Uniwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Sprawozdanie 4: Planowanie procesu wdrożenia produktu informatycznego za pomocą UML

Dawid Roszman, Filip Maćkowiak

Imię i nazwisko:	Dawid Roszman, Filip Maćkowiak
Nr. indeksu:	285808, 288497
Nazwa uczelni:	Uniwersytet Gdański
Kierunek:	Informatyka Praktyczna
Specjalność:	Brak
Prowadzący:	dr. inż. Stanisław Witkowski
Nazwa ćwiczenia:	Planowanie procesu wdrożenia produktu informatycznego za pomocą UML

Data zajęć:	08.11.2024
Data oddania:	14.11.2024
Miejsce na ocenę:	

Spis treści

1 Elementy języka UML

UML to akronim pochodzący od angielskiego określenia Unified Modeling Language. W polskim tłumaczeniu znany jest jako zunifikowany język modelowania.[1]

1.1 Elementy strukturalne

- **Klasa (Class)**:
  - **Atrybuty**: Właściwości klasy.
  - **Metody**: Funkcje lub operacje klasy.
- **Obiekt (Object)**: Konkretna instancja klasy z określonymi wartościami atrybutów.
- **Interfejs (Interface)**: Zbiór operacji, które klasa musi implementować.
- **Pakiet (Package)**: Grupuje powiązane elementy w logiczne jednostki.
- **Komponent (Component)**: Fizyczna część systemu (np. pliki, biblioteki).
- **Węzeł (Node)**: Reprezentuje zasoby obliczeniowe lub środowiska uruchomieniowe (np. serwery).

1.2 Relacje między elementami

- **Związek (Relationship)**: Ogólny termin odnoszący się do powiązań między elementami.
- **Asocjacja**: Połączenie między klasami.
- **Agregacja**: Typ asocjacji, który reprezentuje relację całość-część.
- **Kompozycja**: Silniejsza forma agregacji z regułami żywotności obiektu.
- **Dziedziczenie (Inheritance)**: Relacja między klasą bazową a pochodną.
- **Realizacja (Realization)**: Relacja między interfejsem a jego implementacją.
- **Include**: Relacja między przypadkami użycia, w której jeden przypadek użycia zawsze wywołuje inny przypadek użycia w ramach swojego wykonania. Jest stosowana, gdy pewna funkcjonalność jest zawsze wykorzystywana w kontekście innego przypadku użycia.
- **Extend**: Relacja między przypadkami użycia, w której opcjonalna funkcjonalność może być dodana do podstawowego przypadku użycia w zależności od warunków. Jest stosowana, gdy dodatkowe działania są wykonywane tylko w określonych okolicznościach.

1.3 Elementy behawioralne

- **Przypadek użycia (Use Case)**: Reprezentuje funkcjonalność systemu z punktu widzenia użytkownika.
- **Aktor (Actor)**: Użytkownik lub inny system wchodzący w interakcję z przypadkiem użycia.
- **Stan (State)**: Konkretny etap w cyklu życia obiektu.
- **Wiadomość (Message)**: Komunikacja między obiektami w diagramach sekwencji i komunikacji.

2 Modelowanie wymagań za pomocą przypadków użycia

Modelowanie wymagań za pomocą przypadków użycia to technika, która opisuje interakcje między użytkownikami (aktorami) a systemem w celu osiągnięcia określonych celów. Jest to sposób przedstawienia wymagań systemu, który jest zrozumiały zarówno dla osób technicznych, jak i nietechnicznych, co ułatwia komunikację między zespołami projektowymi a interesariuszami.

2.1 Kluczowe elementy przypadków użycia

- **Przypadki użycia jako narzędzie do zbierania wymagań:** Przypadki użycia pomagają zebrać i uporządkować wymagania systemu. Każdy przypadek użycia odpowiada za jeden scenariusz interakcji użytkownika z systemem, który ma na celu wykonanie określonego zadania.
- **Aktorzy:** Przypadki użycia zaczynają się od określenia aktorów, czyli osób, grup lub innych systemów, które wchodzą w interakcję z systemem. Aktorami mogą być użytkownicy (np. administrator, klient) lub systemy zewnętrzne (np. baza danych).
- **Scenariusze:** Każdy przypadek użycia opisuje scenariusze interakcji. Scenariusz główny pokazuje typową interakcję (tzw. "happy path"), a alternatywne scenariusze uwzględniają wyjątki i możliwe alternatywne działania.
- **Opis systemu:** Przypadki użycia koncentrują się na tym, co system ma robić w odpowiedzi na interakcje z użytkownikami, ale nie opisują szczegółowo, jak system to realizuje. Skupiają się na „co” zamiast na „jak”.
- **Przykład:**
  - Przypadek użycia: „Zaloguj się”
  - Aktor: Użytkownik
  - Opis: Użytkownik wprowadza dane logowania (login i hasło), a system je weryfikuje i udostępnia dostęp, jeśli są poprawne.
- **Cele systemu:** Dzięki przypadkom użycia zespół projektowy i interesariusze mogą określić cele systemu i priorytetyzować wymagania, co pomaga w planowaniu implementacji.
- **Prostota i komunikacja:** Modele przypadków użycia są łatwe do zrozumienia przez wszystkich zaangażowanych, co poprawia komunikację między zespołami projektowymi a klientami.

