ZESZYTY NAUKOWE UNIWERSYTETU SZCZECIŃSKIEGO

NR 476 STUDIA INFORMATICA NR 21 2007

*ŁUKASZ RADLIŃSKI*

# PRZEGLĄD SIECI BAYESA DO SZACOWANIA RYZYKA W INŻYNIERII OPROGRAMOWANIA

## Wprowadzenie

Tworzenie oprogramowania jest skomplikowane, a oprogramowanie, będące jego efektem, jest w dużej mierze niepowtarzalne. Powoduje to, że zaplanowanie odpowiedniej ilości zasobów i czasu dla danego projektu nie jest łatwe. Sytuacja komplikuje się jeszcze bardziej, gdy z góry założono wymagania jakościowe, które musi spełniać oprogramowanie, lub przy prognozowaniu jakiej jakości oprogramowania można się spodziewać przy założonych innych ograniczeniach. W ostatnim czasie prowadzono wiele badań, które miały doprowadzić do zbudowania łatwego w użyciu modelu, zapewniającego wystarczająco precyzyjne wyniki. W artykule skupiono się na modelach opartych na sieciach Bayesa.

Sieci Bayesa to metoda, która ma wiele zalet i stosunkowo niewiele wad w porównaniu z innymi metodami, którymi próbowano rozwiązać ten problem. Do głównych zalet metody można zaliczyć[1]:

a) jawne ujęcie niepewności (zmienne ujmowane są w postaci rozkładów prawdopodobieństw);
b) jawne ujęcie związków przyczynowo skutkowych pomiędzy zmiennymi;
c) działanie modeli przy niepełnych danych (firmy rzadko posiadają dużo danych potrzebnych do zasilenia innych modeli);

[1] N.E. Fenton, M. Neil: *A Critique of Software Defect Prediction Models*. „IEEE Transactions on Software Engineering" 1999, Vol. 25, No 3.

d) możliwość wykorzystania wiedzy eksperta przy budowie modelu,
e) możliwość prowadzenia analiz typu „co-jeśli”,
f) możliwość łączenia różnych typów informacji.

Głównym celem artykułu jest analiza porównawcza najważniejszych istniejących sieci Bayesa zbudowanych przez różnych badaczy. Wyniki tej analizy wykorzystano do budowy kolejnych modeli:

– uwzględniających większą liczbę zmiennych;
– łatwiej dopasowujących się do zgromadzonych empirycznych danych i indywidualnych potrzeb użytkowników;
– łatwiejszych w zrozumieniu i obsłudze przez użytkowników końcowych.

## 1. Sieci Bayesa do szacowania ryzyka w inżynierii oprogramowania

Efektem projektu badawczego MODIST[2] były dwa modele:

– na poziomie projektu (ang. *project-level*),
– do prognozowania defektów (ang. *defect prediction*).

Celem pierwszego z nich jest szacowanie i prognozowanie ryzyka oraz jakości w dużym przedsięwzięciu programistycznym. Docelową grupą użytkowników modelu są kierownicy projektów. Model ten obejmuje sześć podsieci:

– rozproszoną komunikację i zarządzanie,
– wymagania i specyfikację,
– jakość procesu,
– jakość zasobów ludzkich,
– dostarczoną funkcjonalność,
– dostarczoną jakość.

Głównym elementem modelu jest komponent do szacowania zależności między:

a) jakością (przy czym model rozróżnia „liczbę defektów” od „zadowolenia użytkownika”);
b) nakładem pracy (reprezentowanym przez „średnią liczbę pracowników pracujących na pełnym etacie” przy projekcie);
c) czasem (reprezentowanym przez „czas trwania przedsięwzięcia”);
d) funkcjonalnością (w rozumieniu rozmiaru wytworzonego oprogramowania wyrażonego w liczbie punktów funkcyjnych).

[2] MODIST BN models, http://www.modist.org.uk/docs/modist_bn_models.pdf.

Model prognozowania defektów jest skupiony na prognozowaniu i monitorowaniu różnych kategorii defektów pojedynczej fazy (ang. *phase*) lub grupy faz procesu tworzenia oprogramowania. Jest przeznaczony dla pojedynczych grup pracowników realizujących część programu w danej fazie.

