czasem,
* Powszechniejszy dostęp do wiedzy,
* Możliwość indywidualizacji tempa nauki,
* Oszczędności finansowe,
* Monitoring postępów uczącego się,
* Rozwój umiejętności pracy w grupie,
* Możliwość modyfikowania zasobów edukacyjnych,
* Szeroki wachlarz możliwości sprawdzania nabytej wiedzy.

Autorzy dostrzegają również pewne zagrożenia:

* Kontakt ograniczony zazwyczaj do kontaktu pisemnego,
* Trudności w utrzymaniu stałej kontroli nad użytkownikami,
* Problemy z utrzymaniem odpowiedniej motywacji do nauki,
* Niezbędny szeroki zakres umiejętności dla prowadzących kursy.

Reasumując, przeprowadzanie szkoleń w takiej formie daje duże możliwości, ale i niesie ze sobą pewne zagrożenia, które należy mieć na względzie. Najlepiej, żeby proces edukacji był kompleksowy, żeby wymuszał zarówno korzystanie ze źródeł książkowych, jak i z zasobów elektronicznych.

## Mój pomysł na program sprawdzający wiedzę z przedmiotów elektronika i elektrotechnika

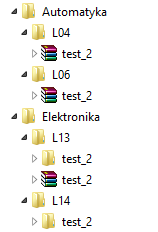
Tworzony na potrzeby tej pracy projekt ma charakter modularny. Składa się z aplikacji serwerowej- obsługiwanej przez prowadzącego zajęcia, oraz aplikacji klienckiej- obsługiwanej przez studentów. Aplikacje prowadzącego i studentów pozostają w relacji 1 do n, komunikując się przez protokół TCP/IP. Aplikacja serwerowa pozwala na podłączanie się klientów i wyświetla klientów, którzy już zostali podłączeni(identyfikuje ich przez imię i nazwisko oraz adres IP). Z poziomu serwera nauczyciel wgrywa test, który jest wewnątrz aplikacji sprawdzany pod względem poprawności składni. Na potrzeby aplikacji został stworzony język skryptowy nazwany „tst”, program będzie przyjmował więc jedynie pliki z rozszerzeniem *.tst*. Przykładowa składnia testu:

\*\*\*\*\*SPRAWDZIAN\_AUTOMATYKA\_2015\*\*\*\*\*

Rodzaj:"Z"; Tresc:"Który z wymienionych wzmacniaczy powinien być użyty do (...)?"; Odpowiedz\_A:"Cos tam"; Odpowiedz\_B:"Jeszcze cos"; Odpowiedz\_C:"Cos jeszcze innego"; Odpowiedz\_D:"Cos jeszcze innego"; Odpowiedz\_E:"Cos jeszcze innego"; Prawidlowa:"B"; Czas:"60".

Rodzaj:"Z"; Tresc:"Jaki jest prad, ktory...?"; Odpowiedz\_A:"Pictures\Pobrane.jpg"; Odpowiedz\_B:"Pictures\Pobrane.jpg"; Odpowiedz\_C:"Pictures\Pobrane.jpg"; Odpowiedz\_D:""; Odpowiedz\_E:""; Prawidlowa:"C"; Czas:"40".

W skrypcie określamy, jakiego rodzaju jest to pytanie – „Z” to zamknięte, „O” to otwarte. Określamy także czas, przez który pytanie i odpowiedzi są wyświetlane, informację o poprawnej odpowiedzi(jeśli jest to pytanie zamknięte), a także wpisujemy treść pytań i ewentualnie odpowiedzi. Interpreter, który wyciąga pojedyncze dane ze skryptu znajduje się po stronie klientów, po stronie serwera znajduje się moduł, który sprawdza, czy wprowadzony test ma poprawną składnię. Po wprowadzeniu testu do programu tworzona jest struktura drzewa folderów dla danego przedmiotu i danej grupy:



Rysunek Przykładowe drzewo folderów dla wprowadzonych testów  
Źródło: opracowanie własne

Aplikacja daje możliwość wyświetlania rysunków jako pytań i jako odpowiedzi. Prowadzący musi wstawić obrazki do folderu *Pictures* i odpowiednie nazwy odzwierciedlić w skrypcie, np:

Odpowiedz\_B:"Pictures\Wzmacniacz-A.jpg";

Cała struktura folderów dla danego testu, zawierająca zdjęcia(grafiki) i plik z testem jest archiwizowana do pliku *.zip* i po wciśnięciu przycisku *„Wyślij test”* paczka jest wysyłana do wszystkich podłączonych klientów.

Po stronie klienta dochodzi rozpakowania pliku *.zip*, oraz do włączenia interpretera, który wyciąga informacje z pliku skryptowego. Dane są bindowane z pliku i wstawiane w odpowiednie miejsce, podczas wykonywania testu. Test rozpoczyna się po wciśnięciu przez użytkownika(ucznia) przycisku *„Rozpocznij test”*. Po wykonaniu testu generowany jest raport, który jest wysyłany na serwer w postaci pliku *.html*. W raporcie znajdują się zaznaczone przez studenta odpowiedzi, ewentualne odpowiedzi na pytania otwarte, sprawdzone odpowiedzi i policzone punkty z wystawioną oceną.

Po zakończeniu testu po stronie serwera, do katalogu w którym znajduje się test dla grupy(w której aktualnie był wykonywany), są kopiowane raporty z wykonania sprawdzianu przez wszystkich studentów(nazwa pliku to *imię\_nazwisko.html*), a także raport zbiorczy z wykonania danego sprawdzianu.

Modułowy charakter tego programu powstał na potrzeby tego projektu. Oczywiście istnieje wiele różnorakich dróg rozwiązania problemu sprawdzania wiedzy. Można stworzyć i skonfigurować pod potrzeby przedmiotów elektronika i elektrotechnika konto na którejś z platform *e-learningowych*. Można stworzyć aplikację webową z bazą pytań, wówczas istniałaby możliwość(przy założeniu, że wszyscy mają dostęp do Internetu) sprawdzania wiedzy zdalnie, bez obecności studentów na zajęciach w sali. Aplikację można by również zrealizować jako jeden moduł, instalowany na komputerze studenta, bez przesyłania wyników do prowadzącego, lub ewentualnie wysyłanie ich na adres e-mail prowadzącego. Kolejną opcją jest wykorzystanie serwera IIS(w dalszej części pracy termin zostanie rozwinięty), który może pełnić rolę serwera ftp w sieci lokalnej – wówczas możliwe byłoby wgrywanie testów na serwer, a program po stronie klienta nasłuchiwałby, czy plik z testem jest dostępny do ściągnięcia, a następnie ściągałby plik i rozpoczynał wykonywanie testu. Wybrana została formuła, która w mojej ocenie sprawdzi się najlepiej. Jednocześnie jest odporna na problemy z połączeniem do Internetu(wykorzystuje sieć LAN), a także daje wygodę związaną z przesyłaniem raportów do prowadzącego. Nie musi on ściągać poczty elektronicznej, od każdego z klientów, ale wchodzi do folderu, gdzie kopiowane są raporty i od razu może wystawić oceny. Nie wymaga również żadnych dodatkowych nakładów w postaci konfiguracji komputerów(komputery, na których testy będą wykonywane, są połączone w sieć LAN). Potencjalnie program może być również adaptowany dla innych przedmiotów. Jedynym założeniem powodzenia adaptacji jest konieczność połączenia komputerów w sieć LAN. Projekt może być również odpowiednio rozwijany przez prowadzących zajęcia, jeśli zechcieliby dołożyć nowe funkcjonalności.

