Inżynieria oprogramowania

Wykład 3
Automatyczne tworzenie dokumentacji

dr inż. Piotr Wrzeciono

Wprowadzenie

Dokumentacja tworzona w trakcie realizacji projektu IT jest sprawą bardzo ważną.

Najważniejszym powodem tworzenia dokumentacji jest poprawa jakości tworzenia oprogramowania oraz samego kodu.

Jedną z najważniejszych celów dokumentacji jest również dokładna specyfikacja zamówionego oprogramowania.

Bardzo istotną sprawą jest szybkość powstawania dokumentacji. W celu jej zwiększenia powstało wiele narzędzi umożliwiających automatyczną generację dokumentacji z kodu programu.

Najprostsze rodzaje dokumentacji

Podstawowym rodzajem dokumentacji w kodzie jest komentarz. W większości języków programowania rozróżniamy komentarze jednoliniowe i wieloliniowe.

W językach programowania wywodzących się z języka C, stosowane są następujące oznaczenia:

```
// Komentarz w jednej linii

/* Komentarz w wielu liniach
*/
```

Komentarze obu typów stosowane są od dawna. Informacje zamieszczone pomiędzy /* */ dotyczą często opisu jakiejś większej funkcjonalności albo na przykład standardu zapisu danych.

Komentarze rozbudowane graficznie

Pierwszy system rozwiniętych komentarzy polegał na umieszczaniu opisu dokumentacyjnego w komentarzu wielowierszowym w tzw. ramkach lub tabelach.

Przykład:

```
/*+-------------------------------------+
 *| Ta funkcja służy do zapisu danych te-  |
 *| kstowych na dysk. Parametrami są: Nazwa|
 *|(zawierająca nazwę pliku) oraz Tablica  |
 *|(zawierająca dane do zapisu → struktura |
 *|DaneDoObliczen.                         |
 *+-------------------------------------+
 */
```

Tego typu komentarz wyróżnia się już w kodzie programu, jednakże nie jest interaktywny. W celu poszukania zależności trzeba przejrzeć dużą część kodu.

JavaDoc

Wspomniane pierwsze rozwiązanie jest już dosyć użyteczne dla osób tworzących oprogramowanie, jednakże jego największą wadą jest potrzeba ręcznego przepisywania do dokumentacji projektu.

Proces ten zajmuje bardzo dużo czasu, który mógłby być przeznaczony na sam proces tworzenia kodu.

Problem ten został dostrzeżony przez programistów z firmy SUN, podczas tworzenia języka programowania o nazwie Java.

W tym języku programowania pojawiły się komentarze dokumentacyjne, z których za pomocą odpowiedniego parsera można generować dokumenty.

Styl dokumentacji JavaDoc

- Komentarz dokumentacyjny znajduje się bezpośrednio przed fragmentem kodu komentowanego.
- Dokumentacja rozpoczyna się symbolem `/**` a kończy symbolem `*/`.
- Wewnątrz komentarza można stosować tagi pomocnicze, na przykład `@author`, `@param`.
- W dokumentacji wolno używać poleceń HTML, przy czym obecnie dopuszcza się również stosowanie CSS.
- Komentarz dokumentacyjny można tworzyć do pakietów, klas, metod i własności.

Zastosowania JavaDoc

Pierwszym celem stosowania komentarzy było tworzenie dokumentacji papierowej za pośrednictwem systemu TEX lub LATEX (wymowa: „tech” lub „latech”).

W trakcie popularyzowania się serwisów WWW oraz coraz większych możliwości przeglądarek internetowych, JavaDoc zaczął służyć również do automatycznej generacji stron z dokumentacją.

Styl ten obowiązuje do tej pory. Cała dokumentacja Javy jest dostępna na stronach internetowych w postaci plików html wygenerowanych automatycznie z kodu źródłowego.

Automatyczne systemy podpowiedzi

Kiedy komputery klasy PC posiadały już odpowiednią moc obliczeniową, aby dokonywać parsowania JavaDoc w czasie rzeczywistym, pojawił się system podpowiedzi, gdzie oprócz pokazywania metod i własności klas zaczęto umieszczać fragmenty dokumentacji bezpośrednio ich dotyczące.