2.2 Korzyści z modelowania wymagań przez przypadki użycia

- **Jasność i zrozumiałość:** Przypadki użycia są zrozumiałe dla osób technicznych i nietechnicznych.
- **Organizacja wymagań:** Pomagają uporządkować wymagania systemu.
- **Spójność:** Zapobiegają pominięciu ważnych funkcji lub interakcji.
- **Dostosowanie do potrzeb użytkowników:** Skupiają się na celach użytkowników, co pomaga lepiej dopasować system do ich potrzeb.
- **Priorytetyzacja:** Pomagają określić, które funkcje są najważniejsze i powinny być realizowane najpierw.

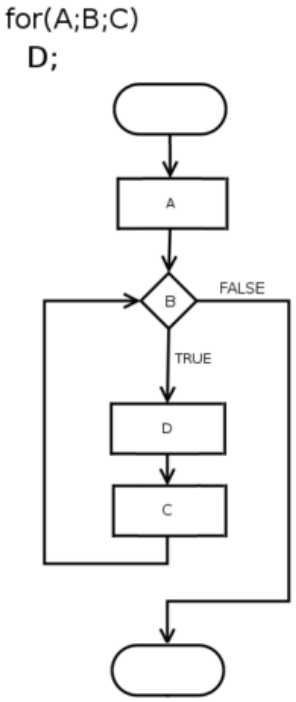
3 Diagramy czynności i sekwencji

3.1 Diagram czynności

Diagram czynności (zwany czasami diagramem aktywności) w języku UML służy do modelowania czynności i zakresu odpowiedzialności elementów bądź użytkowników systemu. Jest niejako podobny do diagramu stanów, jednak w odróżnieniu od niego nie opisuje działań związanych z jednym obiektem a wieloma, pomiędzy którymi może występować komunikacja przy wykonywaniu czynności.[2]

Opis symboli w diagramach aktywności

- **Początkowy i końcowy stan akcji**, przedstawiane odpowiednio jako wypełnione koło oraz wypełnione koło w okręgu.
- **Stan akcji zawierający etykietę ją opisującą**, obrazowany za pomocą prostokąta z zaokrąglonymi narożnikami.
- **Przejście przepływu sterowania** (ciągła strzałka), występujące pomiędzy czynnościami. Zakończenie jednej czynności powoduje rozpoczęcie drugiej.
- **Przejście przepływu obiektów** (przerywana strzałka). Obiekt występuje pomiędzy aktywnościami, co oznacza że jest otrzymywany na wyjściu pierwszej z nich, a pobierany na wejściu drugiej.
- **Tory pływackie**, rysowane za pomocą linii ciągłych. Służą do określania, który element systemu wykonuje dane akcje.
- **Decyzje** (obrazowane za pomocą rombów), służące do wyboru jednego przejścia przepływu sterowania. Odpowiednie wyjścia opisywane są warunkami, które muszą zostać spełnione, by dane przejście mogło zajść.
- **Współbieżność** obrazowana jest za pomocą pogrubionej kreski i dzieli się na dwa elementy:
  - **Synchronizacja sterowania** — aby nastąpiło przejście (lub przejścia) wychodzące, muszą wystąpić wszystkie przejścia przychodzące.
  - **Rozdzielenie sterowania** — po zajściu przejść przychodzących występują jednocześnie wszystkie przejścia wychodzące.
- **Opcjonalnie**, możemy wyznaczyć elementy rozproszone, nadając im symbol :R przy przejściu w kolejny stan.



