

Magazyn programistów i liderów zespołów IT

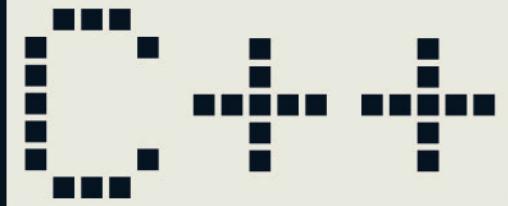


5/2018 (72)

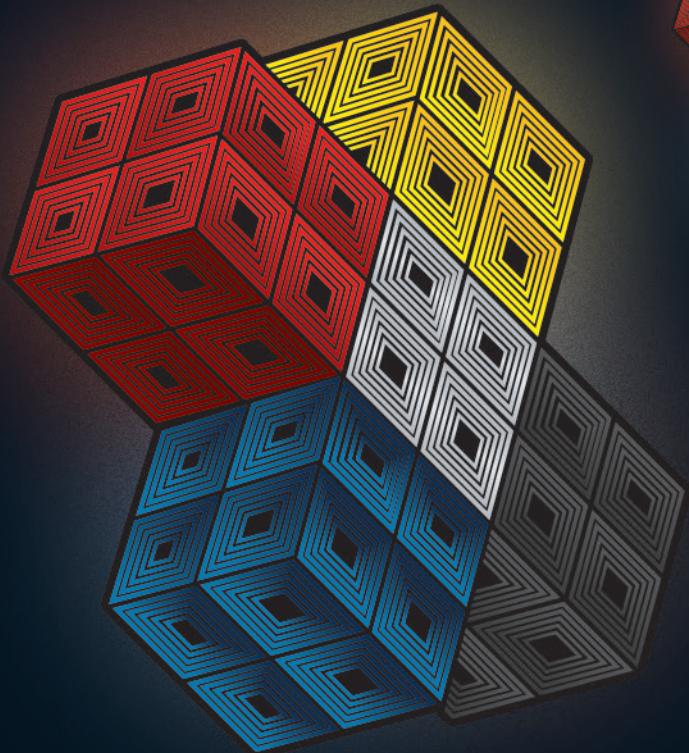
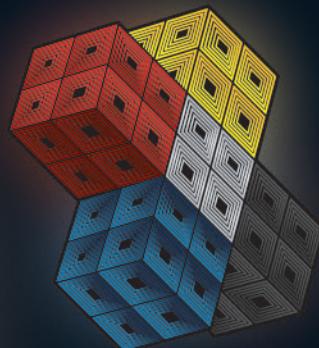
czerwiec/lipiec

Cena 23,90 zł (w tym VAT 5%)

MODULARNE



RAPORT Z POSTĘPÓW



ISSN 2084-9400

05



Zródło: AdobeStock

DPDK NA RASPBERRY PI 3

JAK ZOSTAĆ PROGRAMISTĄ

TIME TRAVEL DEBUGGING

WIELOPLATFORMOWE APLIKACJE
DESKTOPOWE



Join our team of Makers!
career.cybercom.com

Change
tomorrow
with us

www.cybercom.pl

CYBERCOM
GROUP

Cybercom Poland Sp. z o.o.

ul. Hrubieszowska 2, 01-209 Warszawa / ul. Składowa 35, 90-127 Łódź / ul. Unii Lubelskiej 4c, 85-059 Bydgoszcz

A może by...

Wakacje. Prawie wszyscy uciekamy wtedy na urlopy, relaks, wypoczynek. Temat pracy odsuwamy w dali, w końcu czas na coś innego, nowego i odciążenie zmęczonego umysłu. W związku z tym numerem przygotowaliśmy w nieco luźniejszym stylu. Wyjątkowo sporo w nim lżejszych tekstuów, choć stricte technicznych oczywiście nie pominieliśmy.

Na początek proponujemy zapoznać się z obserwem, napisanym z humorem, tekstem Wojtka Sury o tajnikach tego „Jak zostać programistą” z perspektywy własnych doświadczeń i popelnionych błędów.

Kolejny materiał, jaki w tym wydaniu przygotowaliśmy, to tekst o bardzo przewrotnym tytule „Hipsterskie metody wyboru technologii” Michała Lewandowskiego, w którym autor opisuje między innymi wpływy (często zgubne) fascynacji nowymi top trendowymi technologiami na realizację projektu.

Czas na praktykę. Lekturę warto zacząć od artykułu Marka Michalskiego pt. „Wprowadzenie do zautomatyzowanego składu tekstu w systemie LaTeX”, w którym autor opisuje podstawy pracy z jednym z ciekawszych systemów składu tekstu. W końcu dokumentację tworzymy jako programiści sami dla siebie, niech więc wygląda porządnie i będzie czytelna.

Dla tych, co lubią bawić się sieciami, eksperymentować z różnymi implementacjami stosów, analizować masowy ruch sieciowy Rafał Kozik przygotował małe wprowadzenie do framework'u ułatwiającego te zadania, w artykule pod tytułem „Malinowa sieć – DPDK na Raspberry Pi 3”.

W końcu udało się: napisał swój pierwszy większy program, który jednak co jakiś czas sypie błędami. Ciężko jest je znaleźć, zasiadasz więc do debugera i odkrywasz nową metodę sprawdzenia poprawności działania programu – cofnięcie się w czasie. Sposoby wykorzystania tej możliwości, narzędzia wspierające je przedstawił Paweł Łukasik w artykule „Time travel debugging – «nowy» sposób na stare błędy”.

Na zakończenie proponujemy okładkowy materiał naszego redakcyjnego specjalisty od języka C++ Pawła „Krzaka” Zakrzewskiego, który tym razem przygląda się tematyce modułów w tekście pt. „Moduły w C++ – raport z postępów”.

Pozostaje nam życzyć przyjemnie spędzonego urlopu, a w wolnej chwili sięgnąć po nasz magazyn.

Milej lektury!
Mariusz „maryush” Witkowski

BIBLIOTEKI I NARZĘDZIA

Time travel debugging – „nowy” sposób na stare błędy?	4
Paweł Łukasik	

Malinowa sieć – DPDK na Raspberry Pi 3	8
Rafał Kozik	

Wprowadzenie do zautomatyzowanego składu tekstu w systemie LaTeX	14
Marek Michalski	

JĘZYKI PROGRAMOWANIA

Moduły w C++ – raport z postępów	18
Paweł „Krzak” Zakrzewski	

PROGRAMOWANIE APLIKACJI DESKTOPOWYCH

Xwt. Tworzenie wieloplatformowych aplikacji desktopowych	26
Dawid Borycki	

LABORATORIUM IT KONTRAKT

Architektura testów automatycznych dla wielomodułowej aplikacji webowej	32
Piotr Grzesiak	

KLUB LIDERA IT

Współczesne architektury aplikacji biznesowych. Reactive i serverless	36
Mariusz Sieraczkiewicz	

TESTOWANIE I ZARZĄDZANIE JAKOŚCIĄ

Testowanie eksploracyjne – jak robić to dobrze?	42
Marek Walesa	

PLANETA IT

Hipsterskie metody wyboru technologii	44
Michał Lewandowski	

Jak zostać programistą	48
Wojciech Sura	

Microsoft Build 2018. Relacja z konferencji	60
Dawid Borycki	

FELIETON

Letnie nerda przemyślenia	64
Mariusz „maryush” Witkowski	

KLUB DOBREJ KSIĄŻKI

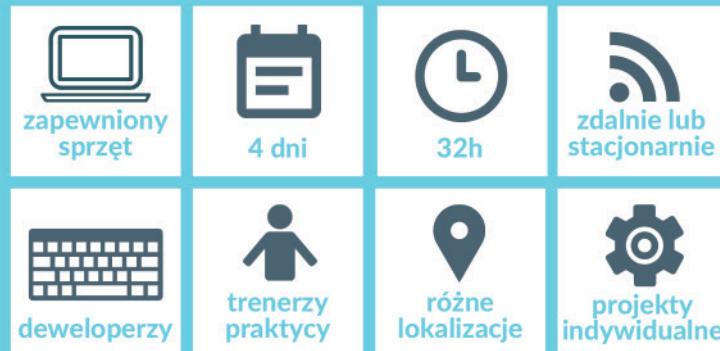
Badanie UX. Praktyczne techniki projektowania bezkonkurencyjnych produktów	68
Katarzyna Małecka	

reklama

Szkolenie dla Ciebie lub Twojego zespołu

Kubernetes w praktyce

Skorzystaj z 10% zniżki na wszystkie szkolenie otwarte z autorskiej oferty Sages ważnej przy zamówieniach złożonych do końca sierpnia 2018 r.
Hasło: PROGRAMISTAMAG



Time travel debugging – „nowy” sposób na stare błędy?

Czytelnicy „Programisty” w kilku pojawiających się na łamach magazynu artykułach mogli zapoznać się z tematem debugowania, jak również poznać ciekawe sztuczki z tym związanego. Ba! Mogli nawet samodzielnie napisać własny debugger¹. W tym artykule zapoznamy się z nie poruszanym dotychczas pojęciem, a mianowicie time travel debugging.

1. Programista 2/2014, Programista 3/2014, Programista 4/2014 oraz Programista 5/2014.

DEBUGOWANIE NIE JEDNO MA IMIĘ

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it. – Brian Kernighan.

Zapewne większość z nas spotkała się z powyższym cytatem. Niektórzy pewnie doświadczyli jego efektów na własne skórze, ślecząc wiele godzin nad błędem, który staramy się zreprodukować, przeanalizować i w końcu naprawić. Czasem z pozytywnym, a czasem negatywnym skutkiem.

Debugując program, śledzimy jego wykonanie, oglądamy je w spowolnieniu i analizujemy efekty poszczególnych akcji. Z tego staramy się wywnioskować, dlaczego nie działa on tak, jak to założyliśmy.

Dzisiejsze narzędzia dają nam duże możliwości, jeśli chodzi o interakcję z programem w trakcie takiej sesji. Choć możliwości te zależne są od poszczególnych środowisk, to jednak dla języków wysokiego poziomu możemy wymienić takie jak: możliwość zmiany kodu w trakcie (Edit & Continue znane chociażby z Visual Studio), pomijać niektóre instrukcje, a także wykonywać je ponownie, przestawiając wirtualny adres, tak aby jeszcze raz wykonać część już uruchomionego kodu. I choć to ostatnie brzmi jak *silver-bullet* rozwiązymy wszystkie nasze problemy, ma jedną podstawową wadę – efekty uboczne. Jeśli instrukcja, jedna lub wiele, którą ponawiamy, posiada jakieś efekty uboczne, to niestety nie zostaną one wycofane w momencie, gdy będziemy chcieli wykonać je raz jeszcze. Z tego powodu ponowne (i kolejne) wykonania nie do końca będą pokrywać się z tymi z pierwszego. Może mieć to kluczowe znaczenie dla części błędów, których analizy się podejmujemy.

I tutaj pojawia się time travel debugging. Z powodu możliwości cofnięcia się w wykonaniu programu czasem występujący także pod nazwą *reverse debugging*. Główną nowością tej techniki oprócz standardowych operacji jak step-into, step-over czy step-out będą ich cofające w czasie odpowiedniki. To właśnie dzięki nim bez problemu możemy cofnąć się w programie do miejsca zawierającego błąd i wielokrotnie wykonywać podejrzane instrukcje bez efektów ubocznych.

CUDA PANIE, CUDA

Manipulacja wskaźnika następnej instrukcji do wykonania – tzw. (E)IP – była dostępna już od dawnego. Jak zostało to wspomniane wyżej, ma ona jednak wadę i nie powoduje cofnięcia efektów ubocznych naszego kodu. Odpowiedzią jest tutaj właśnie time travel debug-

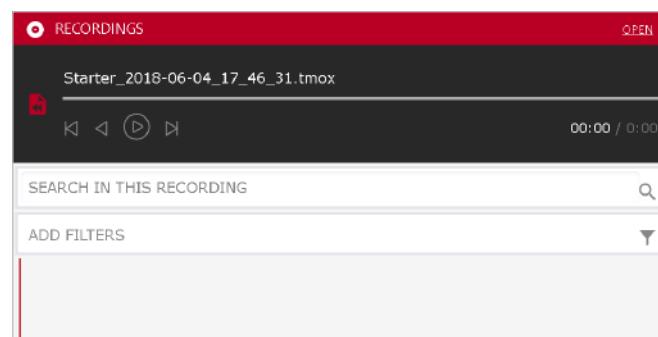
ging. Jak zatem to rozwiązanie działa, że jest w stanie przywrócić naszą aplikację, iż wydaje się, że wykonania instrukcji nigdy nie było? W zasadzie to bardzo prosto. Rozwiązania techniczne różnią się w zależności od narzędzia, które je wspiera, ale główna zasada jest taka sama. Tworzony jest log, który przechowuje wszystkie² zmiany, jakie aplikacja wykonywała podczas sesji debuggera. Log ten, w zależności od poziomu skomplikowania naszej aplikacji, będzie większy lub mniejszy, dlatego też czasem krótkie wykonanie aplikacji spowoduje utworzenie kilkugigabajtowego logu. Następnie pracujemy z takim logiem, który zawiera wszystkie niezbędne informacje, które są konieczne, aby time travel debugging miał sens. Dzięki niemu możemy poruszać się normalnie do przodu, ale także i do tyłu.

NARZĘDZIA

RevDeBug

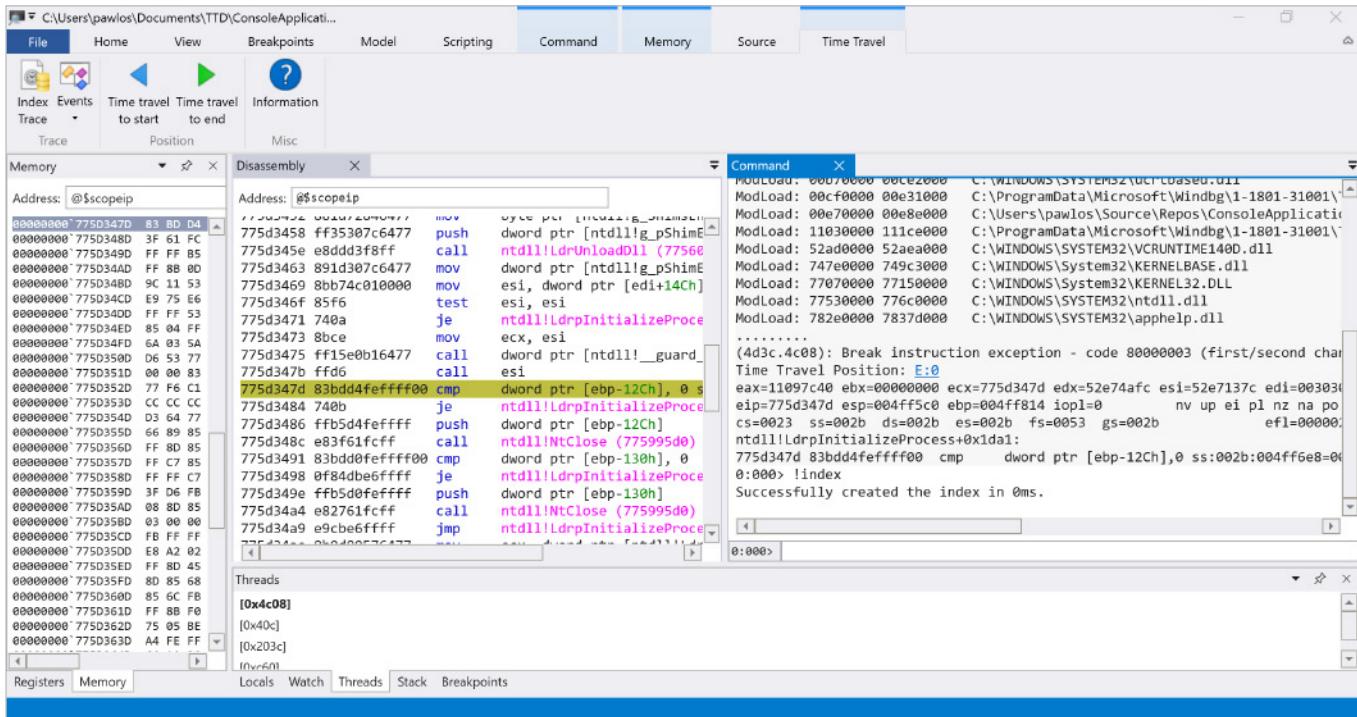
Narzędzie komercyjne dla programistów .NET. Sposób działania to instrumentacja naszego programu, a następnie zebranie logu z jego wykonania. Po zainstalowaniu tego dodatku do Visual Studio otrzymamy kilka nowych przycisków, jak również nowe menu.

Z pomocą standardowych opcji dostępnych jak dotychczas możemy uruchomić naszą aplikację, a dzięki instrumentacji po jej zatrzymaniu zostanie utworzony log z wykonania. Dzięki niemu, oraz dzięki dodatkowym przyciskom i opcjom, będziemy mogli poruszać się po naszej aplikacji w przód oraz w tył, jednocześnie sprawdzając wartości zmiennych oraz wyniki napotkanych po drodze funkcji.



Rysunek 1. Panel RevDeBug umożliwiający sterowanie odtwarzaniem nagranego procesu

2. „Wszystkie” to za dużo powiedziane. Trzymane jest tyle zmian, aby móc cofnąć się w wykonaniu instrukcji bez skutków ubocznych.



Rysunek 2. Interfejs WinDbg z otwartą zakładką Time Travel

WinDbg

Ten znany w środowisku Windows debugger niedawno przeszedł lifting UI, który pozostawał niezmieniony od dawien dawna. Wraz z tą zmianą dostał także wsparcie dla time travel debuggingu. Dodatkowe przyciski w UI pozwalają na przejście w tył oraz w przód po naszej aplikacji. TTD wprowadzony jest wraz z Windows 10 i jest dostępny w wersji WinDbg Preview, który możemy pobrać ze sklepu Microsoft Store.

Jeśli nie zaczynamy pracy z zapisaną sesją, musimy ją nagrać. Time travel debugging jest wspierany tylko w przypadku uruchomienia WinDbg z podniesionymi uprawnieniami. Wtedy dostępna będzie opcja nagrania procesu na potrzeby późniejszego TTD. Po tym możemy już przystąpić do pracy.

WinDbg do standardowych akcji step-out, step-in oraz step-over dodaje ich wsteczne odpowiedniki: Step Out Back, Step Into Back, Step Over Back. Akcje im przypisane są od-

powiednikami tych zwykłych, jednak kierunek ich działania będzie odwrotny.

Dodatkowo na nowej karcie „Time travel” mamy dostępne dwa szybkie przyciski akcji: Time travel to start oraz Time travel to end, które za pomocą jednego kliknięcia zaprowadzą nas odpowiednio na początek lub na koniec wykonania naszej aplikacji.

Oczywiście mamy też możliwość zapisania oraz wczytania logów z przebiegu wykonania aplikacji.

radare2

Ten popularny framework do reverse-engineeringu także posiada opcję debugowania wspierającą pracę w ramach time travel debugging. Samo tworzenie logu odbywa się inaczej niż w przypadku WinDbg. W momencie gdy chcemy zaznaczyć miejsce, do którego będziemy chcieli powrócić, tworzymy je za pomocą polecenia dt\$+.

reklama

devstyle.pl

ŚWIAT OKIEM PROGRAMISTY



```
[0x10000b19c]> dts+
Reading 4096 byte(s) from 0x100005000...
Reading 12288 byte(s) from 0x100055000...
Reading 217088 byte(s) from 0x100058000...
Reading 8388608 byte(s) from 0x7ffffeef40000...
Reading 1212416 byte(s) from 0x7fff874d8000...
Reading 2097152 byte(s) from 0x7fff87600000...
Reading 2097152 byte(s) from 0x7fff87800000...
Reading 2097152 byte(s) from 0x7fff87a00000...
Reading 10485760 byte(s) from 0x7fff87c00000...
Reading 2097152 byte(s) from 0x7fff88600000...
Reading 2097152 byte(s) from 0x7fff88800000...
Reading 2097152 byte(s) from 0x7fff88a00000...
Reading 2097152 byte(s) from 0x7fff88c00000...
Reading 2097152 byte(s) from 0x7fff88e00000...
Reading 2097152 byte(s) from 0x7fff89000000...
Reading 2097152 byte(s) from 0x7fff89200000...
Reading 2097152 byte(s) from 0x7fff89400000...
Reading 2097152 byte(s) from 0x7fff89600000...
Reading 2097152 byte(s) from 0x7fff89800000...
Reading 2097152 byte(s) from 0x7fff89a00000...
Reading 2097152 byte(s) from 0x7fff89c00000...
Reading 4194304 byte(s) from 0x7fff89e00000...
Reading 4194304 byte(s) from 0x7fff89e00000...
```

Rysunek 3. Wynik polecenia dts+ dodającego nowy punkt dla reverse-debugowania

Mając tak utworzony log, możemy się po nim poruszać za pomocą ds (krok w przód) oraz dsb (krok w tył).

Log z wykonania możemy także zapisać do pliku za pomocą polecenia dtst [file] oraz odczytać, korzystając z dtstf [file].

gdb

Okazuje się, że także ten, znany ze środowisk Unix/Linuksowych, debugger posiada wsparcie dla reverse-debuggingu. I to posiada już od całkiem dawna. Według dokumentacji do tego narzędzia pojawiło się ono już w 2009 roku wraz z wydaniem wersji 7.0.

Tutaj, podobnie jak w powyższych narzędziach, potrzebujemy nagrać log z wykonania naszej aplikacji. Możemy tego dokonać za pomocą target record lub samego record. Następnie kontynuujemy działanie naszej aplikacji. W momencie gdy będziemy chcieli się cofnąć do wcześniejszych instrukcji, możemy skorzystać z następujących poleceń:

- » reverse-next
- » reverse-nexti
- » reverse-step
- » reverse-stepi
- » reverse-continue
- » reverse-finish

Pierwsze dwa pozwalają przechodzić po kodzie naszej aplikacji w tył krok po kroku bez wchodzenia w wywołania podprocedur za pomocą instrukcji call. Kolejne dwie mają podobne znaczenie, z tym że jednostką jest linia programu źródłowego. reverse-continue uruchamia wykonanie programu w tył. Jeśli nie mamy żadnej odpowiednio ustawionej pułapki w kodzie, to w tył wykoną się ona tylko do momentu, w którym rozpoczęliśmy nagrywanie loga. Ostatnie polecenie pozwala na powrót tuż przed instrukcją, która jest odpowiedzialna za wywołanie funkcji, w której aktualnie się znajdujemy – czyli mówiąc potocznie, pozwoli nam wyskoczyć z funkcji, w której aktualnie się znajdujemy, ale poruszając się w tył.

```
.edb-peda$ help record
Start recording.

List of record subcommands:
record btrace -- Start branch trace recording
record delete -- Delete the rest of execution log and start recording it anew
record full -- Start full execution recording
record function-call-history -- Prints the execution history at function granularity
record goto -- Restore the program to its state at instruction number N
record instruction-history -- Print disassembled instructions stored in the execution log
record save -- Save the execution log to a file
record stop -- Stop the record/replay target

Type "help record" followed by record subcommand name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
```

Rysunek 4. Pomoc dla polecenia record, dzięki któremu możemy rozpocząć, skasować lub też zapisać sesję wstecznego debugowania

Inne

Choć wymienione powyżej narzędzia są głównymi, z których przyszło mi korzystać, jeśli chodzi o time travel debugging, to jednak szybkie wyszukanie frazy pokazuje, że są także dostępne rozwiązania dla innych języków, jak na przykład Chronon, RevPDB, ElmTTD odpowiednio dla języków Java, Python oraz Elm. I choć nie miałem możliwości pracy z nimi, to jest duża szansa, że działają one na podobnej zasadzie, co te wymienione powyżej, tak więc w momencie gdy poznamy zasadę działania oraz to, jak pracować z time travel debuggerem, to najprawdopodobniej odnajdziemy się także w pracy z innym tego typu narzędziem.

ROZWIĄZANIE IDEALNE?

Jedną z korzyści time travel debuggingu jest to, że taki log możemy przesyłać do naszego bardziej uzdolnionego kolegi lub koleżanki i to oni będą się martwić będą mogli pracować na nim, wspomagając nas w odkryciu źródła problemu i finalnie je naprawić. Minusem, co zostało już też wcześniej wspomniane, może być fakt, że plik taki może zająć sporo miejsca w zależności od skomplikowania naszej aplikacji. Dodatkowo ze względu na fakt, że w niektórych narzędziach wsparcie dla time travel debuggingu jest dopiero rozwijane, nie wszystko może działać jak należy i mogą pojawiać się drobne problemy.

PODSUMOWANIE

Poprawianie błędów jest trudnym i często mozołnym procesem. Poprawianie skomplikowanych błędów jest jeszcze bardziej mozołne i uciążliwe. I nawet jeśli w cytacie Kernighana z początku artykułu jest trochę prawdy, to narzędzia, które mamy dziś do dyspozycji, są w stanie, przynajmniej częściowo, tę różnicę zniwelować. A każde narzędzie, które ułatwia nam zadanie usuwania błędów z naszych produktów, jest na wagę złota.

If debugging is the process of removing software bugs, then programming must be the process of putting them in. – Edsger Dijkstra



PAWEŁ ŁUKASIK

biuro@octal.pl

Programista .NET z 15-letnim doświadczeniem. Ostatnimi czasy bardziej niż programowaniem zainteresowany tematyką security/infosec. Fan zadań CTF oraz narzędzi radare2. Swoje zmagania z tymi dwoma opisuje na blogu – <http://ctfs.ghost.io>. Współprowadzący podcast Ostrapila – <http://ostrapila.pl>.



.NET

DEVELOPER DAYS

Pre-cons
17 września 2018

Konferencja
18-19 września 2018

5 EDYCJA
NAJWIĘKSZEJ
KONFERENCJI

DEDYKOWANEJ PLATFORMIE .NET!

Wśród prelegentów
między innymi:

Scott Hunter:
Director of
Program
Management
at Microsoft

Tim Huckaby:
Executive
Chairman
and Founder of
InterKnowlogy

Gill Cleeren
Visual studio
and development
technologies MVP

Pre-cons:

Neal Ford:
Building
Evolutionary
Architectures

Shawn Wildermuth:
Writing and Securing
APIs with ASP.NET
Core 2

Sasha Goldshtein:
Making .NET
Applications
Faster

Daniel Marbach:
Async/Await and
the Task Parallel
Library

Warszawa | EXPO XXI | net.developerdays.pl

Partner
strategiczny

KMD
KMDPOLAND.PL

Partnerzy
złoci:

Fast Reports
Reporting must be Fast!

Relativity

Partner
srebrny:

ASML
Be part of progress

Malinowa sieć – DPDK na Raspberry Pi 3

Utarło się, iż za przetwarzanie pakietów odpowiedzialny jest stos sieciowy będący częścią systemu operacyjnego. Jednak istnieje też alternatywne rozwiązanie, polegające na obsłudze ruchu w przestrzeni użytkownika. Jednym z narzędzi służącym do tego celu jest framework DPDK. W artykule postaram się przedstawić sposób uruchomienia demonstracyjnej aplikacji test-pmd i generatora pakietów pktgen-dpdk oraz tworzenie własnego programu realizującego dekapsulację tunelu IP.

CO TO JEST DPDK?

Przy prędkości transmisji 10 Gb/s liczba pakietów, które muszą być obsłużone w czasie jednej sekundy, może przekroczyć 14 milionów. Oznacza to, że na obsłuszenie jednej ramki procesor dysponuje przedziałem niecałe 70 ns. Stos sieciowy Linuxa nie jest w stanie nadążyć z obsługą takiego ruchu, ponieważ samo wywołanie przerwania zajmuje aż 50 ns [1].