Należy tutaj wymienić Visual Studio firmy Microsoft, gdzie wspomniany system podpowiedzi pojawił się najpierw dla Visual Basica, a później został zaadoptowany do wszystkich języków programowania.

Później podobna funkcjonalność została wprowadzona do środowiska programistycznego NetBeans, gdzie została bardzo rozbudowana o automatykę autouzupełniania.

XML w dokumentacji. Platforma .NET

Microsoft, wraz z wprowadzeniem platformy .NET wprowadził również nowy język programowania C#, korzystający zarówno z rozwiązań stosowanych w Javie, jak i w samym języku C++.

Rozwiązaniem przeniesionym z Javy były komentarze dokumentacyjne. Jednakże w komentarzach dla platformy .NET używa się standardu XML z kilkoma predefiniowanymi znacznikami.

W ten sposób, do parsowania komentarzy dokumentacyjnych można używać parsera XML, co jest bardzo pomocne w przypadku potrzeby szybkiego generowania dokumentacji „w locie”.

Podstawowe reguły tworzenia komentarza dokumentacyjnego w języku C#.

- Każdy komentarz dokumentacyjny rozpoczyna się tagiem `///` .
- Wspomniany Tag musi się znajdować na początku każdej linii komentarza dokumentacyjnego.
- Podstawowym tagiem, używanym w dokumentacji, jest `<summary>`, oznaczający streszczenie.
- Drugim ważnym tagiem jest `<param>`. W przypadku `<param>` korzystamy dodatkowo z atrybutu `name`. Spowoduje to przypisanie określonego komentarza do określonej zmiennej.
- Dokumentacja musi być umieszczona bezpośrednio przed komentowanym fragmentem kodu.

Przykład komentarza dokumentacyjnego

```
/// <summary>
/// Metoda losująca stan genu z
/// prawdopodobieństwem 0.5
/// </summary>
/// <returns>
/// Zwraca stan genu
/// </returns>
/// <param name='Generator'>
/// Instancja generatora liczb losowych
/// (w programie powinna być tylko jedna
/// instancja klasy Random)
/// </param>
private bool LosujGen(Random Generator)
```

Inne polecenia xml w komentarzu

- W komentarzach dokumentacyjnych wyróżnia się tak zwany krótki opis i dokładny opis. Oba rodzaje opisów znajdują się w ramach tego samego tagu, ale oddzielone są od siebie co najmniej jedną pustą linią.
- Jeżeli chcemy złamać linię, używamy znacznika `<br/>`.
- W przypadku potrzeby powiązania ze sobą opisów lub też możliwości dojścia do innego fragmentu kodu, korzystamy ze znacznika `<see>`. Na przykład:

```
/// <param name='RodzajFunkcjiCelu'>
/// Wybór rodzaju funkcji celu według klasy <see
cref="wyrownanie.Ocenianie"/>
/// </param>
```

Czy same komentarze to wszystko?

Pisanie komentarzy, jakkolwiek bardzo użyteczne, niestety jest niewystarczające. Do pełnego opisu potrzebne są jeszcze relacje istniejące wewnątrz kodu. Zatem dobry parser komentarzy dokumentacyjnych powinien również analizować kod.

Dodatkowo samo pisanie komentarzy powinno zajmować jak najmniej czasu. Zatem potrzebne są dodatkowe funkcjonalności, które nie tylko potrafią zinterpretować komentowany kod, ale również dokonać analizy za pomocą narzędzi lingwistyki komputerowej i wygenerować komentarz automatycznie.

W automatycznie generowanej dokumentacji powinny się również pojawić diagramy UML.

Czy same komentarze to wszystko?

W komentarzach nie można zapominać również o licencji oprogramowania.

Jeżeli piszemy programy oparte o GNU GPL, bezwzględnie musimy podać link do licencji oraz jej wersję (1, 2 lub 3).

Trzecia wersja GNU GPL różni się znacząco od pozostałych. Zatem projekty na laboratorium pisane są pod licencją GNU GPL v.1 lub GNU GPL v.2 .

Informacja o licencji oraz o autorze kodu musi być umieszczona w komentarzu na początku każdego pliku z kodem!