## Potencjał i możliwości rozwoju aplikacji sprawdzających wiedzę

W dobie rozwoju aplikacji mobilnych, dedykowanych dla smartfonów, tabletów, smartwatchów i innych, niebawem zapewne pojawią się programy sprawdzające wiedzę, które mogą być uruchamiane na tych urządzeniach. Coraz częściej w szkołach podstawowych, gimnazjach i szkołach średnich, uczniowie dostają tablety edukacyjne, które są odpowiednio skonfigurowane i mają wgrywane aplikacje służące do nauki, często również do rozwiązywania zadań. Pewnym rozwiązaniem może się stać także przeprowadzanie sprawdzianów na tabletach edukacyjnych, w czasie rzeczywistym.

Kolejną możliwą drogą rozwoju jest droga związana z umieszczaniem aplikacji sprawdzającej wiedzę w chmurze. Wówczas możemy mieć dostęp do pewnych ważnych danych zawsze i mamy możliwość przeprowadzania sprawdzianów, niezależnie od miejsca   
i niezależnie od platformy, z której aktualnie się logujemy do chmury. Jest to wygodne   
i wydajne z punktu widzenia użytkownika – wszystkie obliczenia są wykonywane w chmurze, więc zasoby urządzenia nie są zajmowane. Ograniczeniem może być tu konieczność zapewnienia odpowiedniej infrastruktury w chmurze(korzystanie z wirtualnych maszyn   
w chmurze jest dosyć kosztowne), ograniczeniem może też być kwestia dostępu użytkownika do stabilnego, szybkiego połączenia z Internetem.

# Inżynieria wymagań, proces przygotowawczy do rozpoczęcia projektu informatycznego i nowoczesne metody zarządzania projektami informatycznymi

Proces tworzenia oprogramowania to nie tylko sama implementacja funkcjonalności w systemie. W dzisiejszych czasach duże projekty informatyczne wymagają przeprowadzania wielu czynności, które ułatwiają uzyskać pożądany efekt końcowy. Zaczynając od pozyskania wymagań, przechodzimy do prototypowania, analizy czasu i kosztów, komponowania zespołu, tworzenia niezbędnych diagramów, a dopiero potem przychodzi czas na projektowanie GUI, samą implementację, testowanie i wdrożenie. Wszystkie te czynności mają na celu maksymalizację zysków firmy tworzącej oprogramowanie, ale też poniekąd prowadzą do oszczędności klienta. W związku z potrzebą systematyzacji wiedzy, która odpowiada na pytanie: „jak skutecznie zarządzać projektami ?”, powstało szereg metodyk zarządzania projektami. W dzisiejszych czasach rozpowszechniona jest chyba najbardziej metodyka – **PRINCE2**, która jest z powodzeniem wykorzystywana do wszelkiego rodzaju projektów(nie tylko informatycznych). Definiuje ona kolejne kroki procesu zarządzania projektem. Określa także strukturę jednostki odpowiedzialnej za dany projekt. Na bazie PRINCE2 wyrosły metodyki ściśle związane z zarządzaniem projektami informatycznymi, nazywane metodykami zwinnymi, albo po prostu metodykami **Agile’owymi**.[Krzysztof Sacha, Inż. Opr] Więcej informacji o metodykach zwinnych będzie można przeczytać w podrozdziale zatytułowanym „Zwinne metodyki zarządzania projektami”. W kolejnych rozdziałach czytelnik zostanie przeprowadzany przez wszystkie(niestety w pewnym uproszczeniu) etapy, które towarzyszą każdemu poważnemu przedsięwzięciu informatycznemu.

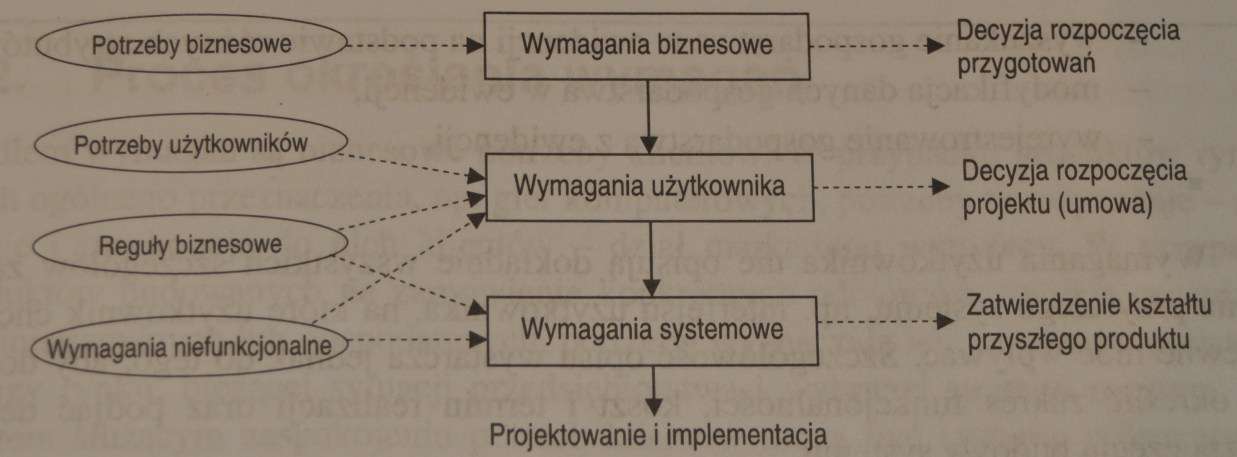
## Inżynieria wymagań

Inżynieria wymagań jest nauką, która definiuje i opisuje systematykę wymagań, a także opisuje proces ich pozyskiwania. Wymagania dla konkretnego systemu informatycznego stają się wykładnią tego, co w danym systemie informatycznym ma zostać wykonane i jak ma zostać wykonane. Wymagania powinny zostać skonstruowane tak, żeby programista mógł po nie sięgnąć, zajrzeć tam i od razu rozumieć funkcjonalność, którą musi zaimplementować. Wymagania powinny być jednoznaczne, poprawne, kompletne, spójne(poszczególne wymagania nie mogą się nawzajem wykluczać), uporządkowane i modyfikowalne(w razie zmiany koncepcji dotyczącej budowanego systemu, musi istnieć możliwość ich modyfikacji). [Krzysztof Sacha, Inż. Opr].

Pozyskiwanie wymagań jest czynnością, która zmierza do zdobycia informacji na temat przyszłego, implementowanego systemu. Informacje te mogą być zdobywane przez:

* Ankiety wypełniane przez przyszłych użytkowników systemu,
* Rozmowy z przyszłymi użytkownikami,
* Oddelegowanie pracowników firmy wytwarzającej oprogramowanie do obserwowania działania procesów, które system ma odzwierciedlać(zwłaszcza jeśli mowa o systemach ERP[[2]](#footnote-3)),
* Studiowanie dostępnej dokumentacji, [Krzysztof Sacha, Inż. Opr]
* Badania rynku i potrzeb użytkowników(bardzo ważne w przypadku systemów użytkowych, które nie są dedykowane tylko dla jednego podmiotu),
* Badanie wymagań formalnych związanych z kontrolą jakości, lub z normami prawnymi,

