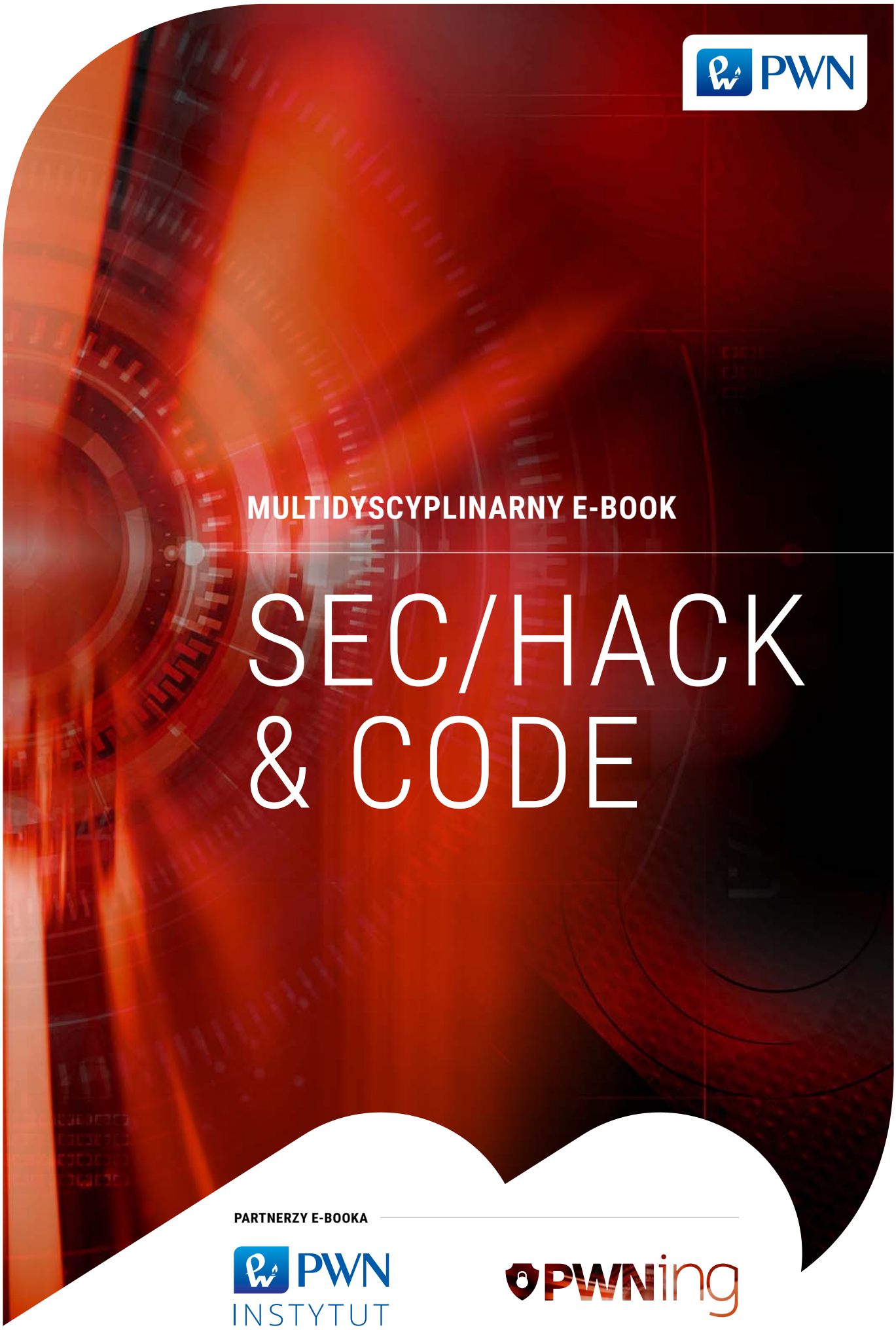


PWN

MULTIDYSCYPLINARNY E-BOOK

SEC/HACK & CODE

PARTNERZY E-BOOKA

PWN INSTYTUT

PWNing

SPIS TREŚCI

GYNVAEL COLDWIND

POSUWAJĄC SIĘ DO OCZYWISTOŚCI, NAPISZĘ, ŻE PROGRAMOWANIE W DZISIEJSZYCH CZASACH JEST ISTOTNE

Co więcej, jego owoce otaczają nas ze wszystkich stron – i to nie tylko w przenośni. Myśląc o niniejszym wstępie, przechadzam się po mieszkaniu i rozglądam się za urządzeniami, w który może być ukryty jakikolwiek – choćby najmniejszy – programowalny mikroprocesor. Pierwsze w oczy rzucają się oczywiście komputery i laptop – każdy z nich ma przynajmniej kilka, jeśli nie kilkanaście procesorów i mikrokontrolerów (na przykład po jednym µC na każdy SSD). Ale wiem, że to tylko wierzchołek góry lodowej. Idąc dalej, znajduję drukarkę bezprzewodową (opartą o procesor ARM z zainstalowanym Linuxem), inteligentne oświetlenie (każda żarówka ma swój mały mikrokontroler), kuchenkę mikrofalową czy czajnik z nadmierną liczbą przycisków. Podejrzliwie spoglądam również w kierunku wentylatora wyświetlającego temperaturę, jak i przysłowiowej lodówki.

Biorąc pod uwagę, jak wiele różnych sprzętów elektronicznych i AGD, serwisów internetowych, czy aplikacji używamy na co dzień, można stwierdzić, że dobrze zaprojektowane i zaimplementowane oprogramowanie bardzo ułatwia nasze życie, a także pozwala zaoszczędzić dużo czasu.

Niestety, informatyka i programowanie nadal są bardzo młodą dziedziną, a co za tym idzie, nie każde oprogramowanie spełnia powyższe kryteria „dobrego zaprojektowania" czy „dobrej implementacji".

Konsekwencje pomyłek programistycznych w zdecydowanej większości przypadków są niegroźne – ot urządzenie czy program się zawieszą albo rzucą błędem i trzeba je zrestartować. Rzadziej skutki wchodzą na terytorium bezpieczeństwa fizycznego czy bezpieczeństwa danych.

I choć chciałbym napisać, że nie jest tak źle – szczególnie jeśli chodzi o te pomniejsze urządzenia AGD – to na myśl od razu przychodzi artykuł z 2017 r. opowiadający o tym, jak cyberprzestępcy wykradli 10 gigabajtów danych z kasyna, zaczynając swój atak od zhakowania inteligentnego akwarium [1]. Również angielskie powiedzenie „S in IoT stands for Security" nie wzięło się znikąd, choć skłamałbym, pisząc, że problem ogranicza się do urządzeń klasy IoT.