Środowisko programistyczne MonoDevelop

Automatyczne generowanie komentarzy dla języka C# zostało nadzwyczaj dobrze opracowane w MonoDevelop. Jest to darmowy, otwarty projekt, służący do programowania dla platformy .NET, dostępne zarówno dla Windows, jak i MacOS oraz Linuksa.

Autouzupełnianie komentarzy dokumentacyjnych zostało rozszerzone o narzędzia lingwistyki komputerowej dla języka angielskiego. Środowisko potrafi dokonać analizy morfologicznej nazw i utworzyć podstawowe zdania używane w komentarzach. Tym samym czas pisania dokumentacji ulega bardzo znacznemu skróceniu.

Przykład cz.1 Klasa przed wpisaniem komentarza

```
using System;

namespace BinaryTree{

    public class TheNode{
        public double StoredValue;
        public TheNode Left;
        public TheNode Right;

        public TheNode(double PriceOfPencil){
            StoredValue = PriceOfPencil;
            Left = null;
            Right = null;
        }//End of constructor
    }
}
```

Przykład cz.2 – po wpisaniu ///

```
using System;

namespace BinaryTree{
    public class TheNode{
        public double StoredValue;
        public TheNode Left;
        public TheNode Right;

        /// <summary>
        /// Initializes a new instance of the <see
cref="BinaryTree.TheNode"/> class.
        /// </summary>
        /// <param name='PriceOfPencil'>
        /// Price of pencil.
        /// </param>
        public TheNode(double PriceOfPencil{
            StoredValue = PriceOfPencil;
            Left = null;
            Right = null;
        }//End of constructor
    }
}
```

Cała dokumentacja została utworzona automatycznie, tylko po wpisaniu znaku /// przed konstruktorem.

Diagramy UML

Obecnie jest już standardem umieszczanie diagramów UML w dokumentacji. Można to realizować na wiele sposób, najlepszym z nich jest wykonanie wszystkich potrzebnych diagramów przed przystąpieniem do realizacji projektu.

Jednakże również realizuje się inżynierię wsteczną projektu poprzez generowanie diagramów UML na podstawie kodu programu.

Taką funkcjonalność posiada wiele programów CASE, przy czym zdecydowanie największa ich liczba obsługuje przede wszystkim języki: Java, C++, C#, PHP oraz JavaScript.

Jednakże czas pracy z tego rodzaju programami jest bardzo długi. Najpierw trzeba przygotować diagramy z kodu, a później włączyć je do dokumentacji.

Diagramy UML

Czas tworzenia dokumentacji do kodu może być sprawą krytyczną, zwłaszcza wtedy, gdy liczba linii kodu jest duża.

Problem automatycznego tworzenia tekstu dokumentacji do kodu został już rozwiązany, poprzez wprowadzenie omówionego wcześniej komentarza dokumentacyjnego.

Zatem, jeżeli można dokonać analizy kodu i wygenerować dokumentację, to czy nie należałoby ubogacić wspomniane narzędzie o inżynierię wsteczną i generowanie diagramów UML?

Rozwiązanie tego problemu pojawiło się stosunkowo niedawno – jest to aplikacja doxygen, służąca do generowania dokumentacji z wielu języków programowania.

Doxygen

Doxygen jest projektem open-source, służącym do generacji dokumentacji, zarówno w postaci strony WWW, jak i w pliku pdf.

Doxygen do działania wymaga zainstalowanego LaTeX'a, jeżeli ktoś chce mieć dokumentacją gotową do wydruku.

Potrafi tworzyć dokumentację w wielu językach narodowych, w tym po polsku.

Automatyzacja procesu posiada wiele parametrów, na przykład można wybrać, czy w dokumentacji ma się znaleźć kod, czy też nie ma być umieszczony.

Doxygen działa pod Linuksem, Windowsem i Mac OS.

Doxygen

Program można pobrać ze strony www.doxygen.org . Doxygen jest też dostępny w repozytoriach najważniejszych dystrybucji linuksowych. Działa zarówno w wersji na konsolę jak i okienkowej.

Działanie aplikacji doxygen omówię na przykładzie mojego własnego programu, przygotowanego do pracy naukowej związanej z MIR (Music Information Retrieval).