Proces pozyskiwania wymagań jest tak naprawdę kluczowy dla całego projektu. Błędy związane ze zrozumieniem potrzeb klienta, złym ich opisem lub niedostrzeżeniem pewnych czynników związanych z jakością, mogą doprowadzić do fiaska całego projektu informatycznego. Należy ten proces przeprowadzić bardzo dokładnie, lub zlecić jego wykonanie zewnętrznej firmie. Przekazując wykonanie pozyskania wymagań zewnętrznej firmie, przekazywana jest również odpowiedzialność. W takim wypadku w razie niezgodności pozyskanych przez firmę *outsourcing’ową* wymagań z rzeczywistymi potrzebami klienta, firma wytwarzająca oprogramowanie może zmniejszyć poziom strat. Pozyskiwanie wymagań zaczyna się zazwyczaj od rozpoznania własnych potrzeb biznesowych, oraz pozyskania i analizy wymagań biznesowych klienta(zbadanie celu biznesowego, który klient chce dzięki systemowi informatycznemu osiągnąć). To rozpoznanie determinuje decyzję o rozpoczęciu przygotowań do projektu informatycznego. W tym miejscu z pomocą przychodzi doskonale sprawdzona technika **SWOT**. Technika analityczna SWOT(Strengths Weaknesses Opportunities Threats) służy analizie środowiska wewnętrznego i zewnętrznego. Analizowane są mocne strony firmy- Strengths, słabe strony firmy- Weaknesses, szanse- Opportunities(np. na korzystne zmiany przy okazji przeprowadzenia danego przedsięwzięcia informatycznego), oraz zagrożenia- Threats(możliwe niekorzystne zmiany).[Krzysztof Sacha Inzop] Następnie są rozpoznawane wymagania przyszłego użytkownika. Wymagania użytkownika odpowiadają na pytania: „co ten system ma robić?”, „jak system ma to robić?”   
i ewentualnie „jak system ma wyglądać pod względem estetycznym?”. Po zebraniu tych wymagań powinna zapaść decyzja o rozpoczęciu projektu. [Krzysztof Sacha, Inż. Opr] Następnie tworzone są wymagania systemowe, czyli ogół tego, co i jak system ma robić. Wymagania systemowe powinny być bardzo obszerne i powinny bardzo dokładnie opisywać przyszły system. Na podstawie analizy wymagań systemowych zatwierdza się kształt przyszłego systemu. Opisany powyżej proces pozyskiwania wymagań doskonale wizualizuje rysunek:



Rysunek Poziomy opisu wymagań   
Źródło: Sacha K.,  *Inżynieria oprogramowania*, str. 55, Wydawnictwo Naukowe PWN, Warszawa 2010.

Najważniejszym podziałem wymagań jest podział na **wymagania** **funkcjonalne**, **niefunkcjonalne** i **wymagania zgodności**.[Sacha K.,  *Inżynieria oprogramowania*] **Wymagania funkcjonalne** opisują, co system ma robić(a nie jak ma to robić). Muszą opisywać wszystkie funkcjonalności, które ma implementować tworzony system. Zwinne metodyki zarządzania projektami optują za opisywaniem wymagań funkcjonalnych jako tzw. *user stories*. Są to po prostu historyjki, które w kilku zdaniach opisują każdą funkcjonalność systemu. **Wymagania niefunkcjonalne** to wymagania, które definiują, jak system powinien coś robić, jak ma wyglądać(np. wymagania związane ze środowiskiem, w którym oprogramowanie ma być wytwarzane, wymagania związane z przenośnością kodu i wiele innych). **Wymagania zgodności** opisują, jak system ma coś wykonywać w zgodzie z pewnym regulacjami prawnymi, standardami i ewentualnie zwyczajami panującymi w danej dziedzinie(np. w przypadku gromadzenia danych o użytkownikach dane powinny być szyfrowane).

W przypadku aplikacji sprawdzającej wiedzę, zgodnej z modelem zaproponowanym w tej pracy, można by wyróżnić następujące wymagania:

1. Wymagania funkcjonalne
   1. System powinien umożliwiać wykonywanie sprawdzianów,
   2. System powinien dawać możliwość kreowania testów przez prowadzącego zajęcia,
   3. System powinien umożliwiać wstawianie grafik jako pytania i jako odpowiedzi,
   4. System powinien sprawdzać zgodność udzielonych odpowiedzi ze wzorcem,
   5. System powinien generować raport z przeprowadzonego testu,
   6. System powinien archiwizować testy i raporty na dysku twardym,
2. Wymagania niefunkcjonalne
   1. System powinien działać w architekturze klient- serwer,
   2. System powinien wykorzystywać do komunikacji protokół TCP/IP,
   3. System powinien wykonywać sprawdziany w czasie rzeczywistym,
   4. System powinien dawać możliwość wykonywania testu w grupie od 1 do 16 osób,
   5. System powinien mieć schludny, czytelny i funkcjonalny interfejs graficzny,
   6. System powinien być odporny na problemy komunikacyjne zachodzące w sieci
3. Wymagania zgodności
   1. Dane, które zawierają imiona i nazwiska uczniów powinny być szyfrowane odpowiednim algorytmem, zapewniającym wysoki poziom bezpieczeństwa przetrzymywanych danych,

Aplikacja do sprawdzania wiedzy stworzona w tym projekcie uwzględnia wszystkie wymienione wyżej wymagania.

Specyfikowanie i opisywanie wymagań to tylko jeden z początkowych procesów wytwarzania oprogramowania. Maksymalizacja zysków osiągniętych z właściwie przeprowadzonego pozyskiwania i analizy wymagań, zachodzi tylko w korelacji z właściwie przeprowadzonymi: analizą czasu i kosztów, stworzeniem odpowiednich diagramów, modeli, prototypu, odpowiednią analizą ryzyka, poprawną implementacją systemu i testami obejmującymi tę implementację. Wszystkie te czynności zostały opisane w kolejnych rozdziałach pracy.

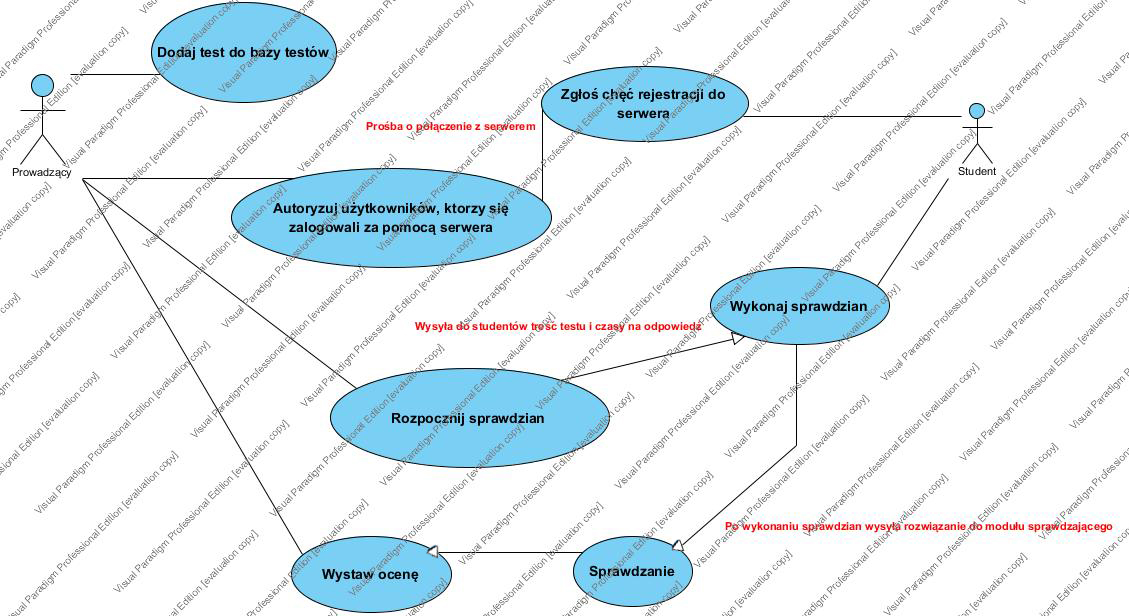
## Diagramy UML jako podstawa modelowania dziedziny problemu

W związku z rozwojem programowania obiektowego powstały techniki modelowania dziedziny problemu zgodne z proponowaną w programowaniu obiektowym wizją świata. Programowanie obiektowe zakłada podzielenie dziedziny problemu na pewne odrębne, współpracujące ze sobą byty, nazywane klasami. Obiekt zaś jest wystąpieniem pewnego abstrakcyjnego bytu jakim jest klasa. Na potrzeby owej koncepcji powstał **język UML**[[3]](#footnote-4), który pozwala modelować za pomocą szeregu diagramów wszystkie procesy zachodzące w systemie, rozmieszczenie komponentów systemu, relacje między obiektami systemy i wiele wiele innych. **Diagramy UML** pozwalają lepiej zrozumieć, jak dany system ma pracować, co ma robić i z czego się składa. Diagramy są tworzone, by ludzie niezwiązani z tworzeniem systemów informatycznych mogli lepiej zrozumieć, jak działa tworzony dla nich system.