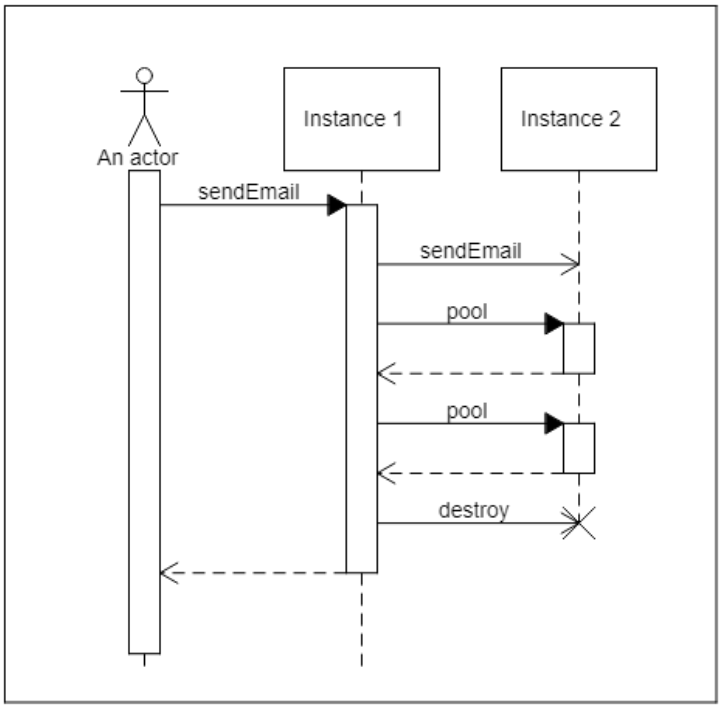
Rysunek 1: Przykład diagramu czynności[2]

3.2 Diagram sekwencji

Diagram sekwencji (ang. sequence diagram) jest jednym z tak zwanych diagramów interakcji. Kładzie on nacisk na komunikację, która odbywa się pomiędzy poszczególnymi klasami/obiektami. Diagram sekwencji pokazuje dokładnie sekwencję wykonania metod w poszczególnych obiektach. Diagram ten przydaje się do pokazania przebiegu skomplikowanej komunikacji.

Elementy diagramu sekwencji

Każdy z obiektów reprezentowany jest jako prostokąt połączony z pionową kreską. Ta linia oznacza „linię życia" – czas życia obiektu. Na diagramie może występować także tak zwany aktor. Aktor to człowiek albo system, który może brać udział w komunikacji.



Rysunek 2: Przykład diagramu sekwencji[1]

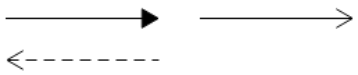
Wąskie pionowe prostokąty na liniach życia oznaczają czas, w którym dany aktor/obiekt był aktywny. Aktywność była niezbędna do wypełnienia żądania, które dany obiekt wysłał/otrzymał.

Niektóre obiekty mogą żyć krócej niż pozostałe. Koniec życia obiektu zaznaczany jest przez znak X na ich linii życia.

Powyższy diagram może służyć jako przykład opisujący mechanizm wysyłania wiadomości e-mail. Na początku aktor inicjalizuje proces, Instance 1 obsługuje akcję sendEmail przekazując ją asynchronicznie do Instance 2. Następnie dwukrotnie sprawdza czy wysłanie wiadomości się powiodło, po czym zwraca informację do aktora.

Rodzaje komunikatów

Pionowe kreski oznaczają linię życia. Im wyżej na diagramie, tym wcześniej coś się wydarzyło. Poziome kreski oznaczają komunikaty.



Rysunek 3: Przykład strzałek na diagramie sekwencyjnym

Strzałki w lewej kolumnie oznaczają komunikaty synchroniczne. Strzałka z ciągłą liną oznacza wysłanie komunikatu, strzałka z przerywaną linią otrzymanie odpowiedzi. W prawej kolumnie znajduje się strzałka reprezentującą asynchroniczne wysłanie komunikatu.[1]

4 Modelowanie klas i powiązań między nimi

Diagram klas (ang. class diagram) pokazuje klasy i zależności między nimi, umożliwiając szczegółowy opis klas z atrybutami i metodami. Jest jednym z najczęściej używanych diagramów w UML, pozwalającym na przedstawienie fragmentu systemu.

Klasa

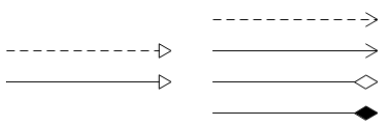
Klasa w diagramie to prostokąt podzielony na trzy sekcje: nazwę klasy, atrybuty i metody. Elementy statyczne są podkreślone, a abstrakcyjne pisane kursywą. Atrybuty i metody mogą mieć modyfikatory dostępu:

- +: publiczny,
- #: chroniony (protected),
- -: prywatny.

Relacje Między Klasami

Pomiędzy klasami mogą występować różne relacje, takie jak:

- **Zależność** (przerywana linia) – jedna klasa używa drugiej,
- **Asocjacja** (ciągła linia) – klasy są powiązane, ale nie zawierają siebie nawzajem,
- **Agregacja** (ciągła linia z pustym rombem) – jedna klasa „własciwie" zawiera inną,
- **Kompozycja** (ciągła linia z wypełnionym rombem) – klasa „zarządza życiem" drugiej.