Omawiany program służy do estymowania oceny subiektywnej tzw. wyrównania między strunami skrzypiec. Parametry obiektywne opisujące to zjawisko można już zmierzyć, jednakże powiązanie ich z oceną subiektywną jeszcze jest na wczesnym etapie badań – aktualnie trwają dalsze prace.

Problem: wyrównanie między strunami skrzypiec

Dla skrzypka bardzo ważną cechą instrumentu jest wyrównanie poziomów dźwięku pomiędzy strunami skrzypiec. Oznacza to, że skrzypek oczekuje, że jeżeli przesuwa smyczek po strunach z jednakową prędkością, to każda ze strun odezwie się jednakowo głośno.

Tego typu założenie trudno zrealizować, jednakże jeszcze trudniej jest to zmierzyć. Skrzypce są bardzo złożonym instrumentem pod względem systemu akustycznego. Pomiar bezpośredni energii przekazanej przez ramię skrzypka do struny jest niemożliwy.

Metoda pomiaru pośredniego została opracowana w 2012 roku, zatem sprawa jest bardzo młoda. Jednak omówienie tego sposobu przekracza ramy wykładu.

Parametry obiektywne

Skrzypce posiadają cztery struny, w układzie kwintowym: G, D, A, E. W wyniku pomiaru wyrównania otrzymuje się różnice w dB pomiędzy energiami dźwięku przy tej samej energii pobudzenia.

Parametry wyrównania dla instrumentu są następujące:

- B_{DG} – wyrównanie pomiędzy struną D i G,
- B_{AG} – wyrównanie pomiędzy struną A i G,
- B_{EG} – wyrównanie pomiędzy struną E i G,
- B_{AD} – wyrównanie pomiędzy struną A i D,
- B_{ED} – wyrównanie pomiędzy struną E i D,
- B_{EA} – wyrównanie pomiędzy struną E i A,
- B_X – średnia arytmetyczna powyższych parametrów
- B_S – odchylenie standardowe z powyższych parametrów.

Oceny subiektywne

Oceny subiektywne zostały wystawione przez jurorów X Międzynarodowego Konkursu Lutniczego im. Henryka Wieniawskiego w Poznaniu, który odbył się w 2001 roku.

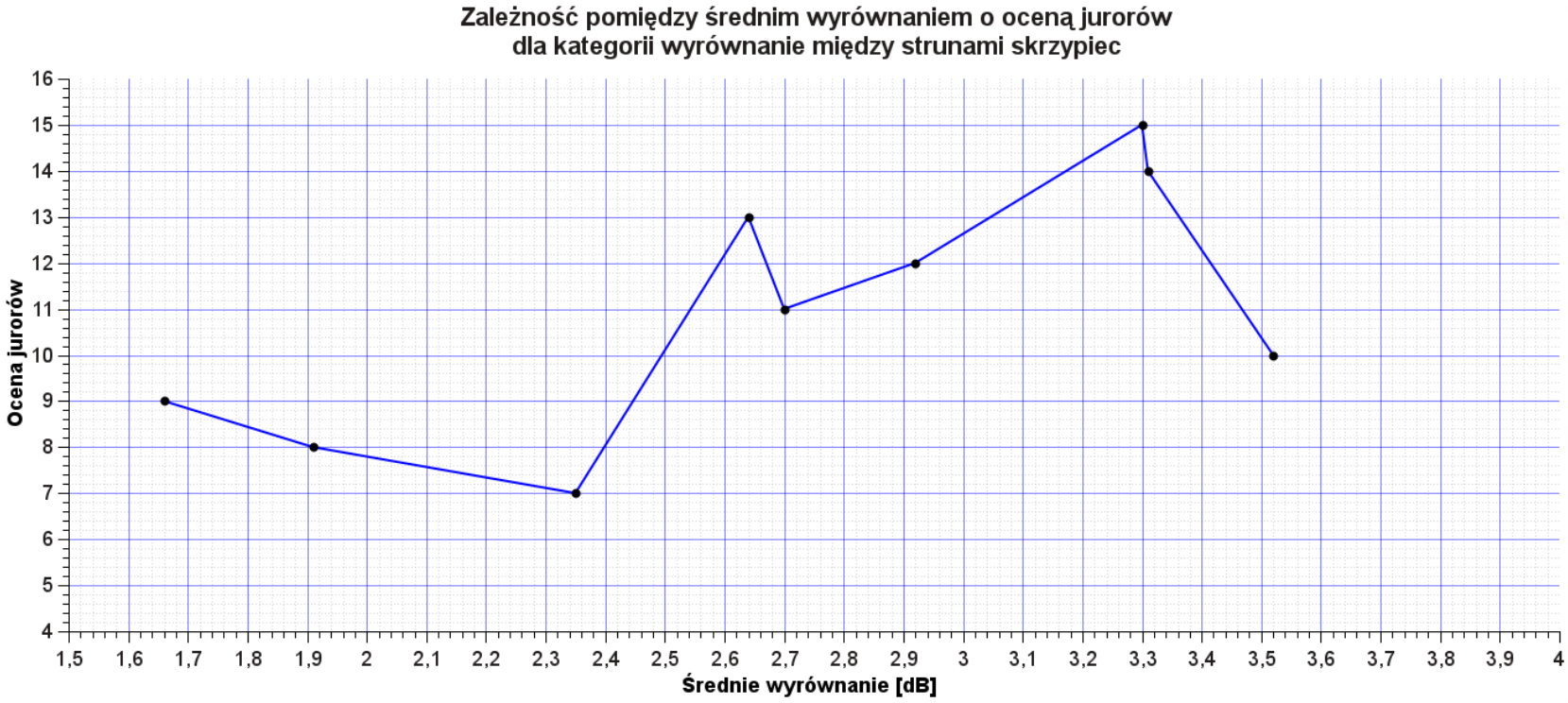
Oceny według regulaminu konkursu przyjmują wartości od 5 do 15 punktów, przy czym im wyższa ocena, tym lepsza opinia o instrumencie.

