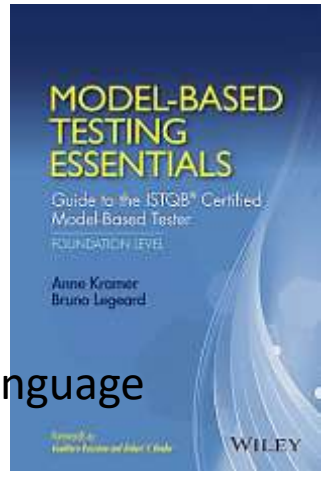
Przegląd MBT

A. Automatyczne generowanie testów z modeli
B. Pokrycie modelu
C. Pokrycie kodu a pokrycie modelu
D. Generowanie testów w wybranym modelu
E. Narzędzie
F. Różne algorytmy generowania testów

MBT według ISTQB



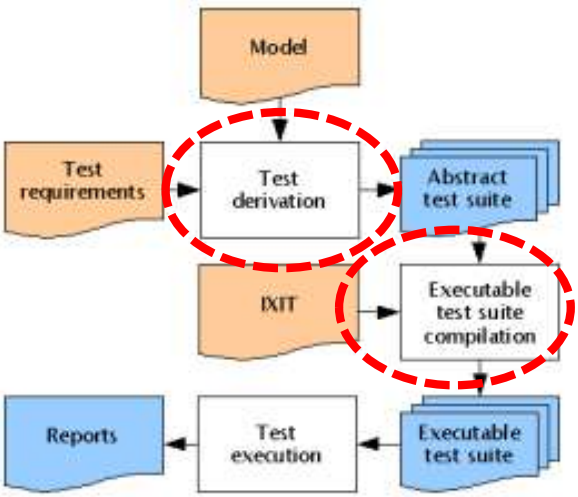
Modeling languages:
UML and BPMN
Other graphical modeling languages
Textual modeling languages
How to select the appropriate modeling language
Linking requirements to the MBT model
Re-using models from other development activities
Tool support for MBT modeling activities

A. Automatyczne generowanie testów z modeli

Zasada MBT

1. **Projektowanie testów z modeli**
2. **Jw., automatyczne**

A. Automatyczne wykonywanie testów
B. Automatyczna kompilacja skryptów



Public Domain, https://en.wikipedia.org/w/index.php?curid=3658377

Automatyzacja na różnych poziomach

1. Projektowanie testów
   1.1 Projektowanie testów z modeli

2. Wykonywanie testów

3. Dla testów wykonywanych automatycznie:
   3.1 Tworzenie skryptów do testów automatycznych

1. Automatyczne projektowanie testów

- Z kodu źródłowego
- Z interfejsu (nagrywanie)
- Ze specyfikacji interfejsu (no, to właściwie z modelu, jeśli specyfikacja jest np. modelem składni)

1. Automatyczne projektowanie testów

- Z kodu źródłowego 1(4)


```
int pięćset_plus(int dochód) {
  wynik ← dochód + 500;
  ładniePokażWynik(wynik);
  return(wynik);
}
```

http://mit.bme.hu/~micskeiz/pages/code_based_test_generation.html

1. Automatyczne projektowanie testów

- Z kodu źródłowego 2(4)

Sterownik testowy:

```
testuj() {
  pięćset_plus(0);
  pięćset_plus (10000);
}
```

1. Automatyczne projektowanie testów

- Z kodu źródłowego 3(4)

Zaślepka testowa:

```
ładniePokażWynik(wynik) {
   sendToRS232(wynik);
}
```


1. Automatyczne projektowanie testów

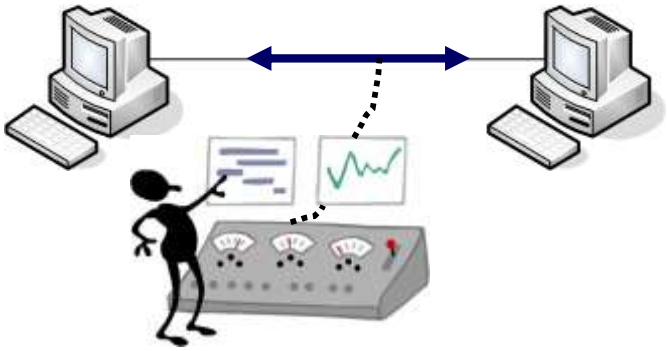
- Z kodu źródłowego 4(4)

Dane testowe:

```
plik17:(-1,499) (0,500)
(500,1000) (-500,0)
(10000,10500)
Testuj(plik17);
```