Rysunek 4: Przykład strzałek na diagramie klasowym[1]

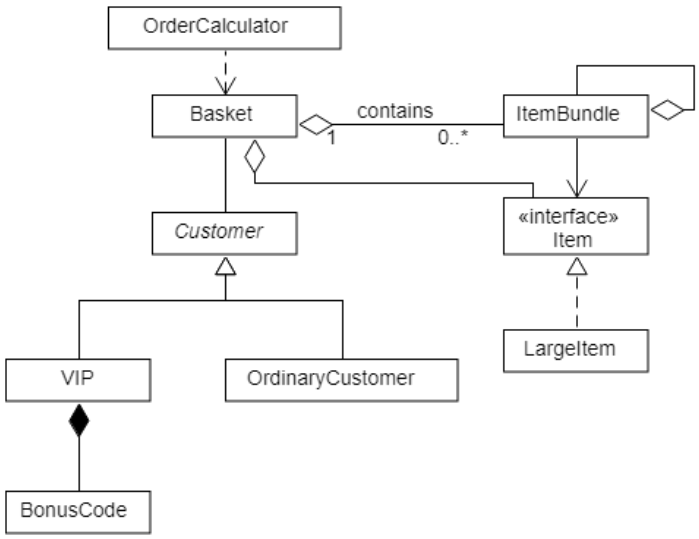
Relacje Dwukierunkowe

Relacja „wiele do wielu" często wymaga wprowadzenia klasy pośredniczącej. Na przykład, w przypadku książki (Book) i autora (Author), mamy dwie klasy pośrednie reprezentujące autorstwo (Authorship). Relacje między klasami mogą być jednokierunkowe lub dwukierunkowe.

Przykład

Przykład relacji w diagramie klas:

- Klasa `LargeItem` implementuje interfejs `Item` (implementacja),
- Klasy `VIP` i `OrdinaryCustomer` dziedziczą po `Customer` (dziedziczenie),
- `OrderCalculator` używa `Basket` (zależność),
- `Basket` ma asocjację z `Customer` (asocjacja),
- `VIP` ma kompozycję z `BonusCode` (kompozycja).



Rysunek 5: Przykład diagramu klasowego[1]

5 Diagramy komponentów

Diagram komponentów (ang. component diagram) przedstawia projekt systemu z wyższej perspektywy. Kluczową rolę w tym diagramie odgrywają komponenty.

Komponent

Komponent to prostokąt z ikonką w prawym górnym rogu. Może wymagać i udostępniać interfejsy. Interfejsy są reprezentowane jako kreski z kółkiem (interfejs udostępniany) lub półkolem (interfejs wymagany). Relacje między komponentami opierają się na interfejsach, co jest rodzajem zależności.

Rysunek 6: Przykład Komponentu[1]

Definicja Komponentu

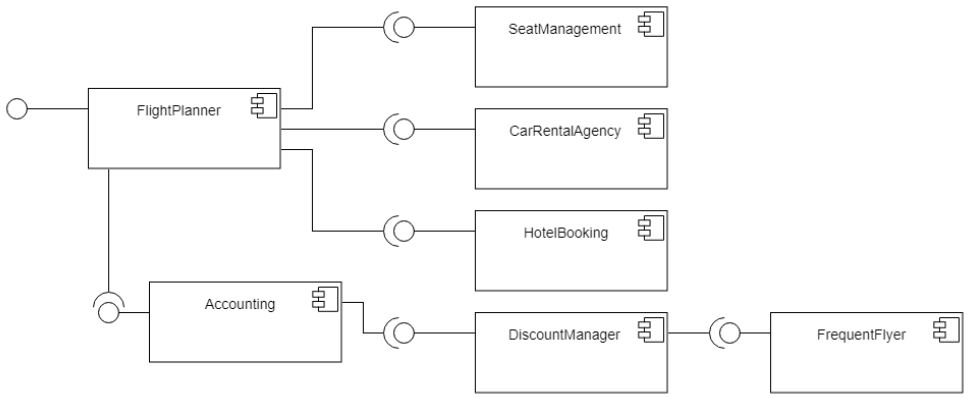
Komponent to wydzielona część systemu, która ma interfejsy komunikacyjne i może być wymieniana w ramach swojego środowiska. Każdy komponent jest zastępowalny przez inną implementację, pod warunkiem że spełnia wymagania interfejsów.

Zastosowanie Komponentów

Komponenty mogą reprezentować różne poziomy abstrakcji: od pojedynczych klas, przez zestawy klas w pakiecie/module, po większe części aplikacji. Ważne jest, aby diagram komponentów zachował spójność i pokazywał komponenty z podobnej odległości.

Przykład Diagramu Komponentów

Przykładowy diagram komponentów systemu do rezerwacji biletów lotniczych pokazuje komponenty zależne od siebie, z interfejsami umożliwiającymi komunikację między nimi. Interfejsy na diagramie zostały pominięte dla uproszczenia.