W kontekście tego modelu ważne jest zrozumienie, co oznacza termin „faza". Nie jest ona tożsama (a w zasadzie nie musi być) z potocznym rozumieniem tego terminu w dziedzinie inżynierii oprogramowania. Nie chodzi bowiem o fazę jako jeden z elementów kaskadowego modelu tworzenia oprogramowania, na przykład projektowanie lub kodowanie. Faza jest tu raczej traktowana jako zespół działań (ang. *activity*) zmierzających do wytworzenia określonej części oprogramowania. Może to być na przykład moduł programu, sama dokumentacja projektowa lub inna część, z których składa się cały program. Faza może zatem odzwierciedlać pojedyncze działanie lub obejmować kombinację takich działań, jak[3]:

- specyfikacja i dokumentacja,
- projektowanie i programowanie,
- testowanie i poprawianie.

Model do prognozowania defektów zawiera węzły wejściowe (ang. *input*) i wyjściowe (ang. *output*). Dzięki nim pojedyncze instancje modelu mogą być łączone w łańcuch modeli, który może odzwierciedlać łańcuch faz (cykl tworzenia) tworzenia danego programu[4]. Model ten zawiera pięć głównych części:

- specyfikację i dokumentację,
- projekt i programowanie,
- testowanie i poprawianie,
- rozmiar nowo wytworzonej funkcjonalności,
- wprowadzanie defektów i regeneracja.

Po zakończeniu projektu MODIST autorzy poddali model do prognozowania defektów następującym modyfikacjom:

[3] Oprócz modelu, który zawiera wszystkie możliwe działania w ramach jednej fazy tworzenia oprogramowania, stworzono również modele zawierające różne kombinacje podzbioru tych działań, np. tylko specyfikacja i dokumentacja oraz projektowanie i programowanie (bez testowania i poprawiania).

[4] Szerzej na temat modelowania różnych cykli tworzenia oprogramowania zob. N.E. Fenton, M. Neil, W. Marsh, P. Krause, R. Mishra: *Predicting Software Defects in Varying Development Lifecycles using Bayesian Nets*. Technical Report No RR-05-11, Queen Mary, University of London, London 2005; Agena: *Software Project Risk Models Manual*, Ver. 01.00, 2004.

a) uproszczono podsieć „specyfikacja i dokumentacja", dzięki czemu przyczyny i efekty stały się bardziej intuicyjne;
b) zmieniono podsieć „rozmiar nowo wytworzonej funkcjonalności", usuwając szacowanie funkcjonalności w punktach funkcyjnych i wprowadzając zmienną „efektywne tysiące linii kodu", która zależy od rzeczywistej wielkości oprogramowania dostrojonej przez „złożoność nowej funkcjonalności", „rozmiar rozproszonej komunikacji" i „integrację z innym oprogramowaniem";
c) zmieniono modelowanie skomplikowanych węzłów z podejścia opartego na „wskaźnikach" (ang. *indicator*) na podejście oparte na „przyczynach" (ang. *causal*)[5];
d) dodano podsieć „istniejący kod programu" zawierającą zmienne opisujące część kodu programu, która powstała przed realizacją danej fazy;
e) dodano podsieć „wspólne wpływy" (ang. *common influences*), zawierającą czynniki związane z efektywnością zarządzania, wpływające na „specyfikację i dokumentację" oraz „projektowanie i programowanie".

C.G. Bai i inni skonstruowali model niezawodności oprogramowania oparty na sieci Bayesa z warunkiem Markova (ang. *Markov Bayesian Model* – MBM)[6]. Punktem wyjścia tworzenia modelu są liczne założenia ograniczające możliwość korzystania z klasycznych modeli niezawodności. Model i inne, oparte na sieciach Bayesa, łączy informacje o związkach przyczynowo-skutkowych z możliwością poprawy predykcji modelu dzięki dostarczaniu nowych obserwacji. W modelu wyróżniono trzy zmienne, których rozkłady są kluczowe:

– liczbę defektów oprogramowania,
– czas między awariami,
– liczbę defektów usuwanych przy danej awarii.