Framework DPDK (Data Plain Development Kit) został stworzony przez firmę Intel, aby rozwiązać ten problem. Dzięki rezygnacji z funkcji sieciowych dostarczanych przez system operacyjny, wcześniejszej alokacji pamięci oraz obsłudze karty sieciowej w trybie odpytywania (ang. *polling*), a także kilku innym sztuczkom pozwala na znaczne przyśpieszenie przetwarzania pakietów [1]. Umożliwia on zastąpienie wyspecjalizowanego routera albo switcha komputerem wyposażonym w odpowiednio szybką kartę sieciową.

Nie jest to jednak stos sieciowy. Jego główną funkcjonalnością jest możliwość analizy pakietów na poziomie bitów. Na podstawie znajdowanych w nich wzorców może realizować inteligentne przekazywanie pakietów, albo poprzez przepisanie lub dodanie nagłówka obsługiwac tunelu [2].

Niestety wspierane są głównie karty sieciowe obsługujące prędkości 10 Gb/s i wyższe. Taki sprzęt jest stosunkowo drogi. W artykule przedstawiono, w jaki sposób można skompilować oraz przeprowadzić proste eksperymenty na platformie Raspberry Pi 3 przy wykorzystaniu wirtualnych interfejsów sieciowych tap. Przedstawione eksperymenty nie pokażą maksymalnej wydajności DPDK, pozwolą jednak zapoznać się z zasadą jego działania. Uruchomienie ich na sprzęcie wyposażonym w szybszą kartę sieciową jest w dużej części analogiczne.

ZACZYNAMY

Eksperymenty zostały przeprowadzone przy wykorzystaniu dystrybucji Raspbian Stretch Lite w wersji z dnia 18.04.2018. Kod źródłowy DPDK możemy pobrać z repozytorium:

```
git clone git://dpdk.org/dpdk
```

Jest on udostępniony na licencji BSD, dzięki czemu możemy z nim zrobić praktycznie to, co chcemy. Wersja developerska znajduje się na branchu master. Jest ona bardzo dynamicznie rozwijana, dlatego lepiej skorzystać z którejś z dostępnych edycji. W momencie powstawania tego artykułu najnowszą była 18.05. Możemy ją wybrać, przechodząc do tagu:

```
cd dpdk
git checkout v18.05
```

W repozytorium znajduje się skrypt, który ułatwia przeprowadzenie komplikacji. Uruchamiamy:

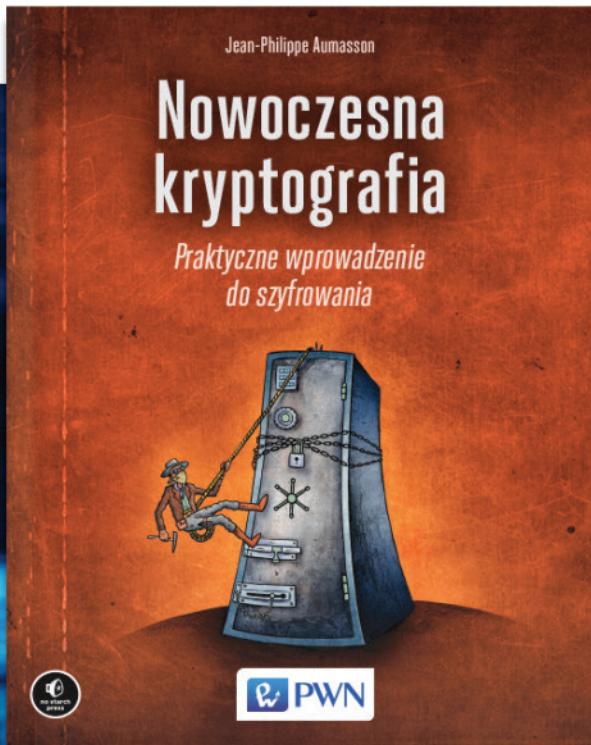
```
user/tools/dpdk-setup.sh
```

Pojawi się lista możliwych opcji pokazana w Listingu 1.

Listing 1. Lista opcji udostępnianych przez skrypt dpdk-setup.sh

```
RTE_SDK exported as /home/pi/dpdk
-----
Step 1: Select the DPDK environment to build
[1] arm64-armv8a-linuxapp-clang
[2] arm64-armv8a-linuxapp-gcc
[3] arm64-dpaa2-linuxapp-gcc
[4] arm64-dpaa-linuxapp-gcc
[5] arm64-stingray-linuxapp-gcc
[6] arm64-thunderx-linuxapp-gcc
[7] arm64-xgene1-linuxapp-gcc
[8] arm-armv7a-linuxapp-gcc
[9] i686-native-linuxapp-gcc
[10] i686-native-linuxapp-icc
[11] ppc_64-power8-linuxapp-gcc
[12] x86_64-native-bsdapp-clang
[13] x86_64-native-bsdapp-gcc
[14] x86_64-native-linuxapp-clang
[15] x86_64-native-linuxapp-gcc
[16] x86_64-native-linuxapp-icc
[17] x86_x32-native-linuxapp-gcc
-----
Step 2: Setup linuxapp environment
[18] Insert IGB UIO module
[19] Insert VFIO module
[20] Insert KNI module
[21] Setup hugepage mappings for non-NUMA systems
[22] Setup hugepage mappings for NUMA systems
[23] Display current Ethernet/Crypto device settings
[24] Bind Ethernet/Crypto device to IGB UIO module
[25] Bind Ethernet/Crypto device to VFIO module
[26] Setup VFIO permissions
-----
Step 3: Run test application for linuxapp environment
[27] Run test application ($RTE_TARGET/app/test)
[28] Run testpmd application in interactive mode ($RTE_TARGET/app/testpmd)
-----
Step 4: Other tools
[29] List hugepage info from /proc/meminfo
-----
Step 5: Uninstall and system cleanup
[30] Unbind devices from IGB UIO or VFIO driver
[31] Remove IGB UIO module
[32] Remove VFIO module
[33] Remove KNI module
[34] Remove hugepage mappings
[35] Exit Script
Option:
```

Najlepsze od



Praktyczny
przewodnik!



ODWIEDŹ NAS NA



IT.PWN.PL i zapisz się na newsletter!



KSIEGARNIA.PWN.PL

Nas interesuje pozycja:

```
[8] arm-armv7a-linuxapp-gcc
```

Pomimo że RaspberryPi 3 jest wyposażony w procesor 64-bitowy, to jednak musimy wybrać opcję dla starszej, 32-bitowej architektury. Zmusza nas do tego wykorzystanie 32-bitowego systemu operacyjnego. Ze względu na to, że DPDK jest stosunkowo dużym frameworkiem, jego komplikacja zajmuje około 20 minut. Po jej zakończeniu wyłączamy skrypt konfiguracyjny, wybierając opcję 35.

PRZYKŁADOWA APLIKACJA

Mamy już skompilowany framework. Jednak aby sprawdzić jego możliwości, potrzebne są korzystające z niego aplikacje. Na szczęście w pakiecie dostarczonych jest kilka przykładowych. Wykorzystamy jedną z bardziej rozbudowanych: testpmd. Do uruchomienia programów wykorzystujących DPDK niezbędne są uprawnienia administratora. Uruchomienie aplikacji wymaga podania kilku parametrów:

```
cd arm-armv7a-linuxapp-gcc/app/  
sudo ./testpmd --no-huge -l 0,1 -m 512 --vdev=net_tap0 -- -i  
--total-num-mbufs=2048
```

Lista parametrów składa się z dwóch grup rozdzielonych podwójnym myślnikiem „--”. Pierwsza część jest przekazywana do EAL (Environment Abstraction Layer), natomiast pozostałe są przekazywane do aplikacji.

Pierwsza opcja --no-huge oznacza, że nie będą wykorzystane duże strony pamięci. Pozwalają one na bardziej efektywne korzystanie z pamięci operacyjnej oraz pamięci podrzędnej rdzenia. Jednakże dla uproszczenia pominiemy ich konfigurację. Parametr -l pozwala zdefiniować listę rdzeni, z których może skorzystać DPDK. Ponieważ nie będą one wykorzystywane przez system operacyjny, nie możemy przekazać wszystkich. W przykładzie zdefiniowano dwa: 0 i 1 z czterech dostępnych. -m pozwala skonfigurować liczbę MB pamięci, która zostanie zaalokowana przez framework. Ponieważ RP3 jest wyposażony w 1 GB, zdecydowano się przeznaczyć połowę. Ostatni argument --vdev umożliwia stworzenie wirtualnego interfejsu sieciowego tap.

Do aplikacji przekazane zostaną dwa parametry. -i oznacza, że będzie ona pracowała w trybie interaktywnym, czyli komendy będą mogły być wprowadzane z linii poleceń. Drugi parametr oznacza liczbę przygotowanych buforów, czyli struktur, w których przechowywane są pakiety.

W konsoli powinien wyświetlić się prompt aplikacji:

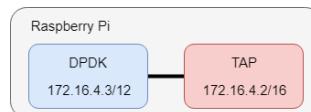
```
testpmd>
```

Powinien się także pojawić nowy interfejs sieciowy. Możemy to sprawdzić, wpisując w drugim oknie terminala:

```
$ ip address show dev dtap0  
21: dtap0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500  
qdisc multiq state UNKNOWN group default qlen 1000  
link/ether 0a:ae:7e:4e:05:54 brd ff:ff:ff:ff:ff:ff  
inet6 fe80::8ae:7eff:fe4e:554/64 scope link  
valid_lft forever preferred_lft forever
```

Możemy ustawić jego adres IP, zwracając uwagę, żeby wybrana podsieć nie była wykorzystana w lokalnej sieci. Przykładowa komenda może mieć postać:

```
$ sudo ip address add 172.16.4.2/12 dev dtap0
```



Rysunek 1. Topologia sieci

Topografię naszej sieci przedstawiono na Rysunku 1. Tak naprawdę w DPDK nie zostanie ustawiony żaden adres IP, ponieważ skorzystamy z trybu icmp echo, który po odebraniu pakietu zamienia miejscami adresy MAC i IP i odsyła go z powrotem:

```
testpmd> set fwd icmp echo  
Set icmp echo packet forwarding mode
```

Można także włączyć tryb „gadatliwy”, co spowoduje wyświetlenie przetworzonych pakietów:

```
testpmd> set verbose 1  
Change verbose level from 0 to 1
```

Następnie możemy włączyć przetwarzanie:

```
testpmd> start
```

Teraz można sprawdzić, czy DPDK odpowiada na pakiet ICMP request, pingując adres należący do skonfigurowanej podsieci:

```
ping 172.16.4.3
```

Na konsoli testpmd powinny pojawić się odebrane pakiety:

```
Port 0 pkt-len=98 nb-segs=1  
ETH: src=0A:AE:7E:4E:05:54 dst=0A:AE:7E:4E:05:54 type=0x0800  
IPV4: src=172.16.4.2 dst=172.16.4.3 proto=1 (ICMP)  
ICMP: echo request seq id=1
```

Natomiast system operacyjny powinien odebrać odpowiedź:

```
64 bytes from 172.16.4.3: icmp_seq=1 ttl=64 time=0.385 ms
```

Przekazywanie pakietów można wyłączyć, wpisując:

```
testpmd> stop
```

Na ekranie zostaną wyświetlone statystyki z liczbą odebranych i wysłanych pakietów:

```
Telling cores to stop...  
Waiting for lcores to finish...  
----- Forward statistics for port 0 -----  
RX-packets: 6 RX-dropped: 0 RX-total: 6  
TX-packets: 4 TX-dropped: 0 TX-total: 4  
-----
```

Natomiast polecenie ping przestanie otrzymywać odpowiedzi.

GENERATOR PAKIETÓW

W oparciu o DPDK jest tworzony generator pakietów. Jego kod źródłowy można pobrać z repozytorium:

```
cd ~  
git clone git://dpdk.org/apps/pktgen-dpdk
```

Przy opisywanych eksperymentach wykorzystana była wersja 3.5.1:

```
cd pktgen-dpdk
git checkout pktgen-3.5.1
```

Do komplikacji niezbędna jest biblioteka:

```
sudo apt-get install libpcap-dev -y
```

oraz zdefiniowanie dwóch zmiennych środowiskowych:

```
export RTE_SDK=/home/pi/dpdk
export RTE_TARGET=arm-armv7a-linuxapp-gcc
```

Pierwsza przechowuje lokalizację, pod którą znajduje się DPDK, a druga określa, z jakiej architektury korzystamy.

Ponieważ komplujemy na platformie 64-bitowej, zmianie uległ rozmiar niektórych zmiennych. Spowodowało to pojawienie się błędów komplikacji przy funkcji printf. Niezbędne jest dodanie patcha, którego można pobrać i zaaplikować:

```
wget goo.gl/enK33V -O patch
git apply patch
```

Kompilację rozpoczynamy poleciem:

```
make
```

Gdy zakończy się ona powodzeniem, można uruchomić generator poleciem:

```
sudo app/arm-armv7a-linuxapp-gcc/pktgen --no-huge -l 0,1,2 -m 512 --vdev=net_tap0 -- -P -m "[1:2].0"
```

Podobnie jak w przypadku testpmd, tutaj także argumenty dzielą się na przekazywane do EAL oraz do aplikacji. Te pierwsze są takie same jak w poprzednim przykładzie. Zwiększyła się jednak liczba użytych rdzeni. Zerowy jest wykorzystywany do obsługi interfejsu użytkownika, a pozostałe do odbierania i wysyłania pakietów. Za pomocą parametru -m przypisujemy, które porty mają być obsługiwane przez które procesory. W tym przypadku rdzeń 1 obsługuje ruch przychodzący, a rdzeń 2 wychodzący z portu 0. -P oznacza, że porty będą pracować w trybie nasłuchu (ang. *promiscuous mode*).

Aby rozpoczęć nadawanie, po znaku zachęty wpisujemy komendę:

```
Pktgen:/> start all
```

Powinny wtedy wzrosnąć statystyki:

Pkts/s	Max/Rx	:	12/0		12/0
Max/Tx	:	265984/265856		265984/265856	
MBits/s	Rx/Tx	:	0/178		0/178

Możemy także wyświetlić liczbę pakietów odbieranych po stronie Linuxa na przykład za pomocą programu bmon:

Interfaces	RX bps	pps
dtap0	15.15MiB	264.69K

Jak wynika z powyższych statystyk, liczba pakietów wysłanych przez DPDK i odebranych przez system Linux są równe. Warto także zwrócić uwagę na wykorzystanie procesora:

PID	USER	PR	NI	VIRT	RES	SHR	S %CPU	%MEM	TIME+	COMMAND
25490	root	20	0	581736	148812	9388	R 300.0	15.7	11:37.77	pktgen

Ponieważ DPDK działa w trybie odpytywania, przejęte przez niego rdzenie są wykorzystane w 100% i niedostępne dla pozostałych procesów działających w systemie Linux. Zakończenie nadawania umożliwia polecenie:

```
Pktgen:/> stop all
```

Można także zmienić rozmiar pakietu:

```
Pktgen:/> set all size 1518
```

Zwiększenie rozmiaru pozwoliło na osiągnięcie prędkości 2400 Mb/s, co odpowiada około 200 tysiącom pakietów na sekundę. Zamknięcie aplikacji umożliwia polecenie:

```
Pktgen:/> quit
~/pktgen-dpdk $
```

WŁASNA APLIKACJA

Po eksperymentach z przykładowymi programami na zakończenie spróbujmy stworzyć własny. Nie będziemy jednak zaczynać całkiem od początku, ale skorzystamy ze szkieletu aplikacji dostarczonego razem z frameworkiem. Znajduje się on w folderze examples/skeleton. Realizuje on most, kopując pakiety pomiędzy dwoma portami.

Składają się na niego 3 pliki. Dzięki meson.build przykłady są komplilowane razem z frameworkiem. My jednak do komplikacji będziemy korzystali z konfiguracji zawartej w pliku Makefile. Kod źródłowy programu znajduje się w pliku basicfwd.c. Jego nazwa została zmieniona na tunnel.c. Kod źródłowy programu można pobrać z repozytorium:

```
git clone https://gitlab.com/kozik/DPDK-decapsulate.git
cd DPDK-decapsulate/
```

W funkcji main następuje inicjalizacja DPDK poprzez wywołanie rte_eal_init, alokacja pamięci oraz konfiguracja portów. Samo przetwarzanie pakietów odbywa się w funkcji lcore_main. Najpierw są one odbierane za pomocą funkcji rte_eth_rx_burst i umieszczane w tablicy bufs. Następnie za pomocą funkcji rte_eth_tx_burst są wysyłane drugim portem. Pomiędzy tymi dwoma funkcjami możemy umieścić kod odpowiedzialny za przetwarzanie pakietów.

Listing 2. Funkcja realizująca dekapsulację pakietów

```
static inline void
decapsulate(struct rte_mbuf *mbuf[], uint16_t n)
{
    struct ether_hdr *eth;
    struct ipv4_hdr *ip;
    uint8_t *data;
    uint16_t i;
    int16_t j;
    uint8_t ip_hdr_size;

    for (i = 0; i < n; i++) {
        eth = rte_pktmbuf_mtod(
            mbuf[i], struct ether_hdr*);
        if (rte_be_to_cpu_16(eth->ether_type) != ETHER_TYPE_IPV4)
            continue;
        ip = (struct ipv4_hdr*) &eth[1];
        if (ip->next_proto_id != 0x04)
            continue;
```

BIBLIOTEKI I NARZĘDZIA

```
ip_hdr_size = (ip->version_ihl &
    IPV4_HDR_IHL_MASK);
ip_hdr_size *= IPV4_IHL_MULTIPLIER;

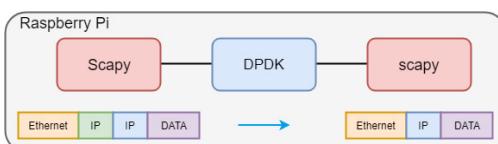
data = rte_pktmbuf_mtod(mbuf[i], uint8_t*);
for(j = sizeof(struct ether_hdr) - 1;
    j >= 0; j--)
    data[j + ip_hdr_size] = data[j];

if(rte_pktmbuf_adj(mbuf[i], ip_hdr_size)
    == NULL)
    printf("Mbuf adj error\n");
}
```

oktety	0	1	2	3
0		dst MAC		
4	dst MAC	src MAC		
8	src MAC			
12	ether type	4 IHL ToS		
16	Length	ID		
20	Flags/Offset	TTL	Prot.	
24	Checksum	src IP		
28	src IP	dst IP		
32	dst IP	4 IHL ToS		
36	Length	ID		
40	Flags/Offset	TTL	Prot.	
44	Checksum	src IP		
48	src IP	dst IP		
52	dst IP	DATA...		
56				
60				
64				

Rysunek 2. Nagłówki sieciowe w tunelu IP

Naszym celem jest dekapsulacja pakietów z tunelu IP. Nagłówki sieciowe takiego pakietu przedstawione są na Rysunku 2, a działanie aplikacji zaprezentowano na Rysunku 3. Polega ono na sprawdzeniu, czy pakiet zawiera nagłówek IP z następnym protokołem ustawionym także na IP, czyli wartość 0x4. Jeżeli tak, zostanie on usunięty. Odpowiada za to funkcja `decapsulate`, jej kod przedstawiono w Listingu 2.



Rysunek 3. Działanie aplikacji dekapsulującej

Przetwarza ona w pętli wszystkie odebrane pakiety. Najpierw następuje rzutowanie danych zawartych w strukturze `mbuf` na strukturę reprezentującą nagłówki Ethernet. Dzięki temu możemy w intuicyjny sposób dostać się do poszczególnych jego pól. Sprawdzamy, czy wartość pola `ether_type` jest równa 4, co oznacza, że następny nagłówek jest typu IP. Ponieważ liczby w protokołach sieciowych są zapisywane w notacji *big-endian*, a nasza platforma działa w trybie *little-endian*, musimy skorzystać z odpowiedniej funkcji konwertującej: `rte_be_to_cpu_16`. Gdy wiemy, że pakiet zawiera nagłówek IP, możemy adres znajdujący się za nagłówkiem Ethernet rzutować na strukturę go reprezentującą. Dzięki temu można znaleźć pole `next_proto_id` i sprawdzić, czy jego wartość jest równa 4, co oznacza, że następny jest także nagłówek IP. Ponieważ to tylko jeden bajt, tym razem nie ma problemu z kolejnością.

Ostatnim etapem jest wycięcie nagłówka IP reprezentującego tunel. Jego długość jest odczytywana z pola `IHL`. Przechowuje ono liczbę 32-bitowych słów, z których składa się nagłówek, dlatego musi ona zostać pomnożona przez stałą `IPV4_IHL_MULTIPLIER` równą 4. Następnie nagłówek Ethernet jest kopowany, ponieważ `mbuf` można skracać tylko o początkowe lub końcowe bajty. Na końcu następuje obcięcie początku pakietu za pomocą funkcji `rte_pktmbuf_adj`.

Aby skompilować aplikację, musimy zdefiniować zmienne `RTE_SDK` i `RTE_TARGET`, analogicznie jak w przypadku programu

`pktgen-dpdk`. Następnie wywołujemy polecenie:

`make`

Jeżeli komplikacja przebiegnie bez błędów, można uruchomić aplikację:

```
sudo build/tunnel --no-huge -l 0 -m 512 --vdev=net_tap0
--vdev=net_tap1
```

Do testów można wykorzystać program `scapy`. Należy otworzyć go dwa razy. W pierwszym oknie można wypisywać pakiety odebrane na interfejsie `dtap1`:

```
sniff(prn=(lambda x: x.summary()), iface="dtap1")
```

A za pomocą drugiego można wysyłać pakiety. Jeżeli wyślemy prosty pakiet IP:

```
sendp( Ether()/IP(src="172.16.4.1", dst="172.16.4.3"), iface="dtap0")
```

to powinien on pojawić się w niezmienionej postaci na drugim porcie:

```
Ether / 172.16.4.1 > 172.16.4.3 hopopt
```

Natomiast jeżeli wyślemy pakiet należący do tunelu:

```
sendp( Ether()/IP(src="172.16.4.1", dst="172.16.4.3")/
IP(src="2.2.2.2", dst="3.3.3.3"), iface="dtap0")
```

to na drugi port dotrze on zdekapsulowany:

```
Ether / 2.2.2.2 > 3.3.3.3 hopopt
```

Enkapsulację pakietów można stworzyć w podobny sposób, korzystając z funkcji `rte_pktmbuf_prepend`, która pozwala dodać przestrzeń na początek `mbufa`.

Zachęcam do własnych eksperymentów z frameworkiem DPDK. Choć pełnię swoich możliwości pokazuje on dopiero przy prędkościach rzędu dziesiątek gigabitów na sekundę, to jednak aby zapoznać się z oferowanymi przez niego funkcjonalnościami, wystarczy niewielka platforma oraz wirtualne interfejsy sieciowe. Dalsze informacje można znaleźć w dokumentacji na stronie projektu [3]. Warto także zapoznać się z listą projektów wykorzystujących DPDK [4]. Znajdziemy tam między innymi stosy sieciowe, generatory pakietów oraz implementację switchy i routerów.

Bibliografia:

- [1] Czekaj M., 10 milionów pakietów na sekundę – poznaj DPDK, „Programista” 2014, nr 11, s. 56–61
- [2] Sujata T. Data Plane Development Kit: Get Started <http://software.intel.com/en-us/articles/data-plane-development-kit-dpdk-getting-started>
- [3] http://dpdk.org/doc/guides/prog_guide/
- [4] <http://dpdk.org/about/ecosystem#associate>

RAFAŁ KOZIK

rko@semihalf.com



Programista systemów wbudowanych w kra-kowskiej firmie Semihalf. Absolwent Automatyki i Robotyki na Akademii Górnictwo-Hutniczej. Zajmuje się systemem operacyjnym FreeBSD oraz frameworkiem DPDK.

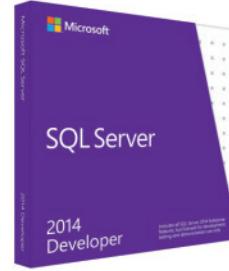
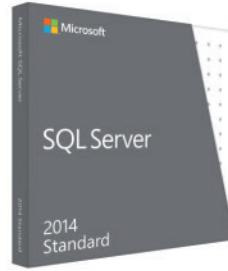
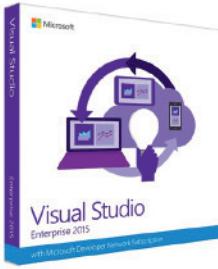
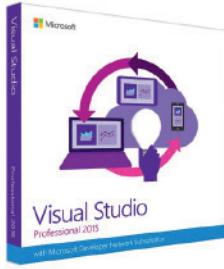
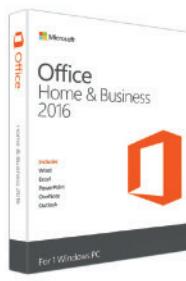
TTS Company rekomenduje oprogramowanie Microsoft ®



Microsoft Partner



Gold Datacenter
Gold Small and Midmarket Cloud Solutions
Silver Data Platform
Silver Data Analytics
Silver Cloud Productivity



www.OprogramowanieKomputerowe.pl

Microsoft Azure

Office 365

OneDrive

Więcej informacji: ☎ (22) 868 40 42 ✉ sales@tts.com.pl

Sprzedaż



Dystrybucja



Import na zamówienie

Wprowadzenie do zautomatyzowanego składu tekstu w systemie LaTeX

Prace związane z procesem produkcji i testowania oprogramowania obejmują również tworzenie różnego rodzaju dokumentów tekstowych. Powstają instrukcje obsługi, specyfikacje wymagań, raporty, procedury będące częścią systemu zarządzania jakością w danej firmie itp. W artykule zaprezentowano narzędzie do zautomatyzowanego składu tekstu LaTeX, które stanowi wartą zainteresowania alternatywę dla popularnych edytorów typu MS Word czy OpenOffice.

TROCHĘ HISTORII

Historia LaTeX-a sięga lat 70. ubiegłego stulecia. W 1977 r. prof. Donald Ervin Knuth (Uniwersytet Stanforda), niezadowolony z wyglądu pierwszych tomów swojej książki „The Art of Computer Programming”, rozpoczął pracę nad własnym systemem zautomatyzowanego składu tekstu TeX (czyt. *tech*). Nazwa jest złożeniem greckich liter *tau*, *epsilon* oraz *chi* i nawiązuje do słów oznaczających sztukę, rzemiosło oraz technikę. Prace nad oprogramowaniem początkowo szacowano na pół roku, jednak ostatecznie trwały one ponad 8 lat. Następujące po sobie wydania, począwszy od wersji 3, oznaczane są kolejnymi przybliżeniami liczby π . Najnowsza (2014 r.) posiada numer 3.14159265. TeX jest przykładem dobrze przemyślanego, wieloplatformowego systemu, którego siła tkwi w zastosowanych algorytmach i skrupulatnym wykonaniu.

W 1983 r. amerykański informatyk Leslie Lamport wydał pierwszą wersję programu LaTeX (czyt. *latech*), będącego zestawem makr rozszerzających system TeX, a co za tym idzie znaczco poprawiających efektywność pracy.

Narzędzie wypełniło lukę rynkową, szybko zyskało na popularności i stało się standardem głównie w obszarze dokumentów technicznych oraz prac naukowych. Aktualna wersja LaTeX-a nosi oznaczenie LaTeXe. Trwają przygotowania do kolejnej. Pomimo że nie pracuje się już bezpośrednio w czystym TeX-u, w powszechnie używanej terminologii określenia LaTeX oraz TeX funkcjonują zamiennie.

CHARAKTERYSTYKA SYSTEMU

LaTeX jest systemem typu WYSIWYM (ang. *what you see is what you mean*). W przeciwieństwie do popularnych edytorów tekstów (MS Word, OpenOffice itp.), będących oprogramowaniem rodzaju WYSIWYG (ang. *what you see is what you get*), efekt końcowy nie jest widoczny od razu. Autor pracuje na pliku źródłowym, w którym za pomocą odpowiednich znaczników wstawia treść oraz definiuje strukturę dokumentu. Pod tym względem LaTeX w wielu aspektach przypomina język HTML. Uzyskanie końcowego rezultatu (np. pliku PDF) wymaga przetworzenia (kompilacji) źródła. System automatycznie wykoná skład wg zadanych reguł. Zadba o poprawne rozmieszczenie tekstu, nada numery stron, rozdziałów, rysunków i tabel. Jeżeli potrzeba, utworzy spis treści, bibliografię itp. W wyniku powyższego otrzymuje się estetyczny i profesjonalny dokument, który nie wymaga ręcznych poprawek wyglądu. Autor może skupić się przede wszystkim na treści.