Na potrzeby przeprowadzenia implementacji aplikacji sprawdzającej wiedzę zostały stworzone następujące diagramy, które obrazują pewien jej wycinek wycinek:

1. **User Case Diagram**- diagram przypadków użycia,
2. **Class Diagram**- diagram klas,
3. **Sequence Diagram**- diagram sekwencji,
4. **Deployment Diagram**- diagram rozmieszczenia,

**User Case Diagram[[4]](#footnote-5)**, jest to diagram który pokazuje pewien konkretny przypadek użycia systemu. Obrazuje, jakie role w powiązaniu z danymi czynnościami przyjmują aktorzy, czyli użytkownicy systemu. W tworzonym systemie budowane są diagramy, które odzwierciedlają kluczowe dla danego systemu przypadki użycia. Poniżej przedstawiony jest diagram przypadku użycia, pokazujący czynności wykonywane przez operatorów aplikacji sprawdzającej wiedzę, a także odpowiedzi systemu na ich działania. Diagram ukazuje czynność przeprowadzania testu:

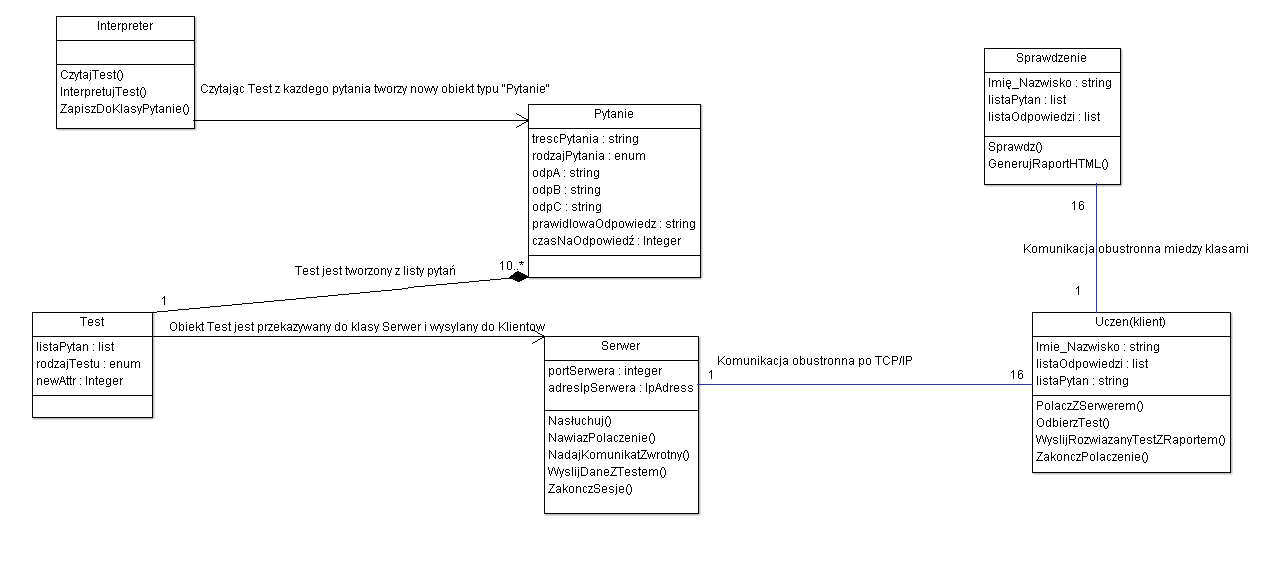


Rysunek Diagram przypadków użycia  
Źródło: opracowanie własne przy pomocy programu Visual Paradigm

Jak widać powyżej diagram dosyć dobrze obrazuje, jak krok po kroku przebiega proces przeprowadzania testu i jakie interakcje zachodzą wewnątrz systemu. Diagram ten abstrahuje od architektury aplikacji, pozostaje ona w domyśle. W eliptycznych blokach przedstawione są czynności wykonywane w systemie, zaś obok bloków czerwoną czcionką opisane są interakcje.

**Class Diagram** obrazuje, w jakiej relacji pozostają podstawowe jednostki syntaktyczne(klasy) znajdujące się w aplikacji. Klasy posiadają metody i pola, które w diagramach są odzwierciedlone. Diagramy klas pokazują także zachodzącą między klasami relację:

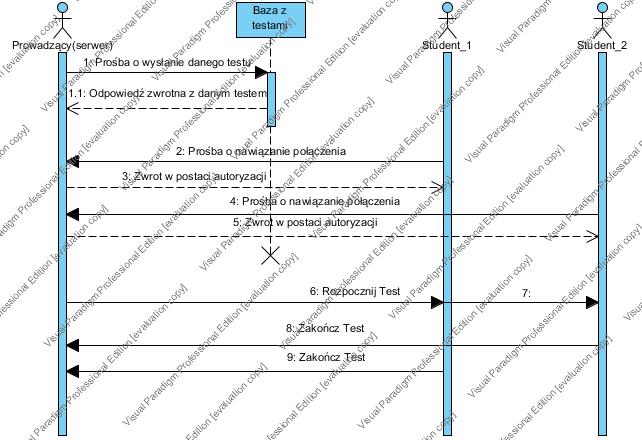
1. Dziedziczenie i dziedziczenie wielokrotne- relacja zgodnie z którą jedna klasa jest uogólnieniem- klasa bazowa, a klasa dziedzicząca jest jej konkretyzacją,
2. Asocjacja- gdy obiekty jednej klasy są w pewien sposób związane z obiektami innej klasy(w przypadku przedstawionego poniżej diagramu asocjacja zachodzi np. między klasą *Interpreter* i klasą *Pytanie*).
3. Agregacja- gdy klasa zagregowana jest częścią kilku różnych całości( nie zachodzi w poniższym diagramie klas),
4. Kompozycja- obiekt jeden klasy jest częścią obiektu jakiejś innej klasy(zachodzi w przypadku relacji między klasami *Pytanie* i  *Test*)



Rysunek Diagram klas  
Źródło: opracowanie własne przy pomocy programu Argo UML

Diagram klas został stworzony w pewnym czytelnym uproszczeniu. Zawiera kluczowe dla działania systemu klasy. Poniekąd uzmysławia też przepływ danych w aplikacji. Między klasami nie zachodzi dziedziczenie ani agregacja, zachodzi zaś kompozycja(zaznaczona czarnym rombem) i asocjacja(zaznaczona prostą linią lub strzałkę, w zależności od kierunku przepływu danych).

