Inżynieria oprogramowania

Wykład 1:

Wprowadzenie do Inżynierii Oprogramowania

Marek Krętowski
pokój 206
e-mail: mkret@ii.pb.bialystok.pl
http://aragorn.pb.bialystok.pl/~mkret

Literatura

- K. Subieta *"Wprowadzenie do inżynierii oprogramowania"*, Wydawnictwo PJWSTK, 2002
- I. Sommerville *"Inżynieria oprogramowania"*, WNT 2003
- S. Szejko (red.) *"Metody wytwarzania oprogramowania"*, Mikom, 2002
- J. Górski (Ed.) *"Inżynieria oprogramowania w projekcie informatycznym"*, Mikom 1999 (wyd. II, 2000)
- A. Jaszkiewicz *"Inżynieria oprogramowania"*, Helion 1997
- G. Booch, J. Rumbaugh, I. Jacobson *"The Unified Modeling Language - User Guide"*, Addison-Wesley 1999 (polskie wydanie "UML: przewodnik użytkownika", WNT 2001)
- M. Fowler, K. Scott *"UML w kropelce"*, Oficyna Wydawnicza LTP, 2002
- J. Schmuller, *"UML dla każdego"*, Helion, 2003

Organizacja zaliczenia wykładu

- Wykład kończy się egzaminem pisemnym:
  - część praktyczna (50% punktów) - 2-3 zadania polegające na zaprojektowaniu na podstawie krótkich scenariuszy elementów systemu inf. (diagramy w UML-u)
  - część teoretyczna (50% punktów) - 3-4 krótkie pytania z materiału prezentowanego na wykładzie
  - zalicza co najmniej 50% punktów
- Terminy:
  - egzamin (najlepiej pierwszy dzień sesji), aby przystąpić do egzaminu należy mieć wpisaną do indeksu pozytywną ocenę z pracowni specjalistycznej
  - egzamin poprawkowy (np. ostatni dzień sesji poprawkowej)

A co to jest oprogramowanie?

Dwie podstawowe grupy produktów programowych:

- produkty **powszechne** ("oprogramowanie w torebkach foliowych") - samodzielne systemy tworzone przez firmy programistyczne a następnie sprzedawane na wolnym rynku wszystkim, którzy zechcą je zakupić; np.:
  - oprogramowanie biurowe, pakiety graficzne, programy narzędziowe, ...
- produkty **na zamówienie** (dostosowywane) - zamawiane przez konkretnego klienta i opracowane specjalnie dla niego (od podstaw lub dostosowanie istniejących rozwiązań ogólnych), np.:
  - systemy do wspomagania określonych procesów przedsiebiorstw czy instytucji, systemy sterujące urządzeniami elektronicznymi

Dobre oprogramowanie jest:

- zgodne z wymaganiami użytkownika,
- niezawodne,
- efektywne,
- łatwe w konserwacji,
- ergonomiczne,
- rozsądne w cenie

?

Definicja inżynierii oprogramowania

- Termin inżynieria oprogramowania został zaproponowany w 1968 roku na konferencji zwołanej w celu przedyskutowania tzw. kryzysu oprogramowania
- **Inżynieria oprogramowania** jest dziedziną zajmującą się tworzeniem i stosowaniem jasno zdefiniowanych zasad i metod inżynierskich służących do wytwarzania w sposób ekonomiczny niezawodnego oprogramowania działającego na rzeczywistych komputerach
- **Inżynieria oprogramowania** jest wiedzą techniczną dotycząca wszystkich faz cyklu życia oprogramowania. Traktuje oprogramowanie jako produkt, który ma spełniać potrzeby techniczne, ekonomiczne lub społeczne

Obszary zainteresowania IO

- Projektowanie i analiza systemów
- Planowanie, szacowanie kosztów, harmonogramowanie
- Zarządzanie przedsięwzięciem (projektem informatycznym)
- Produktywność programisty i projektanta
- Techniki pracy zespołowej
- Zapewnianie niezawodności oprogramowania
- Procedury kontroli jakości (testowanie systemów)
- Sposoby dokumentowania pracy (techniczne i użytkownika)
- Konserwacja oprogramowania (rozszerzanie możliwości oraz usuwanie błędów)
- ...

Produkcja oprogramowania jest procesem składającym się z wielu faz. Kodowanie (pisanie programów) jest tylko jedną z nich, niekoniecznie najważniejszą.