Donald Ervin Knuth (źródło: https://pl.wikipedia.org/wiki/Donald_Knuth)

LATEX

Funkcjonalność LaTeX-a jest rozszerzana przez dołączanie pakietów, które pozwalają m.in. na wstawianie grafik, wzorów matematycznych i chemicznych, schematów elektronicznych, grafów, dynamicznie tworzonych wykresów oraz kodów kreskowych, źródeł programów komputerowych z kolorowaniem składni itp.

Narzędzie doskonale sprawdza się w przypadku obszerniejzych tekstów. Źródło jednego dokumentu można podzielić na wiele plików (np. rozdziałów). Dzięki pełnej automatyczce składu ułatwione są również późniejsze modyfikacje. Jeżeli np. wewnątrz istniejącego tekstu zajdzie konieczność dostawienia kolejnego rozdziału, nie będzie potrzeby późniejszej żmudnej, manualnej zmiany numeracji już istniejących rozdziałów oraz spisu treści. Ten proces przebiega samoczynnie w momencie kompilacji źródła. Dla porównania efektywne tworzenie i modyfikowanie dokumentów powyżej kilkudziesięciu stron w MS Word staje się trudne, a gdy ilość stron wzrośnie do kilkuset – już prawie niemożliwe. Każdy, kto pisał teksty w programach typu WYSIWYG, zdaje sobie sprawę, jakie wyzwania stwarza późniejszy poprawny skład dokumentu, np. zapanowanie nad podpisami pod grafikami, które zostały

przesunięte na początek kolejnej strony. Automatyka dostępna w tego rodzaju aplikacjach jest przeważnie dość uboga.

LaTeX, pomimo swoich zalet i problemów, które rozwiązuje, nie będzie narzędziem odpowiednim dla każdego. Po pierwsze, sposób pracy istotnie różni się od tego znanego z popularnych edytorów. Próg wejścia jest znaczco wyższy. Poznanie niuansów i możliwości systemu wymaga jego przestudiowania. Do LaTeX-a trzeba się też przyzwyczaić. Uzyskanie oczekiwanej efektu końcowego nieawsze jest proste, szczególnie na początku nauki, co może działać zniechęcająco. Pewnym utrudnieniem jest również brak natychmiastowego podglądu ostatecznego rezultatu.

WPROWADZENIE DO SKŁADNI

Źródło dokumentu tworzonego w systemie LaTeX to plik tekstowy o rozszerzeniu *tex*. Na Rysunku 1 przedstawiono prosty przykład kodu wraz z widokiem pliku PDF uzyskanego po jego przetworzeniu.

Polecenia (znaczniki) systemu LaTeX rozpoczynają się od znaku „`\`”. Każdy dokument musi zostać przydzielony do danej klasy, którą w powyższym przykładzie w linii 1 zdefiniowano jako *article* (artykuł). Pozostałe możliwości to m.in. *report* (dłuższe opracowania, np. praca magisterska), *book* (książka), *slides* (prezentacja, pokaz slajdów), *letters* (list). Wybór klasy ma wpływ na szczegóły związane z formatowaniem pliku finalnego oraz możliwość zastosowania niektórych poleceń. Klasa określa również sposoby, których użyje system do wydajnego przetwarzania źródła. Głównym kryterium podziału jest obszerność treści. W zaprezentowanym przykładzie dodatkowo zdefiniowano rozmiar tekstu zasadniczego na 11pt oraz format papieru A4. Argumenty podawane w nawiasach klamrowych (nazwy) są obowiązkowe, natomiast parametry w nawiasach kwadratowych opcjonalne. W liniach 2 i 3 określono język dokumentu oraz system kodowania znaków. W wierszach 5-7 zdefiniowano tytuł, autora oraz datę wydania. Znacznik `\today` spowoduje wstawienie daty systemowej pobranej w momencie budowania pliku wynikowego. Pomiędzy znacznikami `\begin{document}` (linia 9) oraz `\end{document}` (linia 21) umieszcza się ciało dokumentu. Polecenie `\maketitle` (linia 10) wstawia zdefiniowany uprzednio tytuł, autora oraz datę.

```

1 \documentclass[11pt,a4paper]{article}
2 \usepackage[polish]{babel}
3 \usepackage[utf8]{inputenc}
4
5 \title{Przykład dokumentu}
6 \author{Marek Michalski}
7 \date{\today}
8
9 \begin{document}
10 \maketitle
11
12 \tableofcontents
13
14 \section{Rozdział}
15 | Treść rozdziału.
16 \section{Kolejny rozdział}
17 | Treść kolejnego rozdziału.
18 \subsection{Podrozdział}
19 | Treść podrozdziału.
20
21 \end{document}

```

Przykład dokumentu
Marek Michalski
30 czerwca 2018

Spis treści

1	Rozdział	1
2	Kolejny rozdział	1
2.1	Podrozdział	1

1 Rozdział

Treść rozdziału.

2 Kolejny rozdział

Treść kolejnego rozdziału.

2.1 Podrozdział

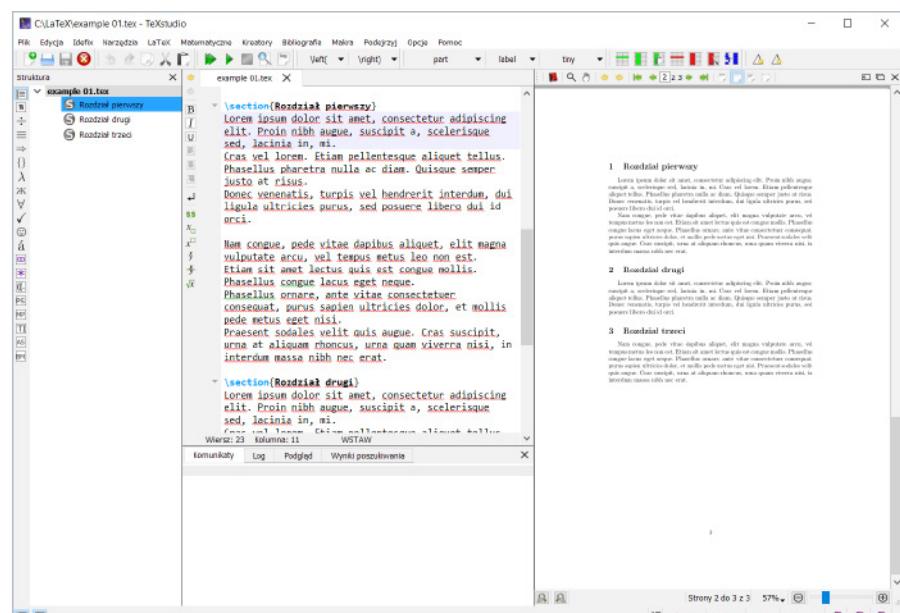
Treść podrozdziału.

Rysunek 1. Przykład kodu źródłowego LaTeX wraz z wynikiem końcowym



Rysunek 1: Model samochodu.

Rysunek 2. Przykład wstawienia grafiki (źródło oraz efekt finalny)



Rysunek 3. Okno główne edytora TeXstudio

```

4 \usepackage{indentfirst} % pierwszy akapit w tekście
5
6 \title{Podstawa} 1655a664 You, 2 months ago (May 1st, 2018 10:17pm)
7 \author{} Pierwsza wersja tekstu.
8 \date{} << ...
9
10 \frenchspacing You, 2 months ago * Pierwsza wersja tekstu.
11
12 \begin{document}

```

Rysunek 4. Wersjonowanie źródeł pozwala m.in. na identyfikację autora oraz czasu wprowadzenia zmian w danej linii pliku. Przykład użycia Gita, Visual Studio Code oraz pluginu GitLens

Znacznik `\tableofcontents` (linia 12) odpowiada za zbudowanie i osadzenie spisu treści. Jego poprawne wygenerowanie może wymagać powtórzenia komplikacji. LaTeX pozwala na umieszczenie spisu treści praktycznie w dowolnej części dokumentu, niekoniecznie na początku. W znacznikach `\section` oraz `\subsection` umieszczono treść rozdziałów i podrozdziałów. Na poziomie źródła format wpisanego tekstu nie ma znaczenia. System sam przetworzy wpisaną treść. Usunięte zostaną wielokrotne spacje, tabulatory, kilkukrotne przejścia do nowych linii itp.

Często uzyskanie oczekiwanych wyników wymaga dołączenia stosownych pakietów, które rozszerzają możliwości LaTeX-a. Na przykład możliwość umieszczania grafik obsługuje paczka `graphicx`. Na Rysunku 2 zaprezentowano przykład zastosowania.

Dokładne omówienie możliwości systemu nie jest możliwe w tak krótkiej prezentacji. Jako uzupełnienie artykułu przygotowano materiały dodatkowe pokazujące rozwiązania typowych problemów. W plikach źródłowych umieszczono komentarze, co ułatwi ich analizę i zrozumienie. Załączono również finalne pliki PDF.

NARZĘDZIA

Jedną z najpopularniejszych implementacji środowiska TeX/LaTeX jest pakiet MiKTeX. Projekt początkowo dedykowany był wyłącznie systemowi Windows. Obecnie dostępny jest również dla Linux i macOS oraz jako obraz Dockera. MiKTeX oferuje podstawową wersję LaTeX-a. Brakujące pakiety, zdefiniowane przez znacznik `\usepackage` w pliku źródłowym, zostaną pobrane z sieci i zainstalowane przy pierwszym użyciu. Z tego powodu wygodnym rozwiązaniem będzie zaznaczenie w trakcie instalacji środowiska MiKTeX, aby powyższy proces odbywał się bez pytania o zgodę.

Do efektywnej pracy niezbędny będzie edytor plików `tex`, który oferuje takie udogodnienia jak kolorowanie składni, system podpowiedzi i autouzupełniania, podgląd dokumentu wynikowego czy automatyzację wybranych operacji, np. wstawiania znaczników. Istnieje wiele tego typu programów – zarówno darmowych, jak i odpłatnych. Wraz z pakietem MiKTeX dostarczany jest edytor TeXworks. Inne popularne programy to m.in. Kile (Linux), LEd (Windows), LyX (edytor graficzny, minimalizuje potrzebę znajomości

LaTeX-a), TexMaker, TeXstudio. Trudno jednoznacznie wskazać najlepsze narzędzie. Wybór zależy w dużej mierze od oczekiwaniń oraz upodobań użytkownika. Również uniwersalne środowiska typu Visual Studio Code oferują pluginy do pracy z LaTeX-em.

Ponieważ źródła LaTeX-a to zwykłe pliki tekstowe, możliwe staje się zarządzanie ich wersjonowaniem za pomocą takich systemów jak Git lub Subversion w identyczny sposób jak ma to miejsce w przypadku kodu oprogramowania. Takie rozwiązanie jest często stosowane w praktyce. Umożliwia pełną identyfikację, kto i kiedy wprowadził daną zmianę, oraz pozwala na jednoczesną pracę kilku osób nad tym samym dokumentem.

Budowanie finalnych plików tekstowych (np. instrukcji obsługi czy warunków licencji) może być wykonywane jako część budowy całej aplikacji w ramach systemu continuous integration.

PODSUMOWANIE

LaTeX to w pełni darmowy system, pozwalający w wydajny sposób tworzyć profesjonalne dokumenty tekstowe, na którego naukę zdecydowanie warto poświęcić czas. Program doskonale sprawdzi się zwłaszcza w przypadku obszernych, złożonych lub często modyfikowanych tekstów, dla których uzyskana efektywność pracy będzie nieporównywalnie większa niż przy zastosowaniu edytorów rodziny WYSIWYG.

Programiści mogą wykorzystać LaTeX-a również jako bibliotekę służącą do generowania plików PDF wewnątrz danej aplikacji. W takim zastosowaniu zadaniem programu będzie utworzenie pliku źródłowego, który po przetworzeniu zostanie zaprezentowany użytkownikowi.

LaTeX-a można spotkać także w wielu innych przypadkach i zastosowaniach. Jest to np. jeden z kilku dostępnych formatów prezentacji raportów generowanych przez popularny program Doxygen, służący do automatycznego tworzenia dokumentacji kodu źródłowego oprogramowania. Przykład takiego dokumentu wraz z opisem sposobu jego uzyskania znajduje się w materiałach dodatkowych przygotowanych do artykułu. Serwis Wikipedia używa składni LaTeX-a m.in. przy tworzeniu i prezentacji wzorów matematycznych.

W sieci:

- ▶ Strona projektu LaTeX: <https://www.latex-project.org/>
- ▶ Strona projektu MiKTeX: <https://miktex.org/>
- ▶ Dodatkowe materiały do artykułu dostępne w serwisie GitHub: <https://github.com/marekmichalski-pl/LaTeX-examples>.



MAREK MICHALSKI

kontakt@marekmichalski.pl

Programista pasjonat, zainteresowany tematyką automatyzacji procesów oraz zagadnieniami na styku programowania i testowania aplikacji. Programuje głównie w języku C#. Tworzył aplikacje bazodanowe oraz oprogramowywał systemy pomiarowe. Lubi poznawać nowe technologie.

5. edycja konferencji .NET DeveloperDays już we wrześniu!

Programiści, architekci oraz specjaliści wykorzystujący do swej pracy platformę .NET po raz piąty spotkają się w warszawskiej hali EXPO XXI.

Najwybitniejsi eksperci z branży IT, aż cztery równolegle ścieżki tematyczne i cztery propozycje całodniowych pre-conów. To właśnie czeka uczestników .NET DeveloperDays – największej w całej Europie Środkowo-Wschodniej konferencji poświęconej platformie .NET.

NOWOŚCI W FORMULE

Od blisko pięciu lat na .NET DeveloperDays przyjeżdżają eksperci i programiści z całego świata. Organizatorzy z roku na rok podnoszą jakość wydarzenia i proponują coraz to nowe rozwiązania zwiększające atrakcyjność konferencji. W tym roku takich nowości będzie kilka. Po raz pierwszy sesje będą się odbywały **równolegle w 4 salach**. Nowością będzie także **sala tematyczna**, a w niej jeden dzień poświęcony w całości technologii **Xamarin**, drugi – **.NET Core**! Kolejna nowinka to zorganizowana strefa ekspertów, w której każdy z uczestników konferencji będzie mógł osobiście porozmawiać z zaproszonymi specjalistami i zapytać o interesującego zagadnienia.

KTO WYSTĄPI NA KONFERENCJI?

O jakość merytoryczną wydarzenia tradycyjnie zadbają eksperci. Swoją wiedzą podzieli się ponad **20 prelegentów** z całego świata. Organizatorzy ujawnili już nazwiska niektórych z nich.

Wśród gwiazd konferencji będzie **Scott Hunter** – Director of Program Management w firmie Microsoft. Na co dzień pracuje nad .NET Core, ASP.NET, Entity Framework i narzędziami internetowymi do Visual Studio. Obok niego wystąpi **Tim Huckaby**, uznany przez prasę za „pioniera rewolucji Smart Client”. Huckaby zajmuje się zagadnieniami z zakresu: AI, Computer Vision, Machine Learning,

AR/MR, & Emerging User Experiences. Podczas 35 lat pracy związany był m.in. z firmą Microsoft – współpracował z Billiem Gatesem i Stevem Ballmerem.

Na scenie po raz kolejny pojawi się także **Gill Cleereen** (Microsoft Regional Director, Visual Studio MVP), który na co dzień pracuje nad rozwojem sieci i urządzeń mobilnych. Ponadto prowadzi kursy video na platformie Pluralsight z zakresu technologii Xamarin. Podczas .NET DeveloperDays to właśnie on będzie prowadził sesję dotyczącą tego zagadnienia.

Wystąpią również: **Wouter de Kort, Jiri Cincura, Michael Kaufmann, Christophe Nasarre oraz Alex Thissen**.

NIE TYLKO KONFERENCJA

W ramach .NET DeveloperDays odbędą się również **całodniowe sesje szkoleniowe**. 17 września **Sasha Goldshtein, Daniel Marbach, Neal Ford i Shawn Wildermuth** poprowadzą autorskie zajęcia z zagadnieniami, w których są uznanymi specjalistami. Co istotne – na część warsztatową obowiązują osobne zapisy, a ilość miejsc jest ograniczona.

Na zakończenie pierwszego dnia konferencji zaplanowana jest impreza integracyjna dla wszystkich uczestników.

Wszystkie sesje podczas wydarzenia odbywać się będą w języku angielskim. Formularz rejestracyjny i wszelkie informacje dostępne są na stronie internetowej <http://net.developerdays.pl/>. Aktualności można śledzić na facebookowym profilu konferencji: <https://www.facebook.com/DeveloperDays/>.

Organizatorem wydarzenia jest firma DATA MASTER, która odpowiada także za produkcję Cloud DeveloperDays oraz Join! The Database Conference.



Moduły w C++ – raport z postępów

O modułach w C++ – jako o rozwiązaniu problemu braku kompartmentalizacji w plikach nagłówkowych – mówi się równie długo, co o koncepcjach w kontekście nieczytelnych błędów komplikacji w przypadku podania niepoprawnych argumentów do szablonów. Oba te pojęcia są również bardzo szeroko rozumiane i mają różne, często wzajemnie sprzeczne interpretacje.

Osoba lub grupa proponująca wprowadzenie modułów musi odpowiedzieć między innymi na następujące pytania:

- » Jaki jest podstawowy cel użycia modułów? Wśród popularnych odpowiedzi znajdują się: przyspieszenie komplikacji, możliwość jednokrotnej komplikacji modułu i udostępnianie go podobnie jak biblioteki .jar w Javie, wersjonowanie bibliotek oraz wyręczenie systemów budowania.
- » Jak będzie wyglądała ścieżka migracji od obecnego systemu nagłówków? Pisanie programów korzystających wyłącznie z modułów jest nerealne, tak samo jak przenoszenie już istniejących projektów w całości.
- » Czy makra i definicje preprocesora będą eksportowane z modułu i, jeśli tak, to w jaki sposób? Tutaj należy zauważać, że właściwie każda znacząca biblioteka/framework w C oraz w C++ w jakiś sposób korzysta z makr, czy jest to POSIX-owy FD_SET, Windowsowe kody błędów, assert z biblioteki standardowej, foreach w Qt, a nawet BOOST_HANA_STRING z, wydawałoby się, czysto C++-owej, nowej biblioteki Boost.Hana. W Listingu 1 pokazano przykładowe użycie makra oferowanego przez bibliotekę.
- » Czy został przewidziany oraz – jeśli tak – jak będzie wyglądał mechanizm sterowania komplikacją załączanych modułów? Obecnie często stosuje się definiowanie odpowiedniej wartości preprocesora, czy to poprzez dyrektywę #define, czy przez argument przekazany kompilatorowi podczas komplikacji. Przykład w Listingu 0.
- » Czy nazwa modułu ma wpływ na przestrzeń nazw zawartych w nim definicji? Tak rozwiązuje to inne języki, dzięki czemu przestrzeń nazw i ścieżka do definicji modułu stają się wymienne. Oznacza to jednak, że przeniesienie starych bibliotek na system modułów będzie się wiązało z przepisaniem (albo przy najmniej refactoringiem) znacznej części istniejącego kodu.

Listing 0. Sterowanie komplikacją załączanych bibliotek

```
#define WIN32_LEAN_AND_MEAN
#include <windows.h>

int main()
{
    return 0;
}
```

Listing 1. Użycie makra BOOST_HANA_STRING eksportowanego z biblioteki Boost.Hana. Źródło: [0]

```
// Copyright Louis Dionne 2013-2017
// Distributed under the Boost Software License, Version 1.0.
// (See accompanying file LICENSE.md or copy at http://boost.org/LICENSE\_1\_0.txt)
```

```
#include <boost/hana/assert.hpp>
#include <boost/hana/equal.hpp>
#include <boost/hana/not_equal.hpp>
#include <boost/hana/string.hpp>
namespace hana = boost::hana;
int main() {
    BOOST_HANA_CONSTANT_CHECK(
        BOOST_HANA_STRING("abcdef") == BOOST_HANA_STRING("abcdef")
    );
    BOOST_HANA_CONSTANT_CHECK(
        BOOST_HANA_STRING("abcdef") != BOOST_HANA_STRING("abef")
    );
}
```

BOJE STANDARYZACYJNE – RYS HISTORYCZNY

Pierwsza propozycja dodania systemu modułów do języka C++ pojawiła się pod koniec roku 2004, a jej autorem był Daveed Vandevoorde. Została ona oznaczona numerem roboczym N1736 [1]. Przedstawiona syntaktyka definiowania oraz importowania modułów może być dla czytelnika zaznajomionego z obecnym stanem rzeczy dość zaskakująca. Zostało to przedstawione w Listingach 2 i 3.

Listing 2. Importowanie modułu według N1736. Źródło: [1]/Example 1

```
namespace << std; // Module import directive.

int main() {
    std::cout << "Hello World\n";
}
```

Listing 3. Deklaracja modułu według N1736. Źródło: [1]/Example 3

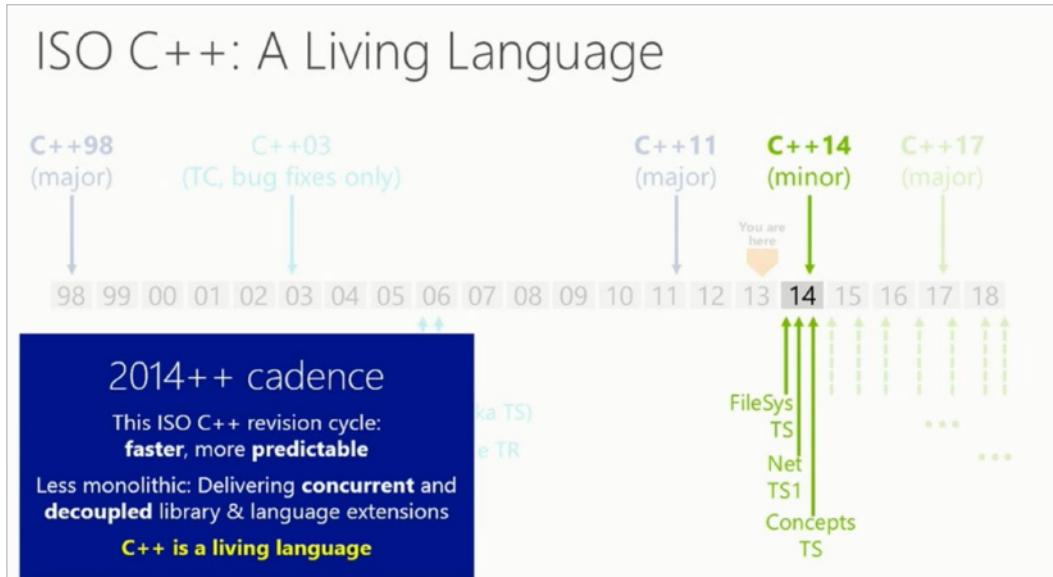
```
// File_1.cpp:
namespace >> M1 {
    export typedef int I1;
}

// File_2.cpp:
namespace >> M2 {
    export typedef int I2;
}

// File_3.cpp:
namespace >> MM {
    export namespace << M1; // Make M1 visible here and
                           // in code that imports MM.
    namespace << M2;      // Make M2 visible here,
                           // but not in clients.
}

// File_4.cpp:
namespace << MM;
M1::I1 i1; // Okay.
M2::I2 i2; // Error: M2 invisible.
```

Propozycja ta została dołączona do głównego szkicu najbliższego standardu, niesławnego C++0x, który ostatecznie stał się C++11.



Rysunek 0. Przewidywania dotyczące rozwoju C++. Źródło: Herb Sutter, [3] 10m 22s

C++14: Constraints aka “Concepts lite”

- How do we specify requirements on template arguments?
 - state intent
 - Explicitly states requirements on argument types
 - provide point-of-use checking
 - No checking of template definitions
 - use `constexpr` functions
- Voted as C++14 Technical Report
- Design by B. Stroustrup, G. Dos Reis, and A. Sutton
- Implemented by Andrew Sutton in GCC
- There are no C++0x concept complexities
 - No concept maps
 - No new syntax for defining concepts
 - No new scope and lookup issues

Stroustrup - Essence - Going Native'13

46

Rysunek 1. Przewidywania dotyczące rozwoju C++. Źródło: Bjarne Stroustrup, [3] 60m 11s

Przez następne lata N1736 zastąpiły kolejne rewizje (dokumenty historyczne mówią o N1736, N1778, N1964, N2006, N2015, N2073, N2074, N2316), ale komisji standaryzacyjnej nie udało się dojść do konsensusu co do ostatecznej definicji oraz kompatybilności z innymi wprowadzonymi w C++0x zmianami.

Pod koniec pierwszej dekady obecnego millenium komisja standaryzacyjna C++ walczyła z czasem, aby C++0x pozostało C++0x (co, jak wiemy, ostatecznie się nie udało i 0x przeistoczyło się w 11). Z tego powodu, po spotkaniu w San Francisco w 2008 r., zdecydowano się na przeniesienie prac nad modułami do osobnego dokumentu standaryzacyjnego [2]. Dokument ten, noszący miano raportu technicznego (ang. *Technical Report* – TR), pozwalał na to, aby prace nad modułami toczyły się własnym tempem, nie zamykając jednak drogi twórcom kompilatorów do dołączania eksperymentalnych implementacji.

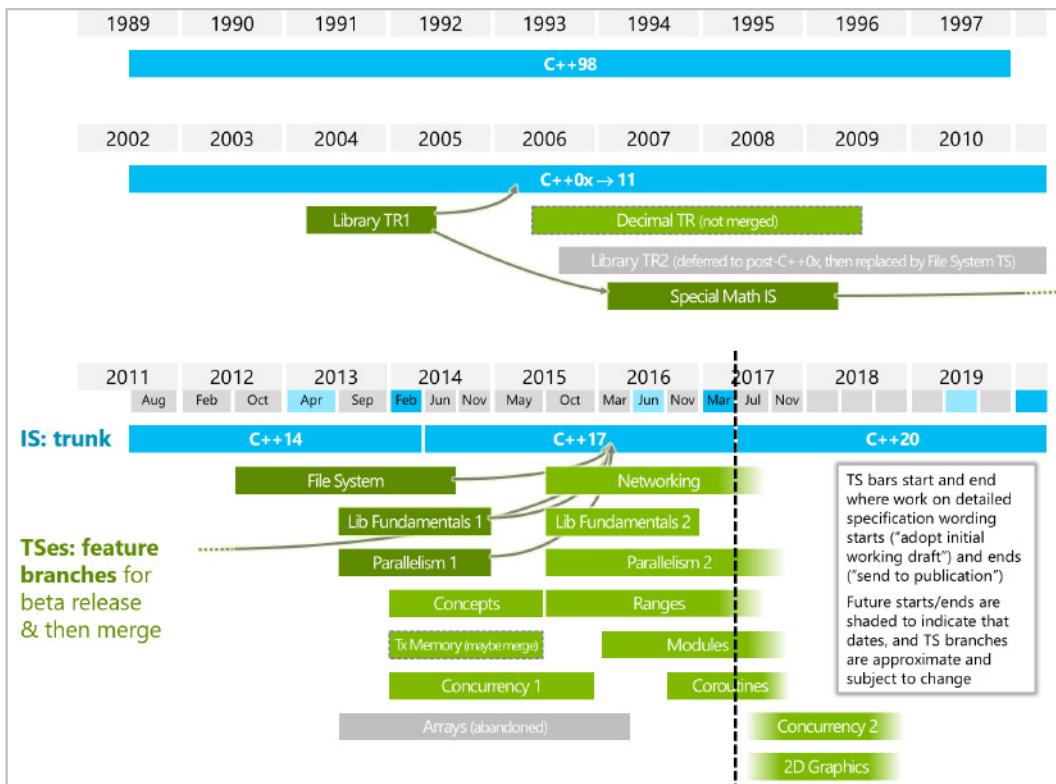
W związku z publikacją C++11 i metaforycznym trzęsieniem ziemi, jakie w związku z tym nastąpiło, sprawa kolejnych dużych

zmian w języku została odłożona na później. Pierwotnym założeniem było rozpoczęcie wydawania nowych standardów w cyklu trzyletnim przestawnym: co drugi miałby być znaczący (C++11, C++17, C++23...), a co drugi pomniejszy/poprawkowy (C++14, C++20, C++26...).