1. Automatyczne projektowanie testów

- Z interfejsu – nagrywanie rzeczywistych danych

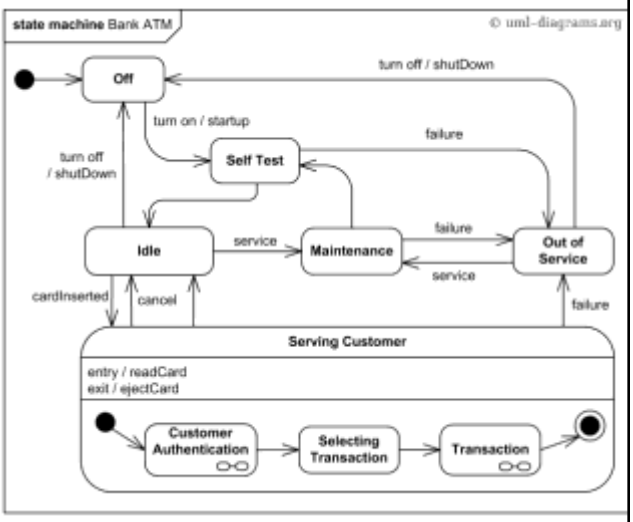


Projektowanie testów z modeli

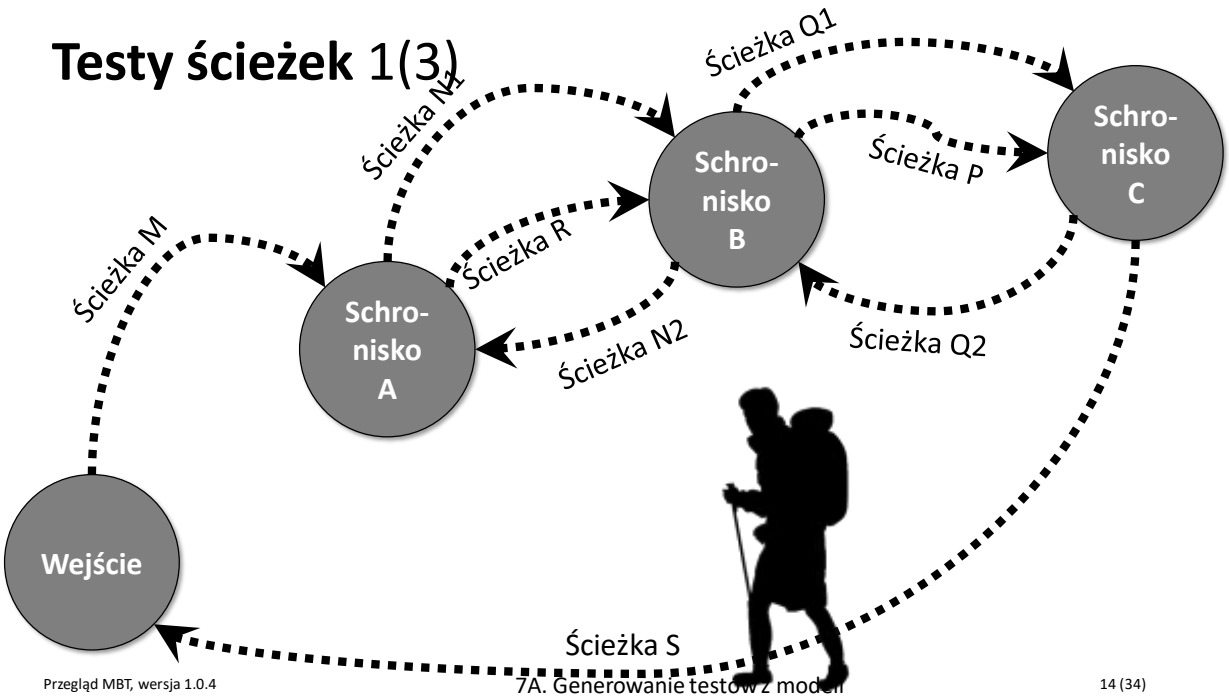
- Metody (algorytmy):
  - Każdego stanu (tzw. „pokrycie stanów”)
  - Każdego przejścia między stanami
  - Każdej ścieżki, składającej się z dwóch kolejnych przejść
  - Każdej ścieżki, składającej się z n kolejnych przejść
  - Różnych kombinacji danych, powodujących dane przejście / zmianę stanu