Programowanie jest zdecydowanie trudną dziedziną, a jego dynamiczny rozwój również nie ułatwia sprawy. To samo można zresztą powiedzieć o bezpieczeństwie aplikacji, jako jednej ze specjalizacji ogólno pojętego bezpieczeństwa IT. Tym bardziej istotne jest, aby komunikacja – i, przede wszystkim, wymiana wiedzy – pomiędzy programistami a hakerami była płynna, a współpraca była korzystna dla obu stron.

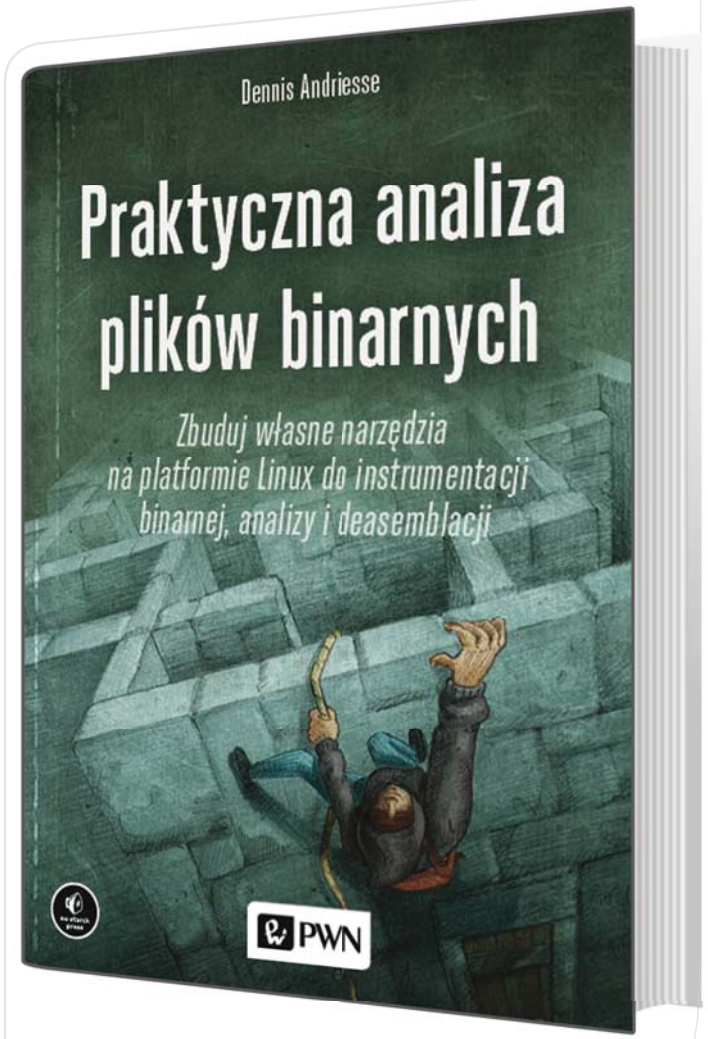
Gynvael Coldwind
Oberrieden, 2020

[1] Mathews L., Forbes, Criminals Hacked A Fish Tank To Steal Data From A Casino, https://www.forbes.com/sites/leemathews/2017/07/27/criminals-hacked-a-fish-tank-to-steal-data-from-a-casino/ (27 czerwca 2017)

DENNIS ANDRIESSE

PRAKTYCZNA ANALIZA PLIKÓW BINARNYCH

Nawet przy najbardziej złożonej analizie binarnej możesz dokonać zadziwiająco zaawansowanych działań, łącząc zestaw podstawowych narzędzi we właściwy sposób. Oszczędzi ci to wielu godzin pracy nad implementacją równoważnej funkcjonalności na własną rękę.



KUP KSIĄŻKĘ • KUP E-BOOK

SZUKANIE WSKAZÓWEK ZA POMOCĄ STRINGS

Aby zorientować się, co robi jakiś plik binarny i jakiego rodzaju danych oczekuje na wejściu, możesz sprawdzić, czy plik ten zawiera jakieś pomocne łańcuchy, które mogą ujawnić jego przeznaczenie. Na przykład jeżeli widzisz łańcuchy zawierające fragmenty żądań HTTP albo URL, możesz bezpiecznie zgadnąć, że plik binarny ma coś wspólnego z siecią. Kiedy masz do czynienia ze złośliwym oprogramowaniem takim jak bot, być może odnajdziesz łańcuchy zawierające polecenia akceptowane przez bota, jeżeli nie zostały zaciemnione. Mógłbyś nawet znaleźć łańcuchy zostawione po debugowaniu, których zapomniał usunąć programista, a wiemy, że tak się zdarzało w prawdziwym złośliwym oprogramowaniu! Możesz użyć narzędzia o nazwie strings, by szukać łańcuchów w pliku binarnym (lub jakimkolwiek innym pliku) na Linuksie. Narzędzie to przyjmuje jako wejście jeden lub więcej plików, a następnie drukuje wszystkie łańcuchy znaków drukowalnych znalezione w tych plikach.

Zauważ, że strings nie sprawdza, czy odnalezione łańcuchy były rzeczywiście przeznaczone jako czytelne dla człowieka, stąd użycie go do plików binarnych może na wyjściu dać jakieś fałszywe łańcuchy jako wynik sekwencji znaków, które przypadkowo okazały się drukowalne. Możesz podregulować zachowanie strings za pomocą opcji. Na przykład możesz użyć przełącznika –d, by strings drukował tylko łańcuchy znalezione w sekcjach danych pliku binarnego, zamiast drukowania wszystkich sekcji. Domyślnie strings wyświetla tylko łańcuchy o długości co najmniej czterech znaków, ale możesz ustawić inną minimalną długość łańcucha, używając opcji –n. Dla naszych celów opcje domyślne wystarczą; zobaczmy, co możesz znaleźć w pliku binarnym ctf używając strings, jak pokazano na poniższym listingu 5.5.

```
$ strings ctf
/lib64/ld–linux–x86–64.so.2
lib5ae9b7f.so
__gmon_start__
_Jv_RegisterClasses
_ITM_deregisterTMCloneTable
_ITM_registerTMCloneTable
_Z8rc4_initP11rc4_state_tPhi
...
DEBUG: argv[1] = %s
checking '%s'
show_me_the_flag
>CMb
–v@P^:
flag = %s
guess again!
It's kinda like Louisiana. Or Dagobah. Dagobah – Where Yoda lives!
;*3$"
zPLR
GCC: (Ubuntu 5.4.0–6ubuntu1~16.04.4) 5.4.0 20160609
.shstrtab
.interp
.note.ABI–tag
.note.gnu.build–id
.gnu.hash
.dynsym
.dynstr
.gnu.version
.gnu.version_r
.rela.dyn
.rela.plt
.init
.plt.got
.text
.fini
.rodata
.eh_frame_hdr
.eh_frame
.gcc_except_table
.init_array
.plt.got
.text
.fini
.rodata
.eh_frame_hdr
.eh_frame
.gcc_except_table
.init_array
.fini_array
.jcr
.dynamic
.got.plt
.data
.bss
.comment
```