Pracę oparto na wcześniejszych badaniach, które doprowadziły do zbudowania modelu o znanym rozkładzie prawdopodobieństwa przed użyciem modelu. W nowym modelu prognozy niezawodności są dokonywane z założeniem, że wartości tych parametrów są niedostępne. Mogą one być jednak oszacowane za

[5] Szerszą dyskusję na temat tych dwóch podejść zob. w Ł. Radliński: *Modelling Complex Nodes in Bayesian Nets for Software Project Risk Assessment*. Suplement do „Polish Journal of Environmental Studiem" 2006.

[6] C.G. Bai, Q.P. Hu, M. Xie, S.H. Ng: *Software Failure Prediction Based on a Markov Bayesian Network Model*. „Journal of Systems and Software" 2005, Vol. 74, Issue 3, February, s. 275–282.

pomocą algorytmu maksymalizacji wartości oczekiwanej, dzięki czemu można użyć modelu przy niekompletnych danych.

Wyniki prognoz modelu autorzy porównali z prognozami klasycznych modeli niezawodności Jelińskiego-Morandy (JM) i Goela-Okumoto (GO). Zestawienie to wykazało, że modele JM i GO nie są w stanie przewidzieć czasu między kolejnymi awariami systemu przy początkowych awariach. Powoduje to, niestety, że nie nadają się one zupełnie w sytuacjach wymagających znajomości danych o awariach systemu w przeszłości. Model MBM może zawsze dokonać predykcji. Ponadto stwierdzono, że wraz ze wzrostem liczby danych o kolejnych awariach modele JM i GO są przeszacowane, a szacunki modelu MBM są stabilne z niewielką wariancją.

T. Cockram skonstruował model do szacowania efektywności inspekcji oprogramowania[7]. Autor wyszedł z założenia, że liczba błędów odkrytych w programie dzięki zastosowaniu inspekcji nie wzbudza zaufania do produktu. Jest tak dlatego, że znaleziono wiele błędów, a niedostateczna inspekcja wykazała część z nich lub, że efektywna inspekcja znalazła większość z nich.

Inne metody, które miały podobny cel, wymagały znajomości liczby wszystkich błędów w produkcie. Niestety, takie wartości nie są dostępne w czasie przeprowadzanie inspekcji i mogą nigdy nie być znane, chyba że działanie programu spowoduje, iż błędy ukażą się w postaci awarii. Celem autora było zatem stworzenie modelu, który nie będzie wymagał znajomości całkowitej liczby błędów. Model zbudowano z następujących grup zmiennych:

- rozmiaru inspekcji,
- poziomu moderatora i zespołu dokonujących inspekcji,
- użycia odpowiedniej metody inspekcji,
- odpowiedniego przygotowania do procesu inspekcji.

Znajomość stopnia efektywności konkretnej inspekcji, jeśli jest on niezadowalający, pozwala na zmianę procesu inspekcji lub ilości materiału poddanego inspekcji w celu poprawienia jej efektywności.

Wyniki obliczeń efektywności inspekcji oprogramowania potwierdziły istotną korelację między wartościami przewidywanymi przez model a rzeczywistą dystrybucją. Efektywność ujęto tu jako stosunek liczby problemów znalezionych podczas inspekcji a całkowitą liczbą problemów wykrytych w kolejnych

[7] T. Cockram: *Gaining Confidence in Software Inspection Using a Bayesian Belief Model*. „Software Quality Journal" 2001, No 9, s. 31–42.

testach inspekcji lub w czasie instalacji. Model zweryfikowano za pomocą reguły logarytmicznej zaproponowanej przez Cowella i innych[8].

G.J. Pai i inni zajęli się problemem niezależnej weryfikacji i walidacji (ang. *independent verification and validation* – IV&V) oprogramowania, a konkretnie – przypadkami użycia UML na etapie specyfikacji wymagań[9]. W pracy tej autorzy wyznaczyli wartości prostych metryk na podstawie przypadków użycia, które później zasiliły stworzoną sieć Bayesa. W sieci tej autorzy modelowali związki między obserwowanymi parametrami procesu IV&V dla przypadków użycia a pożądanymi cechami specyfikacji wymagań.