Co to jest CASE?

- Skrót **CASE** oznacza **Computer-Aided Software Engineering** - Wspomagana Komputerowo Inżynieria Oprogramowania
- Obejmuje różnorodne programy wykorzystywane do wspomagania czynności procesu tworzenia oprogramowania
  - analizy wymagań, modelowania systemu (np. edytory graficzne zgodne z notacją, możliwość weryfikacji modelu zgodnie z regułami metody, raportowanie), generowania kodu na podstawie modelu, wyszukiwania i usuwania błędów oraz testowania
- Zwyczajowo wyróżnia się dwa rodzaje narzędzi CASE:
  - **upper-CASE -** narzędzia wykorzystywane w początkowych fazach rozwoju oprogramowania, czyli wspomaganie analizy i projektowania
  - **lower-CASE -** wspomagające implementowanie i testowanie (m.in.. systemy analizy wydajności, debuggery, generatory przypadków testowych); najczęściej związane z konkretnym środowiskiem implementacyjnym
- Przykłady:
  - IBM Rational Rose, Poseidon for UML, Select Enterprise, Rational Unified Process

Kryzys oprogramowania

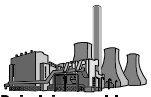
- Sprzeczność pomiędzy odpowiedzialnością, jaka spoczywa na współczesnych SI, a ich zawodnością wynikającą ze złożoności i ciągle niedojrzałych metod tworzenia i weryfikacji
- Niska kultura ponownego użycia wytworzonych komponentów; niski stopień powtarzalności przedsięwzięć
- Długi i kosztowny cykl tworzenia oprogramowania, wysokie prawdopodobieństwo niepowodzenia
- Cykl życia SI, wymagający stałych (często globalnych) zmian => ogromne koszty utrzymania oprogramowania
- Ciągły brak odpowiednich narzędzi i języków programowania
- Frustracje projektantów i programistów wynikające z szybkiego postępu w zakresie języków, narzędzi i metod oraz uciążliwości i długotrwałości procesów produkcji, utrzymania i pielęgnacji opr.
- Uzależnienie organizacji od systemów komp. i przyjętych technologii przetwarzania informacji, które nie są stabilne w długim horyzoncie czasowym
- Problemy współdziałania niezależnie zbudowanego oprogramowania, szczególnie istotne przy dzisiejszych tendencjach integracyjnych
- Problemy przystosowania działających systemów do nowych wymagań i platform sprzętowo-programowych

Walka z kryzysem oprogramowania

- Stosowanie technik i narzędzi ułatwiających pracę nad złożonymi systemami
- Korzystanie z metod wspomagających analizę nieznanych problemów oraz ułatwiających wykorzystanie wcześniejszych doświadczeń
- Usystematyzowanie procesu wytwarzania oprogramowania, tak aby ułatwić jego planowanie i monitorowanie
- Wytworzenie wśród producentów i nabywców przekonania, że budowa dużego systemu wysokiej jakości jest zadaniem wymagającym profesjonalnego podejścia

Podstawowym powodem kryzysu oprogramowania jest złożoność produktów informatyki i procesów ich wytwarzania

Źródła złożoności projektu



Dziedzina problemowa, obejmująca ogromną liczbę wzajemnie uzależnionych aspektów i problemów

Oprogramowanie:
decyzje strategiczne
analiza
projektowanie
konstrukcja
dokumentacja
wdrożenie
szkolenie
eksploatacja
pielęgnacja
modyfikacja

Zespół projektantów podlegający ograniczeniom pamięci, percepcji, wyrażania informacji i komunikacji

Środki i technologie informatyczne: sprzęt, oprogramowanie, sieć, języki, narzędzia, udogodnienia

Potencjalni użytkownicy: czynniki psychologiczne, ergonomia, ograniczenia pamięci i percepcji, skłonność do błędów i nadużyć, tajność, prywatność

Jak walczyć ze złożonością ?

Zasada dekompozycji:
rozdzielenie złożonego problemu na podproblemy, które można rozpatrywać i rozwiązywać niezależnie od siebie i niezależnie od całości.

Zasada abstrakcji:
eliminacja, ukrycie lub pominięcie mniej istotnych szczegółów rozważanego przedmiotu lub mniej istotnej informacji; wyodrębnianie cech wspólnych i niezmiennych dla pewnego zbioru bytów i wprowadzaniu pojęć lub symboli oznaczających takie cechy.