Instrumentami badanymi są skrzypce biorące udział we wspomnianym konkursie.

Każdy instrument oceniany był niezależnie przez czterech jurorów, przy czym instrumenty na każdym etapie konkursu posiadały różne oznaczenia (numery), aby jurorzy nie mogli rozpoznać danego egzemplarza.

Zależności wstępne

Celem sprawdzenia, czy rzeczywiście jest możliwe odtworzenie oceny jurorów na podstawie parametrów obiektywnych, dokonano grupowania instrumentów według mediany ocen i obliczanie średniego współczynnika B_X dla każdej z grup. Otrzymano następujący rezultat:



Wybranie klasyfikatora

Przedstawiona wcześniej zależność przypomina trochę wielomian czwartego rzędu, jednakże jest to zależność uśredniona, zatem należy przyjąć, że liczba parametrów będzie odpowiednio większa.

Przyjęto następującą postać klasyfikatora:

$$\bar{x} = a_1 B_{DG} + a_2 B_{AG} + a_3 B_{EG} + a_4 B_{AD} + a_5 B_{ED} + a_6 B_{EA} + a_7 B_X + a_8 B_S$$

$$Ocena = b_1 |\bar{x}|^{\alpha} + b_2 |\bar{x}|^{\beta} + b_3 |\bar{x}|^{\gamma} + b_4 |\bar{x}|^{\eta} + b_5 |\bar{x}|^{\theta}$$

Klasyfikator jest wzorowany na wielomianie czwartego stopnia. Wartości parametrów muszą zostać znalezione.

Uwaga! W 2014 roku na Forum Acusticum zaprezentowałem kompletny klasyfikator wyrównania, różniący się od tego, który został podany wyżej. Jednakże nowy klasyfikator również został znaleziony za pomocą algorytmu genetycznego.

Własności parametrów

Parametry powinny posiadać jakąś interpretację, przy czym najważniejsze są te, które tworzą kombinację liniową ze współczynników obiektywnie opisujących wyrównanie.

Zatem przyjęto dodatkowe ograniczenie – współczynniki: a_1, a_2, a_3, a_4, a_5, a_6, a_7 oraz a_8 muszą być większe lub równe zero.

Dodatkowo należało wprowadzić możliwość operowania na liniowych wartościach współczynników wyrównania. Wspomniane współczynniki podawane są w dB. Zatem należało je zamienić, na liniowe, według wzoru:

$$y = 10^{\frac{x}{10}}$$

x – oznacza wartość wyrażoną w dB, y – wartość liniową.