**Sequence Diagram-** należy do grupy diagramów modelujących dynamikę przepływu sterowania z uwzględnieniem czasu. U góry diagramu prezentowani są aktorzy, między którymi zachodzi interakcja. Poziome linie określają przepływ komunikatów między aktorami. Pionowa linia jest nazywana linią życia obiektu. Linia życia prezentuje, jak długo dany obiekt będzie istniał w systemie. U góry diagramu, w prostokątnych blokach mogą być też umieszczane obiekty, które funkcjonują w systemie i wykonują pewne operacje, ale nie są utożsamiane z operatorem systemu. W diagramie koniec linii życia obiektu zaznaczana jest znakiem X. Między obiektami mogą występować interakcje, które są prezentowane jako komunikaty asynchroniczne(które nie czekają na komunikat zwrotny, zaznaczane strzałką bez wypełnienia) i komunikaty synchroniczne(którym towarzyszy komunikat zwrotny, oznaczone jako strzałki z wypełnieniem). Poniżej przedstawiony został diagram sekwencji obrazujący wymianę komunikatów między *Prowadzącym*(serwerem)   
a *Studentami*(klienci), a także jako obiekt oznaczona jest *Baza z testami*. Sekwencja zdarzeń w diagramie odpowiada zdarzeniom zachodzącym w aplikacji: *Prowadzący* dodaje test do *Bazy z testami*, następnie *Studenci* podłączają się do serwera i są przez niego autoryzowani, następnie wysyłany jest do nich test, a następnie test jest kończony.



Rysunek Diagram sekwencji  
Źródło: opracowanie własne przy pomocy programu Visual Paradigm

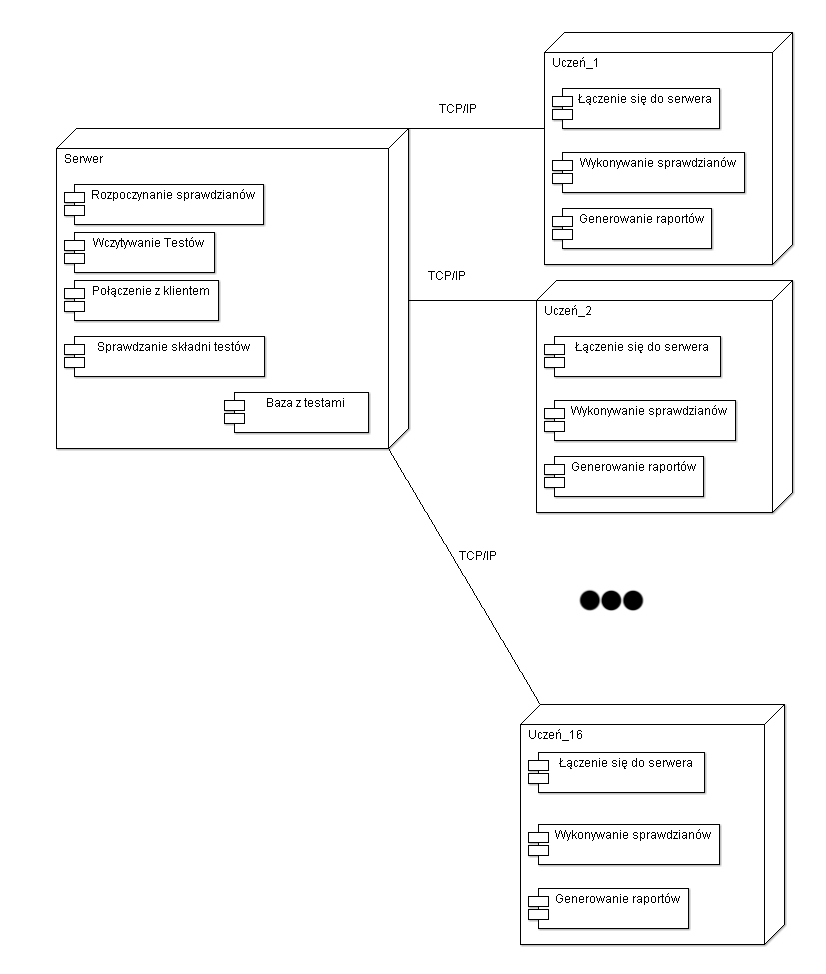
Prezentowany powyżej diagram również jest zbudowany w pewnym uproszczeniu, umożliwiającym przedstawienie pewnej koncepcji związanej z sekwencją zdarzeń w aplikacji. Przedstawienie całego przepływu wymienianych komunikatów zajęłoby niewspółmiernie dużo czasu, w porównaniu do korzyści wynikających z takiego przedstawienia. Przy tworzeniu diagramów UML zawsze należy pamiętać, że potencjalnie czytać to będą ludzie, którzy nie muszą sobie zdawać sprawy, jak dokładnie dana aplikacja będzie działać. Muszą mieć pewien czasem mniej lub bardziej ogólny zarys, by zrozumieć istotę działania programu.

**Deployment Diagram-** jest todiagram, który obrazuje, jak rozmieszczone są węzły danego systemu. Przez węzeł należy rozumieć pewien byt fizyczny, który ma konkretną rolę   
w systemie- np. komputer klienta. W węzłach znajdują się komponenty, czyli programy działające właśnie wewnątrz danego węzła. Diagram rozlokowania ma za zadanie ukazywać architekturę danego systemu. Musi pokazywać dane urządzenie fizyczne i jego korelację   
z innymi urządzeniami. Poniżej przedstawiony został diagram rozlokowania, który wiernie odzwierciedla, jak została zaplanowana architektura w tworzonej na potrzeby pracy inżynierskiej aplikacji. Serwer jest sercem systemu, które łączy się z wszystkimi klientami   
i od niego zależy, kiedy test zostanie rozpoczęty. W serwerze wyróżnia się komponenty takie jak:

* *Rozpoczynanie sprawdzianów*- komponent odpowiedzialny za rozpoczynanie i koordynowanie obiegu informacji związanych z przeprowadzanym sprawdzianem(wysłanie testu, komunikaty zwrotne, odebranie raportu),
* *Wczytywanie testów*- odpowiada za wczytywanie stworzonych przez *Prowadzącego* testów,
* *Połączenie z klientem*- komponent odpowiadający za utrzymanie połączenia z klientami,
* *Sprawdzanie składni testów*- odpowiada za sprawdzenie poprawności wprowadzanych przez *Prowadzącego* testów

W komputerach klienckich wyróżnia się:

* *Łączenie do Serwera*- komponent odpowiedzialny za nawiązanie i utrzymywanie połączenia z *Serwerem*,
* *Wykonywanie sprawdzianów*- odbieranie, interpretowanie składni i wykonywanie sprawdzianów,
* *Generowanie raportów*-generowanie raportów HTML wysyłanych do *Serwera*,

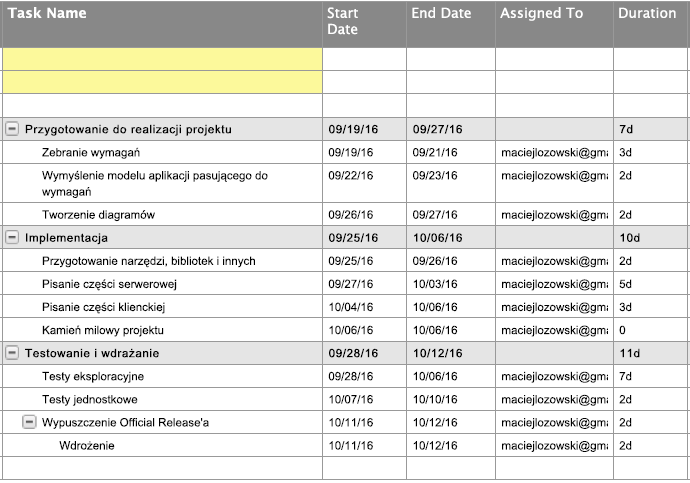


Rysunek Diagram rozmieszczenia  
Źródło: opracowanie własne za pomocą programu Argo UML

Projektowanie systemu za pomocą diagramów UML to proces w dzisiejszych czasach niezbędny, bez którego właściwa implementacja systemu jest niemożliwa. Diagramy UML ułatwiają także interakcje z klientami, dla których tworzymy system. W razie problemów z implementacją pewnej funkcjonalności programista może się wesprzeć odpowiednim diagramem, który wizualizuje pewne niekiedy ciężkie do zrozumienia kwestie.