Widzisz tu pewne łańcuchy, które napotkasz w większości plików ELF. Na przykład, jest tu nazwa interpretera, taka jak znajdowana w sekcji .interp i kilka nazw symbolicznych znajdowanych w sekcji .dynstr. Jednak żaden z tych łańcuchów nie jest szczególnie interesujący dla naszych celów. Na szczęście jest także kilka bardziej użytecznych łańcuchów. Na przykład

pojawia się coś, co wygląda na komunikat debugowania, co sugeruje, że program oczekuje opcji wiersza poleceń. Są także testy pewnego rodzaju, prawdopodobnie wykonywane na łańcuchu wejściowym. Jeszcze nie wiesz, jaka powinna być wartość opcji wiersza poleceń, ale możesz spróbować kilku z innych ciekawie wyglądających łańcuchów, np. show_me_the_flag, które mogłyby zadziałać. Jest także tajemniczy łańcuch, który zawiera wiadomość o niejasnym przeznaczeniu. Na razie nie wiesz, co oznacza ta wiadomość, ale wiesz na pewno z analizy lib5ae9b7f.so, że ten plik binarny używa szyfrowania RC4. Być może ta wiadomość służy jako klucz? Kiedy już wiesz, że plik binarny oczekuje opcji w wierszu polecenia, zobaczmy, czy dodanie przypadkowej opcji przybliży cię do zdobycia flagi.

Z braku lepszych pomysłów użyjmy po prostu łańcucha foobar:

```
$ ./ctf foobar
checking 'foobar'
$ echo $?
1
```

Plik zrobił teraz coś nowego. Informuje, że sprawdza wprowadzony przez ciebie łańcuch. Jednak sprawdzenie nie powiodło się, ponieważ plik binarny nadal zgłasza kod błędu. Zaryzykujmy i spróbujmy jednego z innych interesujących łańcuchów, które znalazłeś, na przykład show_me_the_flag, który wygląda obiecująco.

```
$ ./ctf foobar
checking 'foobar'
$ echo $?
1
```

Udało się! Sprawdzenie się najprawdopodobniej powiodło. Niestety, status wyjścia ciągle wynosi 1, więc czegoś jeszcze musi brakować. Na domiar złego wyniki strings nie dają nam żadnych dalszych wskazówek.

JERZY SURMA

HAKOWANIE SZTUCZNEJ INTELIGENCJI

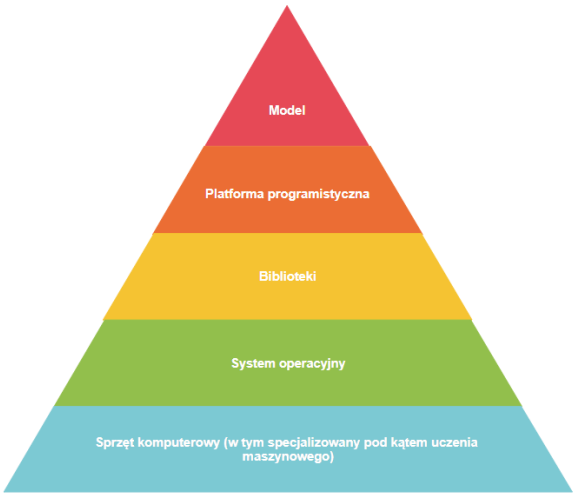
Wraz z rozwojem cyfryzacji, w tym m.in. intensywnego rozwoju Internetu Rzeczy, rośnie znaczenie automatyzacji procesów biznesowych oraz użycia inteligentnych systemów wspomagania decyzji z wykorzystaniem metod sztucznej inteligencji i technik zaawansowanej analizy danych. Ten bezsprzecznie ważny trend rozwojowy implikuje istotne zagrożenia i ryzyka.

KUP KSIĄŻKĘ

5. BEZPIECZEŃSTWO APLIKACJI SYSTEMÓW UCZĄCYCH SIĘ

KAMIL FRANKOWICZ

Klasycznie problematyka bezpieczeństwa systemów maszynowego uczenia się dotyczy głównie zagadnień związanych z uczeniem, testowaniem i użytkowaniem modeli eksploracji danych. Niemniej na to zagadnienie należy spojrzeć znacznie szerzej, biorąc pod uwagę środowiska programistyczne, oprogramowanie systemowe i sprzęt, które są wykorzystywane, aby systemy uczące się (a dokładnie model realizujący określone zadanie) mogły być budowane i użytkowane. Zagadnienie to ilustruje w dużym uproszczeniu rysunek 5.1, gdzie model, umiejscowiony na górze piramidy, powstaje w wyniku użycia środowiska programistycznego[1], które odwołuje się do różnych bibliotek[2], a biblioteki są uruchamiane z wykorzystaniem określonego systemu operacyjnego na konkretnym sprzęcie komputerowym. W książce Hakowanie sztucznej inteligencji zostaną omówione wybrane ataki na środowiska programistyczne i sprzęt. Szczegółowo opisano ataki na wybrane biblioteki, z wykorzystaniem metod testowania oprogramowania.



Rysunek 5.1. Model systemu maszynowego uczenia się w relacji do użytego oprogramowania i sprzętu. Źródło: opracowanie własne.

ATAK NA PLATFORMĘ PROGRAMISTYCZNĄ

Model systemu uczącego się jest implementowany jako program wykonywany w środowisku programistycznym dedykowanym do budowy aplikacji uczenia maszynowego (zob. rysunek 5.1), np. TensorFlow. Powoduje to takie same implikacje dotyczące bezpieczeństwa jak w przypadku uruchamiania klasycznego kodu pobranego z internetu – jest to program niezaufany. Pobieranie gotowych modeli jest obarczone wysokim ryzykiem ataku na platformę programistyczną – w takich przypadkach model może być napisany jak klasyczny malware i próbować doprowadzić do złośliwych aktywności lub przejęcia serwera, na którym został uruchomiony. Nie tylko intencjonalnie złośliwy kod może przysłużyć się do osłabienia jakości pracy platformy. Powody często są prostsze i dotyczą niedojrzałych procesów tworzenia kodu.