Rysunek 7: Przykład diagramu komponentowego[1]

6 Podział modelu na pakiety

Pakiety w Unified Modeling Language służą do grupowania elementów i udostępniania przestrzeni nazw dla zgrupowanych elementów. Pakiet może zawierać inne pakiety, zapewniając w ten sposób hierarchiczną organizację pakietów.

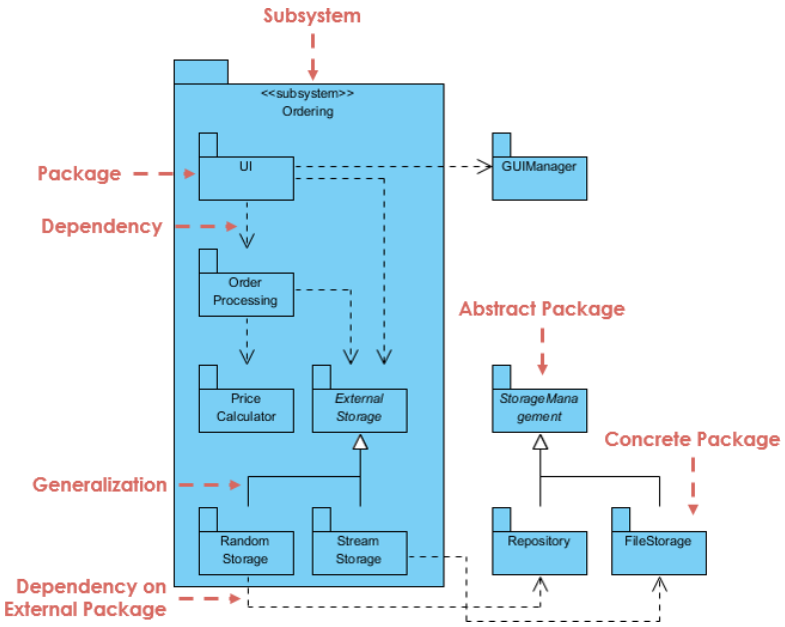
Prawie wszystkie elementy UML można pogrupować w pakiety. W ten sposób klasy, obiekty, przypadki użycia, komponenty, węzły, instancje węzłów itp. Można organizować w pakiety, dzięki czemu organizacja niezliczonych elementów zawartych w rzeczywistym modelu UML jest łatwa do zarządzania.

Duże systemy stanowią szczególne wyzwanie. Rysowanie modelu klasy dla dużego systemu jest zbyt skomplikowane, aby mogły je zrozumieć osoby nieznające szczegółów. Istnieje zbyt wiele powiązań między klasami, by można było je łatwo zrozumieć. Przydatną techniką radzenia sobie z tym problemem jest użycie pakietów UML.

Pakiety w UML służą do:

- grupowania elementów,
- zapewnienia przestrzeni nazw dla zgrupowanych elementów,
- hierarchicznej organizacji, gdy jeden pakiet zawiera inne pakiety.

Diagram pakietów (diagram struktury) pokazuje rozmieszczenie i organizację elementów modelu w średnich i dużych projektach. Może przedstawiać strukturę oraz zależności między podsystemami lub modułami, oferując różne widoki systemu, na przykład jako wielowarstwową aplikację.[3]



Rysunek 8: Przykład diagramu pakietowego[3]

7 Modelowanie wdrożenia systemu

Diagram wdrożenia (ang. deployment diagram) koncentruje się na przedstawieniu zależności między oprogramowaniem a sprzętem, na którym to oprogramowanie będzie działać. W przeciwieństwie do diagramów klas czy komponentów, które skupiają się głównie na aspektach programistycznych systemu, diagram wdrożenia uwzględnia także kwestie związane z infrastrukturą sprzętową, czyli pokazuje, jak aplikacja jest instalowana i wdrażana w konkretnym środowisku.

7.1 Cel diagramu wdrożenia

Diagram wdrożenia służy do ukazania sposobu, w jaki system (lub jego poszczególne komponenty) zostaną uruchomione na odpowiednich urządzeniach i maszynach. Pomaga to w planowaniu wdrożenia systemu, zwłaszcza w kontekście rozmieszczenia oprogramowania na różnych serwerach, urządzeniach czy w chmurze. Dzięki temu zespoły projektowe i inżynierowie wdrożeniowi mogą łatwiej zaplanować proces instalacji i uruchomienia aplikacji.