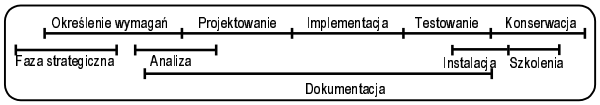
Zasada ponownego użycia:
wykorzystanie wcześniej wytworzonych schematów, metod, wzorców, komponentów projektu, komponentów oprogramowania, itd.

Zasada sprzyjania naturalnym ludzkim własnościom:
dopasowanie modeli pojęciowych i modeli realizacyjnych systemów do wrodzonych ludzkich własności psychologicznych, instynktów oraz mentalnych mechanizmów percepcji i rozumienia świata.

Wyzwania inżynierii oprogramowania

- Wyzwanie dziedzictwa
  - duża część używanych obecnie systemów opracowana została wiele lat temu i pełni ważne funkcje gospodarcze
  - pielęgnacja i rozwój tego typu oprogramowania, aby uniknąć zbędnych kosztów i podtrzymać dostarczanie zasadniczych usług
- Wyzwanie różnorodności
  - coraz częściej wymaga się, aby systemy pracowały jako systemy rozproszone w sieci (różne typy komputerów, systemów operacyjnych i wspomagających)
  - opracowaniemetod budowy niezawodnego oprogramowania, które jest wystarczająco elastyczne, aby radzić sobie z różnorodnością
- Wyzwanie doręczenia
  - wiele tradycyjnych technik inżynierii oprogramowania jest bardzo czasochłonnych (czas jest niezbędny do osiągnięcia dobrej jakości)
  - współczesne procesy gospodarcze są niezwykle dynamiczne
  - skrócenie czasu niezbędnego na dostarczenie wielkich złożonych systemów bez utraty ich jakości

Typowe czynności w cyklu życiowym oprogramowania



- **Faza strategiczna (analiza biznesowa) -** jest zwykle wykonywana przed podjęciem formalnej decyzji o realizacji przedsięwzięcia
  - rozpisanie przetargu
  - zapytania ofertowe
  - określenie wizji produktu
  - analiza potrzeb rynku
- **Określenie wymagań -** określane są cele i szczegółowe wymagania w stosunku do systemu (*Co i przy jakich ograniczeniach system ma robić?*)
- **Analiza (modelowanie) -** budowany jest logiczny model systemu, opisujący sposób realizacji postawionych wymagań *(Jak system ma działać?)*
- **Projektowanie -** opracowanie szczegółowego projektu systemu spełniającego określone wymagania *(Jaki system ma zostać zaimplementowany?)*

Typowe czynności (2)

- **Implementacja -** projekt zostaje zaimplementowany w konkretnym środowisku programistycznym oraz wykonywane są testy poszczególnych modułów
- **Testowanie -** integracja poszczególnych modułów połączona z testowaniem podsystemów oraz całego oprogramowania
- **Instalacja -** następuje przekazanie oprogramowania użytkownikowi (instalacja w siedzibie klienta, inicjowane są struktury danych, ...)
- **Dokumentowanie -** powstają dwa podstawowe typy dokumentacji: techniczna (opisuje proces wytworzenia oprogramowania, istotna dla zespołu projektowego i kierownictwa) oraz użytkowa (opisuje produkt, ukierunkowana na administratorów i przyszłych użytkowników systemu)
- **Wdrożenie** (szkolenia) **-** przeprowadzane są szkolenia przyszłych użytkowników i administratorów
- **Konserwacja (pielęgnacja) -** oprogramowanie jest wykorzystywane przez użytkownika. Dokonuje się modyfikacji polegających na usuwaniu błędów oraz zmian służących rozszerzeniu funkcjonalności systemu

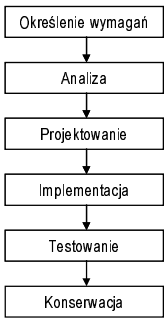
Modele cyklu życia oprogramowania

- Produkcja i eksploatacja oprogramowania jest procesem, którego struktura określana jest modelem cyklu życia
- Modele wprowadzają fazy życia, określają czynności wykonywane w poszczególnych fazach oraz ustalają kolejność tych faz
- Pozwalają uporządkować przebieg prac
- Ułatwiają planowanie zadań oraz monitorowanie przebiegu ich realizacji
- Należy pamiętać, że modele stanowią pewne idealizacje rzeczywistych procesów i w związku z tym nie należy ich nadinterpretowywać

Grupy (typy) modeli:

- sekwencyjne
  - klasyczny model wodospadowy i jego odmiany (z powrotami, realizacja sterowana dokumentami, ...)
  - model V
- iteracyjne:
  - model spiralny
  - programowanie odkrywcze (eksploratywne)
  - realizacja przyrostowa
- tworzenie z użyciem wielokrotnym
- formalne transformacje
- ...