Zhang i in. (2018) przeanalizowali metody wytwórcze oprogramowania wykorzystującego uczenie maszynowe i wyróżnili sześć głównych problemów systemów tworzonych na bazie platformy programistycznej TensorFlow:

- Błędy numeryczne lub brak precyzji obliczeń.
- Złe wykorzystanie API dostarczanego przez platformę programistyczną.
- Niedostosowanie API do aktualnej wersji.
- Niska jakość klasyfikacji przez wyuczony model.
- Złe parametry modelu.
- Brak zrozumienia działania platformy i jej funkcjonalności.

Problemy te bezpośrednio nie prowadzą do obniżenia bezpieczeństwa modelu, lecz do jego wydajności i wrażliwości na zmiany, co może skutkować istotnym obniżeniem jakości klasyfikacji. Ciekawą obserwacją płynącą z badania Zhanga i in. (2018) jest stosunkowo częste wprowadzanie błędów przez programistów przez nieświadome kopiowanie wadliwego kodu z internetowych zasobów, głównie z serwisu Stack Overflow.

```
1.  def _init_parameters(self, input_data, labels):
2.    input_size = tf.shape(input_data)[1]
3.    num_classes = tf.shape(labels)[1]
4.    stddev = 1.0 / tf.cast(input_size, tf.float32)
5.    w_shape = tf.pack([input_size, num_classes], 'w-shape')
6.    normal_dist = tf.truncated_normal(w_shape, stddev=stddev,
        name='normaldist')  // normal_dist.get_shape() gets [Dimension(None), Dimension(None)]
7.    self.w = tf.Variable(normal_dist, name='weights)
```

(a) A faulty TensorFlow example

```
6.    normal_dist = tf.truncated_normal(w_shape, stddev=stddev,
        name='normaldist')
7.    + normal_dist.set_shape([input_data.get_shape()[1], labels.get_shape()[1]])
8.    self.w = tf.Variable(normal_dist, name='weights)
```

(b) A recommended fix

Rysunek 5.3. Przykład błędu w kodzie pobranym z serwisu StackOverflow wraz ze sposobem naprawy wadliwego sposobu uczenia maszynowego
Źródło: Zhang, Y. i in. (2018). An Empirical Study on TensorFlow Program Bugs, http://sccpu2.cse.ust.hk/castle/materials/issta18main-p98-p.pdf (dostęp: 1.04.2020 r.).

1 Środowiska programistycznego dedykowanego do budowy systemów maszynowego uczenia się i sztucznej inteligencji, takiego jak np. TensorFlow czy Keras.
2 Biblioteki numeryczne, obsługa plików graficznych biblioteki systemowe, obsługa archiwów itp.

Kolejnym zagrożeniem są dane wejściowe przetwarzane przez model. Mogą to być pliki graficzne i ich metadane, strumień wideo z kamery internetowej, tekst, produkty serializacji itp. Obsługa różnych formatów danych jest zapewniana dwojako, w zależności od filozofii tworzenia kodu. Najpopularniejszym podejściem jest wykorzystanie zewnętrznych bibliotek do różnych formatów plików. Niektóre platformy programistyczne, np. popularny TensorFlow, samodzielnie implementują obsługę niektórych formatów plików, m.in BMP, JPEG, WAV, CSV lub SQLITE.

Biblioteki (zob. rys. 5.1) pozwalają na szybką budowę oprogramowania i zapewniają obsługę skomplikowanych operacji numerycznych, kryptografii lub elementów prywatyzujących wrażliwe dane, np. medyczne. Zapewniają zamknięcie często specjalistycznej wiedzy w formie API, które jest wygodne dla deweloperów tworzących frameworki uczenia maszynowego. Zasadniczym problemem z bibliotekami jest fakt, że jest to kolejny element komplikujący program oraz wymagający, o czym często deweloperzy zapominają, regularnej aktualizacji. Framework korzystający z podatnych bibliotek również staje się wrażliwy na ataki.

Krytycznym problemem jest podatność dotycząca biblioteki przetwarzającej niezaufane dane wejściowe pochodzące ze źródła będącego pod kontrolą użytkownika. Na przykład w projekcie Tensorflow takie problemy spowodowała biblioteka snappy[3], służąca do szybkiej archiwizacji plików – nieaktualizowana wersja zawierała podatność typu memory corruption, pozwalającą na odczyt pamięci procesu lub awarię programu podczas przetwarzania specjalnie spreparowanego archiwum[4].

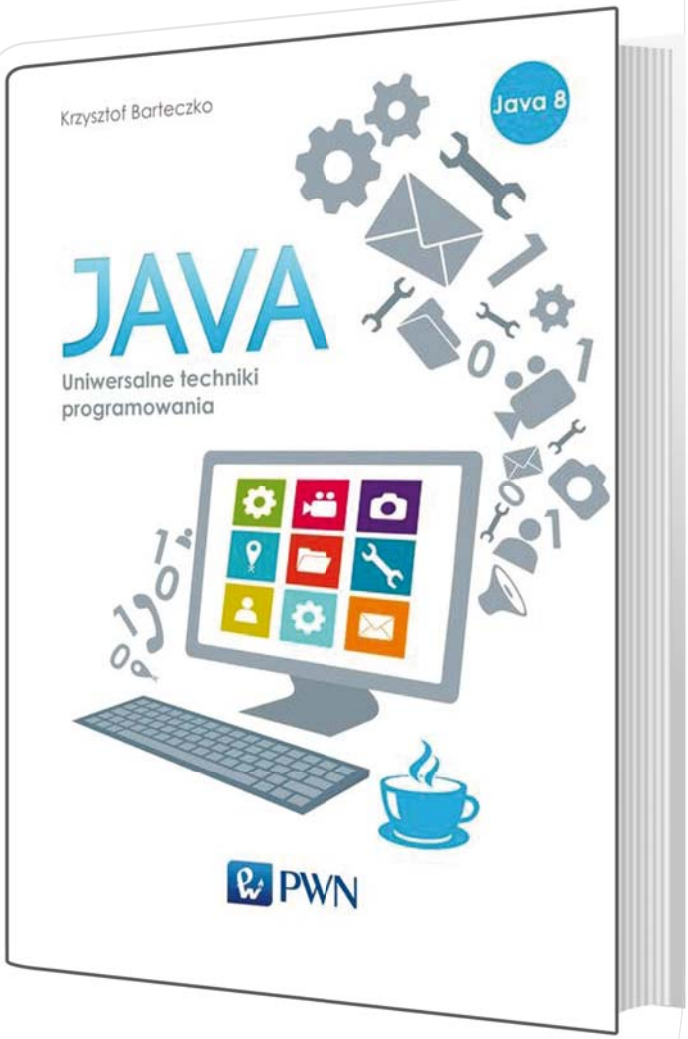
Z perspektywy rozwiązań uczenia maszynowego jesteśmy podatni na dwóch poziomach w przeciwieństwie do klasycznego oprogramowania. Pierwszym punktem wejścia do systemu jest model, który korzystając z gotowych rozwiązań, może zawierać w sobie złośliwą zawartość. Drugim jest klasyczne oprogramowanie i błędy w nim zawarte, w tym przypadku platformy programistyczne oraz zależności, z których korzystają.