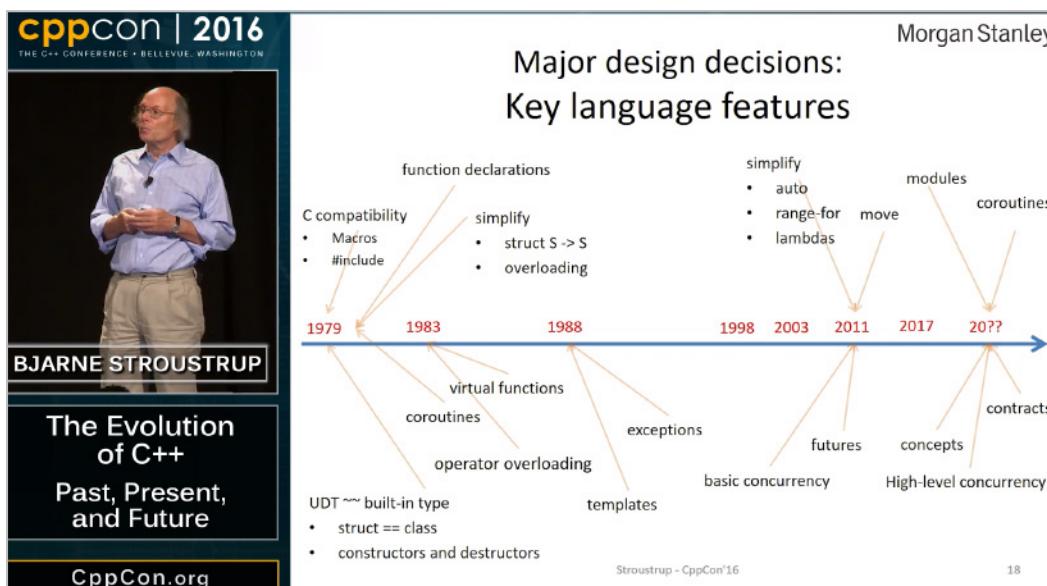
Ewangelici języka przewidywali wprowadzenie kolejnych przełomowych zmian już w C++14, a najpóźniej w C++17. Można to zobaczyć na przykładzie Herba Suttera z jego prezentacji „One C++” podczas konferencji *Going Native 2013* [3] (Rysunek 0) oraz na przykładzie prelekcji „The Essence of C++” Bjarne'a Stroustrupa podczas tej samej konferencji [4] (Rysunek 1).

Bardzo szybko jednak zorientowano się, że oczekiwania te są nierealne. Już pod koniec roku 2013 Herb Sutter zamieścił nowe przewidywania, zdejmujące z C++14 brzemień wprowadzania przełomowych zmian i przenoszące specyfikację istotnych funkcjonalności na „po C++17” (Rysunek 2). Podobne zjawisko można zauważać w prelekcji Bjarne'a Stroustrupa z C++Con 2016 [6] (Rysunek 3),

JĘZYKI PROGRAMOWANIA



Rysunek 2. Przewidywania Herba Suterra dotyczące rozwoju C++ z października 2013. Źródło: [5]



Rysunek 3. Przewidywania Bjarne'a Stroustrupa dotyczące rozwoju języka z września 2016. Źródło: [6] 22m 02s

choć on jeszcze w 2015 roku przewidywał, że C++17 będzie zawierał między innymi moduły (Rysunek 4).

Ostatecznie C++17 w opinii wielu osób nie spełnił pokładanych w nim nadziei. Bjarne Stroustrup w [6] stwierdził, że z dziesięciu najistotniejszych nowości, których oczekiwali od najnowszego standardu, udało się w nim zatrzymać dwie, i to na dodatek tylko częściowo. Autor tego artykułu opisał swoje przemyślenia dotyczące C++17 – także w kontekście modułów – w artykułach [7] i [8].

MODUŁY DZISIAJ

Najnowsza wersja specyfikacji modułów nosi nazwę N4720 [9] i datowana jest na 29 stycznia obecnego roku. Udostępniono ją jako

ISO/IEC TS 21544:2018 [A] (treść jest identyczna poza nagłówkami). Jej istnienie nie oznacza jednak, że znajdzie się ona w następnym standardzie, a także że pozostanie bez zmian.

W chwili pisania tego artykułu istnieją eksperymentalne implementacje wyżej wspomnianej specyfikacji. Jest tak w kompilatorze Microsoftu dołączanym do Visual Studio w wersji 15.3¹[B] i nowszych, gcc natomiast posiada osobnego brancha *cxx-modules* [C], z implementacją w wersji alpha. Niestety, clang od kilku lat zawiera implementację niekompatybilną z N4720 [D].

1. Autor testował za pomocą Visual Studio Community 2017 w wersji 15.7.4, z wersją kompilatora 19.14.26431

Morgan Stanley

My top-ten list for C++17 (in early 2015)

- Concepts
 - Concept-based generic programming, good error messages
- Modules
 - Fast compilation through cleaner code
- Ranges (library)
- Uniform call syntax
- Co-routines
 - Fast and simple
- Networking (library)
- Contracts
- SIMD vector and parallel algorithms (mostly library)
- Library “vocabulary types”
 - such as *optional*, *variant*, *string_span*, and *span*
- A “magic type” *stack_array*

It's hard to make predictions, especially about the future

Stroustrup - CppCon'16 42

Rysunek 4. Przewidywania Bjarne'a Stroustrupa dotyczące rozwoju języka z początku 2015. Źródło: [6] 58m 54s

Clang

Zespół tworzący kompilator clang już kilka lat temu zaproponował implementację modułów [E]. Rozwiązania clanga oraz Microsoftu, choć niekompatybilne, to jednak są do siebie znaczco zblżone. Przykładowy kod modułu w pliku o nazwie *mod.cppm* znajduje się w Listingu 4.

Listing 4. Kod przykładowego modułu mod.cppm w implementacji kompilatora clang

```
export module mod;
#define ANSWER 42
export int answer()
{
    return ANSWER;
}
```

Jak widać, kod ten definiuje funkcję *answer()* oraz definicję preprocesora *ANSWER*. Obie ewaluują się do wartości 42, a funkcja bezpośrednio korzysta ze starej zdefiniowanej w preprocesorze. Kompilację modułu oraz pliku *main.cpp* zawierającego główną funkcję programu przedstawia plik Makefile z Listingu 5.

Listing 5. Makefile dla przykładu z modułami clanga

```
all: mod.o
clang++ -fmodules-ts --precompile mod.cppm -o mod.pcm
mod.o: mod.pcm
clang++ -fmodules-ts -c mod.pcm -o mod.o
mod.pcm:
clang++ -fmodules-ts --precompile mod.cppm -o mod.pcm
clean:
rm -f a.out
rm -f mod.pcm
rm -f mod.o
```

Aby pokazać działanie modułów, najpierw zostanie opisany kod błędny. W Listingu 6 przedstawiono kod próbujący użyć definicji preprocesora *ANSWER*, zdefiniowanej wewnątrz pliku *mod.cppm*.

Autor zauważa, że gdyby plik *mod.cppm* był plikiem nagłówkowym załączanym za pomocą dyrektywy `#include <mod.hpp>`, to byłby on w pełni poprawny.

Listing 6. Kod używający definicji preprocesora

```
#include <iostream>
import mod;
int main()
{
    std::cout << "Hello, Modular World!\n";
    std::cout << "The answer is " << ANSWER << '\n';
}
```

Przy próbie komplikacji użytkownik utrzymuje następujący komunikat:

Listing 7. Wynik komplikacji kodu z Listingu 6

```
> make
clang++ -fmodules-ts --precompile mod.cppm -o mod.pcm
clang++ -fmodules-ts -c mod.pcm -o mod.o
clang++ -fmodules-ts -fprebuilt-module-path=. mod.o main.cpp
main.cpp:8:38: error: use of undeclared identifier 'ANSWER'
    std::cout << "The answer is " << ANSWER << '\n';
                                         ^
1 error generated.
make: *** [Makefile:3: all] Error 1
```

Wystarczy użyć jawnie wyeksportowanej funkcji, aby uzyskać pożądany wynik:

Listing 8. Poprawiony kod z Listingu 6

```
#include <iostream>
import mod;
int main()
{
    std::cout << "Hello, Modular World!\n";
    std::cout << "The answer is " << answer() << '\n';
}
```

Teraz komplikacja i uruchomienie przebiegają pomyślnie:

Listing 9. Wynik komplikacji i uruchomienia kodu z Listingu 8

```
% make
clang++ -fmodules-ts --precompile mod.cppm -o mod.pcm
clang++ -fmodules-ts -c mod.pcm -o mod.o
clang++ -fmodules-ts -fprebuilt-module-path=. mod.o main.cpp

% ./a.out
Hello, Modular World!
The answer is 42
```

Autor uznał, że na tym przykładzie udało się pokazać jedną z cech charakterystycznych modułów, tj. oddzielenie prywatnych definicji (nawet jeśli są makrami preprocesora) modułu od jego eksportowanego interfejsu.

Visual C++

Implementacja *Modules TS* kompilatora Microsoftu wspiera importowanie biblioteki standardowej. Zamiast imitować podział nagłówków, została ona podzielona na pięć modułów:

- » `std.regex` – odpowiednik nagłówka `<regex>`,
- » `std.filesystem` – odpowiednik nagłówka `<filesystem>` (a w obecnej implementacji – `<experimental/filesystem>`),
- » `std.memory` – odpowiednik nagłówka `<memory>`,
- » `std.threading` – amalgamat nagłówków `<atomic>`, `<condition_variable>`, `<future>`, `<mutex>`, `<shared_mutex>` oraz `<thread>`,
- » `std.core` – wszystko inne.

W Listingu A przedstawiono przykładowy program wykorzystujący bibliotekę standardową:

Listing A. Przykładowy program korzystający z biblioteki standardowej za pomocą modułów

```
import std.core;

using namespace std::literals;

int main()
{
    auto hello = "Hello, Modular World!"s;
    std::cout << hello << '\n';

    std::map<char, int> counts;

    for(char c : hello)
        if(isalpha(c))
            counts[tolower(c)]++;

    std::cout << "In the previous message...\n";
    for(auto const& p : counts)
    {
        std::cout << "Letter " << p.first <<
                    " was found " <<
                    p.second << " times\n";
    }
}
```

Proces komplikacji i uruchomienia wygląda następująco:

Listing B. Kompilacja i uruchomienie programu z Listingu A

```
> cl /experimental:module /EHsc /MD /std:c++latest vs1.cpp
Microsoft (R) C/C++ Optimizing Compiler Version 19.14.26431 for x86
Copyright (C) Microsoft Corporation. All rights reserved.
```

Experimental features are provided as a preview of proposed language features, and we're eager to hear about bugs and suggestions for improvements. However, note that these experimental features are non-standard, provided as-is without support, and subject to breaking changes or removal without notice. See <http://go.microsoft.com/fwlink/?LinkID=691081> for details.

```
vs1.cpp
Microsoft (R) Incremental Linker Version 14.14.26431.0
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
/out:vs1.exe
vs1.obj
```

```
> vs1
Hello, Modular World!
In the previous message...
Letter a found 1 times
Letter d found 2 times
Letter e found 1 times
Letter h found 1 times
Letter l found 4 times
Letter m found 1 times
Letter o found 3 times
Letter r found 2 times
Letter u found 1 times
Letter w found 1 times
```

Biorąc za przykład kod modułu z Listingu C (pod nazwą `mod.ixx`), oraz kod głównego pliku Listingu D (pod nazwą `vs2.cpp`), komplikacja przedstawiona jest w Listingu E.

Listing C. Kod przykładowego modułu mod.cppm w implementacji kompilatora Visual C++

```
export module mod;

#define ANSWER 42

export int answer()
{
    return ANSWER;
}
```

Listing D. Kod używający definicji preprocesora

```
import std.core;
import mod;

int main()
{
    std::cout << "Hello, Modular World!\n";
    std::cout << "The answer is " << ANSWER << '\n';
}
```

Listing E. Kompilacja błędного kodu (zawierającego definicję preprocesora)

```
> cl /c /EHsc /MD /std:c++latest /experimental:module mod.ixx
Microsoft (R) C/C++ Optimizing Compiler Version 19.14.26431 for x86
Copyright (C) Microsoft Corporation. All rights reserved.
```

Experimental features are provided as a preview of proposed language features, and we're eager to hear about bugs and suggestions for improvements.

However, note that these experimental features are non-standard, provided as-is without support, and subject to breaking changes or removal without notice. See <http://go.microsoft.com/fwlink/?LinkID=691081> for details.

mod.ixx

```
> cl /experimental:module /EHsc /MD /std:c++latest /
module:reference mod.ifc vs2.cpp mod.obj
Microsoft (R) C/C++ Optimizing Compiler Version 19.14.26431 for x86
Copyright (C) Microsoft Corporation. All rights reserved.
```

Experimental features are provided as a preview of proposed language features, and we're eager to hear about bugs and suggestions for improvements. However, note that these experimental features are non-standard, provided as-is without support, and subject to breaking changes or removal without notice. See <http://go.microsoft.com/fwlink/?LinkID=691081> for details.

```
vs2.cpp
vs2.cpp(7): error C2065: 'ANSWER': undeclared identifier
```

Jak widać, również tutaj nie udaje się użyć makra, które w razie skorzystania z dyrektywy `#include <mod.hpp>` przedostałoby się do widoku pliku `vs2.cpp`. Po dokonaniu poprawki poprzez wywołanie funkcji, przedstawionej w Listingu F, udane komplikacja i uruchomienie przedstawione zostały w Listingu 10.

Listing F. Poprawiony kod z Listingu D

```
import std.core;
import mod;

int main()
{
    std::cout << "Hello, Modular World!\n";
    std::cout << "The answer is " << answer() << '\n';
}
```

Listing 10. Kompilacja i uruchomienie poprawnego kodu

```
> cl /c /EHsc /MD /std:c++latest /experimental:module mod.ixx
Microsoft (R) C/C++ Optimizing Compiler Version 19.14.26431 for
x86
Copyright (C) Microsoft Corporation. All rights reserved.

Experimental features are provided as a preview of proposed
language features,
and we're eager to hear about bugs and suggestions for
improvements.
However, note that these experimental features are non-
standard, provided as-is without
support, and subject to breaking changes or removal without
notice. See
http://go.microsoft.com/fwlink/?LinkID=691081 for details.

mod.ixx

> cl /experimental:module /EHsc /MD /std:c++latest /
module:reference mod.ifc vs2.cpp mod.obj
Microsoft (R) C/C++ Optimizing Compiler Version 19.14.26431 for
x86
Copyright (C) Microsoft Corporation. All rights reserved.

Experimental features are provided as a preview of proposed
language features,
and we're eager to hear about bugs and suggestions for
improvements.
However, note that these experimental features are non-
standard, provided as-is without
support, and subject to breaking changes or removal without
notice. See
http://go.microsoft.com/fwlink/?LinkID=691081 for details.

vs2.cpp
Microsoft (R) Incremental Linker Version 14.14.26431.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:vs2.exe
vs2.obj
mod.obj

> vs2
Hello, Modular World!
The answer is 42
```

Powyzszy eksperyment można uznać za sukces. Nowa wersja Visual C++ pozwala skompilować i wykonać prosty program korzystający z biblioteki standardowej, jak również definiować własne

moduły. Poprawnie zadziała odseparowanie definicji preprocesora wewnętrz modułu od plików importujących dany moduł.

gcc

Zgodnie z opisem [C] wsparcie dla modułów w gcc jest na razie bardzo eksperymentalne. Dość powiedzieć, że trzeba samodzielnie skompilować źródła kompilatora ze specjalnie na tę okazję utworzonego repozytorium – na głównym nie wprowadzono odpowiednich zmian, nawet w wersji *master*. Przedstawiony przez twórcę zmian przykładowy kod lekko różni się również od tych działających z wcześniej opisanymi kompilatorami.

W Listingach 11 i 12 przedstawiono kod wykorzystany do testowania wsparcia modułów w testowej wersji gcc. W Listingu 13 przedstawiono próbę komplikacji.

Listing 11. Kod modułu mod.cpp

```
module;
export module mod;

#define ANSWER 42

export int answer()
{
    return ANSWER;
}
```

Listing 12. Kod używający definicji preprocesora

```
#include <iostream>
import mod;

int main()
{
    std::cout << "Hello, Modular World!\n";
    std::cout << "The answer is " << ANSWER << '\n';
}
```

Listing 13. Próba komplikacji kodu z Listingów 11 i 12

```
> make
g++ -fmodules-ts main.cpp mod.cpp
In file included from /path/gcc/include/c++/9.0.0/bits/move.h:55,
                 from /path/gcc/include/c++/9.0.0/bits/
nested_exception.h:40,
                 from /path/gcc/include/c++/9.0.0/exception:144,
                 from /path/gcc/include/c++/9.0.0/ios:39,
                 from /path/gcc/include/c++/9.0.0/ostream:38,
                 from /path/gcc/include/c++/9.0.0/iostream:39,
                 from main.cpp:1
/path/gcc/include/c++/9.0.0/type_traits:2267:12: internal compiler error:
in operator[], at vec.h:841
  struct __inv_unwrap
  ~~~~~
0x61a5b1 vec<module_state*, va_gc, v1_embed>::operator[](unsigned int)
  ../../gcc-modules/gcc/vec.h:841
0x61a5b1 set_module_owner(tree_node*)
  ../../gcc-modules/gcc/cp/module.c:10347
0x914733 do_pushtag
  ../../gcc-modules/gcc/cp/name-lookup.c:7209
0x914733 pushtag(tree_node*, tree_node*, tag_scope)
  ../../gcc-modules/gcc/cp/name-lookup.c:7309
0x87ce68 xref_tag_1
  ../../gcc-modules/gcc/cp/decl.c:13817
0x87ce68 xref_tag(tag_types, tree_node*, tag_scope, bool)
  ../../gcc-modules/gcc/cp/decl.c:13874
0x93a57e cp_parser_class_head
  ../../gcc-modules/gcc/cp/parser.c:23640
0x93a57e cp_parser_class_specifier_1
  ../../gcc-modules/gcc/cp/parser.c:22908
0x93aa5d cp_parser_class_specifier
  ../../gcc-modules/gcc/cp/parser.c:23232
0x93aa5d cp_parser_type_specifier
  ../../gcc-modules/gcc/cp/parser.c:17213
0x947893 cp_parser_decl_specifier_seq
  ../../gcc-modules/gcc/cp/parser.c:14056
0x94bfc5 cp_parser_single_declaration
  ../../gcc-modules/gcc/cp/parser.c:27644
0x94c31c cp_parser_template_declaration_after_parameters
  ../../gcc-modules/gcc/cp/parser.c:27336
0x94cbae cp_parser_explicit_template_declaration
```

```
../../../../gcc-modules/gcc/cp/parser.c:27573
0x94cbae cp_parser_template_declarator_after_export
../../../../gcc-modules/gcc/cp/parser.c:27592
0x952451 cp_parser_declarator
../../../../gcc-modules/gcc/cp/parser.c:13150
0x9527fd cp_parser_declarator_seq_opt
../../../../gcc-modules/gcc/cp/parser.c:13066
0x952abc cp_parser_namespace_body
../../../../gcc-modules/gcc/cp/parser.c:19059
0x952abc cp_parser_namespace_definition
../../../../gcc-modules/gcc/cp/parser.c:19037
0x95215f cp_parser_declarator
../../../../gcc-modules/gcc/cp/parser.c:13191
We are damaged, This is broken.
Partial device,
Pliable design,
Hunt through the rubble for what once was.
See <https://gcc.gnu.org/wiki/cxx-modules#Bugs> for instructions.
make: *** [Makefile:2: all] Error 1
```

Powyższy kod się nie kompliuje. Ponieważ komunikat informuje o wewnętrzny błędzie kompilatora spowodowanym użyciem wyjątków, autor postanowił użyć funkcji obsługujących wyjście standardowe z biblioteki C. Zmieniony kod znajduje się w Listingu 14.

Listing 14. Zmodyfikowany kod main.cpp

```
#include <cstdio>
import mod;

int main()
{
    printf("Hello, Modular World!\n");
    printf("The answer is %d!\n", ANSWER);
}
```

Próba komplikacji:

Listing 15. Próba komplikacji kodu z Listingów 14 i 11

```
% make
g++ -fmodules-ts main.cpp mod.cpp
main.cpp: In function 'int main()':
main.cpp:8:35: error: 'ANSWER' was not declared in this scope
    printf("The answer is %d!\n", ANSWER);
                           ^
make: *** [Makefile:2: all] Error 1
```

Bibliografia

- [0]: <https://goo.gl/1ePGA9>
- [1]: <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2004/n1736.pdf>
- [2]: <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n2869.html>
- [3]: <https://goo.gl/c5SHPA>
- [4]: <https://goo.gl/nkLqZn>
- [5]: <https://herbsutter.com/2013/10/03/trip-report-fall-iso-c-standards-meeting/>
- [6]: https://youtu.be/_wzc7a3McOs
- [7]: Paweł „KrzaQ” Zakrzewski, C++17, co znajdziemy w nowym standardzie, „Programista”, 10, 2016 s. 18-23
- [8]: Paweł „KrzaQ” Zakrzewski, C++17 – nowy, miłośnicie panujący nam standard C++, „Programista”, 11, 2017 s. 20-32
- [9]: <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/n4720.pdf>
- [A]: <https://www.iso.org/standard/71051.html>
- [B]: <https://blogs.msdn.microsoft.com/vcblog/2017/05/05/cpp-modules-in-visual-studio-2017/>
- [C]: <https://gcc.gnu.org/wiki/cxx-modules>
- [D]: <https://clang.llvm.org/docs/Modules.html>
- [E]: <https://news.ycombinator.com/item?id=4832568>

Zgodnie z oczekiwaniami użycie ANSWER spowodowało błąd komplikacji. W Listingu 16 przedstawiono poprawiony kod, a w Listingu 17 jego komplikację i użycie.

Listing 16. Poprawiony kod z Listingu 14

```
#include <cstdio>
import mod;

int main()
{
    printf("Hello, Modular World!\n");
    printf("The answer is %d!\n", ANSWER);
}
```

Listing 17. Komplikacja kodu z Listingów 16 i 11 oraz jego uruchomienie

```
> make
g++ -fmodules-ts main.cpp mod.cpp
> ./a.out
Hello, Modular World!
The answer is 42!
```

Test modułów z eksperymentalnym gcc można uznać za umiarkowany sukces.

PODSUMOWANIE

Implementacja modułów wśród najpopularniejszych kompilatorów jest na stosunkowo zaawansowanym poziomie. Sposób ich użycia przez programistę okazał się dużo bardziej zbliżony składowiowo pomiędzy kompilatorami, niż autor artykułu się spodziewał przed rozpoczęciem prac. Należy jednak podkreślić, że sama specyfikacja może jeszcze ulec wielokrotnym zmianom zanim wejdzie do standardu, o ile w ogóle tak się stanie, więc całą zabawę z modułami należy na razie traktować właśnie tak – jako zabawę.



Największy wybór profesjonalnego oprogramowania w Polsce !

... w ofercie programy ponad 500 producentów ...



Więcej informacji:

📞 (22) 868 40 42



sales@tts.com.pl

Sprzedaż



Dystrybucja



Import na zamówienie

TTS Company Sp. z o.o.

ul. Domaniewska 44A

02-672 Warszawa

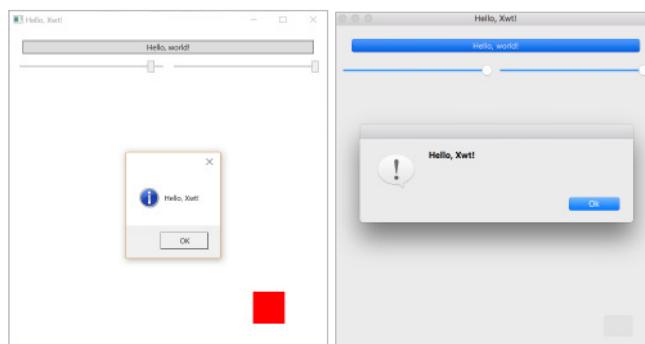
www.tts.com.pl

Xwt. Tworzenie wieloplatformowych aplikacji desktopowych

Firma Xamarin stworzyła jedną z najbardziej popularnych technologii tworzenia wieloplatformowych aplikacji mobilnych. Jednak produkty tej firmy nie ograniczają się wyłącznie do platform mobilnych. Dobrym przykładem jest tutaj framework Xwt, który, analogicznie jak technologia Xamarin, umożliwia tworzenie wieloplatformowych aplikacji desktopowych w oparciu o wspólny kod C#. W tym artykule zaprezentuję przykładowe użycie tej technologii do zaimplementowania stosunkowo prostej aplikacji dla Windows i Mac.

WSTĘP

Xwt jest technologią umożliwiającą tworzenie wieloplatformowych aplikacji desktopowych w oparciu o spójny interfejs programistyczny. Interfejs ten jest dostępny z poziomu jednego z języków programowania platformy .NET. W trakcie działania aplikacji polecenia z interfejsu Xwt są mapowane na natywne kontrolki danej platformy. To mapowanie jest realizowane poprzez odpowiedni silnik uruchomieniowy Xwt, który jest zależny od platformy. W efekcie, jak pokazano na Rysunku 1, aplikacja powstaje przy użyciu tego samego kodu, a jej wygląd jest charakterystyczny dla danej platformy.



Rysunek 1. Wieloplatformowa aplikacja, którą zaimplementujemy w tym artykule. Po lewej stronie przedstawiono aplikację uruchomioną w systemie Windows 10, a po prawej aplikację uruchomioną na platformie Mac

Typowa struktura rozwiązania aplikacji Xwt posiada wspólny projekt, w którym definiujemy UI aplikacji. UI tworzy się za pomocą widżetów. To one są następnie konwertowane na natywne kontrolki. Dostęp do widżetów oraz API Xwt jest możliwy poprzez pakiet NuGet o nazwie Xwt. Dodatkowo rozwiązanie zawiera projekty specyficzne dla danej platformy. Te projekty posiadają referencję do projektu wspólnego oraz wykorzystują pakiety NuGet Xwt, które dostarczają silniki uruchomieniowe dla konkretnej platformy.

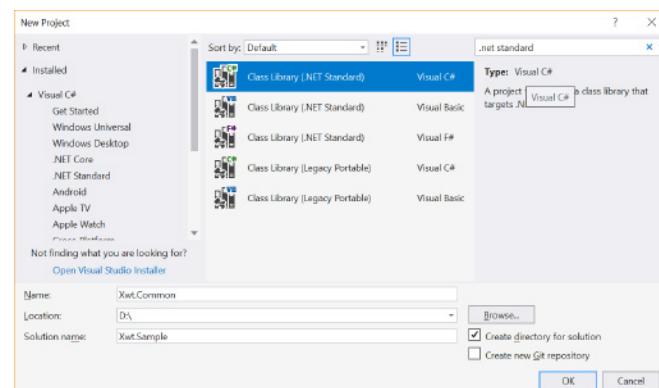
W tym artykule pokażę, w jaki sposób skorzystać z opisanego mechanizmu. Utworzę aplikację z Rysunku 1. Działa ona na Windows i Mac. Celem aplikacji jest wyświetlanie komunikatów (po kliknięciu przycisku) oraz dynamiczna zmiana pozycji prostokąta za pomocą dwóch suwaków. Ta funkcjonalność będzie zawarta w projekcie wspólnym .NET Standard. Całe rozwiązanie będzie zawierało jeszcze dwa projekty. Jeden dla Windows, a drugi

dla Mac. Aplikację dla Windows utworzę na komputerze z zainstalowanym Windows 10 (April 2018 Update) oraz Visual Studio 2017 Community. Z kolei wersja dla Maca powstanie w macOS High Sierra (10.13.4) oraz Visual Studio 2017 Community for Mac (wersja 7.5.3). Pełen kod jest dostępny pod adresem: <https://github.com/dawidborycki/Xwt.Sample>.

STRUKTURA PROJEKTU I INSTALACJA XWT

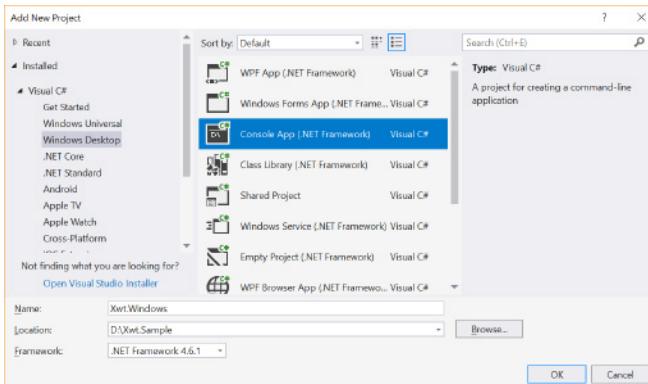
Implementację rozpoczynamy w środowisku Windows od utworzenia wspólnego projektu .NET Standard o nazwie Xwt.Common (Rysunek 2). Nazwę rozwiązania ustalamy na Xwt.Sample, a następnie instalujemy pakiet NuGet. W tym celu w konsoli menedżera pakietów NuGet (NuGet Package Manager Console) wydajemy następujące polecenie:

Install-Package Xwt



Rysunek 2. Tworzenie wspólnego projektu

Po utworzeniu projektu Xwt.Common uzupełniamy rozwiązanie o kolejny projekt aplikacji konsolowej Windows (Rysunek 3). W tym celu przechodzimy do Solution Explorer, gdzie klikamy prawym przyciskiem Xwt-Sample, a następnie z menu kontekstowego wybieramy Add->New Project... W oknie Add New Project odnajdujemy szablon Console App (.NET Framework), zmieniamy nazwę projektu na Xwt.Windows i klikamy przycisk OK. Teraz pozostało już tylko dodać referencję do projektu Xwt.Common i zainstalować dwa pakiety: NuGet Xwt oraz Xwt.Wpf (Install-Package Xwt.Wpf).



Rysunek 3. Tworzenie aplikacji Windows

Korzystamy tutaj z szablonu aplikacji konsolowej, gdyż jak za chwilę zobaczymy, cały interfejs UI zostanie utworzony z poziomu kodu C#. Dlatego nie potrzebujemy żadnych dodatkowych elementów, które implementowałby UI. Jednakże szablon Console App powoduje, że podczas uruchamiania aplikacji będzie również widoczna linia komend. Aby ją zablokować, wystarczy we właściwościach projektu Xwt.Windows zmienić *Output type* z *Console Application* na *Windows Application*.

Reasumując powyższe operacje, mamy teraz rozwiązanie złożone z dwóch projektów:

- » Xwt.Common – biblioteka .NET Standard, w której zaimplementujemy część wspólną,
- » Xwt.Windows – projekt aplikacji Windows, który odwołuje się do Xwt.Common.

PUNKT WEJŚCIA APLIKACJI XWT

Mając skonfigurowane rozwiązanie, przejdźmy teraz do właściwej implementacji. W tym celu projekt Xwt.Common uzupełniamy o kolejny plik *XwtApp.cs*, w którym umieszczamy polecenia z Listingu 1:

Listing 1. Uruchamianie aplikacji Xwt