## Planowanie czasu i kosztów

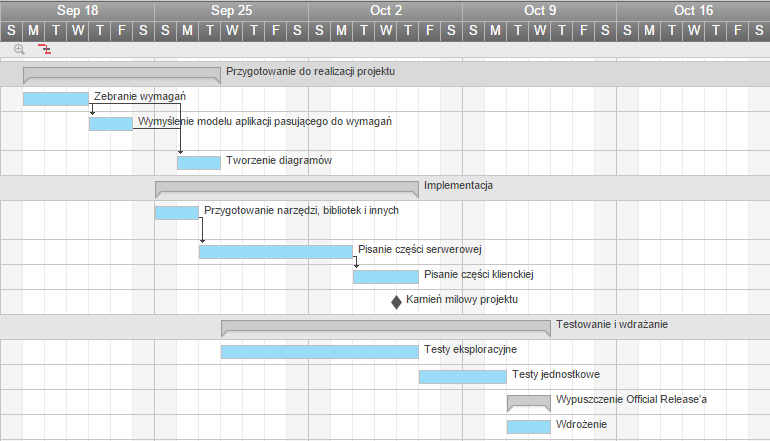
Planowanie jest etapem wytwarzania oprogramowania, w którym firma wytwarzająca oprogramowanie, po dogłębnej analizie dostrzega sens takiego przedsięwzięcia. Jednym z dostępnych diagramów wspomagających harmonogramowanie jest **diagram Gannta**. Jest to prosty diagram, w którym na podstawie oszacowanych czasów wykonania poszczególnych czynności prowadzących do zakończenia projektu, uwidaczniane są kolejne etapy. Celem tej pracy inżynierskiej, oprócz samego stworzenia konkretnej aplikacji i jej opisania, jest również przeprowadzenie czytelnika przez niemal wszystkie etapy, które towarzyszą tworzeniu oprogramowania. W związku z tym stworzony został diagram Gantta, który obrazuje, jak mógłby wyglądać plan przeprowadzania poszczególnych czynności związanych z projektem, dla komercyjnej aplikacji sprawdzającej wiedzę. Tak wygląda tabela zgodnie, z którą został stworzony diagram:



Rysunek Tabela, na podstawie której stworzony został diagram Gantta  
Źródło: opracowanie własne, za pomocą aplikacji smartsheet dostępnej na stronie internetowej: <https://app.smartsheet.com/>

Jest w tym rzecz jasna pewna abstrakcja, czas potrzebny do wykonania poszczególnych zadań został ustalony jedynie na bazie własnych doświadczeń .Zgodne z prawdą są kolejno następujące po sobie czynności w danych etapach.

Na podstawie tabeli został stworzony diagram Gantta:



Rysunek Diagram Gantta  
Źródło: opracowanie własne, za pomocą aplikacji smartsheet dostępnej na stronie internetowej: <https://app.smartsheet.com/>

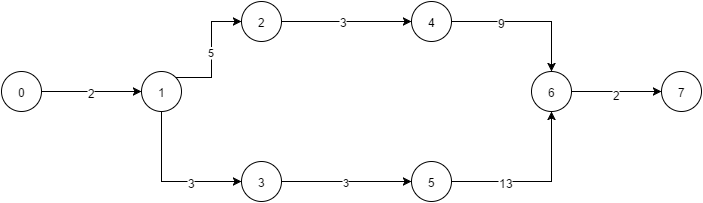
W diagramie został użyty termin **Kamień milowy projektu**. Termin jest niewątpliwie warty wyjaśnienia. Kamień milowy jest to podkreślenie zakończenia pewnego ważnego, niemal kluczowego dla danego projektu etapu. Oznacza, że duża część projektu jest już za nami. W projekcie przeprowadzanym na potrzeby tej pracy kamień milowy został postawiony po zakończeniu implementacji. Pozostałe po implementacji czynności są jakby formalnością i staje się niemal oczywistym, że projekt zostanie doprowadzony do końca. W dużych projektach, zwłaszcza zarządzanych z wykorzystaniem metodyk zwinnych, kamieni milowych może być więcej. Tak naprawdę wypuszczeniu każdego *Release’a[[5]](#footnote-6)* może towarzyszyć ich kilka.

Kolejną metodą związaną z harmonogramowaniem i estymacją czasu potrzebnego na projekt jest **CPM**(Critical Path Method), czyli **Ścieżka krytyczna**. Planując pewne przedsięwzięcie informatyczne możemy zakładać kilka możliwych dróg jego przeprowadzenia. Możemy oszacować czas dla każdej z możliwych dróg, a najdłuższa możliwa droga będzie nazywana Ścieżką krytyczną. Tworzenie diagramu z wykorzystaniem CPM musi uwzględniać wszystkie zagrożenia, które mogą towarzyszyć projektowi. Poniżej znajduje się tabela, zgodnie z którą stworzony został diagram pozwalający obliczyć Ścieżkę krytyczną:

|  |  |  |  |
| --- | --- | --- | --- |
| Nr czynności | Opis | Koszt czynności | Czynność poprzedzająca |
| 0->1 | Zestawienie środowiska, bibliotek, narzędzi | 2 |  |
| 1->2 | Implementacja części serwerowej z wykorzystaniem architektury klient serwer i protokołu TCP/IP | 5 | 0->1 |
| 1->3 | Wykorzystanie IIS jako części serwerowej, adaptacja | 3 | 0->1 |
| 2->4 | Implementacja części klienckiej z wykorzystaniem architektury klient serwer i protokołu TCP/IP | 3 | 1->2 |
| 3->5 | Implementacja klienta korzystającego z serwera IIS | 3 | 1->3 |
| 4->6 | Testowanie aplikacji, która została zaimplementowana w architekturze klient-serwer z wykorzystaniem protokołu TCP/IP | 9 | 2->4 |
| 5->6 | Testowanie aplikacji, która wykorzystuje serwer IIS | 13 | 3->5 |
| 6->7 | Wdrażanie | 2 | 4->6, 5->6 |
| **CPM** | | Dla architektury klient-serwer | Dla IIS |
| 21 | 23 |

Tabela Tabela przedstawiająca kolejne etapy implementacji systemu  
Źródło: opracowanie własne

Powyższa tabela ukazuje dwie możliwe drogi implementacji aplikacji sprawdzającej wiedzę   
i związane z nimi koszty[[6]](#footnote-7). Porównane zostały czasy dla drogi, która zakłada stworzenie aplikacji w architekturze klient- serwer, a także dla drogi, która zakłada stworzenie aplikacji do sprawdzania wiedzy przy założeniu wykorzystania serwera **IIS**[[7]](#footnote-8). Poniżej diagram bazujący na tabeli:



Rysunek Diagram Ścieżki krytycznej  
Źródło: opracowanie własne

W kółkach znajdują się poszczególne kroki, które można zidentyfikować w tabeli, zaś na liniach znajdują się koszty przeprowadzania danego etapu. Górna ścieżka diagramu to droga zakładająca wykorzystanie do implementacji aplikacji architektury klient- serwer, dolna ścieżka zakłada wykorzystanie IIS. Z przeprowadzonej analizy wynika, że korzystniejszą drogą jest droga, w której system jest implementowany w architekturze klient- serwer.