Motywacją do badań była chęć stworzenia metodologii, która ujmuje istniejące techniki inspekcji i subiektywnej analizy w celu systematycznej i ilościowej analizy stopnia gotowości wymagań ujętych w postaci przypadków użycia. Model skonstruowano ręcznie przez powiązanie głównych działań procesowych z cechami wymagań, które weryfikują te działania. Autorzy przyznają, że w miarę potrzeby model może być, rozbudowywany i nie wyczerpuje zbioru czynników niezbędnych do analizy dojrzałości procesu IV&V[10].

Model ujmuje następujące cechy specyfikacji wymagań:

- przejrzystość,
- poziom skomplikowania,
- kompletność,
- spójność,
- poprawność,
- identyfikację czynników ograniczających (ang. *constraint*),
- możliwość śledzenia zmian i wzajemnych zależności.

Dzięki zastosowaniu sieci Bayesa do analizy dojrzałości wymagań można zaobserwować, że każda cecha specyfikacji wymagań ma przypisany rozkład prawdopodobieństwa, mówiący o stopniu spełnienia tej cechy. W związku z tym łatwe jest odczytanie, czy dana cecha osiągnęła wartość co najmniej 0,95, która jest kryterium wyjściowym IV&V[11].

[8] R.G. Cowell, A.P. Dawid, D.J. Spiegelhalter: *Sequential Model criticism in Probabilistic Export Systems*. „IEEE Trans. Pattern Analysis and Machine Intelligence" 1993, No 15(3), s. 209–219.

[9] G.J. Pai, J.B. Dugan, K. Lateef: *Bayesian Networks Applied to Software IV&V*. Proc. 29th Annual IEEE/NASA Software Engineering Workshop 2005, s. 293–304.

[10] G.J. Pai i in.: *op.cit.*

[11] *Ibidem.*

Autorzy zdają sobie sprawę z tego, że ze względu na wczesny etap dokonywania analizy w procesie tworzenia oprogramowania prawdziwa natura wzajemnych zależności w modelu może nie być znana. Zakładają zatem wystąpienie wariancji w wynikach modelu w odniesieniu do rzeczywistych wartości.

S. Bibi i I. Stamelos skonstruowali sieć Bayesa do szacowania nakładów pracy w czasie tworzenia oprogramowania[12]. Model ten umożliwia odzwierciedlenie etapowej natury procesu tworzenia oprogramowania przez wykorzystywanie wiedzy o procesie w poprzednich etapach do prognozowania kolejnych etapów. W artykule jako przykładowym modelem procesu tworzenia oprogramowania posłużono się popularnym *Rational Unified Process* (RUP). Jest to proces iteracyjny – dziewięć obszarów jest powtarzanych w każdej z czterech faz. Dzięki temu cały proces może być odzwierciedlony w naturalny sposób jako dynamiczna sieć Bayesa. Autorzy celowo nie ujęli w swoim modelu dwóch obszarów („konfiguracja i zarządzanie wersjami" i „środowisko") z powodu ich niewielkiego udziału w każdej iteracji, a także dla utrzymania prostoty modelu. Model prognozuje nakład pracy dla każdego przepływu obszaru w każdej fazie. Na każdym etapie jest również dostarczana wartość całkowitego nakładu na zakończenie projektu. W miarę upływu czasu, coraz większa liczba danych wprowadzonych do modelu prowadzi do coraz dokładniejszych szacunków.

Model może być uogólniony przez ominięcie liczby iteracji. Jeśli będzie realizowana tylko jedna iteracja, model odzwierciedli podejście kaskadowe. Model może również odzwierciedlać dowolny przyrostowy lub iteracyjny cykl życia projektu, gdzie liczba iteracji jest zależna od konkretnego projektu.

W pracy autorzy przedstawili również bardziej szczegółową wersję modelu, niż wyżej opisana. Najważniejsze działania w ramach poszczególnych obszarów powiązano ze zmiennymi odzwierciedlającymi ich rezultaty. W ten sposób model pokazuje kolejność wszystkich działań. W modelu tym również oszacowano nakłady dla każdego obszaru.

Autorzy świadomie położyli nacisk na konstrukcję modelu, ponieważ wzięli pod uwagę fakt, że to ludzie będą końcowymi użytkownikami takich modeli. Modele powinny być zatem stosunkowo proste i intuicyjnie zrozumiałe przez użytkowników.