3 Repozytorium GitHub projektu snappy, https://github.com/google/snappy (dostęp: 1.04.2020 r.).
4 *TFSA–2018–005: Old Snappy Library Usage Resulting in Memcpy Parameter Overlap*, https://github.com/tensorflow/tensorflow/blob/master/tensorflow/security/advisory/tfsa-2018-005.md (dostęp: 4.04.2020 r.).

KRZYSZTOF BARTECZKO

JAVA. UNIWERSALNE TECHNIKI PROGRAMOWANIA

Najważniejszą innowacją języka w wersji 8 było wprowadzenie elementów programowania funkcyjnego. Tutaj przyjrzymy się im z „lotu ptaka”, zwracając uwagę na koncepcje i wybrane praktyczne narzędzia. Powinno to dopomóc w dalszej lekturze bardziej szczegółowych i technicznych rozdziałów, dotyczących elementów programowania funkcyjnego.



KUP KSIĄŻKĘ

KUP E-BOOK

LAMBDA-WYRAŻENIA: PIERWSZE SPOTKANIE

Zanim poznamy lambda-wyrażenia w szczegółach, możemy o nich myśleć jako o bezpośrednio podawanych fragmentach kodu, swoistych funkcjach bez nazwy, które są traktowane jak „prawdziwe obiekty", czyli np. mogą być przekazywane metodom. Dzięki zastosowaniu lambda wyrażeń poprzedni przykład wywołania metody create() upraszczamy praktycznie do jednego wiersza:

```
List<Integer>
target = create(src, n -> n < 10, n -> n*n );
```

Tu zapis: n -> n < 10 jest lambda-wyrażeniem, które możemy traktować jako anonimową funkcję o parametrze n, która zwraca wynik wyrażenia n<10.
Nie wchodząc na razie w szczegóły, można powiedzieć, że kompilator „dopasowuje" to lambda-wyrażenie do drugiego parametru (Filter<S>) metody create(...), w wyniku czego użyta tam metoda test(...) interfejsu Filter „pod spodem" uzyskuje odpowiednią implementację (boolean test(Integer n) { return n < 10; }).
Podobnie się dzieje z lambda-wyrażeniem:

```
n -> n*n
```

pasującym do wywołania metody transform(...) interfejsu Transformer.

Przykłady wykorzystania lambda-wyrażeń w różnych kontekstach zawiera kod 6.2

```
List<Integer> num = Arrays.asList( 1, 3, 5, 10, 9, 12, 7);
List<String> txt = Arrays.asList(„ala", „ma", „kota",
                    „aleksandra", „psa", „azora" );
List<Employee> emp = Arrays.asList(
          new Employee(„Kowal", „Jan", 34, 3400.0),
          new Employee(„As", „Ala", 27, 4100.0),
          new Employee(„Kot", „Zofi a", 33, 3700.0),
          new Employee(„Puchacz", „Jan", 41 , 3600.0)
          );
System.out.println(
    create(num, n-> n%2!=0, n->n*100)
);

System.out.println(
    create(txt,
          s -> s.startsWith(„a"),
          s -> s.toUpperCase() + „ „ + s.length())
);

List<Employee> doPodwyzki =
  create(emp,
      e -> e.getAge() > 30 && e.getSalary() < 4000,
      e -> e
  );
System.out.println(„Podwyzki powinni uzyskac:");
System.out.println(doPodwyzki);
```

Kod 6.2. Wprowadzający przykład zastosowania lambda-wyrażeń
Uwaga. W kodzie 6.2 wykorzystano prostą klasę Employee o następującej postaci:

```
public class Employee {
    private String lname;
    private String fname;
    private Integer age;
    private Double salary;

     public Employee(String lname, String fname, Integer age, Double salary) {
          this.lname = lname;
          this.fname = fname;
          this.age = age;
          this.salary = salary;
}

public String getLname() {
    return lname;
}
public String getFname() {
    return fname;
}
public Integer getAge() {
    return age;
}
public Double getSalary() {
    return salary;
}
public void setSalary(Double salary) {
    this.salary = salary;
}
@Override
public String toString() {
    return lname + „ „ + fname;
 }
}
```

Kod 6.2 wyświetli na konsoli następujące wyniki:

```
[100, 300, 500, 900, 700]
[ALA 3, ALEKSANDRA 10, AZORA 5]
Podwyżki powinni uzyskać:

[Kowal Jan, Kot Zofi a, Puchacz Jan]
```

Jak widać, utworzona wcześniej metoda create() jest tu wykorzystywana dosyć elastycznie (dla różnych typów danych, różnych warunków, różnych operacji), a dzięki lambda-wyrażeniom sformułowanie tego, co chcemy osiągnąć, jest bardzo proste i czytelne. Co więcej, metoda create jest tak napisana, że wynikowa lista wcale nie musi zawierać elementów tego samego typu, co lista źródłowa. Możemy np. łatwo uzyskać listę pensji wszystkich pracowników; w wyniku wykonania.

```
List<Double> sal =
        create(emp,
                e -> true,
                e -> e.getSalary()
        );
System.out.println(sal);
```

otrzymamy:

```
[3400.0, 4100.0, 3700.0, 3600.0]
```

Przy tej okazji warto wspomnieć o referencjach do metod (zapisywanych z użyciem podwójnego dwukropka), które też są lambda-wyrażeniami. Zapis e -> e.getSalary() jest równoważny następującej referencji do metody getSalary z klasy Employee:

```
Employee::getSalary
```

i kod możemy zapisać nieco prościej:

```
create(emp,
        e -> true,
        Employee::getSalary
);
```