7.2 Elementy diagramu wdrożenia

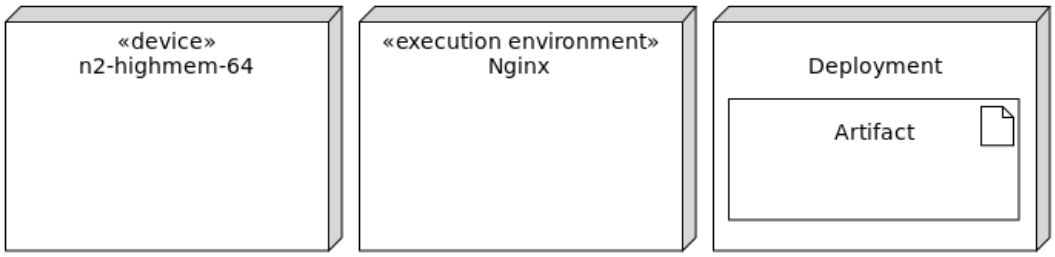
Na diagramie wdrożenia spotykamy następujące kluczowe elementy:

- **Sprzęt (hardware):** Reprezentuje maszyny, serwery lub urządzenia, na których będzie działać aplikacja. Na przykład, może to być serwer typu n2-highmem-64
- **Oprogramowanie (software):** Obejmuje różne komponenty aplikacji, takie jak serwery, bazy danych czy usługi zewnętrzne.
- **Artefakty (artifacts):** To konkretne pliki lub zasoby związane z aplikacją, takie jak pliki konfiguracyjne, aplikacje czy bazy danych. Artefakty są przechowywane w deployment (element wskazujący na proces wdrożenia) i mogą obejmować pliki binarne lub skrypty.

Przykład poniżej pokazuje elementy, które możemy spotkać na diagramach wdrożenia:

Kolejno od lewej na rysunku możesz zobaczyć:

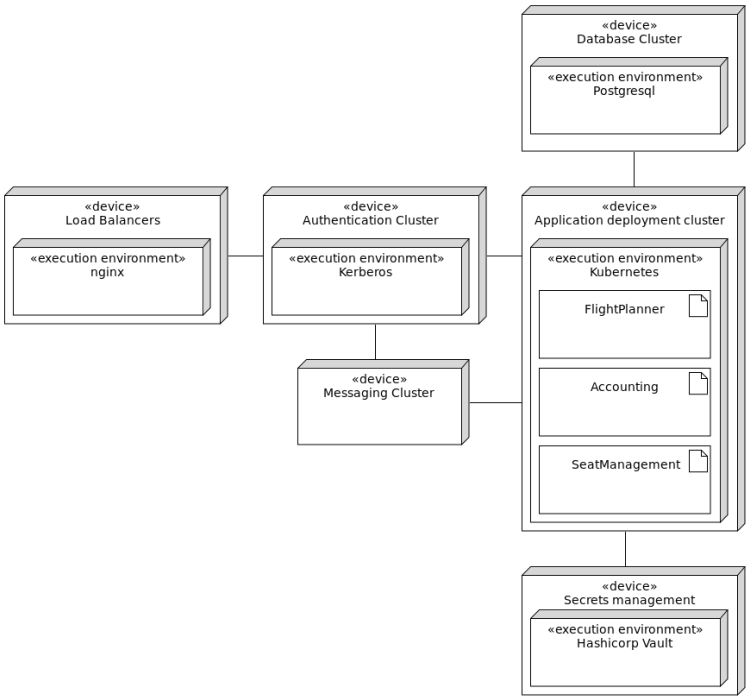
- serwer typu n2-highmem-64, który jest odpowiedzialny za hostowanie aplikacji.
- element o nazwie Nginx, który reprezentuje serwer HTTP,
- element Deployment, który wewnątrz zawiera artefakt o nazwie Artifact.



Rysunek 9: Elementy diagramu wdrożenia[1]

Na diagramie mogą pojawić się również stereotypy, które służą jako dodatkowe oznaczenia, pozwalające na lepsze zrozumienie specyfiki poszczególnych elementów.

7.3 Przykład diagramu wdrożenia

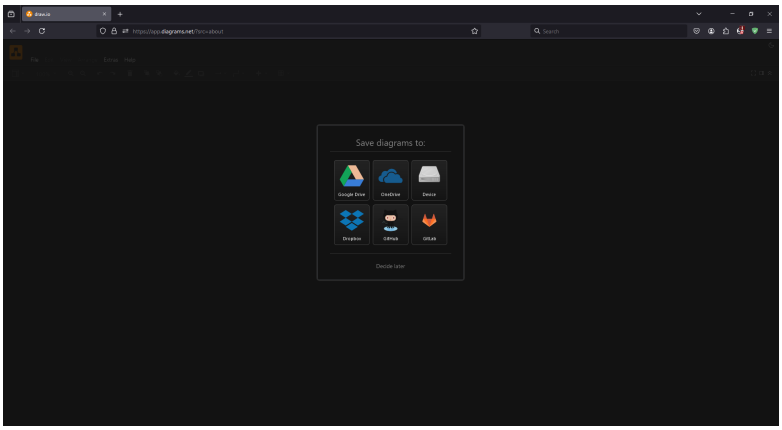


Rysunek 10: Przykład diagramu wdrożenia[1]

Na diagramie wdrożenia widać zestawy maszyn (klastry), które są przeznaczone do wdrożenia różnych komponentów aplikacji. Łącza między elementami pokazują, jak te komponenty komunikują się ze sobą w ramach wdrożonego systemu. Może to obejmować powiązania między bazą danych, serwerem aplikacji i serwerem HTTP, ilustrując całą infrastrukturę potrzebną do uruchomienia aplikacji.