[12] S. Bibi, I. Stamelos: *Software Process Modeling with Bayesian Belief Networks*. Proc. of 10th International Software Metrics Symposium. Chicago 2004.

D.A. Wooff i inni zaproponowali modele oparte na sieciach Bayesa do testowania oprogramowania[13]. Ujmują one najważniejsze aspekty testowania oprogramowania, czyli:

– liczbę defektów w oprogramowaniu,
– stopień dopasowania przypadków testowych do danej sytuacji,
– wymagany poziom nakładów na testowanie.

Autorzy zanalizowali dwa studia przypadków w jednej z większych firm w Wielkiej Brytanii. Pierwsze dotyczyło zarządzania bazą danych kart kredytowych, a drugie – zmiany numeracji rekordów w bazie danych spowodowanej rosnącym popytem klientów i ujawnioną w ten sposób niewystarczającą liczbą cyfr do identyfikowania rekordów.

Naturalną cechą modeli opartych na sieciach Bayesa jest ujęcie niepewności za pomocą rozkładu prawdopodobieństwa wystąpienia danego zdarzenia. Istotną cechą modeli Wooffa i innych jest ujęcie w nich dodatkowo miar użyteczności (ang. *utility*). Odzwierciedlają one koszt (niekoniecznie ujęty finansowo), jaki ponosi organizacja w związku z udostępnieniem oprogramowania użytkownikowi w danym momencie.

## 2. Porównanie modeli

W tabeli 1 zestawiono najważniejsze zalety i wady analizowanych w pracy modeli.

Tabela 1

Zestawienie głównych zalet i wad analizowanych sieci Bayesa

| Model | Zalety | Wady |
|---|---|---|
| 1 | 2 | 3 |
| MODIST – poziom projektu | – zawiera komponent do analizy zależności ograniczeń w przedsięwzięciu informatycznym<br>– zawiera ogólny poziom zarządzania projektem<br>– zawiera możliwość predykcji jakości oprogramowania (defekty i satysfakcja użytkownika) | – słabo rozbudowana część do predykcji defektów<br>– konieczność użycia punktów funkcyjnych do ujęcia rozmiaru oprogramowania |

[13] D.A. Wooff, M. Goldstein, F.P.A. Coolen: *Bayesian Graphical Models for Software Testing*. „IEEE Transactions on Software Engineering" 2002, Vol. 28, Issue 5, May, s. 510–525.

| 1 | 2 | 3 |
|---|---|---|
| MODIST – prognozowanie defektów | – ujęcie różnych kategorii defektów<br>– możliwość łączenia modeli zgodnie z określonym cyklem tworzenia oprogramowania<br>– oddzielne modele do różnych kombinacji działań w ramach fazy tworzenia oprogramowania<br>– pozytywnie zweryfikowana dokładność prognoz modelu (ale dla określonego przedziału wartości parametrów) | – nacisk jedynie na prognozowanie defektów<br>– brak komponentu do analizy zależności ograniczeń w przedsięwzięciu informatycznym<br>– brak odwołania do harmonogramu przedsięwzięcia<br>– słabo rozbudowana część o jakości zarządzania projektem<br>– konieczność użycia punktów funkcyjnych do ujęcia rozmiaru oprogramowania<br>– nakłady wyrażone jedynie w postaci węzłów stopniowanych |
| Ulepszony model prognozowania defektów | – ulepszona podsieć „nowa funkcjonalność”<br>– bardziej intuicyjna podsieć specyfikacja/dokumentacja<br>– bardziej szczegółowe dane na temat zarządzania projektem<br>– odwołanie do metryk opisujących kod poddany rozbudowie | – brak komponentu do analizy zależności ograniczeń w przedsięwzięciu informatycznym<br>– nacisk jedynie na prognozowanie defektów<br>– brak odwołania do harmonogramu przedsięwzięcia<br>– nakłady wyrażone jedynie w postaci węzłów stopniowanych |
| Model niezawodności oprogramowania | – możliwość zastosowania modelu przy braku znajomości niektórych parametrów modelu<br>– pozytywna weryfikacja prognoz modelu (ale na niewielkiej ilości danych) | – odzwierciedlenie tylko jednego z obszarów inżynierii oprogramowania<br>– niewielka liczba czynników ujętych w modelu |
| Model do szacowania efektywności inspekcji | – duża liczba istotnych czynników wpływających na wyniki modelu<br>– ujęcie jako parametry zmiennych znanych w momencie użycia modelu<br>– pozytywna weryfikacja prognoz modelu | – odzwierciedlenie tylko jednego obszaru inżynierii oprogramowania |
| Model niezależnej weryfikacji i walidacji | – dokładnie odzwierciedlone czynniki charakteryzujące specyfikację wymagań | – model odzwierciedla tylko jeden obszar inżynierii oprogramowania<br>– brak zależności między poszczególnymi cechami specyfikacji (z jednym wyjątkiem) |