Oczywiście nie zawsze musimy tworzyć nową listę z wynikami przetwarzania. Możemy np. modyfikować elementy istniejącej. Czy nie warto by np. od razu zmienić pensję naszym pracownikom? Aby dokonywać modyfikacji, możemy napisać metodę change, która będzie otrzymywać listę źródłową i zmieniać te jej elementy, które spełniają podany warunek. Naturalnie po to, by móc korzystać z lambda-wyrażeń, potrzebny będzie interfejs. Niech się nazywa Modifier i ma metodę modify(), która przyjmuje argument typu S i nie zwraca żadnego wyniku.

```
public interface Modifier<S> {
    void modify(S v);
}
```

Metoda change() może wyglądać tak:

```
static <S> void change(List<S> list, Filter<S> f, Modifier<S> mod) {
   for (S e : list) {
      if (f.test(e)) {
      mod.modify(e);
      }
   }
}
```

a jej wykorzystanie do zmiany pensji pracowników z użyciem lambda-wyrażenia tak:

```
List<Employee> emp =
Arrays.asList(
                    new Employee(„Kowal", „Jan", 34, 3400.0),
                new Employee(„As", „Ala", 27, 4100.0),
                    new Employee(„Kot", „Zofi a", 33, 3700.0),
                new Employee(„Puchacz", „Jan", 41, 3600.0)
                );
change(emp,
                e -> e.getAge() > 30 && e.getSalary() < 4000,
                e -> e.setSalary(e.getSalary()+200)
                );
for (Employee e : emp) System.out.println(e + „ „ + e.getSalary());
```

Uwaga. W tym przypadku elementy listy muszą być obiektami modyfikowalnymi (i są, ponieważ klasa Employee ma metodę setSalary).

Interfejsy z pakietu *java.util.function* a wyjątki kontrolowane

Deklaracje metod interfejsów z pakietu java.util.function nie zawierają klauzuli throws Exception. W konsekwencji wszystkie wyjątki kontrolowane, które mogą powstać przy wykonaniu lambda-wyrażeń dobieranych do tych interfejsów funkcyjnych, muszą być obsługiwane w kodzie lambda-wyrażeń za pomocą konstrukcji try-catch. Na przykład, zastosujmy pokazaną w poprzednim punkcie metodę transform, przekształcającą elementy kolekcji za pomocą podanej funkcji, w następującym ilustracyjnym kodzie 7.13.

```
import java.util.*;
import java.util.function.*;
// ...

static <S, T> List<T> transform(Collection<S> src,
Function<S, T> f) {
    // ...
}
static <T> T sameAfterSec(T arg) throws InterruptedE-
xception {
   Thread.sleep(1000);
   return arg;
}

public static void main(String[] args) throws Exception
{
    System.out.println(
        transform(Arrays.asList(1, 2, 3),
                  n -> n == 2 ? sameAfterSec(n) : n+1)
    );
}
```

Kod 7.13. Brak obsługi wyjątków kontrolowanych w kodzie lambda-wyrażenia odpowiadającego interfejsowi Function

Ten kod nie skompiluje się poprawnie, ponieważ metoda sameAfterSec może zgłosić kontrolowany wyjątek InterruptedException. Obsługi tego wyjątku nie możemy zawrzeć w try-catch okalającym wywołanie metody println ani przerzucić tej obsługi na JVM za pomocą klauzuli throws metody main.

Dzieje się tak dlatego, że deklaracja metody apply(..) interfejsu Function nie zawiera klauzuli throws, wobec tego jedyną możliwością jest obsługa wyjątku w ciele lambdy, co czyni kod mniej zgrabnym:

```
transform(Arrays.asList(1, 2, 3),
          n -> {
            if (n == 2)
              try {
                return sameAfterSec(n);
              } catch (Exception exc) {
                return n;
              }
            else return n+1;
          })
```

Sytuacja ta jest szczególnie nieprzyjemna w przetwarzaniu strumieniowym: przy wyjątkach kontrolowanych, które mogą powstawać w lambda-wyrażeniach stosowanych w funkcjach mapowania, fi ltrowania, redukcji, kod zdecydowanie traci na klarowności i elastyczności. Może się zatem okazać, że warto będzie poświęcić chwilę na przygotowanie dodatkowych konstrukcji ad hoc, które zdejmą z nas obowiązek obsługi wyjątków kontrolowanych w ciele lambda-wyrażeń. Jednym z możliwych rozwiązań jest dostarczenie własnych interfejsów funkcyjnych rozszerzających te z pakietu java.util.function.

Przykładowe rozszerzenie interfejsu Function przedstawia kod 7.14.

```
@FunctionalInterface
interface CEF<T,R> extends Function<T,R> {
   R checkedApply(T arg) throws Exception;

   default R apply(T arg) {

      try {
         return checkedApply(arg);
      } catch (RuntimeException exc) {
         throw exc;
      } catch (Exception exc) {
         throw new RuntimeException(exc);
   }
}
```

Kod 7.14. Rozszerzenie interfejsu Function w celu uniknięcia konieczności obsługi wyjątków kontrolowanych w ciele lambdy

Uwaga. Pokazane rozwiązanie, wzorowane na projekcie throwing-lambdas (https://github.com/fge/throwing-lambdas) jest uproszczone, ponieważ pomija

przypadki Throwable, które nie są ani typu Error, ani Exception.

Interfejs CEF (od controlled exception function) rozszarza interfejs Function. Dostarczono w nim nową abstrakcyjną metodę przetwarzania (checkedApply), która w sygnaturze zawiera klauzulę throws sygnalizująca możliwość powstania wyjątku kontrolowanego. Jednocześnie zdefiniowano (jako domyślną) standardową metodę apply interfejsuFunction w taki sposób, by delegowała przetwarzanie do nowej metody i w razie wyjątku kontrolowanego opakowywała go w wyjątek niekontrolowany i zgłaszała. Jeśli powstanie jakiś wyjątek niekontrolowany, to po prostu jest ponownie zgłaszany (rethrowing) bez żadnego opakowania. Przy tym ważne jest w sekwencji klauzul catch, by zapewnić najpierw obsługę RuntimeException (nie wymaga opakowania), a później dopiero Exception. W przeciwnym razie ewentualne powstające wyjątki typu Runtime-Exception byłyby niepotrzebnie pakowane w dodatkowy obiekt. Używając tego prostego rozszerzenia interfejsu Function, możemy nasz program zapisać tak:

```
static <S, T> List<T> transform(Collection<S> src,
Function<S, T> f) {
   List<T> trg = new ArrayList<>();
   for (S e : src) trg.add(f.apply(e));
   return trg;
}
static <T> T sameAfterSec(T arg) throws InterruptedE-
xception {
   Thread.sleep(1000);
   return arg;
}
public static void main(String[] args) {
    System.out.println(
       transform(Arrays.asList(1, 2, 3),
                    (CEF<Integer, Integer>) n -> n == 2 ?
sameAfterSec(n) : n+1)
    );
}
```