Ciekawymi metodami pomagającymi estymować koszty są metody bazujące na metodzie CPM- są to metody CPM PERT i CPM COST. Chętnych odsyłam do literatury związanej z prognozowaniem i symulacjami biznesowymi.

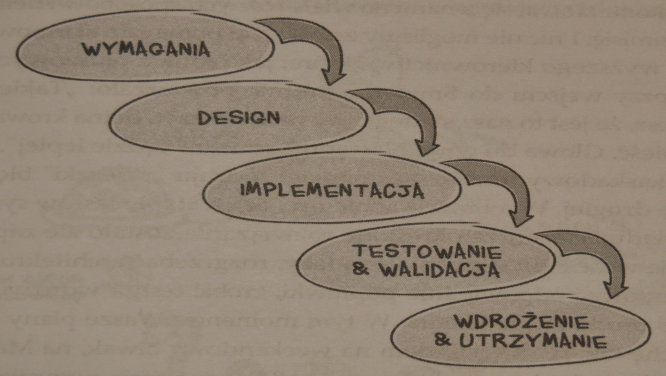
## Ryzyko towarzyszące projektom informatycznym

Etapem, który także musi towarzyszyć przeprowadzaniu projektów jest analiza ryzyka. Przez ryzyko rozumiana jest każda możliwa sytuacja, która może zaistnieć podczas przeprowadzania projektu, mogąca wywrzeć negatywny wpływ na to przedsięwzięcie. [Sacha, InzOp]. By przeciwdziałać ryzyku należy uczyć się nim zarządzać. Zarządzanie ryzykiem polega na jego identyfikacji i na posiadaniu scenariuszów, które następują, gdy krytyczne sytuacje się ziszczą. Wszystkie potencjalne zagrożenia muszą zostać uwzględnione w harmonogramie i w budżecie projektu. Aby zapewnić odpowiednią ochronę projektowi należy stworzyć dokumenty: **plan zarządzania ryzykiem** i **rejestr ryzyka**.[Sacha InzOp] Plan zarządzania ryzykiem opisuje metody wykrywania ryzyka, metody oceny ryzyka i osoby odpowiedzialne za wykonanie zadań związanych z ryzykiem. Kolejnym dokumentem jest rejestr ryzyka. Jest to dokument, który obejmuje:

* Identyfikację ryzyka,
* Ocenę stopnia zagrożenia(ocena prawdopodobieństwa wystąpienia ryzyka i ocena negatywnych skutków tego wystąpienia),
* Sposoby postępowania pozwalające uniknąć ryzyka, lub ograniczyć skutki wystąpienia ryzyka.

## Zwinne metodyki zarządzania projektami

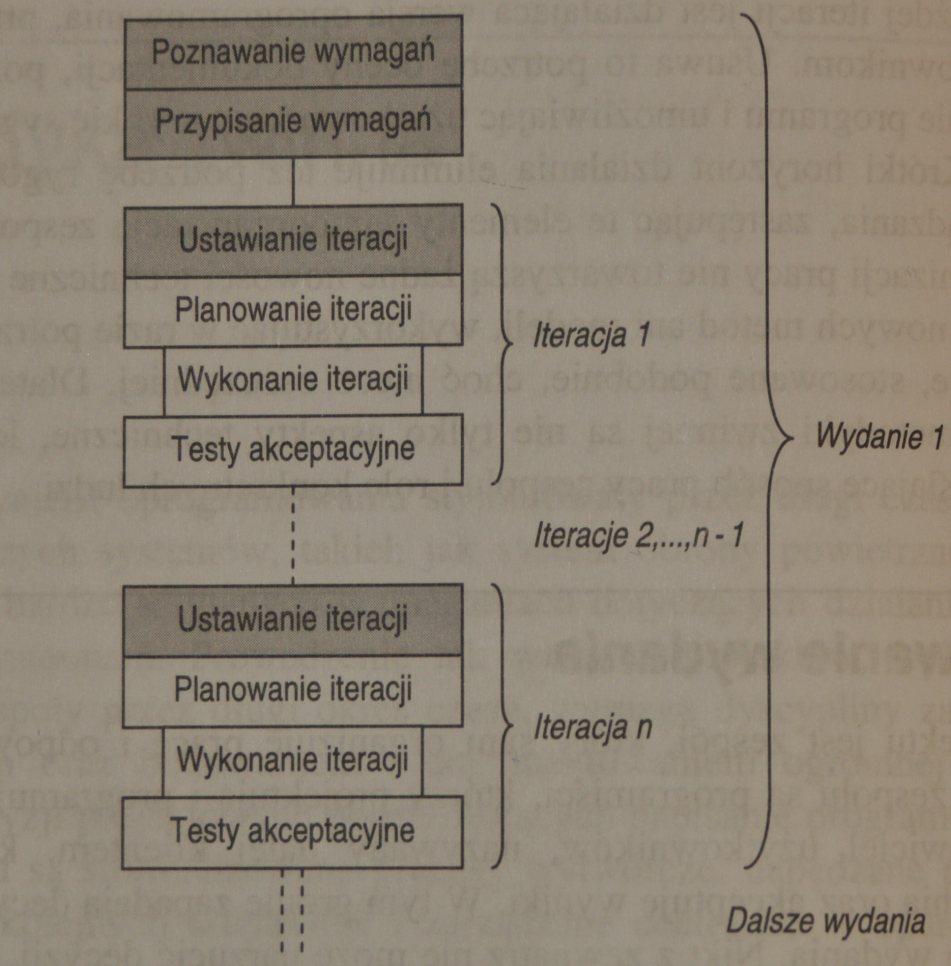
Do niedawna projekty informatyczne przeprowadzane były głównie z wykorzystaniem metod, które zakładały, że istnieje jedna sekwencja następujących po sobie zdarzeń związanych z przeprowadzanym projektem. Taką metodyką jest metodyka zgodnie, z którą przeprowadzony został projekt aplikacji sprawdzającej wiedzę. Ta metodyka to metodyka kaskadowa:



Rysunek Sekwencja zdarzeń przy metodyce kaskadowej  
Źródło: Chrapko M., SCRUM, O zwinnym zarządzaniu projektami, str. 19, Helion, 2013.

Metodyka kaskadowa sprawdza się przy małych projektach, nastręcza jednak zbyt wielu problemów w projekcie, w którym mogą zmieniać się wymagania. Załóżmy, że przeszliśmy już przez etap implementacji, stworzyliśmy kompletny system i nagle dowiadujemy się od kontrahenta, że jednak musimy zmienić pewne wymagania. System jest już niemal gotowy, ale firma dostarczająca oprogramowanie wie, że jeśli się nie podporządkuje, to system nie zostanie zaakceptowany. Wymagane są zmiany, które niewątpliwie będą bardzo kosztowne. Pewnym rozwiązaniem jest *outsourcing*, czyli zlecanie niezależnym podmiotom wytwarzania poszczególnych części oprogramowania. *Outsourcing* ma swoje zalety- na wypadek zmiany wymagań podczas projektu odpowiedzialność za zmiany oraz koszty rozkładają się na więcej podmiotów. Niewątpliwą wadą jest natomiast problem z komunikacją między poszczególnymi podmiotami, a także możliwe rozmywanie się odpowiedzialności.