Projektowanie testów z modeli

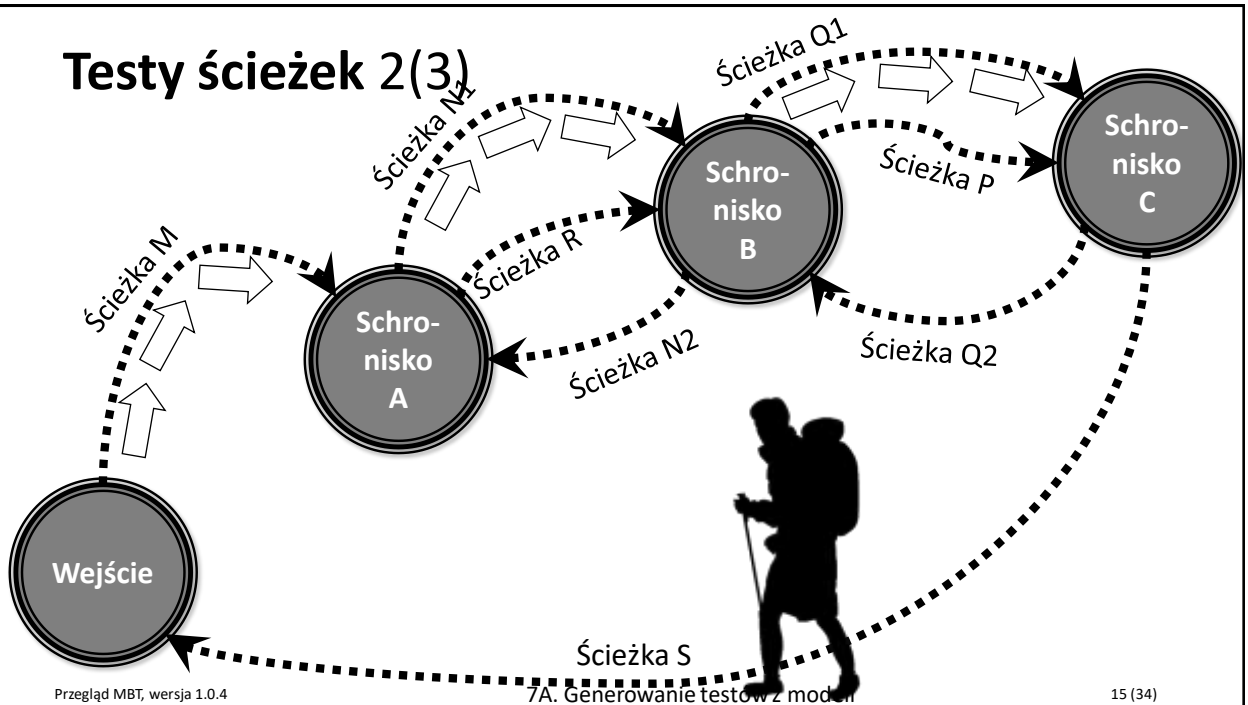
- Metody (algorytmy):
  - Każdego stanu (tzw. „pokrycie stanów”)
  - Każdego przejścia między stanami
  - Każdej ścieżki, składającej się z dwóch kolejnych przejść
  - Każdej ścieżki, składającej się z n kolejnych przejść
  - Różnych kombinacji danych, powodujących dane przejście / zmianę stanu



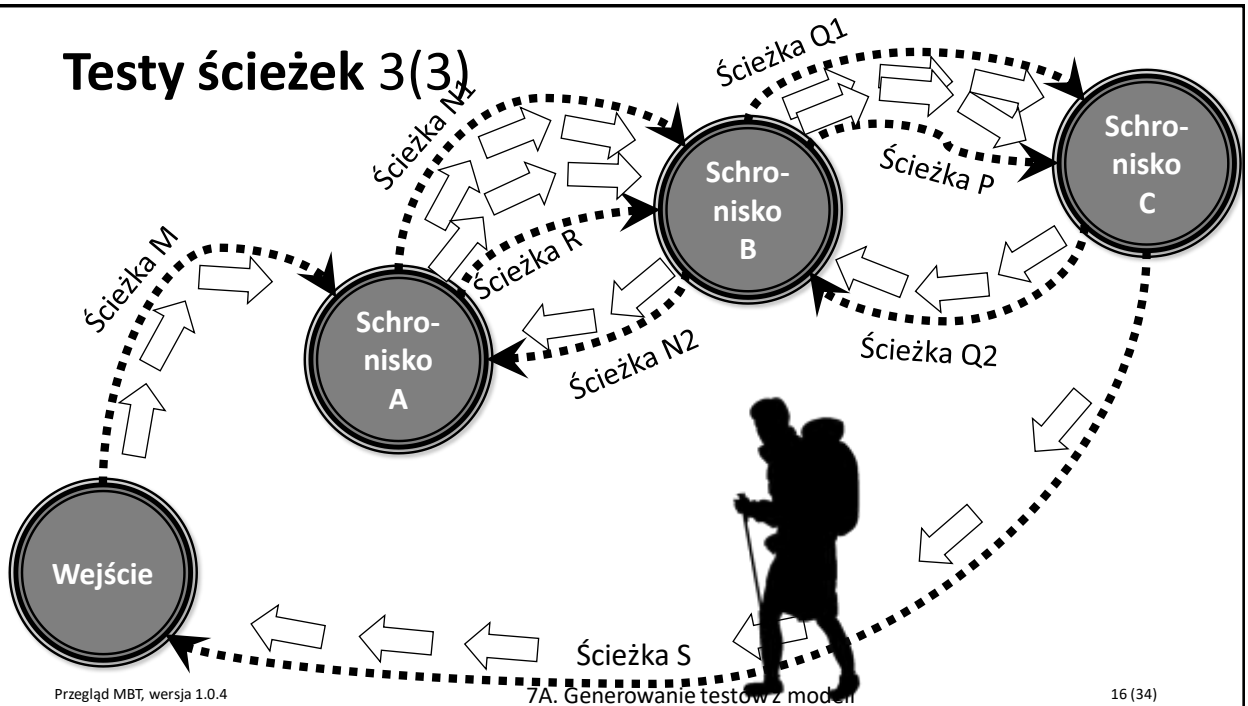
Testy ścieżek 1(3)