| 1 | 2 | 3 |
|---|---|---|
| Model do szacowania nakładu pracy | – ujęcie nakładów pracy na różnych etapach tworzenia oprogramowania<br>– możliwość odzwierciedlenia różnych cykli tworzenia oprogramowania<br>– celowe uniknięcie nadmiarowego skomplikowania modelu prowadzącego do czasochłonnych obliczeń | skupienie się jedynie na nakładach z pominięciem innych czynników |
| Model do testowania oprogramowania | – ujęcie wielu aspektów procesu testowania<br>– ujęcie miar użyteczności | odzwierciedlenie jedynie procesu testowania |

Źródło: opracowanie własne.

## Podsumowanie

Przeprowadzona analiza sieci Bayesa wykorzystywanych do szacowania ryzyka w przedsięwzięciach informatycznych wykazała, że:

1. Modele te najczęściej są przeznaczone dla odmiennych obszarów inżynierii oprogramowania, co praktycznie uniemożliwia bezpośrednie porównanie tych modeli w celu odpowiedzi na pytanie: który model jest najlepszy?
2. Niektóre modele są zdecydowanie bardziej rozbudowane niż inne, zatem uwzględniają większą liczbę czynników, które mogą zostać poddane analizie.
3. Niektóre modele są jedynie na etapie koncepcyjnym – nie poddano ich procesowi weryfikacji.
4. Każdy z modeli ma następujące wady:

a) brak ścisłej integracji danych dotyczących poziomu projektu i poziomu jakości (testowanie, defekty) w jednym modelu;
b) brak ujęcia wielu istotnych czynników wpływających na modelowany obszar, na przykład jakości procesów (CMMI, ISO 9001 lub innych);
c) brak możliwości łatwego ujęcia w modelu pojawiających się dostępnych danych empirycznych;
d) brak ujęcia taksonomii ryzyka – węzły modelu nie mają przypisanych kategorii w zależności od użytkownika modelu;
e) brak ujęcia parametrów, dzięki którym możliwe byłoby dopasowanie przez użytkownika modelu do swoich potrzeb;

f) brak ujęcia rozmiaru oprogramowania w dowolnej jednostce miary wybranej przez użytkownika modelu (często są to trudne w użyciu punkty funkcyjne);
g) częsty brak wyróżnienia typów defektów i spowodowanych przez nie awarii;
h) brak łatwego sposobu obsługi modeli przez użytkowników końcowych, szczególnie przy bardziej rozbudowanych modelach (jest to wada nie tylko samych modeli, ale również narzędzi do ich obsługi).

Wymienione wady istniejących modeli są przyczyną dalszych badań autora nad konstrukcją modeli eliminujących te wady.

## A SURVEY OF BAYESIAN NETWORKS FOR RISK ASSESSMENT IN SOFTWARE ENGINEERING

**Summary**

The aim of this work is to compare selected Bayesian Nets for software project risk assessment. Previous studies have shown that Bayesian Nets have several advantages over other methods of creating models in the domain of software engineering. Thus, I did not analyze other models in this work. The results of this analysis have shown that direct comparison is difficult because of big differences between the models caused by different motivations for building them. Many of those models reflect only small part of software engineering. All of them suffer common weaknesses which are the motivations for my future work focused on creating models overcoming those weaknesses.

*Translated by Łukasz Radliński*