Model wodospadowy
(ang. waterfall model)



- Inne nazwy: model kaskadowy, liniowy, klasyczny
- Zaproponowany poprzez analogię z realizacją przedsięwzięć w tradycyjnych dziedzinach
- Cykl życia jest procesem liniowym, w którym poszczególne fazy występują jedna po drugiej
- Każda z faz musi być zakończona przed rozpoczęciem kolejnej
- Podział na konkretne fazy może się różnić w zależności od autora (np. dodatkowe fazy: analiza i modelowanie, instalacja, ...)
- W praktyce, jeżeli ściśle stosowany, może być użyty jedynie w krótkich projektach, w których wymagania są precyzyjnie określone i zrozumiałe

Model wodospadowy (2)

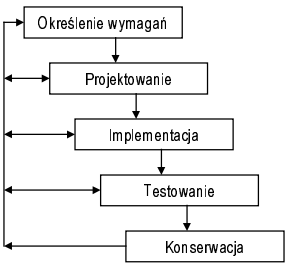
Zalety:

- stosunkowo łatwe zarządzanie przedsięwzięciem (planowanie, harmonogramowanie, monitorowanie)
- ułatwia rozliczenia finansowe, możliwość regulowania płatności po kolejnych etapach

Wady:

- wysoki koszt błędów popełnionych we wcześniejszych fazach i brak możliwości ich korekty w ramach modelu
- długa przerwa w kontaktach z klientem (brak interakcji z klientem od momentu określenia wymagań do instalacji systemu)
- narzucenie twórcom oprogramowania ścisłej kolejności wykonywania prac

Model kaskadowy z powrotami

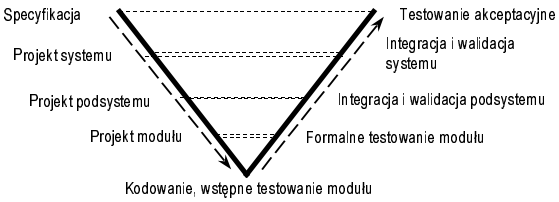


- Nazywany również wariantem iteracyjnym
- Umożliwia na każdym z etapów powrót do poprzedzających faz w wyniku stwierdzenia rozbieżności pomiędzy realizowanym systemem a oczekiwaniami klienta
- Znacznie bardziej realistyczny od modelu oryginalnego
- Ceną za likwidację podstawowej wady jest niestety utrata pewnych zalet (głównie pochodnych prostoty modelu)

Realizacja sterowana dokumentami
(ang. document-driven approach)

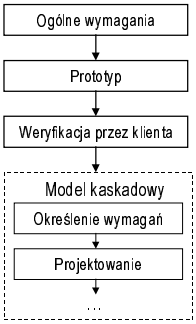
- Bardzo sformalizowana odmiana (realizacja) modelu kaskadowego
- Każda faza kończy się opracowaniem szeregu ściśle określonych dokumentów, w których opisuje się wyniki danej fazy; dokumenty te powinny być wystarczającą podstawą do realizacji kolejnych faz
- Przejście do następnej fazy jedynie po zaakceptowaniu przez zleceniodawcę wszystkich dokumentów
- Zalety:
  - istnieje możliwość realizacji kolejnych faz przez inną firmę
  - ścisła współpraca z klientem (zleceniodawcą)
- Wady:
  - bardzo duży nakład pracy niezbędny do opracowania dokumentacji (ponad 50% całkowitego wysiłku)
  - niezbędne przerwy w realizacji służące do weryfikacji dokumentów przez klienta
- Wykorzystywany przez armię amerykańską do realizacji projektów w Adzie