Testy ścieżek 2(3)



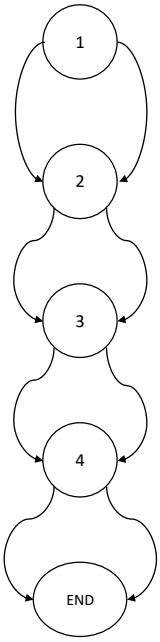
Testy ścieżek 3(3)



Testy ze schematu blokowego

Albo diagramu aktywności UML

```
if (warunek_1) then
  czynność 1A
else czynność 1B;
if (warunek_2) then
  czynność 2A
else czynność 2B;
if (warunek_3) then
  czynność 3A
else czynność 3B;
if (warunek_4) then
  czynność 4A
else czynność 4B
```



B. Pokrycie modelu

A. Automatyczne generowanie testów z modeli
B. Pokrycie modelu
C. Pokrycie kodu a pokrycie modelu
D. Generowanie testów w wybranym modelu
E. Narzędzie
F. Różne algorytmy generowania testów

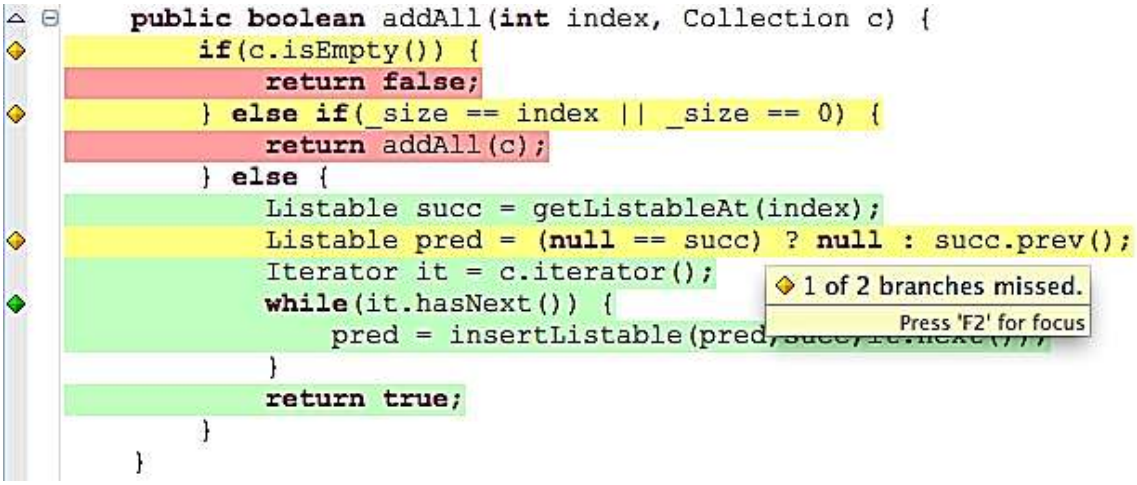
Różne miary pokrycia dla różnych modeli

- Mieliśmy już pewne miary pokrycia dla maszyny stanów i diagramu aktywności
- Ciekawe, ale słabo opracowane (nie mówiąc już o automatycznym generowaniu, którego brak) są diagramy struktur, danych oraz och przepływów i diagramy interakcji
- Wyjątki to tzw. podział na klasy równoważności oraz testowanie na podstawie składni