8 Draw.io - opis działania

Draw.io (obecnie znane jako **diagrams.net**) to popularne narzędzie online do tworzenia diagramów, które może być używane do rysowania różnych typów diagramów, w tym diagramów UML (Unified Modeling Language). Jest to intuicyjna i łatwa w użyciu platforma, która pozwala na szybkie tworzenie diagramów, współpracę z innymi użytkownikami oraz integrację z różnymi systemami przechowywania danych (takimi jak Google Drive, GitHub, OneDrive).

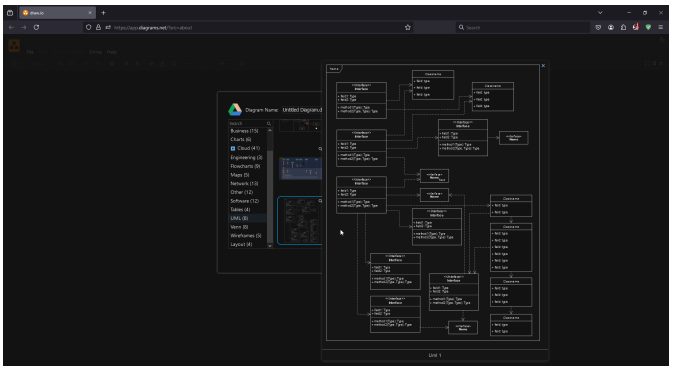


Rysunek 11: Możliwość integracji z systemami przechowywania danych on-line

8.1 Draw.io w kontekście diagramów UML

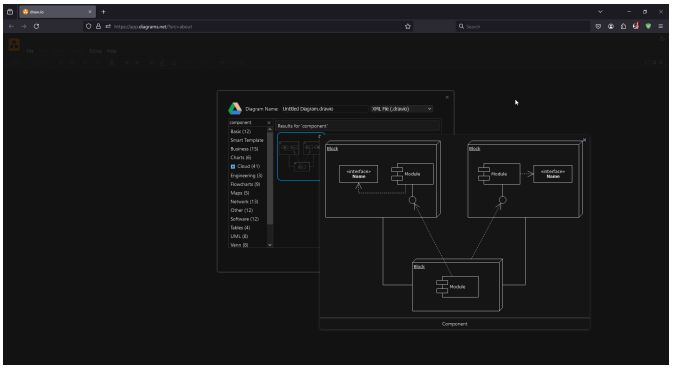
Diagramy UML służą do wizualizacji różnych aspektów systemu i jego struktury. Draw.io to narzędzie, które umożliwia tworzenie wszystkich standardowych diagramów UML, w tym:

- **Diagram klas (Class Diagram)**: Draw.io umożliwia łatwe tworzenie diagramów klas, które pokazują strukturę klas w systemie oraz ich relacje. Można dodać klasy, atrybuty, metody, a także różne typy powiązań między klasami, takie jak dziedziczenie, agregacja czy kompozycja.



Rysunek 12: Szablon diagramu klasowego na Draw.io

- **Diagram komponentów (Component Diagram)**: Umożliwia tworzenie diagramów, które przedstawiają podział systemu na komponenty i zależności między nimi. Można z łatwością dodać różne komponenty, interfejsy i zależności między nimi.

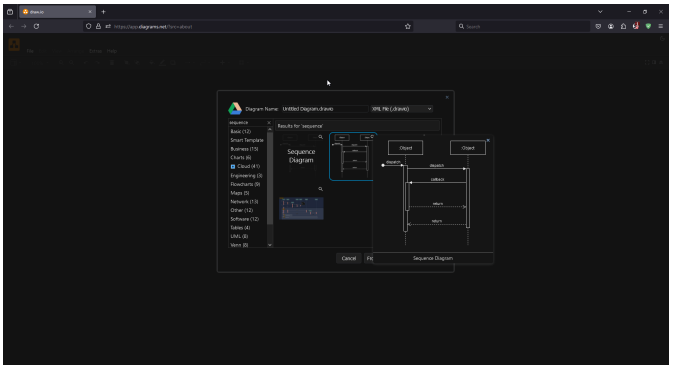


Rysunek 13: Szablon diagramu komponentów na Draw.io

- **Diagram przypadków użycia (Use Case Diagram)**: Na Draw.io można bardzo łatwo stworzyć diagramy przypadków użycia, które obrazują interakcje między aktorami (użytkownikami lub systemami zewnętrznymi), a systemem. Pozwala to na łatwe przedstawienie funkcji, które system ma realizować, z perspektywy użytkownika.