Operator rzutowania zapewnia, że lambda-wyrażenie będzie typu CEF, a jego kod będzie pasował do deskryptora funkcji checkedApply, która ma klauzulę throws Exception. Wobec tego nieobsłużone w lambda kontrolowane wyjątki (typu Exception) będą przekazywane do obsługi przez wywołującego. Tym wywołującym jest (w transform) domyślna metoda apply interfejsu CEF, która wyjątki kontrolowane obsługuje przez opakowanie ich w RuntimeException. Kompilator się nie skarży i wszystko działa, jak powinno, w danym przypadku produkując na wyjściu: [2, 2, 4]

MICHAEL KEELING

ZOSTAŃ ARCHITEKTEM OPROGRAMOWANIA

W poszukiwaniu wymagań istotnych dla architektury – Definiowanie atrybutów jakościowych – Dowiedzmy się, co jeszcze wpływa na architekturę

Wybór architektury (zanim ona wybierze nas) – Wspieranie pożądanych atrybutów jakościowych

KUP KSIĄŻKĘ

KUP E-BOOK

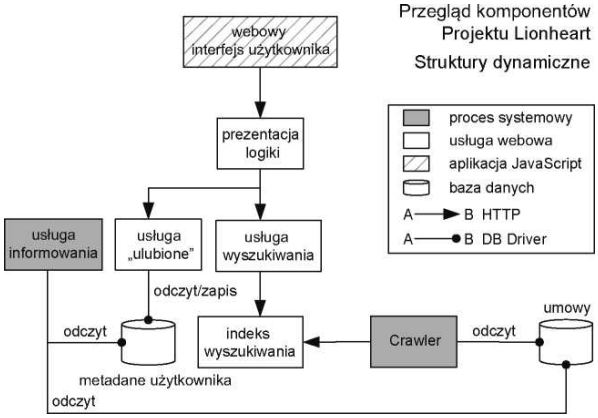
PRZYPISYWANIE ELEMENTOM FUNKCJONALNYCH OBOWIĄZKÓW

Każdy element w architekturze ma zadanie do wykonania. Wybierając struktury, przypisujemy konkretne obowiązki funkcjonalne do każdego elementu, dzięki czemu możemy osiągnąć wszystkie niezbędne wymagania funkcjonalne.

Spójrzmy na przykład z naszego studium przypadku, Projektu Lionheart. Oto niektóre wymagania funkcjonalne zebrane w wywiadzie z osobami z Biura Zarządzania i Finansów.

Użytkownik z Biura Zarządzania i Finansów może:

- Wyszukiwać istniejące i przeszłe miejskie umowy.
- Dzielić wyniki na strony.
- Przeglądać podstawowe informacje o firmie, w tym nazwę, numer telefonu, adres i listę wcześniejszych i aktywnych umów.
- Przeglądać podstawowe informacje o umowie, w tym rodzaj, status, datę zakończenia, numer identyfikacyjny, firmy licytujące i kto wygrał kontrakt.
- Zapisać się, aby otrzymywać powiadomienia o aktualizacjach umów.



Oprócz wyżej wymienionych te wymagania funkcjonalne implikują kilka dodatkowych obowiązków:

- Ponieważ użytkownicy będą mieli możliwość wyszukiwania, trzeba będzie utworzyć indeksy.
- Aby wyświetlać szczegółowe informacje o umowie i firmie, trzeba będzie je przechowywać.
- Możliwość subskrybowania będzie wymagać, aby system zapisywał adresy e-mail.
- Aby informować użytkowników o zmianach, będzie potrzebne coś, co rozpozna, że zmiana została wprowadzona.

Oto widok zestawu elementów, który pozwoli nam osiągnąć te wymagania funkcjonalne, pokazany na schemacie na stronie 82.

Katalog odpowiedzialności elementów ze wspomnianego schematu:

Element	Odpowiedzialność
Webowy interfejs użytkownika	Renderuje interfejs dla użytkownika w przeglądarce, obsługuje interakcje użytkownika.
Prezentacja logiki	Autoryzacja i uwierzytelnianie, proxy dla innych usług backendu, weryfikuje logikę biznesową dla użycia w aplikacji.
Usługa wyszukiwania	Przetwarzanie w celu analizowania zapytań, wyszukiwania, stronicowania, filtrowania.
Usługa „ulubione"	Normalizuje znaczniki, zapisuje ulubione do pamięci.
Usługa informowania	Zaplanowana, aby wyszukiwać ostatnie zmiany, wysyłać wiadomości e-mail na podstawie subskrybcji przechowywanych w bazie danych metadanych użytkownika.
Crawler	Odczytuje dane z bazy umów, przekształca je w celu wyszukiwania i dodaje do indeksu.
Baza metadanych użytkowników	Trwała pamięć dla subskrybcji i innych treści dodawanych przez użytkowników.
Indeks wyszukiwania	Zoptymalizowana, przeznaczona do przeszukiwania reprezentacja danych o umowie. Wszystkie dane umów wyświetlane w interfejsie użytkownika można przeszukiwać, sortować i przechowywać.
Baza umów	Pamięć trwała. System rekordów dla miejskich danych o zapytaniach ofertowych.
HTTP	Komunikacja między usługami za pośrednictwem standardowych protokołów HTTP. Przyjmuje się, że interfejsy API są typu RESTful.
Sterownik bazy danych	Natywny sterownik/klient dla wybranej bazy danych.

Katalog odpowiedzialności elementów opisuje podstawowe obowiązki, jakie każdy element w architekturze ma prawo wykonywać. Stworzyliśmy ten katalog, przeglądając listę znanych, wpływowych wymagań funkcjonalnych i pilnując, aby każda funkcjonalność była własnością jednego i tylko jednego elementu. Ponadto każdy element w architekturze powinien mieć co najmniej jedną funkcjonalność, za którą jest odpowiedzialny; w przeciwnym razie ten element jest zbędny.

Wpływowe wymagania funkcjonalne tworzą świetną listę kontrolną przy przypisywaniu odpowiedzialności do elementów. Jednym ze sposobów identyfikacji obowiązków jest modelowanie systemu za pomocą kart CRC, co opisano na stronie 246.