```
namespace Xwt.Common
{
    public class XwtApp
    {
        public static void Run(ToolkitType toolkitType)
        {
            Application.Initialize(toolkitType);

            var mainWindow = new MainWindow();

            Application.Run();
            Application.Dispose();
        }
    }
}
```

Klasa XwtApp z Listingu 1 stanowi punkt wejścia aplikacji Xwt. Jedynym składnikiem klasy jest statyczna metoda Run. Przyjmuje ona jeden argument typu Xwt.ToolkitType. Jest to typ wyliczeniowy, który określa silnik Xwt, który ma zostać użyty do mapowania interfejsu API Xwt na kontrolki dla danej platformy. Silnik Xwt jest inicjalizowany za pomocą statycznej metody Initialize klasy Xwt.Application. W kolejnym kroku zazwyczaj tworzy się główne okno aplikacji. W moim przykładzie okno to będzie zaimplementowane w ramach klasy MainWindow (o czym za chwilę). Dalej uruchamia się aplikację za pomocą metody Run klasy Application. W ostatnim kroku wystarczy zwolnić niezarządzane zasoby z wykorzystaniem metody Application.Dispose.

Warto zauważyć, że do zaimplementowania aplikacji nie musimy jawnie wskazywać silnika Xwt. Jest on wykorzystywany dopiero podczas działania aplikacji. Dlatego też metodę XwtApp.Run wykorzystamy później w projektach dedykowanych konkretnym platformom, zmieniając jedynie argument toolkitType. Sama implementacja MainWindow pozostanie bez zmian.

TWORZENIE OKNA I FUNKCJONALNOŚĆ TYPU WITAJ, ŚWIECIE

Implementację klasy MainWindow rozpoczynamy od uzupełnienia projektu Xwt.Common o dodatkowy plik *MainWindow.cs*. Następnie modyfikujemy go zgodnie z Listingiem 2:

Listing 2. Tworzenie okna z przyciskiem

```
using Xwt;

public class MainWindow
{
    private Window window;

    public MainWindow()
    {
        // Window
        window = new Window()
        {
            Title = "Hello, Xwt!",
            Width = 500,
            Height = 500
        };

        window.Closed += (sender, e) =>
        {
            Application.Exit();
        };

        // Layout
        var tableLayout = new Table();
        window.Content = tableLayout;

        // Button
        tableLayout.Add(CreateButton(), 0, 0, colspan: 2, hexpand: true);

        // Show window
        window.Show();
    }

    private Button CreateButton()
    {
        var button = new Button("Hello, world!");
        {
            Margin = new WidgetSpacing(10, 5, 10, 5)
        };

        button.Clicked += (sender, e) =>
        {
            MessageDialog.ShowMessage("Hello, Xwt!");
        };

        return button;
    }
}
```

Kod z Listingu 2 działa następująco: w ramach konstruktora klasy MainWindow tworzona jest nowa instancja klasy Xwt.Window. Ta ostatnia jest abstrakcyjną reprezentacją okna w Xwt. Jej właściwości i metody pozwalają kontrolować sposób wyświetlanego i działania okna. W tym przykładzie ograniczam się jedynie do ustawienia tytułu oraz rozmiarów okna. W kolejnym etapie wiążę anonimową metodę ze zdarzeniem Closed okna, aby zakończyć działanie aplikacji (Xwt.Application.Exit) po zamknięciu okna.

Aby uzupełnić okno o przycisk, stworzyłem instancję klasy Xwt.Table (jest ona następnie przepisywana do właściwości Content instancji klasy Window). Obiekt typu Table umożliwia

rozmieszczanie kontrollek w sposób tabelaryczny. Kontrolki dodaje się do tabeli za pomocą metody Add instancji klasy Table. Jak pokazano w Listingu 2, użycie metody Add sprowadza się do wskazania kontrolki, która ma zostać umieszczona w tabeli (pierwszy argument metody Add), a także pozycji w tabeli (drugi i trzeci argument metody Add). Dodatkowo możemy skonfigurować scalanie komórek tabeli (argumenty rowspan i colspan), a także zdefiniować sposób rozmieszczenia kontrolki w komórce (argumenty vexpand, hexpand, vpos i hpos). Ponadto metoda Add pozwala zdefiniować marginesy. W tym przypadku marginesy mają wartości domyślne.

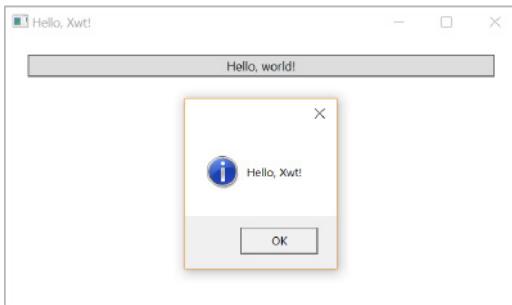
Samo tworzenie przycisku jest realizowane za pomocą metody CreateButton (dolina część Listingu 2). Metoda ta tworzy instancję klasy Xwt.Button, a następnie zmienia jego etykietę na „Hello, world!” i ustawia lewy, górny, prawy i dolny margines odpowiednio na 10, 5, 10 i 5 pikseli. W ostatnim kroku tworzona jest anonimowa metoda zdarzeniowa. Jej celem jest wyświetlenie okna z komunikatem „Hello, Xwt!” w przypadku, gdy użytkownik kliknie przycisk.

URUCHAMIANIE APLIKACJI

Aby przetestować działanie aplikacji, należy skorzystać z zaimplementowanej wcześniej metody XwtApp.Run w ramach statycznej metody Main klasy Program z projektu Xwt.Windows (Listing 3). Po wykonaniu tych zmian możemy uruchomić aplikację. Wynik jej działania powinien być analogiczny do przedstawionego na Rysunku 4.

Listing 3. Uruchamianie aplikacji Xwt

```
using Xwt.Common;
namespace Xwt.Windows
{
    class Program
    {
        [STAThread]
        static void Main(string[] args)
        {
            XwtApp.Run(ToolkitType.Wpf);
        }
    }
}
```



Rysunek 4. Aplikacja Xwt w trakcie działania

SUWAKI I DYNAMICZNA AKTUALIZACJA POZYCJI PROSTOKĄTA

Przejdziemy teraz do zaimplementowania funkcjonalności przesuwania prostokąta za pomocą suwaków. Prostokąt utworzymy za pomocą obiektu Xwt.Frame, a suwaki za pomocą Xwt.HSlider. Najpierw uzupełniamy deklarację klasy MainWindow o cztery pola, jeden typ wyliczeniowy oraz kilka metod pomocniczych (Listing 4).

Listing 4. Dodatkowe składniki klasy MainWindow

```
private Frame box;
private HSlider horizontalShiftSlider;
private HSlider verticalShiftSlider;

private int boxSize = 50;
private enum Orientation
{
    Horizontal, Vertical
}
private HSlider CreateSlider(Orientation orientation)
{
    var hSlider = new HSlider();

    hSlider.ValueChanged += (sender, e) =>
    {
        var slider = sender as Slider;
        if (orientation == Orientation.Horizontal)
        {
            box.MarginLeft = slider.Value;
        }
        else
        {
            box.MarginTop = slider.Value;
        }
    };
    return hSlider;
}

private Frame CreateBox()
{
    return new Frame()
    {
        BackgroundColor = Xwt.Drawing.Color.FromBytes(255, 0, 0),
        HeightRequest = boxSize,
        WidthRequest = boxSize,
        BorderWidth = 0
    };
}

private void UpdateSliderRanges()
{
    var horizontalRange = window.Content.WindowBounds.Right
        - window.Content.WindowBounds.Left - 2 * boxSize;
    var verticalRange = window.Content.WindowBounds.Bottom
        - verticalShiftSlider.WindowBounds.Bottom - 2 * boxSize;
    verticalShiftSlider.MinimumValue = -verticalRange;
    verticalShiftSlider.MaximumValue = verticalRange;
    horizontalShiftSlider.MinimumValue = -horizontalRange;
    horizontalShiftSlider.MaximumValue = horizontalRange;
}
```

Dodatkowe pola służą do przechowywania referencji do prostokąta (box) oraz suwaków (horizontalShiftSlider i verticalShiftSlider). Z kolei pole boxSize określa rozmiar prostokąta. Dla wygody zadeklarowaliśmy sobie typ wyliczeniowy Orientation. Posiada on dwie wartości: Horizontal i Vertical. Określają one kierunek przesuwania prostokąta. Wykorzystujemy je następnie w anonimowych metodach zdarzeniowych skojarzonych z suwakami (metoda CreateSlider z Listingu 4).

Za utworzenie suwaków oraz prostokąta odpowiedzialne są metody CreateSlider oraz CreateBox. Nie wymagają one jednak dłuższego komentarza, gdyż ograniczają się do utworzenia i skonfigurowania odpowiednich obiektów.

Dodatkowego komentarza wymaga jedynie metoda UpdateSliderRanges. Służy ona do dynamicznego określenia zakresów dla suwaków. Zakresy te są ustalane w taki sposób, aby prostokąt mógł poruszać się wyłącznie po powierzchni, zlokalizowanej pomiędzy suwakami a dolną krawędzią okna.

Mając te obiekty i metody pomocnicze, przechodzimy do zmodyfikowania konstruktora klasy zgodnie z Listingiem 5.

Listing 5. Zmodyfikowany konstruktor klasy MainWindow

```

public MainWindow()
{
    // Window
    window = new Window()
    {
        Title = "Hello, Xwt!",
        Width = 500,
        Height = 500
    };

    window.Closed += (sender, e) =>
    {
        Application.Exit();
    };

    // Layout
    var tableLayout = new Table();
    window.Content = tableLayout;

    // Button
    tableLayout.Add(CreateButton(), 0, 0, colspan: 2, hexpand: true);

    // Horizontal slider
    horizontalShiftSlider = CreateSlider(Orientation.Horizontal);
    tableLayout.Add(horizontalShiftSlider, 0, 1, hexpand: true);

    // Vertical slider
    verticalShiftSlider = CreateSlider(Orientation.Vertical);
    tableLayout.Add(verticalShiftSlider, 1, 1, hexpand: true);

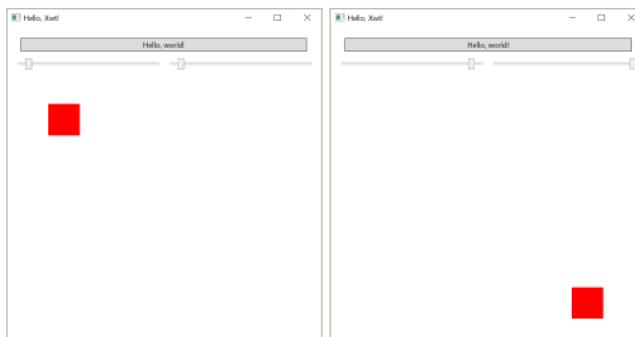
    // Box
    box = CreateBox();
    tableLayout.Add(box, 0, 2, colspan: 2, vexpand: true,
        vpos: WidgetPlacement.Center, hpos: WidgetPlacement.Center);

    // Show window
    window.Show();

    // Update sliders
    UpdateSliderRanges();
}

```

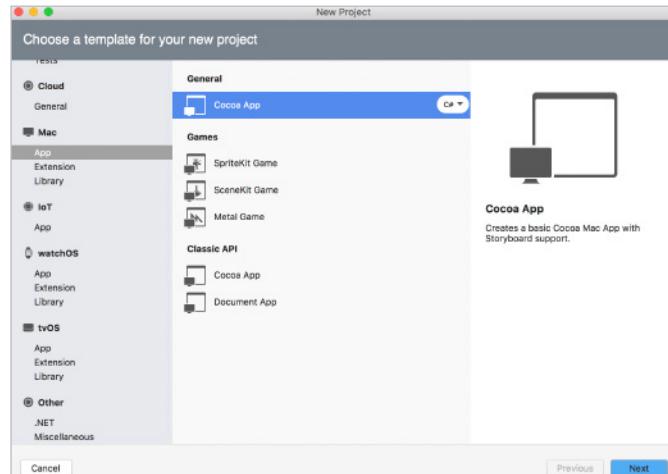
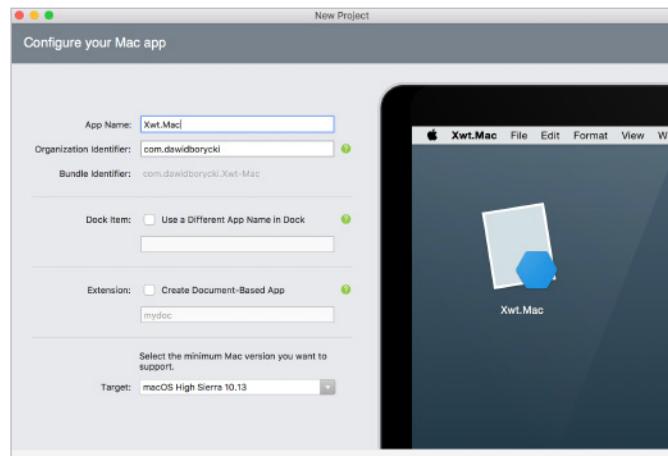
Powyższe zmiany spowodują uzupełnienie okna o dwa suwaki oraz prostokąt. Wobec tego po ponownym uruchomieniu aplikacji użyjemy efekt z Rysunku 5.

**Rysunek 5. Ostateczna postać aplikacji Hello, Xwt. Suwaki umożliwiają dynamiczne przesuwanie prostokąta**

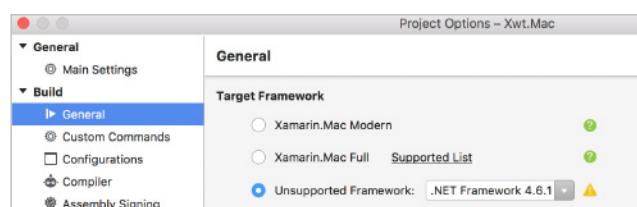
APLIKACJA DLA PLATFORMY MAC

W ostatnim kroku uzupełniamy rozwiązanie o projekt dedykowany platformie Mac. Na potrzeby tego przykładu skorzystamy z silnika Xwt.XamMac. Postępujemy następująco: otwieramy rozwiązanie Xwt.Sample w Visual Studio 2017 Community for Mac. Następnie w Solution Explorer klikamy prawym przyciskiem myszy Add->New Project.... Spowoduje to uaktywnienie kreatora New Project, w którym klikamy zakładkę Mac/App, po czym z listy dostępnych szablonów wybieramy Cocoa App C# i klikamy przycisk z etykietą Next (Rysunek 6). W kolejnym oknie kreatora zmieniamy nazwę aplikacji

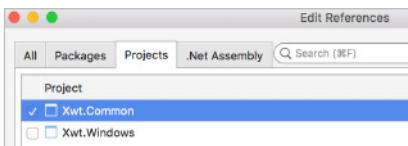
(App Name) na Xwt.Mac i ponownie klikamy przycisk Next (Rysunek 7). Spowoduje to wyświetlenie ostatniego już okna kreatora, w którym klikamy przycisk z etykietą Create.

**Rysunek 6. Tworzenie projektu Cocoa App****Rysunek 7. Konfigurowanie projektu**

Po utworzeniu projektu musimy wybrać docelowy framework, który będzie kompatybilny z Xwt. W tym celu w Solution Explorer klikamy prawym przyciskiem myszy Xwt.Mac, a następnie z menu kontekstowego wybieramy Options. W opcjach projektu klikamy zakładkę General, a w polu Target Framework zaznaczamy .NET Framework 4.6.1 (Rysunek 8).

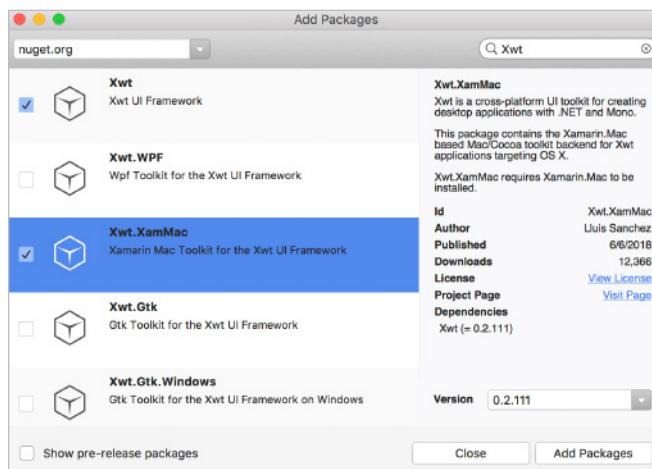
**Rysunek 8. Konfiguracja docelowego frameworku**

W kolejnym kroku ustawiamy referencję do projektu Xwt.Common. W tym celu w Solution Explorer klikamy prawym przyciskiem myszy pozycję References, a następnie wybieramy Edit References... Uaktywni to widok Edit References, w którym przechodzimy na zakładkę Projects, gdzie zaznaczamy Xwt.Common (Rysunek 9).



Rysunek 9. Edycja referencji

Możemy teraz zainstalować wymagane pakiety NuGet: Xwt oraz Xwt.XamMac. W tym celu w Solution Explorer klikamy prawym przyciskiem myszy pozycję *Packages* i wybieramy *Add packages...* z menu, które się pojawi. W efekcie nastąpi uruchomienie okna z Rysunku 10, w którym w polu wyszukiwania wpisujemy Xwt, a następnie zaznaczamy Xwt oraz Xwt.XamMac i klikamy przycisk z etyktą *Add Packages*.



Rysunek 10. Instalacja Xwt i Xwt.XamMac

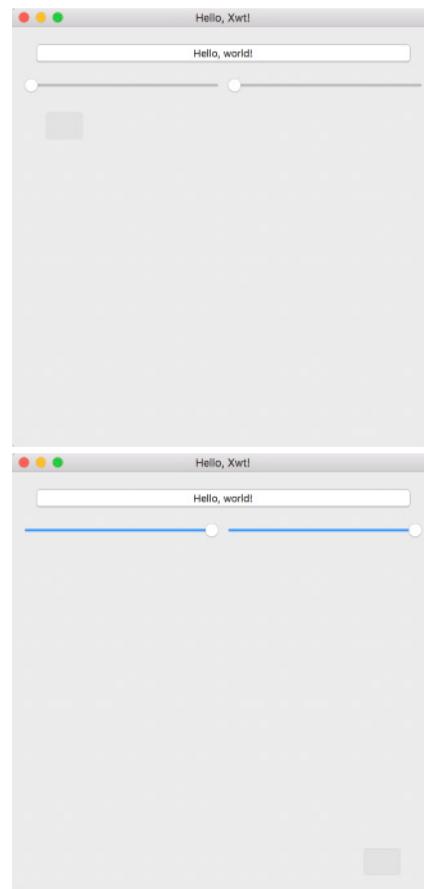
Ostatnią czynnością, którą musimy wykonać, jest wywołanie metody *XwtApp.Run*, co realizujemy w pliku *Main.cs* zgodnie ze wzorem z Listingu 6.

Listing 6. Punkt wejścia aplikacji Xwt.Mac

```
using Xwt.Common;
namespace Xwt.Mac
{
    static class MainClass
    {
        static void Main(string[] args)
        {
            XwtApp.Run(ToolkitType.XamMac);
        }
    }
}
```

Po wykonaniu tych zmian uruchamiamy aplikację. Powinniśmy uzyskać efekt podobny do przedstawionego na Rysunku 11. Jak widać, aplikacja wygląda analogicznie jak w poprzednim przypadku. Jedynym problemem jest to, że kolor prostokąta nie uległ zmianie (nie jest czerwony jak w przypadku Windows). Jednak pozostałe funkcjonalności działają poprawnie, co potwierdza możliwości Xwt w zakresie tworzenia wieloplatformowych aplikacji desktopowych.

W trakcie uruchamiania aplikacji zauważymy również, że oprócz naszego okna wyświetlane jest jeszcze jedno. Pochodzi ono ze szeniorusu, utworzonego z wykorzystanego szablonu. To zbędne okno można łatwo wyłączyć poprzez usunięcie klucza *NSMainStoryboardFile* z pliku *Info.plist*.



Rysunek 11. Aplikacja w trakcie działania

PODSUMOWANIE

W tym artykule pokazałem, w jaki sposób wykorzystać technologię Xwt do utworzenia wieloplatformowej aplikacji desktopowej. Proces tworzenia takiej aplikacji jest podobny jak w przypadku aplikacji mobilnych Xamarin. Mianowicie mamy jeden wspólnie dostępny projekt, w którym implementujemy UI. Ten projekt jest wykorzystywany w projektach dedykowanych konkretnym platformom (np. Windows i Mac). Funkcjonalność tych projektów ogranicza się jedynie do wskazania i uruchomienia silnika Xwt, który będzie mapował polecenia ze zunifikowanego API na kontrolki specyficzne dla danej platformy.

Chociaż, jak się przekonaliśmy, nie wszystko w Xwt działa poprawnie, to jednak ta technologia wydaje się być dobrym rozwiązaniem dla programistów C#, którzy chcą tworzyć wieloplatformowe aplikacje desktopowe w oparciu o standardy wypracowane dla aplikacji mobilnych.



DAWID BORYCKI

Naukowiec, programista, autor wielu książek, amerykańskich patentów, artykułów i videotutoriali o programowaniu w różnych technologiach. Pierwszy polski autor w Microsoft Press i MSDN Magazine. Z wykształcenia doktor fizyki teoretycznej. Obecnie pracuje w Polskiej Akademii Nauk, gdzie zajmuje się rozwojem nowoczesnych, nieinwazyjnych urządzeń do analizowania mikroprzepływu krwi w mózgu i obrazowania przez ośrodkie nieprzezroczyste optycznie.



Zapraszamy na autorskie szkolenia
z zakresu **bezpieczeństwa IT**

- { Bezpieczeństwo aplikacji WWW }
- { Offensive HTML, SVG, CSS and other Browser-Evil }
- { Wprowadzenie do bezpieczeństwa IT }
- { Szkolenie przygotowujące do egzaminu CEH
(Certified Ethical Hacker) }

www.securitum.pl/oferta/szkolenia

Patroni medialni: sekurak.pl



rozwal.to



Architektura testów automatycznych dla wielomodułowej aplikacji webowej

Większość aplikacji webowych pisanych jest przez jednego dostawcę, który zajmuje się także automatyzacją testów, przez co ich rozwój i utrzymanie przezewnętrzny, kilkuosobowy zespół nie stanowi problemu, pod warunkiem umiejętności posługiwania się systemem kontroli wersji.

Jedna z polskich firm zleciła jednak stworzenie aplikacji kilku zewnętrznym firmom. Każda z nich była odpowiedzialna za napisanie i przetestowanie jednego lub kilku modułów aplikacji. Jednocześnie klient chciał mieć kontrolę nad testami automatycznymi tworzonymi przez firmy w ramach poszczególnych modułów, uruchamiania ich oraz budowania z nich własnych testów typu *end to end* zgodnie z zasadą, że grupa działających niezależnie modułów nie gwarantuje jeszcze prawidłowo działającej aplikacji jako całości. Poniżej przedstawiam case study rozwiązania, które zapewniło powyższe wymagania.

Framework testów automatycznych został oparty na następujących technologiach:

- » Java, jako język programowania,
- » Junit, jako runner testów,
- » WebDriver, jako biblioteka pozwalająca na interakcję z przeglądarką,
- » oraz Maven jako sposób budowania projektu.

Framework został stworzony przez komórkę QA klienta. Zakładał pisanie testów automatycznych zgodnie ze wzorcem Page Object Pattern, a jego główne klasy bazowe: *AbstractTest*, *AbstractPage*, oraz klasy pozwalające na raportowanie wykonania testów do Jiry, zostały wrzucone do repozytorium binariów – Artifactory, jako „core” projektowy.

STRUKTURA PROJEKTU

Jako zasadę przyjęto standardowe nazewnictwo pakietów dla projektów testów we wszystkich modułach w formacie: eu.nazwa_domeny.nazwa_modulu oraz identyczny podział na foldery źródłowe:

- » folder *src/main/java* – klasy stron Page Object Pattern, konfiguracje, obiekty danych,
- » folder *src/main/resources* – pliki konfiguracyjne, dane testowe,
- » folder *src/test/java* – testy, suity testowe.

Wspólna struktura w połączeniu z korzystaniem z projektu „core” pozwoliła na standaryzację wszystkich projektów, co umożliwiło bezproblemową kontrolę i analizę testów przez komórkę QA klienta, oraz współdzielenie pracy pomiędzy firmami – ich opis można znaleźć poniżej.

JAVADOCS

Kolejnym działaniem mającym ułatwić współpracę było postawienie przez klienta wymogu opisywania wszystkich metod w klasach

stron, które realizują biznesowe akcje, za pomocą komentarza zgodnego z JavaDocs. Nakład przeznaczony na tworzenie i utrzymywanie komentarzy zwraca się, gdy członek któregoś z pozostałych zespołów projektowych chce użyć metody z innego modułu. Dzięki komentarzom podczas wywołania metody, IDE pokazuje adnotacje w przystępnej formie.

Według szablonu komentarze powinny zawierać następujące pola:

- » opis,
- » entry point – w jakim stanie musi znajdować się moduł, żeby można było użyć metody,
- » exit point – w jakim stanie metoda zostawia moduł,
- » @param – parametry wejściowe metody,
- » @return – jaki obiekt zwraca metoda.

Choć taki zapis może się wydawać niepotrzebny dla prostych metod, stanowi on spore ułatwienie w przypadku metod bardziej skomplikowanych. Szczególnie takich, których działania biznesowe nie możemy w całości określić w samej nazwie. Pamiętajmy, że aplikacja składała się z wielu modułów, z których każdy realizuje inne akcje biznesowe.

FORMATTER KODU

Uzupełnieniem podejścia było stworzenie formattera kodu, który był używany w IDE przez członków wszystkich projektów. Dzięki identycznym długościom linii i obsłudze wcięć różnicę przy analizie zmian w kodzie są efektem rzeczywistych modyfikacji samego kodu, a nie różnej ilości lub wielkości wcięć w klasach.

PAGE OBJECT PATTERN „DO KWADRATU”

Wzorzec Page Object Pattern zakłada odseparowanie klas stron (zawierających lokatory do kontrolek na danej stronie oraz metody, które można na nich wykonać) od samych testów automatycznych, co ma ułatwić dodawanie nowych testów, oraz utrzymanie w przypadku zmian w aplikacji. Ustandaryzowanie projektów pisanych we wszystkich zespołach zgodnie z tym, co opisałem powyżej, umożliwiło odseparowanie klas stron nie tylko od kodu testów, ale również od pojedynczego projektu testów automatycznych.

Działające klasy stron, znajdujące się w folderze źródłowym *src/main/java*, zostają przez zespoły zdeployowane do Artifactory i gotowe do pobrania i użycia przez pozostałe zespoły, a także komórkę QA klienta.

Warto tutaj zaznaczyć, że tworzenie aplikacji przez kilka niezależnych scrumowych zespołów powodowało, że w danej chwili każdy z modułów mógł być na danym środowisku integracyjnym w innej wersji. Dlatego projekty testów automatycznych miały być numerowane zgodnie z wersjami danego modułu, a odpowiednio sterując wersjami zaciąganych modułów w *pom.xml*, można z łatwością pobrać klasy stron dla danego modułu w odpowiedniej wersji.

PRZYKŁADY ZASTOSOWANIA

Dobrym przykładem pokazującym przydatność takiego rozwiązania jest wysłanie do Artifactory klasy strony logowania zawierającej metody pozwalające na poprawne zalogowanie się do aplikacji.

W efekcie inżynierowie testów z innych projektów nie muszą kodować własnych metod logowania, a tylko załączają do swojego projektu odpowiedni pakiet z Artifactory – takie rozwiązanie pozwala oszczędzić czas potrzebny na ich zakodowanie i utrzymanie oraz zmniejsza duplikację kodu.

Przykład 1. Projekt testów modułu nawigacji po serwisie – Nav

W *pom.xml* projektu testów tego modułu zawarta jest zależność do strony modułu logowania w wymaganej wersji:

```
<dependency>
  <groupId>eu.firma.login</groupId>
  <artifactId>UI-tests</artifactId>
  <version>3.4.5</version>
</dependency>
```

W klasie testu modułu Nav importujemy klasę modułu *LoginPage*:

```
import eu.firma.login.webpages.LoginPage;
```

Następnie w @Before naszego testu za pomocą metody zaimportowanej klasy logujemy się do serwisu, przechodząc do naszej strony nawigacyjnej:

```
LoginPage navPage = LoginPage.login(username, password);
```

Po tym wywołaniu jesteśmy zalogowani w serwisie na jego głównej stronie i możemy wykonywać akcje na stronie nawigacyjnej.

Ponadto z racji tego, że każdy moduł jest pisany przez inną firmę, automatyzacja testów może się wiązać z koniecznością zastosowania różnych praktyk radzenia sobie z problemami synchronizacji lub oczekiwania na pojawienie się elementu na stronie. Różnice te mogą być tym większe, im większa jest różnorodność technologiczna pomiędzy modułami. Współdzielając gotowe klasy stron, inżynierowie automatyzujący jeden moduł nie muszą ponownie rozwijać problemów z automatyzacją w innej technologii.

Strona logowania to bardzo prosty przykład przygotowania stanu aplikacji do testów któregoś z modułów i zapewne nie nastręczałyby problemów ze zrozumieniem biznesowego procesu, jednak część modułów potrzebuje stanu aplikacji, który można uzyskać jedynie poprzez wykonanie bardziej skomplikowanych akcji biznesowych w innych częściach aplikacji. Zespół automatyzujący dany moduł jest odpowiedzialny za dostarczenie do Artifactory metod poprawnie działających pod względem technicznym i biznesowym. Odpowiada także za ich utrzymanie. Opisanie zgod-

ne z JavaDocs umożliwiło łatwe zorientowanie się, której metody potrzebujemy, aby ustawić żądaną stan aplikacji, i jak jej użyć.

Przykład 2. Projekt testów modułu faktur – Invoicing

Jeżeli do rozpoczęcia testów naszego modułu w systemie musimy przejść przez kilka stron, załączamy wszystkie projekty, na których się znajdują w *pom.xml*:

```
<dependency>
  <groupId>eu.firma.login</groupId>
  <artifactId>UI-tests</artifactId>
  <version>3.4.5</version>
</dependency>
<dependency>
  <groupId>eu.firma.nav</groupId>
  <artifactId>UI-tests</artifactId>
  <version>3.0.1</version>
</dependency>
```

A następnie, po importie w klasie naszego testu:

```
import eu.firma.login.webpages.LoginPage;
import eu.firma.login.webpages.NavPage;
wywołujemy ich metody:
LoginPage navPage = LoginPage.login(username, password);
Invoicing invoicing = navPage.navigateTo(Modules.Invoicing);
```

I dopiero wtedy zaczynamy prawdziwe testy naszego modułu. Warto zauważać, że metody wykorzystane do tego momentu nie były tworzone przez nas.

Ponadto niektóre zespoły wystawiały do Artifactory, oprócz klas stron swojego modułu, również aplikacje klienckie API pozwalające na przygotowanie lub manipulację danymi testowymi, co dodatkowo upraszcza proces ustawiania aplikacji w żądanym dla danego testu stanie.

W praktyce najbardziej rozpowszechnionymi klasami, zaimportowanymi we wszystkich projektach, były klasy strony logowania oraz nawigacji.

Kolejną korzyścią z zastosowania takiego podejścia jest możliwość zautomatyzowania testów *end to end* przez komórkę QA klienta, wykorzystując jedynie metody zakodowane przez poszczególne zespoły w ramach swoich modułów. Logowanie i nawigacja mogą być w takich testach rozszerzone o kolejne metody biznesowe poszczególnych modułów realizujące biznesowy przypadek przechodzący przez całą aplikację.

Przykład 3. Testy e2e w aplikacji

Test wystawiania faktur przechodzi przez moduły logowania, nawigacji, zamówień i faktur. Wszystkie one importujemy w *pom.xml*, podobnie jak w poprzednich przykładach.

Importujemy w klasie naszego testu fakturowania:

```
import eu.firma.login.webpages.LoginPage;
import eu.firma.login.webpages.NavPage;
import eu.firma.login.webpages.Orders;
import eu.firma.login.webpages.Invoicing;
```

W @Before testu logujemy się i przechodzimy do modułu zamówień:

```
LoginPage navPage = LoginPage.login(username, password);
OrdersPage ordersPage = navPage.navigateTo(Modules.Orders);
```

A następnie korzystając z metod przygotowanych przez zespół testujący moduł zamówień i fakturowania, przeprowadzamy nasz test:

```
String orderId = ordersPage.createDefaultOrder();
InvoicingPage invoicingPage = navPage.navigateTo(Modules.Invoicing);
invoicingPage = invoicingPage.searchInvoiceByOrderId(orderId);
assertTrue(invoicingPage.verifyDefaultInvoiceData());
```

KONFIGURACJA ARTIFACTORY

Aby zapewnić bezpieczny dostęp do zasobów, należało skonfigurować uprawnienia dla korzystających z niego użytkowników.

Każdy z zespołów posiadał dwa typy użytkowników. Pierwszy to użytkownik techniczny, który ma uprawnienia do pobierania wszystkich paczek z Artifactory oraz do wysyłania nowych (z nowym numerem). Ze względu na ograniczone uprawnienia ten użytkownik może być stosowany w automatycznych zadaniach budowania projektu w CI. Drugi typ użytkownika to użytkownik oparty o firmowy serwer LDAP, który oprócz odczytywania i wysyłania paczek może kasować i podmieniać paczki w repozytorium.

Obydwa rodzaje użytkowników mają ograniczone uprawnienia wysyłania, kasowania i podmianiania paczek tylko do gałęzi własnego projektu. Ma to swoje odzwierciedlenie w nazwie pakietu:

```
eu/nazwa_domeny/nazwa_modulu /**
```

Dzięki takiemu rozwiązaniu nie ma możliwości wgrania i modyfikacji paczki w nieswoim projekcie.

Jednoznaczna identyfikacja modułu danego projektu w następujący sposób:

```
<groupId>eu.nazwa_domeny.nazwa_modulu </groupId>
<artifactId>UI-tests</artifactId>
<version>0.0.1</version>
```

umożliwia łatwe przeglądanie Artifactory w poszukiwaniu interesującego nas modułu.

DOBRE PRAKTYKI AUTOMATYZACJI

Uzupełnieniem powyższych wytycznych przekazanych przez klienta do pozostałych firm było utworzenie dokumentu dobrych praktyk automatyzacji, którym miały się podporządkować wszystkie zespoły. Były to między innymi:

WRAPPERY NA AKCJE WEBDRIVERA

Aplikacje webowe zazwyczaj zawierają elementy, które asynchronicznie pobierają dane już po załadowaniu całej strony. Obsługa tego typu elementów z wykorzystaniem WebDrivera może być problematyczna. Aby dać możliwość dowolnego rozszerzania metod, które wywołują akcje w przeglądarce, na przykład poprzez wprowadzenie dynamicznego oczekiwania na elementy przed kliknięciem, wszystkie akcje WebDrivera są wywoływane przez metody wrappujące z pakietu „core”.

Metody klikania, wysyłania i pobierania tekstu z pól, jak również wybierania wartości z kontrolek znajdują się w „corowym” AbstractPage'u. Dzięki temu, że klasa ta jest dziedziczona przez wszystkie pozostałe klasy stron, każda z metod ma do nich dostęp. Zamiast więc stosować metodę WebDrivera do klikania w element loginButton.click() wywołujemy naszą metodę z AbstractPage'a: click(loginButton).

W jej implementacji przed właściwym kliknięciem może być zawarte wymienione wcześniej oczekiwanie na pojawienie się i aktywację danego elementu, albo na przykład sprawdzenie, czy zniknął już loader (spinner) aplikacji.

Przykład 4. Metoda click z AbstractPage:

```
public void click(WebElement element){
    waitForLoader();
    waitForElementClickable();
    element.click();
}
```

NIEZALEŻNOŚĆ TESTÓW

Chcąc umożliwić szybkie retesty pojedynczych funkcjonalności aplikacji (na przykład po naprawieniu błędu lub zmianach w module) oraz pozwolić na równoległe uruchamianie wielu testów, konieczne jest zadbanie o to, aby wszystkie testy były niezależne od siebie.

Wprowadziłem zasadę, zgodnie z którą żaden test nie może korzystać z danych lub stanu aplikacji pochodzących z innego testu. Każdy z nich musi sam pobrać dane testowe, z których tylko on będzie korzystał. Zasada ta pozwala na uruchomienie dowolnego testu na dowolnym środowisku oraz na wyłączenie dowolnej ilości testów równocześnie, co znacznie skraca czas uruchamiania całości suity testowej.

DYNAMICZNE OCZEKIWANIE

Absolutny zakaz stosowania bezwzględnego zatrzymania testu, który wpływa na niestabilność testów oraz wydłuża czas ich trwania. Wszystkie oczekiwania na elementy aplikacji, lub konkretny stan, powinny być dynamiczne (fluent bądź explicit wait).

ZALETY ROZWIĄZANIA

Zastosowanie szablonu uwzględniającego dobre praktyki kodowania oraz dzielenia się klasami stron pomiędzy zespołami projektowymi umożliwia ścisłe rozgraniczenie odpowiedzialności za części testów automatycznych oraz skupienie się na automatyzacji własnych modułów. W razie konieczności przeprowadzenia części procesu biznesowego w innym module uruchamiamy jedynie metodę przygotowaną przez odpowiedni zespół, co pozwala ograniczyć duplikację kodu do minimum.

Jeżeli konieczne są zmiany w danym module, zespół automatyzujący konkretny obszar wie od razu, czego i w jakiej wersji się spodziewać. Dzięki temu jest w stanie szybko dopasować do niej klasy stron i wysłać do Artifactory ich działające wersje z odpowiednim numerem.

WADY ROZWIĄZANIA

Jedną z zasad wzorca projektowego Page Object Pattern jest zwracanie przez wszystkie metody klas stron obiektów, na których kończy się dana akcja. Powoduje to, że praktycznie wszystkie klasy stron zawierają metody, które zwracają obiekty stron z innych modułów, czego efektem są wzajemne zależności w pakietach. Ponadto każdy z projektów może zatrzymać w swoim pom.xml niezbędne mu do testów biblioteki w konkretnej wersji. Jeżeli my również (albo pakiet „core”) używamy tej biblioteki, ale w innej wersji,

zaciągnięcie takiego pakietu do swojego projektu może spowodować znany problem „Jar Hell”, którego rozwiązanie (na przykład za pomocą <exclude> w pom.xml) może nastręczyć bardzo dużo kłopotów, czego doświadczyliśmy w praktyce.

PODSUMOWANIE

Zastosowanie powyższego rozwiązania pozwoliło klientowi na łatwą kontrolę prac prowadzonych w ramach automatyzacji testów we wszystkich zespołach scrumowych dostarczających moduły zamówionej aplikacji. Idea współdzielenia się klasami stron poza projektami znacznie ograniczyła konieczność duplikowania kodu oraz umożliwiła komórce QA klienta pisanie własnych testów *end-to-end* na ich podstawie.

Wprowadzanie takiego rozwiązania pokazało, że potrzebna jest ścisła współpraca pomiędzy klientem a dostawcami, także z dostawcami między sobą. Na okresowych spotkaniach należy rozstrzygać niejasności pojawiające się na poziomie zapisu kodu klas, aby nie odchodzić od obowiązującego standardu.

Framework w projekcie był wprowadzany stopniowo. Najpierw nastąpiło współdzielenie stron pomiędzy niektórymi zespołami, a następnie uwspólnienie części „core” projektu. Ostatnim etapem było wpinanie go do kolejnych zespołów. Takie podejście spowodowało duże problemy z kompatybilnością używanych przez zespoły bibliotek oraz wzajemnymi zależnościami klas. Doświadczenie to pokazuje, że rozwiązanie powinno być wprowadzone na początku życia projektu, a jeżeli to niemożliwe, to na zasadzie „big-bang”, szczególnie w obszarze „core”.

PIOTR GRZESIAK

Test Automation Engineer w IT Kontrakt. Ma ponad 8-letnie doświadczenie w pisaniu testów automatycznych dla aplikacji webowych i desktopowych. Pracował m.in. w technologiach: Java, Webdriver, SoapUI, ALM, Robotframework. Dwukrotny prelegent konferencji TestWarez.

reklama

Obserwuj nas na Twitterze



@PROGRAMISTAMAG



Współczesne architektury aplikacji biznesowych. Reactive i serverless

Ten cykl artykułów ma na celu dokonać przeglądu różnych trendów architektonicznych, które pojawiły się w ciągu ostatnich kilku lat, po to aby je uporządkować, zestawić ze sobą, wskazać główne powody zastosowania, jednocześnie układaając je w ewolucyjną ścieżkę, którą może podążać system na tle zmian architektonicznych. Poprzednio analizowaliśmy klasyczną architekturę trójwarstwową, Domain-Driven Design, Ports and Adapters oraz microservices. W niniejszym artykule zajmiemy się architekturami reactive oraz serverless.

ARCHITEKTURA REACTIVE

Architektura reactive to architektura oparta o asynchroniczną wymianę komunikatów (messaging) pomiędzy niezależnymi procesami, obecnie najczęściej będących microservice'ami. W dużym stopniu zainspirowane przez The Reactive Manifesto¹, które zachęca do rozwijaniu systemów, które są:

- » responsywne (ang. *responsive*) – są w stanie obsłużyć przychodzące żądania w krótkim czasie,
- » odporne (ang. *resilient*) – są w stanie obsłużyć żądania, nawet w przypadku awarii,
- » elastyczne (ang. *elastic*) – są w stanie obsługiwać żądania, nawet przy dużym obciążeniu,
- » oparte o wymianę komunikatów (ang. *message driven*) – komunikacja odbywa się asynchronicznie z użyciem komunikatów.

O ile samo pojęcie architektury reaktywnej jest abstrakcyjne, tzn. nie określa konkretnego sposobu wykonania postawionych przed nią założeń, to najczęściej jest ona implementowana jako system rozproszyony oparty o zdarzenia przesyłane z użyciem serwerów kolejek (ewentualnie realizowany z użyciem koncepcji aktorów). W niniejszym artykule skupimy się na wersji zdarzeniowej (event-driven).

Ponieważ w zakresie nazewnictwa w tym obszarze pojawia się bardzo wiele nieporozumień, zdefiniujemy kluczowe pojęcia:

- » komunikat (ang. *message*) – to forma, w jakiej przesyłane dane pomiędzy różnymi procesami (pewnego rodzaju nośnik), która służy do przekazywania zawartości typu zdarzenie lub komenda; w kontekście architektury reaktywnej – do przesyłania komunikatów potrzebna jest odpowiednia infrastruktura, tj. serwer kolejki komunikatów (ang. *message queue*), np. RabbitMQ, Kafka;
- » zdarzenie (ang. *event*) – informacja nt. tego, co się wydarzyło w systemie, fizycznie jest to najczęściej struktura danych przesyłana z użyciem systemu messagingowego; nadawca nie kieruje zdarzenia do konkretnego odbiorcy, a raczej ogłasza je; zdarzenie może mieć wielu odbiorców. Przykłady zdarzeń: złożono zamówienie, dodano użytkownika, zamknięto konto;
- » komenda (ang. *command*) – informacja odnośnie żądania przesyłana do innego procesu, które ma wywołać działanie lub zmianę; komenda jest kierowana do konkretnego odbiorcy, w związku z tym odbiorca jest jeden. Przykłady: wystaw fakturę, wygeneruj raport, zamknij konto.

Ponadto często mylnie używa się zamiennie terminów: „architektura reaktywna” oraz „programowanie reaktywne”. Drugi z wymienionych konceptów dotyczy bardziej niskopoziomowych mechanizmów na poziomie języka lub bibliotek, które umożliwiają asynchroniczne wykonywanie metod.

Dla przykładu założmy, że tworzymy system w stylu microservices, przy czym komunikacja pomiędzy poszczególnymi częściami systemu nie odbywa się bezpośrednio poprzez zdalne wywołania (np. REST), a z użyciem komunikatów. Nadawca jest w ten sposób odseparowany od odbiorcy, moment wygenerowania zdarzenia i odpowiedzi na nie muszą być ze sobą powiązane. Przykład realizacji żądania w tym podejściu przedstawiony został na Rysunku 1.

Klient chcąc zainicjować złożenie zamówienia, wysyła zdarzenie Order Event, które następnie jest przetwarzane przez usługę Order Service. Poprawne przetworzenie zamówienia powoduje wygenerowanie kolejnego zdarzenia Order Created Event, które jest dalej rozsyłane. Jego odbiorcą są usługi: Delivery Service, Customer Points Service, Customer Notification Service, Inventory Service. Jedna z usług Delivery Service wskutek przetworzenia tego zdarzenia generuje kolejne: Tracking Service.

Architektura reactive to architektoniczne promowanie luźnych zależności. Części systemu można niezależnie skalować, niezależnie rozwijać, uruchamiać i zamykać (np. wprowadzając nową wersję), gdyż są niezależnymi aplikacjami i komunikują się poprzez serwer komunikatów. W zasadzie zbliżamy się do inżynierijnego raju. Jednak nie ma nic za darmo, ponieważ tego typu rozwiązania są niezwykle trudne w implementacji właśnie m. in. z powodu nie bezpośredni komunikacji. Poza tym musimy opracować strategie radzenia sobie m.in. z następującymi kwestiami:

- » jak będziemy reagować na sytuacje wyjątkowe (odpowiednik obsługi wyjątków);
- » jak będziemy obsługiwać scenariusze o charakterze transakcyjnym (klasyczne transakcje typu ACID w środowisku rozproszonym są niezwykle kosztowne);
- » jak będziemy odtwarzać kolejność wysyłania komunikatów (ze względu na kwestie wydajnościowe musimy liczyć się z brakiem gwarancji zachowania kolejności);
- » jak zabezpieczymy się w sytuacjach, gdy ten sam komunikat zostanie dostarczony do odbiorcy więcej niż raz.

Podejście typu reactive jest komplementarne z innymi opisanymi w cyklu artykułów strategiami:

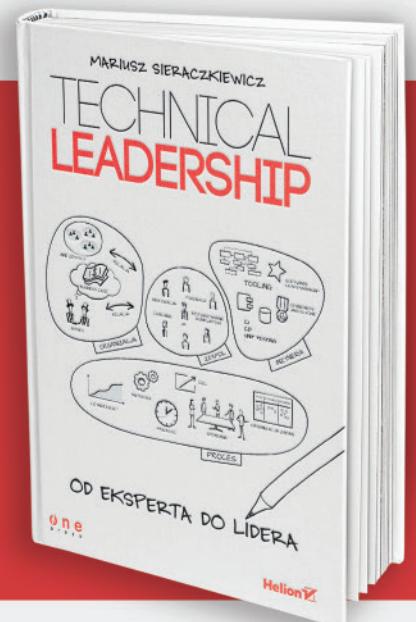
- » Domain-Driven Design podsuwa pomysły na strategie organi-

1. <https://www.reactivemanifesto.org>

BNS IT - SZKOLENIA OTWARTE

WARSZAWA / 04-06.07.2018
TECHNICAL LEADERSHIP™
ROLA LIDERA TECHNICZNEGO

1. Rola lidera technicznego
2. Motywacja własna i innych
3. Ludzie
4. Zespół
5. Kompetencje lidera



ŁÓDŹ / 11-13.06.2018
WZORCE PROJEKTOWE
I REFAKTORYZACJA DO WZORCÓW

1. Wprowadzenie do wzorców projektowych
2. Jakość kodu źródłowego
3. Refaktoryzacja
4. Wzorce GoF

WARSZAWA / 11-13.07.2018
NOWOCZESNE
ARCHITEKTURY APLIKACJI

1. Wprowadzenie do pojęcia architektura oprogramowania
2. Domain-Driven Design
3. Micro Services
4. Ports & Adapters
5. Clean and Onion Architecture
6. Reactive Architecture
7. Serverless Architecture

P O Z O S T A Ł E T E R M I N Y :

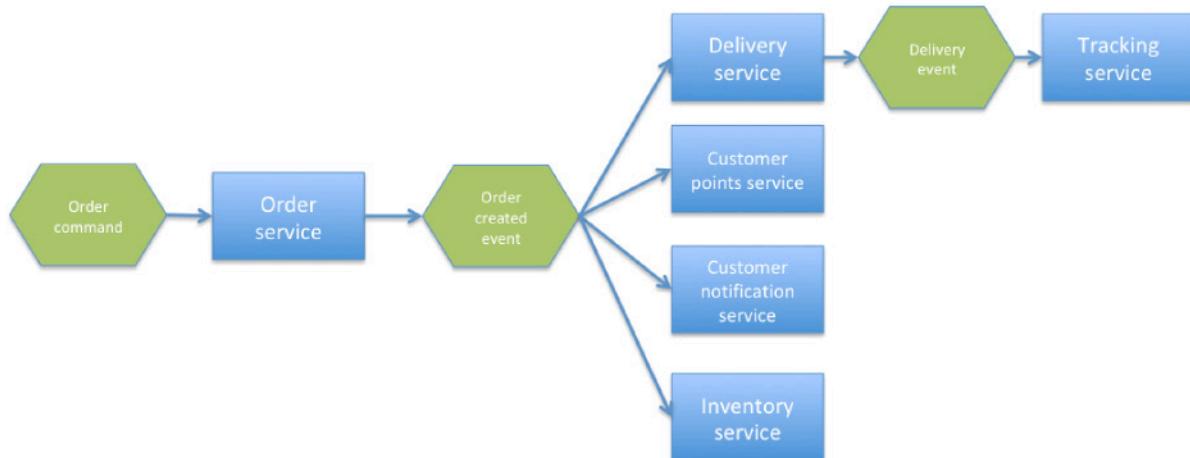
Zbieranie wymagań i współpraca z klientem

Łódź 14-16.11.2018 2100,00 PLN

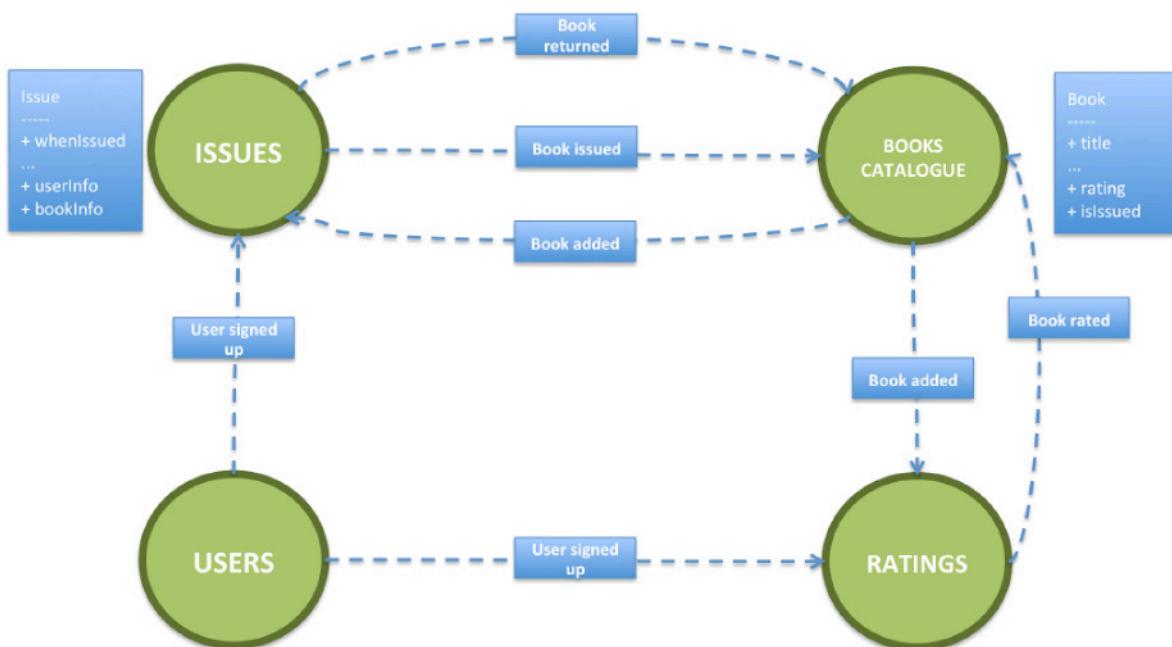
Wzorce projektowe i refaktoryzacja do wzorców

Łódź 17-19.12.2018 2100,00 PLN

CENY NETTO



Rysunek 1. Przykład realizacji żądania z użyciem zdarzeń i komend



Rysunek 2. Przykładowa komunikacja z użyciem zdarzeń

zowania komunikacji z użyciem obiektów zdarzeń i komend, ich wysyłania oraz obsługi z użyciem command handlerów i event handlerów;

- » architektura Ports and Adapters umożliwia łatwą i jednocześnie przezroczystą dla aplikacji implementację komunikacji poprzez komunikaty, obiekt publikujący zdarzenia to adapter wyjściowy, a obiekt odbierający to adapter wejściowy;
- » w podejściu microservices zdarzenia są jednym ze sposobów komunikacji między usługami;
- » podejście serverless jest jednym ze sposobów zrealizowania architektury reactive.

Gdybyśmy chcieli zrealizować wspomniany w pierwszym artykule tej serii system do wypożyczania książek z użyciem podejścia reactive, komunikacja mogłaby wyglądać następująco:

Wady i zalety

Architektura reactive umożliwia niemal dowolne możliwości skalowania i niezależność w rozwoju konkretnych składowych aplikacji. Obec-

nie istnieje wiele wysokowydajnych narzędzi, również open-source, które umożliwiają tworzenie rozwiązań w tej architekturze, m.in.:

- » RabbitMQ, Kafka, Celery, ActiveMQ, Kestrel jako przykłady systemów kolejkowych przesyłających komunikaty,
- » Akka, Netty, Orleans jako przykłady bibliotek ułatwiających tworzenie systemów reactive.

Wadą rozwiązania jest duży poziom skomplikowania tego typu podejścia – zasadniczo nie należy go stosować jako głównego stylu architektonicznego, jeśli wysoka skalowalność czy niezależny rozwój części systemu nie jest kluczowym wymaganiem wobec systemu. Tego typu podejście wymaga również bardzo kompetentnego zespołu, który tworzył już aplikacje tego typu, m.in. dlatego, iż dekompozycja aplikacji i zaprojektowanie komunikacji wymagają specyficznych technik dedykowanych tej architekturze.

SERVERLESS

Ostatnią z przedstawionych architektur jest serverless, która jest jedną z najnowszych mysił tworzenia systemów IT, będąca najbardziej

posuniętą koncepcją tworzenia aplikacji w chmurze i architektury microservices w stylu reaktywnym. W tym przypadku wdrażając kod rozwiązania, nie zastanawiamy się nad konfiguracją maszyny czy też systemu operacyjnego, ani konfiguracją kontenera czy serwera aplikacyjnego. Po prostu wdrażamy kod bezpośrednio na serwer, infrastruktura nas nie interesuje. To platforma dostarczająca rozwijanie serverless zadba o to, aby powstał i uruchomił się odpowiedni kontener wykonujący nasz kod oraz jeśli trzeba – przeskalała.

Pierwotnie serverless powstał z myślą o systemach, które nie potrzebują ciągle aktywnych zasobów, a operacje pojawiają się nieregularnie lub niezbyt często. Dobrym przykładem mogą być systemy IoT, gdzie pojedynczy czujnik może wzbudzać się, odczytywać dane raz na godzinę i wtedy wysyłać dane pomiaru do chmury. Ostatnio jednak coraz częściej zaczyna się myśleć o zastosowaniu serverless do tworzenia aplikacji biznesowych.

System oparty o serverless tworzymy jako zbiór powiązanych usług zewnętrznych (Backend as a Service) takich jak Auth0 (Single Sign On), chmurowych baz danych (np. Firebase) oraz funkcje wdrażanych bezpośrednio na serwer (ang. *Function as a Service* – FaaS). Architektura ta jest ze swojej natury oparta o zdarzenia. Wywołanie funkcji następuje wskutek zdarzenia (np. wywołania REST API, czasu, odebranego komunikatu, zmiany w pliku). Serwery, a w zasadzie kontenery z serwerami, nie muszą być cały czas w stanie aktywnym, wzbudzane są jedynie w momencie wywołania określonego zdarzenia (np. wywołania REST). Jednym z bardziej popularnych rozwiązań tego typu jest AWS Lambda.

Oczywiście sama nazwa serverless nie oznacza dosłownie tego, że nie ma żadnych serwerów, a bardziej to, że użytkownik tego typu usługi nie ma żadnej świadomości istnienia, a co za tym idzie – potrzeby ich konfiguracji. Brak serwerów jest poprawnym stwierdzeniem, jeśli mówimy o publicznych dostawcach tego typu rozwiązań (jak AWS Lambda, Microsoft Azure, Google Functions), a już nie do końca w przypadku prywatnych platform, które można wdrożyć na własnej infrastrukturze (np. OpenWhisk, OpenFaaS, Nuclio). Wtedy organizacja musi się zająć zarządzaniem środowiskiem. Niemniej jednak dla programistów nawet w tym przypadku cały czas istnienie serwerów jest niewidoczne.

Jednym z dużych minusów platform tego typu jest znaczne uzależnienie od dostawcy. Jeśli zaczynamy korzystać np. z AWS Lambda, najczęściej potrzebujemy innych usług (takich jak Gateway API,

bazy danych, systemu plików), z których możemy skorzystać w wydajny i prosty sposób, jeśli pochodzą ze środowiska AWS. Choć jest możliwość skorzystania z usług zewnętrznych, jest to dość trudne w implementacji oraz może wpływać znacząco na wydajność rozwiązania.

Funkcje w modelu FaaS mają też swoje ograniczenia, których warto mieć świadomość, np. mają ograniczony czas wykonania (AWS Lambda to 5 minut na moment pisania artykułu), funkcje są praktycznie bezstanowe (nie ma czegoś takiego jak sesja). Jeśli potrzebujemy przechowywać stan, potrzebujemy usług przechowywania danych.

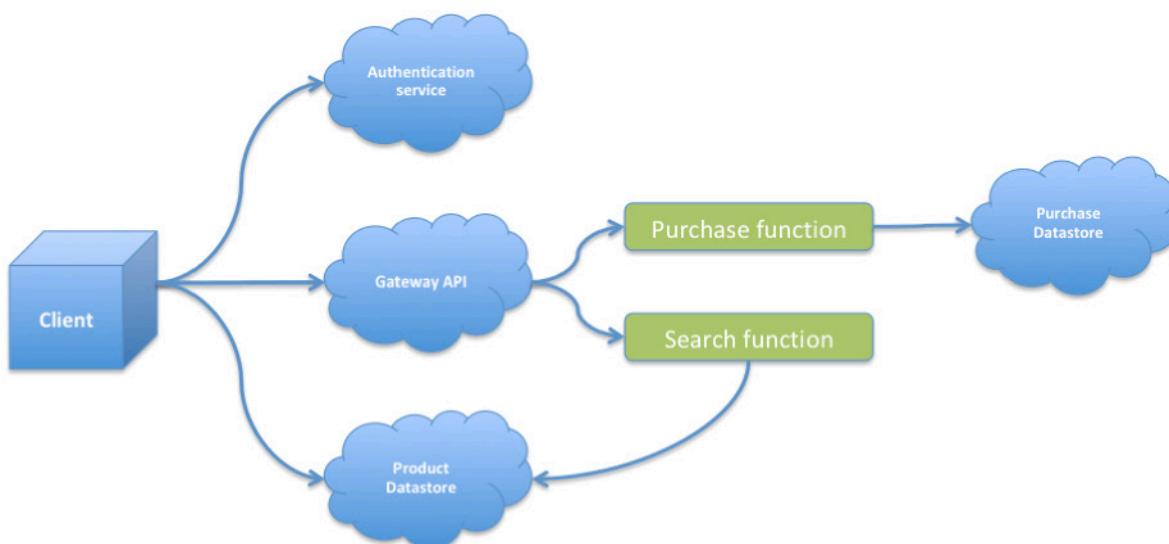
Jak użytkowanie serverless wpływa na design aplikacji? Ponieważ główną jednostką wdrożeniową, którą się posługujemy, są funkcje, zaczynamy myśleć o systemie jako zbiorze wielu funkcji. Z jednej strony będą one nieduże (bardzo małe microservices), z drugiej strony będzie ich wiele, co powoduje, że w przypadku bardziej złożonych rozwiązań będą one trudne w utrzymaniu i zarządzaniu. Przykład tego, jak myślimy o systemie w tej architekturze, można znaleźć na Rysunku 3.

W tym przypadku bierzemy pod uwagę usługę autentykacji, Gateway API, poszczególne bazy danych, funkcję związaną z zakupem, funkcję wyszukiwania produktów.

Zalety i wady

Główną zaletą tego typu podejścia jest możliwość całkowitego pozbicia się zarządzania infrastrukturą oraz możliwość niemal natychmiastowego wdrażania na produkcję. Ponieważ serverless zachęca do wykorzystywania usług w modelu BaaS (autentykacja, baza danych, przechowywanie plików) i ma dla nich natywne wsparcie, oszczędzamy czas na implementację generycznych rozwiązań. Nie sposób przecenić sposobu działania platformy, tj. automatycznego podnoszenia i zamknięcia kontenerów, mechanizmów autoskalowania.

Minusów, tudzież trudnych elementów, jest jednak sporo. Jest to rozwijające się podejście, toteż wsparcie narzędziowe jeszcze nie jest zbyt dobre. Istnieje spore ryzyko związania się z jednym konkretnym dostawcą. O ile testowanie jednostkowe poszczególnych funkcji jest dość proste, to testy integracyjne i funkcjonalne mogą być bardzo trudne do zrealizowania. Bardzo istotną kwestią jest



Rysunek 3. Przykładowy przebieg komunikacji w architekturze serverless

to, że serverless wymusza drobnoziarnistą organizację systemu, co może się okazać kłopotliwe do opanowania, szczególnie przez mniejszą kompetentny zespół. Na obecny moment przykłady użycia dotyczą wybranych zastosowań, jak: przetwarzanie pojedynczych obrazów, pojedynczych filmów, przetwarzanie zdarzeń z mediów społecznościowych, kategoryzacja elementów z użyciem wyuczonej sieci neuronowej i jeszcze brak dobrze ukształtowanych praktyk odnośnie tego, jak szerzej stosować tego typu podejście.

PODSUMOWANIE

W niniejszym artykule zostały opisane dwa ostatnie podejścia architektoniczne, dające największe możliwości skalowania i rozwoju aplikacji. Podejście w architekturze reactive polega na komunikacji pomiędzy częściami aplikacji z użyciem komunikatorów, co umożliwia ich pełne rozdzielenie i uniezależnienie, a co za tym idzie – uzyskanie wysokiego poziomu responsywności. Podejście serverless idzie krok dalej, gdyż zwalnia twórców kodu z myślenia o infrastrukturze – o serwerach, systemach operacyjnych, kontenerach, instancjach usług, co daje możliwość zminimalizowania czasu od implementacji do końcowego wdrożenia. Niemniej jednak

należy mieć na uwadze koszty związane z tymi podejściami: duży poziom skomplikowania rozwiązań tego typu. To architektura dla zespołu o bardzo wysokich kompetencjach i doświadczeniu w zakresie tworzenia systemów rozproszonych opartych o zdarzenia.

PODSUMOWANIE SERII

Jest to ostatni z serii trzech artykułów, toteż podsumujemy poniżej wszystkie przedstawiane podejścia. O ile orientacja i znajomość różnych stylów architektonicznych jest istotna, o tyle należy unikać sytuacji, kiedy podejmujemy decyzje w oparciu o panujące mody. Do architektury warto podejść pragmatycznie z pewnym pomysłem na jej ewolucję. W taki też sposób zostały przedstawione architektury w niniejszej serii, z pewną sugestią, jak systemy mogą z czasem ewoluować (choć oczywiście nie jest to jedyna ścieżka): od klasycznej architektury warstwowej, poprzez wprowadzenie technik Domain-Driven Design, oddzielenie aplikacji od framework'ów z użyciem Ports and Adapters, rozproszenie aplikacji z użyciem microservices, zmaksymalizowanie jej responsywności z architekturą reactive oraz uniezależnienie od infrastruktury z pomocą serverless. Poniżej zamieszczono zestawienie cech poszczególnych podejść.

Rodzaj architektury	Główne wyróżniki, zalety, wady	Kiedy korzystać
Klasyczna architektura trójwarstwowa (w wersji monolitycznej)	<ul style="list-style-type: none"> » jasno określa kluczowe odpowiedzialności w systemie » znane, mocno rozpowszechnione podejście » prostota » łatwa do zrozumienia » komplikuje się znacząco dla złożonych domen » najczęściej monolityczny złożony model danych 	<ul style="list-style-type: none"> » proste aplikacje (typu CRUD) » początkowa faza rozwoju systemu » mało kompetentny, zmienny skład zespołu
Domain-Driven Design	<ul style="list-style-type: none"> » podział systemu na mniejsze części (Bounded Contexts) » mniejsze, bogate modele dziedziny » techniki i praktyki obiektowego projektowania aplikacji, bloki budujące » modelowanie złożonych dziedzin » podwaliny dla rozproszonych architektur typu microservices i reactive » duża bariera wejścia - wymaga wysokich kompetencji związanych z obiektywnością i DDD 	<ul style="list-style-type: none"> » złożona domena » system rozwijany przez wiele lat » kompetentny zespół lub plan inwestycji w rozwój kompetencji zespołu
Ports and Adapters	<ul style="list-style-type: none"> » separacja części domenowej aplikacji od technologii » możliwość testowania systemu z różnych perspektyw » klienci systemu i infrastruktura może być przepięta na różne implementacje » konstrukcja nasycona wieloma abstrakcjami (interfejsami), co wpływa na jego większą złożoność » aby uzyskać pełną czystość rozwiązania, trzeba stosować różne modele w różnych częściach systemu 	<ul style="list-style-type: none"> » ogromne znaczenie ma testowanie automatyczne systemu » różne klienci dla tego samego systemu
Microservices	<ul style="list-style-type: none"> » każda część systemu to osobna aplikacja, gdzie można zastosować inną technologię i architekturę » wprowadza fizyczne granice między częściami systemu » niezależna skalowalność i rozwój różnych części systemu 	<ul style="list-style-type: none"> » systemy o dużym obciążeniu » systemy ogólnodostępne » pojedyncze moduły, które wymagają innego poziomu skalowalności » kompetentny zespół lub plan inwestycji w rozwój kompetencji zespołu
Reactive	<ul style="list-style-type: none"> » każdy element systemu jest niezależny » komunikacja z użyciem systemów kolejkowych do przesyłania komunikatorów » często stosowany model event-driven » wysoka skalowalność części systemu » bardzo duża złożoność systemu » bardzo duża bariera wejścia dla nowych osób 	<ul style="list-style-type: none"> » systemy o bardzo dużym obciążeniu » skalowalność jest krytycznym wymaganiem » kompetentny zespół lub plan inwestycji w rozwój kompetencji zespołu
Serverless	<ul style="list-style-type: none"> » wdrożenia bez infrastruktury » efemeryczne kontenery są podnoszone w momencie obsługi żądania » architektura typu event-driven » aplikacje tworzy się z użyciem gotowych komponentów chmurowych (baza danych, systemy kolejek, usługi mail, sms etc.) » łatwy rozwój złożonej aplikacji, gdy korzysta się ze składników od tego samego dostawcy » rozdrobnione składniki systemu (podstawowym budulcem są funkcje) 	<ul style="list-style-type: none"> » systemy, gdzie żądania są generowane niezbyt często (IoT) i ciągła gotowość serwera do działania nie jest niezbędna » eksperymentalne: systemy, gdzie jest potrzeba natychmiastowego wdrażania na produkcję



MARIUSZ SIERACZKIEWICZ

m.sieraczewicz@bnsit.pl

Od ponad dwunastu lat profesjonalnie zajmuje się tworzeniem oprogramowania. Zdobyte w tym czasie doświadczenie przenosi na pole zawodowe w BNS IT, gdzie jako trener i konsultant współpracuje z czołowymi polskimi zespołami programistycznymi. Jego obszary specjalizacji to: refaktoryzacja, czysty kod oraz technical leadership. Autor metody Naturalnego Porządku Refaktoryzacji ®. Autor książki „Technical Leadership. Od eksperta do lidera”.

13-14 WRZEŚNIA 2018
WARSZAWA
Hotel Sound Garden

SECURITY CASE 2018 STUDY



V DOROCZNA KONFERENCJA IT SECURITY

SCS EXPO, SCS FREE, ELEVATOR PITCH – Wstęp wolny
ZAREJESTRUJ SIE JUŻ DZIŚ

CZYM JEST KONFERENCJA SCS?

SECURITY CASE STUDY to spotkanie społeczności IT Security przyciągające setki uczestników, 5 ścieżek tematycznych, konkurs CTF (Capture the Flag), specjalistyczne warsztaty oraz EXPO ze stoiskami wiodących firm branży IT Security.

Ścieżki konferencji SCS 2018:

- **SCS PRO** – IT Security, cyberbezpieczeństwo
- **SCS URDI** – informatyka śledcza
- **SCS INFOOPS** – ścieżka dedykowana zagadnieniom operacji informacyjnych w cyberprzestrzeni
- **SCS FREE** – prelekcje, część wystawiennicza i stoiska partnerskie (**część bezpłatna**)
- **ELEVATOR PITCH** czyli otwarta scena, na której będzie można w luźniejszej formie przedstawić swój projekt, produkt czy też ideę (**część bezpłatna**)
- **konkurs CTF (Capture the Flag)**, każda drużyna może spróbować swoich sił, dla trzech pierwszych drużyn nagrody pieniężne (**część bezpłatna**).

www.securitycasestudy.pl
#SCSconference

Testowanie eksploracyjne – jak robić to dobrze?

Tak zwane projekty zwinne są już w zasadzie codziennością. Jak wiadomo, ich częścią składową są Sprinty. W każdym Sprincie dostarczamy działającą funkcjonalnie część aplikacji. Nowo dostarczone funkcjonalności muszą być jednak odpowiednio przetestowane. Tymczasem mało czasu na realizację projektu nie zawsze na to pozwala. Rozwiązaniem może być testowanie eksploracyjne.

Testy wykonywane są z reguły na końcu Sprintu. I chociaż nie jest to dobra praktyka, to właśnie tak wygląda zdecydowana większość projektów. Niestety, jest to etap, który może generować nieco stresu. Z tego względu warto zainteresować się technikami i metodami testowania, które pomogą nam ten stres na koniec każdego Sprintu zmniejszyć oraz zapewnić wysoką jakość realizacji całego projektu.

Zgodnie z aktualną definicją International Software Testing Qualifications Board testowanie eksploracyjne to „podejście do testowania, w którym osoba testująca dynamicznie projektuje i wykonuje test, bazując na swoim doświadczeniu, wiedzy, eksploracji testowanego oprogramowania i wynikach poprzednich testów”. Oznacza to zatem, że ten rodzaj testowania kodu oparty jest na doświadczeniu i nauce w projekcie.

Jak prawidłowo przeprowadzać testy eksploracyjne? Z wykorzystaniem *test charterów* (nazywanych po polsku, zgodnie ze Statutem Stowarzyszenia Jakości Systemów Informatycznych, statutami testu).

Z CZEGO SKŁADA SIĘ CHARTER?

Jedną z metod zarządzania testowaniem eksploracyjnym jest testowanie eksploracyjne w sesjach. Warto przed rozpoczęciem poznać jego zasady. Polega ono na wyznaczeniu zakresu testów i ram czasowych na wykonanie tych testów. Po każdej zakończonej sesji powinniśmy przygotować odpowiedni raport z sesji, który – oprócz innych informacji – zawierać będzie dane o znalezionych błędach.

Do sesji powinniśmy się dobrze przygotować. Pомocne w tym celu są tzw. *chartery*. Charter jest swego rodzaju mapą, w której musimy zatrudnić określone dane. Może przyjąć formę papierową lub elektroniczną. Podstawowy charter powinien składać się z następujących elementów:

1. Cel testowania – czyli krótka informacja o tym, co jest celem testu, jaki element aplikacji sprawdzamy.
2. Pomoce – musimy zastanowić się, jakich narzędzi będziemy używać, z jakich technik będziemy korzystać, jakich danych testowych użyjemy oraz – jakich konfiguracji.
3. Atrybuty jakości – należy wiedzieć, jakiego rodzaju informacje chcemy sprawdzić. Do wyboru jest m.in. wydajność, bezpieczeństwo, spójność danych, zgodność UI ze standardem.

JAK TWORZYĆ CHARTERY?

Przy tworzeniu charterów należy zwrócić uwagę na kilka elementów:

1. Zakres testów – należy zdefiniować, co leży w zakresie testowania, a co jest poza jego zakresem. Na przykład, gdy testo-

wana jest validacja danych na formularzu, nie należy skupiać się na testowaniu, czy dane w formularzu są przesyłane w sposób bezpieczny – do tego celu warto stworzyć osobny charter.

2. Ramy czasowe – należy zdefiniować konkretny maksymalny nakład czasu, jaki chcemy poświęcić na przetestowanie danego fragmentu aplikacji. Sesje dzielą się z reguły na krótkie (do 45 minut) i długie (trwające nawet do dwóch godzin). Ważne jest, aby tester po skończonej sesji pamiętał, co testował i z jakim wynikiem. Jeśli zauważamy, że w danej części aplikacji, której dotyczy chart, znajdujemy dużo defektów, możemy stworzyć dodatkowy chart, na dokładniejsze testy, nie należy wydłużać istniejącej sesji. Przyjmuje się, że sesja trwa średnio około 30–90 min.
3. Zbiór informacji – opis tego, co chcemy testować, również musi być wyważony. Zbyt dokładny nie pozwoli na eksplorację, natomiast zbyt ubogi spowoduje, że część funkcjonalności może pozostać bez sprawdzenia.

DEBRIEFING

Po zakończeniu każdej sesji testowania należy przeanalizować wyniki i spojrzeć na wykonane czynności testowe z innej perspektywy. Aby to dobrze zrobić, należy przedstawić wyniki sesji wszystkim zainteresowanym *stakeholderom* w formie prezentacji nazywanej *debriefingiem*.

Podczas debriefingu twórcy (*reporter*) przedstawia osobie przeglądającej (*reviewer*) informacje o tym, co zostało przetestowane, czego nauczył się podczas przeprowadzania testu, jakie problemy napotkał po drodze i jakie błędy udało mu się znaleźć.

Debriefing pozwala określić kolejne obszary do testowania, zidentyfikować nowe ryzyka, wreszcie odkryć obszary aplikacji, do których wymagana będzie automatyzacja testów.

ROZWIAZANIE SZYBKIE, ALE NIEDOSKONAŁE

Testy eksploracyjne oparte są o wiedzę i doświadczenie testera. Wiążą się z tym pewne wady i zalety. Z pewnością warto zauważyć, że:
» testowanie eksploracyjne jest szybkie,
» ta metoda nastawiona jest na wykrywanie błędów.

Są jednak i wady testowania eksploracyjnego. I jest ich więcej niż zalet:

- » wykonywane testy są często niepowtarzalne, bo ścieżki nie opisuje się krok po kroku. Jeśli zatem doprowadzimy do tego, że aplikacja przestanie działać, wpisując w polu formularza



jakiś znak specjalny i nie zapamiętamy, jaki był to znak, nie będziemy w stanie ponownie doprowadzić do wystąpienia tego samego błędu. Jedyną możliwością na uniknięcie tego problemu jest rejestrowanie sesji w formie przechwytywania wpisywanych znaków lub tworzenia wideo z sesji.

- » często brakuje informacji, jakie jeszcze testy muszą zostać wykonane – w przeciwieństwie do standardowych testów pomijany jest etap tworzenia przypadków testowych, czy przeglądania przypadków, choć może powstać plan określający, jakie testy mają zostać wykonane; często wiele zależy od doświadczenia testera: im jest ono większe, tym większa jest też skuteczność wykonywanego testu,

» niska znajomość działania produktu powoduje dużą liczbę zgłoszonych problemów, które wynikają z braku znajomości działania aplikacji czy systemu i tak naprawdę nie są defektami.

Dlatego warto pamiętać, że chociaż testowanie eksploracyjne może być z powodzeniem wdrożone do projektów zwinnych, to powinno stanowić tylko jeden z wielu elementów testowania. Nie powinno być to jedyne podejście do kontroli jakości. To po prostu szybka i nastawiona na wykrywanie błędów technika, która ułatwia pracę testera, ale nie jest pozabawiona wad.

MAREK WALESIA

Testowaniem zajmuje się od ponad siedmiu lat. W tym czasie przeżył wiele bugów, w tym kilka identycznych. Aktualnie zajmuje się projektowaniem i przygotowaniem architektury w projektach, w których planuje się wprowadzić testy automatyczne. Oprócz testowania i automatyzacji od kilku lat zajmuje się bezpieczeństwem aplikacji. Jego motto brzmi: „Dzień bez NullPointer'a – dniem nietestowanym”.

Hipsterskie metody wyboru technologii

Czy zdarzyło Ci się kiedyś pracować pod wpływem fascynacji jakąś technologią? Wracając z konferencji, czytając artykuł, oglądając wideo, biorąc udział w szkoleniu, słuchając rozmów znajomych z pracy – czy czułeś ekscytację nową technologią? Mnie zawsze przechodzą ciarki po plecach, kiedy siadam do nowego projektu lub nowego fragmentu kodu i chcę wykorzystać wszystko to, czego się dowiedziałem przez ostatnie pół roku. Wiem, że jeżeli tak zrobię, jest spora szansa, że nie wszystko pójdzie tak dobrze jak to było obiecywane, a ja będę zastanawiał się, dlaczego tak się stało.

ZNAJ PRZESZŁOŚĆ, PATRZ W PRZYSZŁOŚĆ

Jaki był początek twojej kariery zawodowej? Jaki był pierwszy projekt, za który otrzymałeś wynagrodzenie? Jakie technologie były w nim użyte?

Może zaczynałeś swoją pracę od budowania stron w PHP? Może zaczynałeś od programowania w C? A może pamiętasz, jakby to było wczoraj, kiedy wyszła pierwsza wersja Springa? Spójrz na swoje życie zawodowe z innej perspektywy. Czy dostrzegasz całą historię swojego życia zawodowego? Czy widzisz, że świat wokół Ciebie się zmienił?

Swoją karierę zawodową zaczynałem w czasach, w których SOAP oraz SOA były na pierwszych stronach gazet. WSDLle pisało się z palca, a schematy XSD to był chleb powszedni. Wtedy byłem świecie przekonany, że ten sposób tworzenia usług sieciowych będzie wieczny. Że tak będziemy robili przez przynajmniej kilka dekad. Dzisiaj świat jest inny i nie stosuję już żadnej z wymienionych powyżej technologii, z których korzystałem kilka lat temu. Technologie i narzędzia przychodzą i odchodzą. Jedne są z nami krócej, inne są z nami dłużej. Pamiętam, jakby to było wczoraj, kiedy metodyki Agile były wynoszone na piedestał, jakby miały rozwiązać wszystkie problemy tego świata. Dziś otwarcie mówi się o tym, żeby programiści porzucili Agile¹.

Każda technologia, z której korzystamy, przedzej czy później zostanie zastąpiona przez nową. Tak powinno być. Kiedy się nad tym zastanawiam, jest to najskuteczniejsza metoda rozwoju. W czasach pradawnych czerpalismy światło ze spalanego drewna, następnie wynaleziono świece, elektryczność i żarówkę żarnikową, potem świetłówki, a teraz diody led. Tak samo jest z naszą branżą IT. Pytanie brzmi: co zrobić, żeby być zawsze na czasie i nie pozostać przy świecach, kiedy na oko wszyscy korzystają z oświetlenia ledowego?

Jedną z metod bycia na czasie, z jaką się spotkałem, jest branie wszystkiego do projektu, co jest najnowsze, bez względu na konsekwencje. Niektórzy wybierają nawet wersje alpha nowych bibliotek i zależności. Inni starannie wybierają swój stos technologiczny i są zamknęci na dyskusję. Czasami jedziemy na konferencję i pod wpływem oczarowania nowymi rzeczami zmieniamy architekturę

projektu. Jeżeli pracujemy w doświadczenym zespole, możemy dyskutować na temat zmian i tego, co one nam dadzą. Na podstawie wymiany doświadczeń możemy podjąć decyzję, co dalej.

„NEVER ASK A BARBER IF YOU NEED A HAIRCUT”

Ponieważ mój już prawie pełnoletni samochód był objęty jakąś akcją serwisową, odbyłem wizytę w salonie samochodowym. Salon ten sprzedawał zarówno nowe samochody, jak i używane. Nie ukrywałem przed sprzedawcą, że przymierzę się do zakupu samochodu, nowszego i większego, ze względu na powiększającą się rodzinę. Niestety moja 10-minutowa rozmowa ze sprzedawcą nie zakończyła się dobrze, ponieważ czułem się manipulowany. Przedstawiłem mu swoje wymagania, a on usilnie starał się mi sprzedać nowy i najdroższy samochód. Ja takiego nie chciałem. Chciałem zakupić 2-, 3-letnie auto, które nie stoi w salonie, tylko przed salonem, gdzie klient oddał go w rozliczeniu. Nasłuchałem się, jak to 3-letni samochód jest przestarzały, nie na czasie i tym podobnych, kiedy tak naprawdę chciałem coś zupełnie innego. To zdarzenie nauczyło mnie jednego: nieważne, z kim rozmawiasz, pamiętaj o tym, kto siedzi po drugiej stronie i jakie ma motyw. Sprzedawca chciał zarobić, nie interesowały go zupełnie moje potrzeby.

Jak to się ma do branży IT? Każdy z nas słucha prezentacji nowych rozwiązań, czyta blogi tematyczne, rozmawia z ludźmi przy okazji konferencji lub spotkania, albo dostaje maile informujące o nowych rozwiązaniach. Czy zdajesz sobie sprawę, kto jest po drugiej stronie? Jakie ma motyw do tego, co robi? Na ile treści, które dostajesz, są przypudrowane i ubrane w ładne słowa, a na ile to, co słyszę, rozwiązuje prawdziwe problemy, które mam lub mogę mieć w niedalekiej przyszłości.

My, programiści, software developerzy, możemy być całkowicie obiektywni, jeżeli tylko chcemy. Nie można zaprzeczyć, że mamy wrodzoną skłonność do inżynierii i wymyślania koła na nowo. Barizo łatwo ulegamy trendom, modzie, sezonowemu nastawieniu na nowe rozwiązanie, tylko po to, żeby... za rok przepisać system od nowa, ponieważ będzie coś nowego. Sam też często zastanawiam się na kodem, który popełnię. Czy nie dałoby się prościej? Czy czasami nie przesadzam? A może jest biblioteka, która rozwiązuje ten problem?

1. <https://ronjeffries.com/articles/018-01ff/abandon-1/>

ENCJA NA TWARZ I PCHASZ

Mamy dużo szczęścia, że pracujemy w IT. Ja doceniam ten zawód i nie wiem, co innego mógłbym robić w życiu. Kiedy rozmawiam z innymi znajomymi z czasów szkolnych, większość z nich wykonuje codziennie te same, powtarzalne zadania, codziennie, dzień za dniem, przez cały rok.

Tymczasem my, inżynierowie oprogramowania, jak tylko pojawi się coś nieciekawego do zrobienia – narzekamy. Że trzeba rozwiązywać bugi, że jest kolejny JSON do wysłania, lub robimy systemy w architekturze, która nam się nie podoba – często nie podejmując żadnych działań, żeby to zmienić.

Doceńmy, że nasza branża jest otwarta na ciągłe zmiany i ciągły rozwój. Nie wszędzie tak jest, nie w każdym sektorze jest taka kultura pracy.

Niemniej każde zadanie, każdy system, nad którym pracowałem, miał interesujące problemy do rozwiązania. Zawsze jest miejsce, żeby nauczyć się czegoś wartościowego. Podam prosty przykład. Od kilku lat przeżywamy „boom” na NoSQL. Każdy, dosłownie każdy programista chciał robić Cassandra, mimo że nie było ku temu żadnych racjonalnych powodów. Większość z nas robiła systemy ze zwykłymi bazami relacyjnymi oraz jakimś ORMem – u mnie to był Hibernate.

Co można zrobić? Można narzekać, że Cassandra rozwiązałaby wszystkie nasze problemy (a zapewne tak nie będzie). Można też wykorzystać ten czas, żeby zgłębić, jak działa to, co mamy, żeby móc z pełną świadomością powiedzieć, że panujemy nad tym. Ja zainwestowałem w wiedzę z Hibernate. Czy to była dobra inwestycja? Zdecydowanie tak! Ogromna ilość systemów stosowała, stosuje i będzie stosować ORMy. Wiedza, którą zdobyłem, okazała się później niezbędna i przydatna, mimo że wszyscy zapowiadaли koniec relacyjnych baz danych.

Dla pozostawienia czystości obrazu dobrze jest wiedzieć, kiedy i w jakich warunkach wykorzystać nierelacyjne bazy danych. Warto też zaznajomić się z nimi, żeby sprawnie ich używać. Nie należy ich stosować wszędzie, gdzie popadnie.

ZAKŁAD PRACY

Kiedy wychodzimy z zespołem na lunch, w jaki sposób dokonujemy wyboru tego, co zjemy? Bierzemy to, co braliśmy wczoraj, lub to, co wybrał kolega lub koleżanka. Nie przywiązuje się do tej decyzji dużej uwagi.

Spójrzmy prawdzie w oczy. Czy poświęcamy dużo uwagi temu, w jakich technologiach pracujemy? Jak wybieramy narzędzia, w których spędzimy następne miesiące naszego życia? Tak samo! Na przykład przejdźmy do wyboru języka programowania. Jakie są kryteria wyboru języka programowania do nowej usługi? Wybieramy to, co jest znane w naszym „zakładzie pracy”, lub wybieramy

to, co robiliśmy w ostatnim projekcie. Tak samo jak wybór potrawy na lunch. Wszystko jest w porządku, jeżeli wiemy, że wybór, który podejmujemy, jest dopasowany do potrzeb projektu. Gorzej, jeżeli znamy tylko jedno narzędzie i cały czas używamy tylko tego jednego narzędzia do wszystkich rodzajów problemów.

CV DRIVEN DEVELOPMENT

Otwieram teraz swoje CV sprzed kilku lat – kiedy zaczynałem swoją karierę. Co tam znajduję? Dużą ilość technologii. W zasadzie same wypunktowania i listy technologii, z którymi miałem do czynienia. Otwieram teraz swoje CV. Zostawiłem w nim kilka kluczowych technologii i narzędzi, z którymi pracuję i które uznaję za dobre. Te technologie, które wykreśliłem, zastąpiłem znajomością podstaw inżynierii oprogramowania. W sekcji z doświadczeniem pojawia się więcej wpisów na temat tego, jak oprogramowanie, które tworzyłem, miało wpływ na upraszczanie procesów i biznesów w firmie.

Dlaczego tak? Ponieważ nie chcę być oceniany tylko i wyłącznie na podstawie znajomości frameworków. Narzędzia przychodzą i odchodzą. Zdaję sobie z tego sprawę. W zdecydowanej większości przypadków biblioteka, która ukazała się miesiąc temu, nie pozwoli nam zbudować lepszego projektu. Ważne jest to, żeby się rozwijać i iść z duchem czasu, właśnie dlatego, żeby wiedzieć, które narzędzia i rozwiązania stosować w których sytuacjach.

Kiedy patrzysz na swoje CV, czy widzisz tam zbiór luźno wrzuconych technologii, czy dostrzegasz to, co zrobiłeś? Czy jesteś z tego dumny? Warto – i opłaca się – wybierać takie projekty i takie technologie, żeby być dumnym z tego, czego się dokonało.

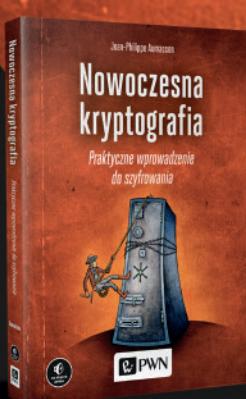
HYPE DRIVEN DEVELOPMENT

Według mnie konferencje techniczne są bardziej niebezpieczne dla naszych projektów, niż dostęp do bazy produkcyjnej z prawami kasowania danych. Przed każdą konferencją uczestnik powinien dostać ostrzeżenie, że po powrocie do pracy z konferencji nie należy wywrać projektu do góry nogami. Hype Driven Development to najbardziej beztroskie, a zarazem najbardziej nieodpowiedzialne podejście do wytwarzania oprogramowania.

Programiści dzielą się na tych, którzy tworzą projekty na fali mody, oraz tych, którzy potem muszą to utrzymywać. Oczywiście, ci, co wybrali pracę „na fali”, szybko muszą zmieniać projekty. Po pół roku pracy szukają dla siebie nowego projektu, ponieważ ten, nad którym pracują obecnie, jest już przestarzały i nieatrakcyjny.

Jakie jest największe zagrożenie? Nie czytamy dokumentacji, nie sprawdzamy technologii przed jej produkcyjnym użyciem, nie zagłębiamy się dokładnie w „nowość”, którą trzymamy w ręku przed jej zastosowaniem. Często kończy się to robieniem projektu bez przemyślenia, ponieważ rzeczy, które są w projekcie, nie do końca do siebie pasują.

reklama



**ZAPISZ SIĘ
NA NASZ NEWSLETTER
I BĄDŹ NA BIEŻĄCO**

Odwiedź nas na:
IT.PWN.PL

KSIEGARNIA.PWN.PL

Wcale nie trzeba być uczestnikiem konferencji, żeby zachłysnąć się jakąś technologią i położyć przez to cały projekt. Na przykład pracując w iteracyjnym podejściu, ze Sprintu na Sprint, bardzo łatwo jest stracić wizję, którą chcemy zrealizować. Każdego dnia w pracy podejmujemy dziesiątki decyzji architektonicznych, które w przyszłości mogą mieć negatywny wpływ na cały projekt, ale bardzo pozytywny i mile widziany wpływ na ten Sprint. Nie chodzi o to, żeby wygrać bitwę. Chodzi o to, żeby wygrać wojnę.

MALOWANIE

Robiłem remont, zatrudniłem sprawdzoną przeze mnie ekipę fachowców. Pracę wykonywali szybko i sprawnie. Wiedzieli dokładnie, co i jak będą robili. Prace były już mocno zaawansowane, zostało tylko malowanie ścian. Tego dnia dostaję telefon, że zjawiają się godzinę później niż zwykle. Jako wytlumaczenie podali, że w miejscu, w którym zazwyczaj robią zakupy, nie było farby podkładowej i muszą jechać do innego sklepu. Zadałem pytanie: „Nie było żadnej farby podkładowej?”. Fachowiec przyznał, że inne były. Dodał też, że największe zaufanie ma do tej, która chwilowo była niedostępna, że nie będzie na mnie eksperymentował z inną farbą, że pojedzie do innego sklepu.

Byłem w szoku. Naszła mnie też wtedy refleksja, ile razy to my, programiści, eksperymentujemy na naszych projektach, zostawiamy funkcjonalności nieprzetestowane, które przecież trafiają na produkcję. Mówimy wtedy sobie, że taka branża, że jak nie spróbujemy, to nie będziemy wiedzieli, robimy to przecież, żeby się rozwijać itp. Bardzo często jest to prawda. Próbowanie czegoś nowego jest skuteczną metodą poznawania innych rozwiązań. Również bardzo często – za często – jest to tylko naszą wymówką, ponieważ czegoś nie chce nam się sprawdzić, napisać dobrych testów lub dowiedzieć się, jak powinniśmy to zrobić poprawnie. Jedni twierdzą, że w zawodzie programisty występuje niska odpowiedzialność za popełnione błędy. Inni próbują zwalić winę na wymagania, które były nieprecyzyjne. Jak temu zaradzić? Należy utrzymywać czystą architekturę naszego systemu. Powinniśmy mieć na uwadze to, że jak system będzie się rozrastał, architektura będzie stawała się coraz bardziej zagmatwana. Jedyne wyjście, jakie widzę, to ciągły refactoring, tak żeby tempo pracy nad systemem było stałe.

BLOG DRIVEN DEVELOPMENT

Na czym polega Blog Driven Development? Założmy, że masz jakieś wyzwanie lub decyzję projektową do podjęcia. Możesz zrobić to, co zwykle, czyli podjąć decyzję na podstawie intuicji. Albo możesz zacząć tworzyć wpis na blogu, w którym opiszesz:

- » Problem lub wyzwanie, z którym się mierzycie – Ty lub zespół,
- » Analizę możliwych rozwiązań,
- » Wybrane rozwiązanie – wraz z argumentami za oraz przeciw,
- » Skutki podjętej decyzji.

Nie musi to być poemat na kilka stron, może to być drobna notatka złożona z kilku punktów. Nie jest istotne to, czy kiedykolwiek opublikujesz ten wpis na blogu, ale założ, że tak może się zdarzyć.

Takie ćwiczenie jest formą futurospekcji. Formą zreflektowania się nad swoimi najbliższymi działaniami, czy doprowadzą one do tego, co chcesz osiągnąć.

Czy będziesz się czuł dumny z tego, co napisałeś? Czy może kiedy patrzysz na to, co napisałeś, to widzisz, że to nie będzie profesjonalne podejście?

Taka metoda pozwoli spojrzeć nam z boku na to, co robimy. Pozwala nam uspokoić emocje, wziąć głęboki oddech oraz jeszcze raz na spokojnie wszystko przemyśleć.

TRADE OFF

Pytanie jest proste. Czy powinieneś na każdym kroku inwestować w najnowsze rozwiązania, czy powinieneś skupić się na szlifowaniu swoich umiejętności z tego, co jest tu i teraz?

Odpowiedź nie jest łatwa, na pewno nie jest zero-jedynkowa. Branża IT zmienia się szybko. Jeśli prześpisz jakiś moment, możesz się obudzić w zupełnie innym świecie i być przestarzałym. Z drugiej strony, nie znając dobrze swoich narzędzi i technologii, możemy nigdy nie być docenieni za bycie prawdziwym ekspertem. To każdy z nas powinien sam sobie odpowiedzieć na pytanie, jak chce pokierować swoją ścieżką kariery.

ZAKOŃCZENIE

I would advise students to pay more attention to the fundamental ideas rather than the latest technology. The technology will be out-of-date before they graduate. Fundamental ideas never get out of date.

David Parnas

Czym w takim razie powinniśmy się kierować, tworząc nasze systemy? Największą uwagę powinniśmy zwrócić na design i architekturę naszego systemu, a nie na stosowaniu w nim wszystkiego, co najnowsze. Każdy system z biegiem czasu robi się coraz bardziej skomplikowany oraz zawiera coraz więcej rozwiązań „na sznurki”. Twoim zadaniem jako programisty jest dbanie o wysoką jakość oprogramowania, przeciwdziałanie temu zjawisku poprzez ciągły refactoring. Dodawanie nowych narzędzi niewiele nam tu pomoże, a jest duże zagrożenie, że przyspieszy proces erozji naszego oprogramowania.

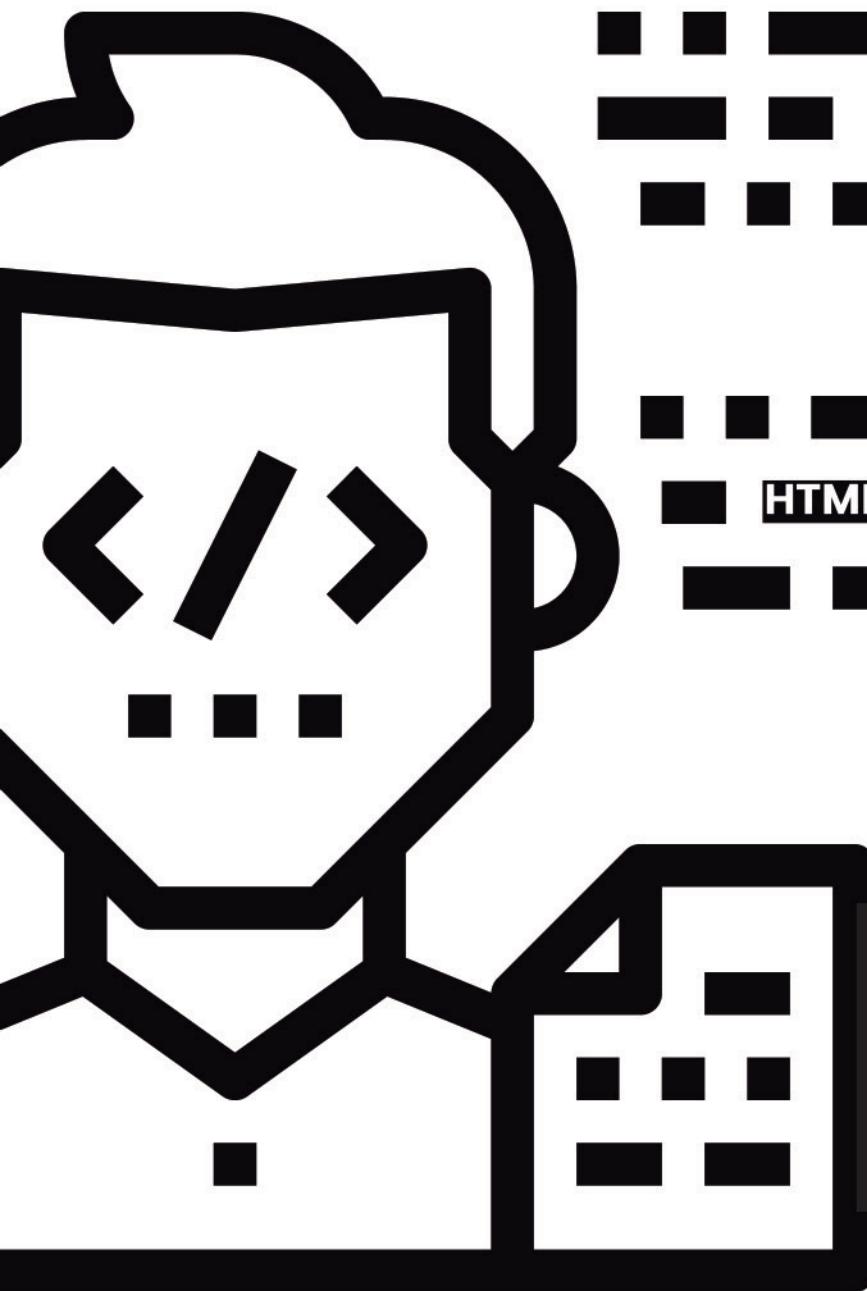
Skąd wiem, że mój warsztat jako programisty jest dobry? Na to pytanie odpowiem pytaniem: skąd wiesz, że fachowiec, który przychodzi do domu wykonać u Ciebie jakieś prace, jest dobry? Jeżeli jego skrzynka z narzędziami jest duża, jest poukładana, a fachowiec wie, jakiego narzędzia do czego użyć, to wiesz, że trafiles na osobę, która poważnie podchodzi do swojej pracy. Naprawa może zajść 10 minut, a fachowiec może użyć jednego narzędzia ze swojej skrzynki z narzędziami – tego jedynego i odpowiedniego do tego zadania. Właśnie na tym polega profesjonalizm – znam swoje narzędzia, które umiem dobrze wykorzystać, dzięki temu jestem skuteczny w tym, co robię.

Nie spoczywam na laurach. Wiem, że świat się zmienia, a ja razem z nim. Swój rozwój planuję, robię to w sposób świadomym. Dzięki temu rozwijam się szybko i w odpowiednich dla siebie kierunkach.

MICHAŁ LEWANDOWSKI

Software developer. Przez ostatnie kilka lat zajmował się różnymi rzeczami związanymi z wytwarzaniem oprogramowania, począwszy od analizy wymagań, przez prowadzenie zespołu, samo kodowanie, skończywszy na utrzymaniu i odpalaniu systemu. Jest przekonany, że każdy profesjonalista, oprócz twardych umiejętności związanych z kodowaniem, powinien mieć rozwinięte umiejętności miękkie.

Audyt kodu źródłowego



■ ■ ■ **Java**
■ ■ ■ **PHP C / C++ / C#**
■ ■ ■ **Assembler**

■ ■ ■ **MySQL**
■ ■ ■ **HTML5, CSS3, JS, jQuery**
■ ■ ■ **SOAP / REST**



Usługi.
testerzy.pl

Jak zostać programistą

Nie ma co ukrywać, że programowanie ostatnimi czasy stało się bardzo popularnym zawodem. Wynika to między innymi z tego, że razem z informatyzacją coraz to nowych obszarów życia potrzeba programistów, którzy napiszą odpowiednie oprogramowanie. Razem z podażą rośnie oczywiście popyt – na rynku namnożyło się mnóstwo kursów, kursików, szkoleń, a nawet kierunków na uczelniach wyższych, przygotowujących do tego zawodu, i w efekcie pojawia się również coraz więcej developerów. Pomimo tego o naprawdę dobrych programistów wciąż jest trudno. Co więc zrobić, by wyróżnić się wśród innych i stać się dobrym programistą?

ZACZNIJ JAK NAJCZEŚNIEJ

Listing 1. Mój pierwszy program (ever)