Model V



- Każdy etap prowadzi do zwiększenia szczegółowości definicji, aż do osiągnięcia dolnego punktu oznaczającego wytworzenie kodu modułów
- Następnie realizowane są kroki intergracyjne umieszczone na drugim ramieniu litery 'V' (są to sprzężone z kolejnymi etapami procesy weryfikacyjne i walidacyjne)

Prototypowanie - makietowanie

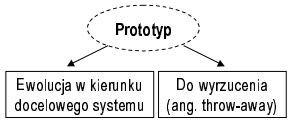


- Cele jest minimalizacja ryzyka związanego z niewłaściwym określeniem wymagań a w szczególności:
  - wykrycie nieporozumeń między klientem a twórcami systemu
  - wykrycie brakujących funkcji i trudnych usług
  - wykrycie braków w specyfikacji wymagań
- Zalecane, gdy określenie wstępnych wymagań jest stosunkowo proste
- Dodatkowe zalety:
  - możliwość demonstracji pracującej wersji systemu
  - możliwość rozpoczęcia szkoleń przed powstaniem pełnej wersji systemu

Prototypowanie (2)

Metody protypowania:

- Niepełna realizacja (tylko wybrane funkcje systemu)
- Języki wysokiego poziomu
- Użycie gotowych komponentów
- Generatory interfejsu użytkownika (wykonywany jest wyłącznie interfejs)
- Szybkie programowanie (ang. quick-and-dirty) - normalne programowanie, ale np. pominięcie obsługi błędów, ...
- Programowanie odkrywcze
- Papier (przedstawienie jedynie interfejsu użytkownika)



- Dość często następuje ewolucyjne przejście od prototypu do końcowego systemu.
- Należy starać się nie dopuścić do sytuacji, aby klient miał wrażenie, że prototyp jest prawie ukończonym produktem.
- Po fazie prototypowania najlepiej prototyp skierować do archiwum.

Model spiralny

- Po raz pierwszy zaproponowany przez Boehma w 1988
- Każde okrążenie po spirali reprezentuje planowanie i wytworzenie pewnego elementu produktu projektu
- Cały projekt składa się z szeregu obrotów po rozkręcającej się spirali
- Jawne potraktowanie zagrożeń:
  - zidentyfikowanie zagrożeń i ich nazwanie
  - ocena prawdopodobieństwa ich wystąpienia oraz oszacowanie ewentualnych szkód
  - opracowanie metod zapobiegania oraz zachowania w przypadkach wystąpienia ryzyka
- Istnieje wiele wariantów tego modelu

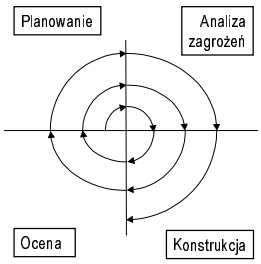
Zalety:

- możliwość zmiany kierunków rozwoju systemu pomiędzy obrotami
- ścisła współpraca z klientem
- uświadomienie konieczności zarządzania ryzykiem

Wada:

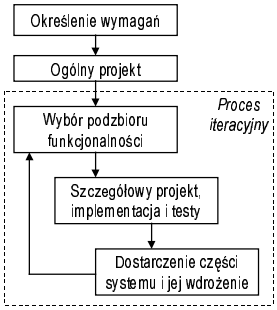
- przydatny do systemów, które mogą być wdrażane przy okrojonej funkcjonalności i obniżonej jakości
- osiągnięcie docelowej wersji systemu może wymagać długiego czasu

Model spiralny (2)



- **Planowanie -** ustalenie celów przygotowania elementu lub produkcji kolejnej wersji systemu, identyfikowanie alternatyw i ograniczeń
- **Analiza zagrożeń -** ocena alternatyw, identyfikacja i oszacowanie ryzyka, kroki zmierzające do redukcji zagrożeń (ew. budowa prototypu)
- **Konstrukcja -** wytworzenie i weryfikacja następnego przybliżenia produktu
- **Ocena -** atestowanie (przez klienta). Jeżeli ocena nie jest w pełni pozytywna, rozpoczynany jest kolejny cykl; w przeciwnym razie przechodzimy do planowania następnej fazy

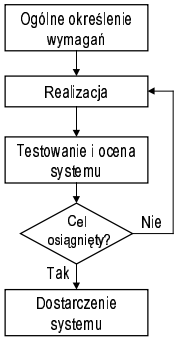
Realizacja przyrostowa
(ang. incremental development)