W odpowiedzi na potrzebę elastyczności projektów powstały metodyki zwinne nazywane również metodykami **Agile’owymi**. Zakładają one zupełnie inny model wytwarzania oprogramowania i dużo lepiej dostosowują się do aktualnych potrzeb. Metodyki zwinne zakładają istnienie określonej ilości cykli(iteracji), w której wypuszczana jest kolejna wersja oprogramowania- *Release*.[Chrapko Scrum] Planowanie i wykonanie wydania w metodykach **Agile’owych** doskonale obrazuje poniższy rysunek:



Rysunek Iteracje i planowanie w metodykach zwinnych  
Źródło: Sacha K.,  *Inżynieria oprogramowania*, str. 336, Wydawnictwo Naukowe PWN, Warszawa 2010.

W tym modelu przed każdą iteracją wymagania mogą zostać zmodyfikowane, by w danym etapie- na diagramie nazwanym *Wykonanie iteracji* (w rzeczywistych projektach ten etap nazywany jest *Sprint*) wykonać część projektu, która zakłada zmianę wymagań. Tutaj zakłada się krótki horyzont działania- po każdej iteracji wypuszcza się *Release*, który po wykonaniu testów akceptacyjnych zostaje wysłany do kontrahenta do akceptacji. Każdy *Release* jest rozliczany na bieżąco, a nie po dostarczeniu całego projektu. Testy akceptacyjne poprzedzające wypuszczenie *Release’a* do akceptacji, to sprawdzenie zgodności stworzonego oprogramowania z wymaganiami, założeniami co w danym *Sprincie* miało zostać wykonane, oraz sprawdzenie czy nie ma błędów merytorycznych. Ta metodyka sprawdza się zwłaszcza w olbrzymich projektach, w których niezbędne jest szybkie reagowanie na wszelkie zmiany. *Agile* jest wygodny zarówno dla klientów, jak i dla firm dostarczających oprogramowanie. Firmy zlecające zbudowanie jakiegoś systemu mogą na bieżąco uczestniczyć w wytwarzaniu oprogramowania, nadzorować wszystkie etapy. Nie narażają się na fiasko całego przedsięwzięcia, które może zaistnieć z powodu nie dostarczenia oprogramowania w ogóle. Dostając co pewien określony czas *Release*, zawsze mogą skończyć projekt na danym etapie i zlecić jego dokończenie innemu podmiotowi. Dla firmy wytwarzającej oprogramowanie wykorzystanie metodyk *Agile’owych* jest wygodne, bo wypuszczając kolejną wersję oprogramowania, po jej akceptacji realizowana jest faktura. Nie naraża to firmy wytwarzającej oprogramowanie na czekanie na realizację płatności, a co za tym idzie potencjalne utracenie płynności finansowej. Metodyki *Agile’owe* dostarczają także szereg gotowych praktyk, które można implementować w firmie. Mowa tutaj o praktykach związanych z budowaniem zespołów, związanych z pozyskiwaniem wymagań, czy po prostu o pewnych założeniach wynikających z tzw. **Agile Manifesto**[[8]](#footnote-9). Jest to temat niezwykle obszerny, ale i niesamowicie interesujący, mający jak najbardziej praktyczny wymiar. Chętni chcący zgłębić wiedzę w danej materii, powinni sięgnąć po książkę opisującą konkretną metodykę *Agile’ową*- „*Scrum. O zwinnym zarządzaniu projektami.”*. Pozycja cieszącego się ogromną popularnością specjalisty od metodyk zwinnych- Mariusza Chrapko, przeprowadza czytelnika przez wiele tajników wiedzy o *Scrum*, podpierając wszystko bardzo ciekawymi i zrozumiałymi przykładami.

## Podsumowanie rozdziału

Reasumując, wytwarzanie oprogramowania to nie tylko pisanie kodu, właściwie pisanie kodu nie jest czynnością, która dominuje w pracy programisty. Niestety w realnych projektach o wiele więcej czasu jest poświęcane na kontakty z klientami, na dyskusje wewnątrz zespołu, czy na tzw. *Maintenance[[9]](#footnote-10).* Wytwarzanie oprogramowania to olbrzymie przedsięwzięcie, które składa się z ogromnej ilości pojedynczych procesów, które niezwykle ciężko byłoby okiełznać nie stosując pewnych dobrze już znanych praktyk. Jak widać nawet w tak małym projekcie, jak tworzona na potrzeby pracy aplikacja sprawdzająca wiedzę, inżynieria oprogramowania może być niezwykle przydatna. Wyspecyfikowanie wymagań nakazuje trzymać się pewnych ścisłych założeń, zapobiega tym samym niepotrzebnemu chaosowi. Stworzenie odpowiednich diagramów, nawet tak minimalistycznych, jak te zaprezentowane w owej pracy, pozwala po pierwsze trzymać się pewnych wyznaczonych ram, po drugie pozwala zrozumieć istotę działania systemu czytelnikowi, który na co dzień nie zajmuje się wytwarzaniem oprogramowania, oraz po trzecie jest pewną formą wsparcia dla kogoś, kto chciałby przeprowadzić modyfikacje w systemie. Diagramy są bardzo pomocne przy tzw. inżynierii wstecznej[[10]](#footnote-11). Ów rozdział miał na celu przeprowadzić czytelnika przez poszczególne etapy związane z specyfikacją wymagań, tworzeniem użytecznych diagramów, czy planowaniem czasu, które miały ścisły związek z aplikacją sprawdzającą wiedzę. Rozdział miał też na celu dostarczenie pewnych ciekawostek ze świata inżynierii oprogramowania. Podrozdziały opisane pokrótce, niemające ścisłego związku z tworzoną aplikacją, wymagają rozwinięcia i osobnego wyjaśnienia, niestety nie są tematem tej pracy. Zainteresowanych odsyłam do literatury technicznej poświęconej inżynierii oprogramowania.

1. Przez ticket należy rozumieć problem konfiguracyjny, który napotkaliśmy na naszej drodze i który został przez nas zaraportowany. Powinien on zostać rozwiązany przez suport team. [↑](#footnote-ref-2)
2. ERP- Enterprise Resource Planning, systemy informatyczne wspomagające zarządzanie przedsiębiorstwem(odzwierciedlenie zachodzących procesów logistycznych, tworzenie łańcuchów dostaw, etc.). [↑](#footnote-ref-3)
3. UML- Unified Modeling Language, język modelowania systemów informatycznych, oparty o koncepcję obiektowości. [↑](#footnote-ref-4)
4. Celowo stosowane jest nazewnictwo w języku angielskim, gdyż tym właśnie językiem operuje się zazwyczaj przy okazji tworzenia projektów informatycznych. [↑](#footnote-ref-5)
5. Release- wersja oprogramowania, która jest wypuszczana do użytku. [↑](#footnote-ref-6)
6. Przez koszt w tym wypadku należy rozumieć czas potrzebny na wykonanie danej czynności liczony w dniach. [↑](#footnote-ref-7)
7. IIS- Internet Information Services jest to zbiór usług sieciowych dla systemów Microsoftu, które udostępniają protokoły, dzięki którym można w łatwy sposób uruchomić serwery FTP, http czy SMTP. [↑](#footnote-ref-8)
8. Manifest zwinnego wytwarzania oprogramowania, deklaracja pewnych zasad, które powinny być przestrzegane w każdym projekcie informatycznym budowanym z wykorzystaniem metodyk zwinnych. [↑](#footnote-ref-9)
9. Maintenance, czyli konserwacja oprogramowania. Czynności mające na celu usprawnienie oprogramowania, jego adaptację. Ogólnie opiekowanie się oprogramowaniem. [↑](#footnote-ref-10)
10. Inżynieria wsteczna, reengineering- proces odwrotny do tworzenia oprogramowania. Jest to analiza istniejącego już programu i zrozumienie jego działania, tak by wprowadzić niezbędne modyfikacje(związane np. z bezpieczeństwem). [↑](#footnote-ref-11)