C. Pokrycie kodu a pokrycie modelu

A. Automatyczne generowanie testów z modeli
B. Pokrycie modelu
C. Pokrycie kodu a pokrycie modelu
D. Generowanie testów w wybranym modelu
E. Narzędzie
F. Różne algorytmy generowania testów

Pokrycie kodu, przykład



Pokrycie kodu - ograniczenia

```
integer a, b, c;
integer vector[75];
if (a < b) then
  c = a / b;
  write (file, vector[a]);
else
  printf („a is too big!”);
end
```

Test 1: a==2, b==3
Test 2: a==3, b==2
→ 100% pokrycia instrukcji i decyzji ☺

Podczas gdy to b==0 oraz a>74 powodują solidne awarie

D. Generowanie testów w wybranym modelu

A. Automatyczne generowanie testów z modeli
B. Pokrycie modelu
C. Pokrycie kodu a pokrycie modelu
D. Generowanie testów w wybranym modelu
E. Narzędzie
F. Różne algorytmy generowania testów

Jakie modele są dostępne?

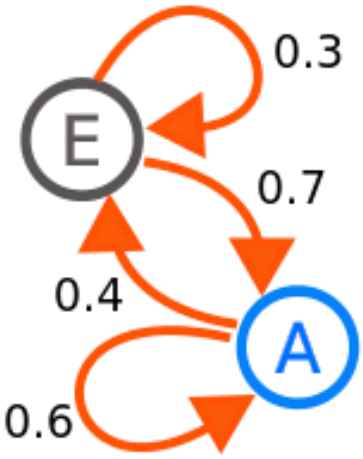
- Informacja w rozdziale „E. Narzędzia” poniżej

E. Narzędzie

A. Automatyczne generowanie testów z modeli
B. Pokrycie modelu
C. Pokrycie kodu a pokrycie modelu
D. Generowanie testów w wybranym modelu
E. Narzędzie
F. Różne algorytmy generowania testów

en.wikipedia.org/wiki/Model-based_testing

- Istniejące narzędzia do generowania testów z modeli najczęściej wykorzystują diagramy przejść stanów lub łańcuchy Markowa:



mit.bme.hu/~micskeiz/pages/modelbased_testing.html

I. Modele
II. Import modeli z innych narzędzi
III. Algorytmy tworzenia testów

I. Modele

I. Modele
II. Import modeli z innych narzędzi
III. Algorytmy tworzenia testów

Modele tekstowe w formie języka typu „Gherkin”
XML
„Diagramy UML”
C#
Łańcuchy Markowa
Diagramy BPMN
Java
Matlab
Diagramy stanów
Modele własne
Sieci Petriego
Phyton
SysML
MatthWorks Simulink

II. Import modeli z innych narzędzi

I. Modele

II. Import modeli z innych narzędzi

III. Algorytmy tworzenia testów

Także: eksport testów do różnych narzędzi

„From several modeling tools”

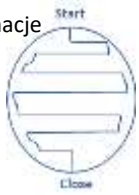
„Narzędzia BPM”

„Multiple formats”

„Narzędzia UML”

III. Algorytmy tworzenia testów

Kombinacje



I. Modele

II. Import modeli z innych narzędzi

III. Algorytmy tworzenia testów

Według modelu rozkładu P

statement, branch, path, condition

„pokrycie strukturalne”

Obszary ryzyka

Typowe scenariusze

robertvbinder.com/open-source-tools-for-model-based-testing/

Open Source Tools for Model-Based Testing

APRIL 17, 2012 | BLOG, MODEL-BASED TESTING, SOFTWARE PRODUCTS, SOFTWARE TESTING

I promised to provide a list of open source and free model-based testing tools in a QUEST panel session, and have updated this list a few times since. There is wide variation in maturity, stability, and provisioning.

Nearly all of following information is from provider's web sites - YMMV. The Environment column indicates the source code language and/or run-time environment(s); "*nix" indicates a tool written to run on most Linux/Unix variants. Open source licenses vary widely, click here for an overview.

If you note any errors or omissions, please post a comment and I will revise accordingly. A revision history follows the last table.

F. Różne algorytmy generowania testów

A. Automatyczne generowanie testów z modeli
B. Pokrycie modelu
C. Pokrycie kodu a pokrycie modelu
D. Generowanie testów w wybranym modelu
E. Narzędzie
F. Różne algorytmy generowania testów

Już było ☺

Automatyczne generowanie kodu z modeli

wersja 1.0.4