Odmiana modelu spiralnego

- Na wstępie identyfikowana jest i następnie realizowana podstawowa funkcjonalność systemu (podstawowy zestaw funkcji)
- W kolejnych iteracjach po akceptacji zestawu zrealizowanych funkcji następuje wykonanie i dostarczenie dalszej części budowanego systemu
- Zalety:
  - wczesne wykorzystanie przez klienta dostarczonych fragmentów
  - skrócenie przerw w kontaktach z klientem
  - możliwość elastycznego reagowania na powstałe opóźnienia
- Wada:
  - dodatkowy koszt związany z niezależną realizacją fragmentów systemu

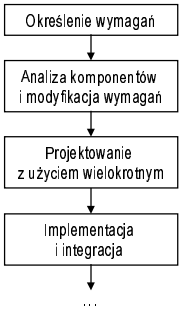
Programowanie odkrywcze
(ang. exploratory programming)



- Praca odbywa się niejako pod dyktando klienta (zleceniodawcy), zakres kolejnych iteracji jest ustalany ad hoc na podstawie aktualnej wizji, która zwykle ulega zmianom wraz z rozwojem systemu
- Zaleta:
  - możliwość stosowania nawet w przypadku trudności w określeniu oczekiwań klienta
- Wady:
  - praktycznie niemożliwe zachowanie sensownej struktury systemu
  - testowanie możliwe prawie wyłącznie w obecności klienta
- Model ten dobrze opisuje amatorski sposób tworzenia oprogramowania

Montaż z gotowych elementów
(ang. commercial off-the-shelf, COTS)

- Możliwość redukcji nakładów kosztów poprzez maksymalne wykorzystanie podobieństwa tworzonego oprogramowania do wcześniej tworzonych systemów oraz wykorzystanie gotowych komponentów dostępnych na rynku
- Przykładowe komponenty:
  - biblioteki
  - języki czwartej generacji (4GL)
  - gotowe aplikacje
- Metody pozyskiwania:
  - zakup elementów od zewnętrznych dostawców
  - przygotowanie elementów poprzednich przedsięwzięć do ponownego użycia



Montaż z gotowych elementów (2)

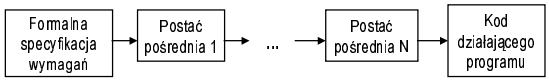
Zalety:
- wysoka niezawodność (potencjalnie)
- zmniejszenie ryzyka niepowodzenia projektu
- efektywne wykorzystanie specjalistów
- narzucenie standardów
- potencjalna redukcja kosztów

Wady:
- dodatkowy koszt przystosowania elementów do ponownego wykorzystania
- ryzyko uzależnienia się od dostawcy elementów
- brak rzeczywistej kontroli nad komponentami pozyskanymi spoza organizacji (szczególnie istotne podczas wykrywania błędów)
- niedostatki narzędzi wspomagających ten rodzaj pracy

Możliwość ponownego użycia (ang. reusability)

Formalne transformacje

- Postulowany w ramach tzw. *nurtu formalnego* w inżynierii oprogramowania
- Wymagania na system są formułowane w pewnym formalnym języku a następnie poddawane kolejnym transformacjom aż do uzyskania działającego programu
- Zakłada się, że kolejne transformacje są wykonywane bez udziału ludzi (w praktyce oznacza to, że język specyfikacji jest nowym "cudownym" językiem programowania)
- Najbardziej znanym przykładem takiego procesu wytwórczego jest Cleanroom, pierwotnie opracowany w IBM; inny przykład to proces oparty na metodzie B



Formalne transformacje (2)

Zalety:
- stworzone oprogramowanie charakteryzować się powinno wysoką niezawodnością

Wady:
- trudność formalnego wyspecyfikowania wymagań
- mała efektywność tak wytworzonego kodu
- brak dobrze rozwiniętych uniwersalnych języków formalnej specyfikacji zadań

- Metody matematyczne nie były w stanie utworzyć pełnej metodyki projektowania, gdyż metodyki włączają wiele elementów (np. psychologicznych) nie podlegających formalnemu traktowaniu
- W związku z powyższym jak na razie nie sprawdziły się w praktyce - nie są znane szersze ich zastosowania
- Mogą jednak z powodzeniem wspomagać bardziej szczegółowe zagadnienia np. obliczanie pewnych mierzalnych charakterystyk oprogramowania