Poszukiwanie wartości parametrów

Parametry obiektywne wyrównania natężenia dźwięku pomiędzy strunami są już pomierzone.

Zatem, jak znaleźć wartości pozostałych osiemnastu parametrów?

W tym celu można zastosować różnego rodzaju algorytmy, na przykład z rodziny metod Monte Carlo, czy też algorytmy genetyczne lub ewolucyjne.

Jako rozwiązanie wybrano algorytm genetyczny, gdyż posiada on pewne możliwości pozwalające „wyjść” z minimum lokalnego.

Ogólny algorytm genetyczny

Funkcja celu

Ocena osobnika odbywa się za pomocą funkcji celu. Jest to wyrażenie, które informuje, jak bardzo osobnik jest dopasowany do pożądanego rezultatu. Zazwyczaj, im mniejsza wartość funkcji celu, tym lepsze dopasowanie.

W przypadku omawianego programu, zastosowano trzy funkcje celu do wyboru:

- Średnia arytmetyczna modułów różnic pomiędzy oceną wyliczoną a oceną jurorów.
- Maksymalny moduł z różnicy pomiędzy ocenami wyliczonymi a oceną jurorów.
- Liczba instrumentów, dla których moduł z różnicy pomiędzy oceną wyliczoną a oceną wystawioną przez jurorów jest mniejszy od 1.

Projektowanie oprogramowania

Jako metodologię tworzenia oprogramowania przyjęto następujące zasady:

1) Program jest dzielony na trzy pakiety: algorytm genetyczny, program właściwy i pakiet z testami jednostkowymi.

2) Kod programu pisany jest stylem systematycznym.

3) Przyjęto, że modelowanie następuje metodą wstępującą, czyli na początku najprostsze klasy, a na końcu interface.

4) Jako język programowania wybrano C#, przy czym dokumentacja jest wpisana w kod i następnie wygenerowana z użyciem doxygen'u.

Pakiet algorytmu genetycznego

Kolejność budowania klas jest następująca:

1) Klasa kodująca pojedynczy gen binarny (dwuwartościowy)

2) Klasa kodująca genom (tablicę genów)

3) Klasa kodująca osobnika.

4) Klasa kodująca populację.

Pakiet do algorytmów genetycznych został zaprojektowany jako ogólny.

Pakiet obliczający wyrównanie

Kolejność powstawania klas:

1) Klasa przechowująca parametry instrumentów wraz z ocenami jurorów.
2) Klasa klasyfikatora.
3) Klasa dekodująca i kodująca genom.
4) Osobnik reprezentujący parametry klasyfikatora.
5) Klasa kodująca i dekodująca genom.
6) Klasa oceniająca.
7) Klasa wczytująca dane z pliku.
8) Klasa zapisująca wyniki.
9) Klasa zawierająca główny program (z metodą Main).

Testy jednostkowe

Testy jednostkowe znajdują się w jednej klasie i obejmują kilka kluczowych metod.

Są to:

- Wczytywanie danych wejściowych z pliku.
- Zapisywanie wyników do pliku wyjściowego.
- Kodowanie i dekodowanie genomu.
- Sortowanie (testowanie metody Array.sort)

Funkcjonalność interfejsu użytkownika

Przewidziano następujące możliwości:

- Wpisanie nazwy pliku z danymi.
- Wpisanie nazwy pliku wyjściowego.
- Podanie parametrów populacji (liczba osobników, maksymalna wartość współczynników, prawdopodobieństwo mutacji, maksymalna liczba pokoleń bez poprawy).
- Wybór estymowanej oceny (juror 1, juror 2, juror 3, juror 4, mediana ocen, średnia ocen).
- Wybór funkcji celu.

Dane wejściowe

Dane wejściowe to parametry wyrównania oraz oceny instrumentów biorących udział w X Międzynarodowym Konkursie Lutniczym im. Henryka Wieniawskiego w Poznaniu.

Nagrania wspomnianych skrzypiec są przechowywane w multimedialnej bazie danych AMATI.

Wszystkie badane instrumenty konkursowe były nowymi konstrukcjami.

Co dalej?

Gdy już mniej więcej wiadomo, czas uruchomić doxygen, wygenerować dokumentację i omówić program korzystając z tak utworzonego dokumentu.