DEFINIOWANIE ATRYBUTÓW JAKOŚCIOWYCH

Atrybuty jakościowe opisują właściwości systemu widoczne na zewnątrz i oczekiwania dotyczące jego działania. Określają, jak dobrze system powinien wykonywać pewne działania. Te cechy systemu są czasami nazywane *wymaganiami jakościowymi*. Oto lista niektórych typowych atrybutów jakościowych z *Software Architecture in Practice [BCK12].*

Właściwości etapu projektowania	Właściwości środowiska wykonawczego	Właściwości konceptualne
Zdolność do modyfikowania	Dostępność	Łatwość zarządzania
Łatwość utrzymania	Niezawodność	Zgodność z wymaganiami
Możliwość ponownego użycia	Wydajność	Prostota
Testowalność	Skalowalność	Zdolność do uczenia
Konstruowalność lub czas wytworzenia	Bezpieczeństwo	

Każda decyzja architektoniczna promuje lub ogranicza co najmniej jeden atrybut jakościowy. Wiele decyzji projektowych promuje jeden zbiór atrybutów jakościowych, jednocześnie blokując inne, równie ważne! Kiedy tak się dzieje, wymieniamy jeden atrybut jakościowy na inny, wybierając strukturę architektoniczną, która faworyzuje jeden atrybut jakości, ale szkodzi innym.

Podczas poszukiwania ASR większość czasu poświęcamy na pracę z atrybutami jakościowymi. Są one używane w całym procesie projektowania, aby kierować wyborem technologii, wybierać struktury, wzorce i oceniać przydatność naszych decyzji projektowych.

V/ Pyta Joe: Czy atrybuty jakościowe są wymaganiami niefunkcjonalnymi?

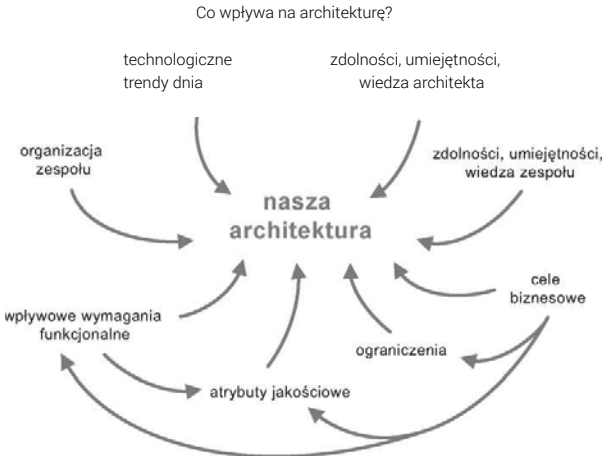
Tradycyjne podręczniki inżynierii oprogramowania zazwyczaj omawiają dwie klasy wymagań. *Wymagania funkcjonalne* opisują zachowanie systemu oprogramowania. *Wymagania niefunkcjonalne* opisują wszystkie wymagania systemowe, które nie są wymaganiami funkcjonalnymi, w tym to, co nazywamy atrybutami jakościowymi i ograniczeniami.

Podczas projektowania architektury oprogramowania przydatne jest rozróżnienie między funkcjonalnościami, ograniczeniami i atrybutami jakościowymi, ponieważ każdy typ wymagań oznacza inny zestaw sił wpływających na projekt. Na przykład ograniczenia nie podlegają negocjacjom, natomiast atrybuty jakościowe można dopracować i wymagają one znacznych kompromisów.

Tak, atrybuty jakościowe są wymaganiami niefunkcjonalnymi, ale też niewłaściwe jest używanie tego terminu do ich opisu, ponieważ scenariusze atrybutów jakościowych (czasami nazywane wymaganiamijakościowymi) zawierają pewien aspekt funkcjonalny. Atrybuty jakościowe mają sens tylko w kontekście działania systemu. W scenariuszu atrybutu jakościowego odpowiedź artefaktu jest bezpośrednim rezultatem pewnej funkcjonalności.

DOWIEDZMY SIĘ, CO JESZCZE WPŁYWA NA ARCHITEKTURĘ

Oprócz ASR istnieje wiele innych czynników, które wpływają na architekturę zarówno bezpośrednio, jak i pośrednio. Oto lista wybranych:



Nasze umiejętności i doświadczenie jako architekta określają, w jaki sposób podchodzimy do projektowania i dostępnych możliwości architektonicznych. Wiedza nasza i naszego zespołu na temat technologii określa nasz słownik projektowy. Jeśli znamy tylko Ruby on Rails, wtedy są spore szanse na zaklinowanie się w architekturze. Gdy młotek jest wszystkim, czym dysponujemy, to znajdziemy mnóstwo gwoździ, które możemy wbić.

Architektura zawsze wydaje się podążać za najnowszymi trendami technologicznymi. Wraz z pojawieniem się nowych paradygmatów sprzętowych, programowych i projektowych, niektóre z nich na stałe zmieniają krajobraz inżynierii oprogramowania. Inne mogą być jedynie marketingowym opakowaniem starych pomysłów. Jest spora szansa na to, że twoja architektura już stała się projektową starocią.

ZAINTERESOWAŁY CIĘ NASZE KSIĄŻKI?

ZNAJDZIESZ JE W:

IBUK Libra

IBUK Libra to czytelnia on-line czynna całą dobę. Dostępne w niej są tysiące e-booków oraz e-czasopism z niemal każdej dziedziny. Do IBUKA Libry możesz zalogować się z dowolnego miejsca, o każdej porze. Korzystanie z IBUKA Libry jest bezpłatne – poproś o dostęp w swojej bibliotece.

PRZEJDŹ DO IBUK LIBRA

IBUK

IBUK.pl jest platformą pozwalającą kupować i wypożyczać e-booki. Można je wypożyczać zarówno pojedynczo – już od 4,92 PLN za dobę oraz w abonamentach – ceny zaczynają się od 19,90 PLN miesięcznie. W ofercie dostępne są także audiobooki.

PRZEJDŹ DO IBUK.PL

PWN KSIĘGARNIA INTERNETOWA

Księgarnia Internetowa PWN oferuje szeroki zakres publikacji: podręczniki akademickie, książki naukowe i popularnonaukowe, słowniki języka polskiego i słowniki języków obcych. Znajdziesz w niej zarówno publikacje papierowe, jak i książki w wersji elektronicznej – e-booki i audiobooki.

PRZEJDŹ DO KSIĘGARNI INTERNETOWEJ PWN

ZNAJDŹ NAS NA PWN STEM