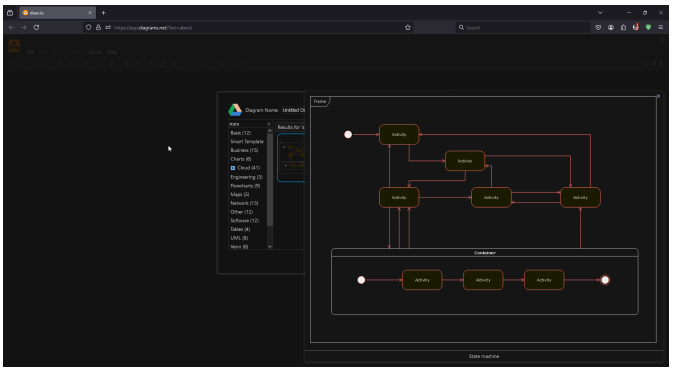
Rysunek 14: Przykładowy diagram UML stworzony w Draw.io

- **Diagram sekwencji (Sequence Diagram)**: W Draw.io można tworzyć diagramy sekwencji, które przedstawiają kolejność interakcji między obiektami w systemie. Użytkownik może dodać aktorów, obiekty i komunikaty między nimi, co umożliwia modelowanie przepływu informacji w systemie.



Rysunek 15: Szablon diagramu sekwencji na Draw.io

- **Diagram stanu (State Diagram)**: Draw.io pozwala na tworzenie diagramów stanów, które przedstawiają możliwe stany obiektów w systemie oraz przejścia między nimi w odpowiedzi na zdarzenia.



Rysunek 16: Szablon diagramu stanu na Draw.io

8.2 Funkcje Draw.io wspierające tworzenie diagramów UML

- **Szablony i elementy UML**: Draw.io oferuje gotowe szablony i zestawy symboli UML, co ułatwia tworzenie diagramów. Istnieją wbudowane biblioteki z symbolami odpowiednimi do różnych typów diagramów UML, takich jak klasy, obiekty, interfejsy, aktorzy czy komponenty.
- **Łatwość użycia**: Draw.io oferuje prosty interfejs *przeciągnij i upuść*, który pozwala na łatwe dodawanie i edytowanie elementów diagramu. Dzięki temu nawet osoby, które nie mają dużego doświadczenia w tworzeniu diagramów UML, mogą szybko zrozumieć i stworzyć wymagane modele.

- **Współpraca w czasie rzeczywistym**: Draw.io umożliwia współpracę z innymi użytkownikami w czasie rzeczywistym, co jest przydatne w pracy zespołowej. Dzięki tej funkcji wiele osób może jednocześnie edytować ten sam diagram, co ułatwia wspólne tworzenie i weryfikację diagramów UML.
- **Integracje z chmurą**: Możliwość zapisywania diagramów w chmurze (Google Drive, GitHub, OneDrive) oraz eksportowania ich do różnych formatów (np. PNG, PDF, SVG) sprawia, że Draw.io jest bardzo wygodne w pracy nad projektami w zespole.
- **Możliwość personalizacji**: Draw.io pozwala na pełną personalizację diagramów, zarówno pod względem kolorów, czcionek, jak i układu elementów. Dzięki temu można dostosować diagramy do indywidualnych potrzeb i preferencji.

8.3 Zastosowanie Draw.io do diagramów UML

- **Prototypowanie i modelowanie systemów**: Draw.io jest idealnym narzędziem do szybkiego prototypowania i modelowania różnych aspektów systemu, szczególnie w fazie analizy i projektowania oprogramowania. Dzięki możliwości łatwego rysowania diagramów UML, zespół projektowy może szybko zwizualizować strukturę systemu, procesy i interakcje.
- **Dokumentacja systemów**: Diagramy UML są powszechnie wykorzystywane w dokumentacji systemów. Dzięki Draw.io, zespoły mogą tworzyć profesjonalne diagramy, które stanowią część dokumentacji technicznej i pomagają w zrozumieniu struktury i działania systemu.

9 Wnioski

- Poznaliśmy różne rodzaje diagramów UML (przypadków użycia, czynności, sekwencji, klas, komponentów, pakietów, wdrożenia systemów)
- Nauczyliśmy się tworzyć diagramy przypadków użycia, bazując na istniejących systemach
- Zapoznaliśmy się z działaniem profesjonalnego narzędzia do prototypowania i modelowania systemów Draw.io

Literatura

[1] Marcin Pietraszek. Podstawy uml, 11 2020.

[2] Wikipedia. Diagram czynności.

[3] VisialParadigm. Co to jest pakiet? co to jest diagram pakietów w uml?, 2 2022.