```
10 PRINT "CZESC"
20 GOTO 10
```

Swój pierwszy program napisałem mniej więcej w wieku 8 lat w Basicu na Atari i mogę chyba uczciwie powiedzieć, że od tego czasu programuję prawie bez przerwy – z Basica przerzuciłem się w podstawówce na QBasic, by potem zahaczyć jeszcze o Visual Basic; następnie zainteresowałem się Pascalem, z którego jeszcze w liceum przesiadłem się na Delphi; na studiach zacząłem pisać w C# i C++, a później już w zasadzie we wszystkim, co mi wpadło w ręce albo co było na daną chwilę potrzebne. Z perspektywy czasu myślę, że moje pokolenie programistów w pewien sposób jest uprzewilejowane, ponieważ to właśnie lata 80. i 90. stanowiły okres dynamicznego rozwoju informatyki – a my mogliśmy być w centrum tego wszystkiego i uczyć się na bieżąco. Współczesni młodzi programiści mają przed sobą naprawdę niełatwwe zadanie – nie dosyć, że pojawiło się dużo platform, na które można pisać kod (embedded, wearables, desktop, web, tv, konsole, rozwiązania chmurowe itp.) i samych wiodących języków programowania jest już w tej chwili przy najmniej kilkanaście, to jeszcze w każdym z nich zderzymy się z multum bibliotek, frameworków, architektur, wzorców i rozwiązań, które po prostu trzeba znać.

Podejrzewam, że większość dzieci posadzona dzisiaj przed Atari i jego Basiciem nie wykazałaby entuzjazmu równego mojemu, ale przecież współczesny Internet dostarcza wielu znacznie bardziej atrakcyjnych narzędzi. Jakiś czas temu natknąłem się na bardzo ciekawą webową grę o nazwie CodeMonkey, pozwalającą powoli

wdrożyć dzieci w świat programowania. Jest to wariacja na temat Logo – za pomocą odpowiednich komend staramy się poprowadzić małpę w taki sposób, by pozbierać z poziomu wszystkie banany. Co przykuło moją uwagę, to fakt, że składnia jest bardziej czytelna od oryginalnego Logo i jednocześnie odważnie wprowadza gracza od razu w świat programowania obiektowego, omijając po cichu większość jego komplikacji, które mogłyby na początku zniechęcić. Przykładowe rozwiązywanie jednej z plansz znajduje się na Rysunku 1.

Listing 2. Przykładowy kod CodeMonkey

```
x = 15
4.times ->
  monkey.step x
  monkey.step -x
  turtle.step 8
```

Podrzuciłem ostatnio tę grę bratanicom mojej żony (9 i 12 lat) i powiem szczerze, że byłem zdumiony, bo tłukły poziom za poziomem i nie można ich było oderwać od komputera. Abstrahując od tego, że czas w ten sposób spędzony jest na pewno o wiele bardziej wartościowy od grania w Minecrafta czy Simsów, taka gra uczy również analitycznego podejścia do rozwiązywanych problemów, co potem przydaje się bodaj we wszystkich przedmiotach ścisłych. CodeMonkey to oczywiście tylko jedno z wielu narzędzi, a wybór jest naprawdę duży, włączając w to na przykład Lego Mindstorms, które pozwalają projektować i programować mniej lub bardziej skomplikowane roboty, albo popularnego – dostępnego na przykład na Raspberry, a zaprojektowanego na MIT – Scratcha.

Jeżeli kogoś zniechęca fakt, że żadne z zaprezentowanych rozwiązań nie przynosi żadnej praktycznej wiedzy, niech ma na uwadze, że sama umiejętność skorzystania z danego języka czy środowiska to tak naprawdę tylko niewielki ułamek programowania jako takiego. Gdybym miał ocenić, powiedziałbym, że dobre 40-50% procesu programowania polega na rozłożeniu dużego problemu na mniejsze, do których można dopasować posiadane lub znane narzędzia; kolejne 30-40% polega na zaprojektowaniu abstrakcji obiektów grających rolę w danym zagadnienu oraz zależności pomiędzy nimi tak, aby powstała spójna architektura; do tego momentu praca jest czysto koncepcyjna. Samo wprowadzanie kodu zajmie pozostałe 10-30% i w przypadku bardziej doświadczonych programistów staje się często formalnością.

Żebyśmy mieli jasność: nie mówię tu bynajmniej o czasie spędżonym na realizowaniu danego procesu, chodzi mi bardziej



Rysunek 1. CodeMonkey (źródło: <https://www.playcodemonkey.com/challenges/36>)

code::dive 2018



7-8 November 2018
Kino Nowe Horyzonty
Wrocław, Poland

Python

C++

ML

Agile

DevOps

Go

Security

Would you like to influence the world and share your
experience with other programming enthusiasts?
Become a code::dive speaker!

call for papers

Our conference covers topics important for software development and software management. We talk about low-level issues, programming languages, new tools and methods of creating software.

Brought to you by

NOKIA

Submit your topic propositions to our webpage:
<http://codedive.pl/register/cfp>

o zaangażowanie w niego konkretnej umiejętności programisty. W środowisku programistycznym słyszy się wręcz o stanowisku kodera – od „pełnowymiarowego” programisty różniącego się tym, że efekt dwóch pierwszych etapów dostaje na tacy, zaś jego zadaniem jest tylko wprowadzenie go do komputera w postaci kodu źródłowego.

Dlatego też wykształcenie w dziecku umiejętności analitycznego myślenia jest znacznie bardziej wartościowe od nauczenia go konkretnego, współczesnego języka programowania. A pamiętajmy, że zrobić to możemy, nie tylko sadząc je przed komputerem, ale na przykład wspólnie budując budkę dla ptaków – od przygotowania projektu, przez zakup materiałów i narzędzi, aż po jej wykonanie i triumfalne zawieszenie w ogródku (a ponieważ żyjemy w XXI wieku, można tam schować małą kamerkę z radiowym przekaznikiem video i razem z dziećmi obserwować rozwijające się pisklaki).

ANGIELSKI

To będzie najkrótsza i najnudniejsza sekcja, więc mniejmy ją szybko za sobą. 99% dokumentacji technicznej bibliotek i frameworków jest po angielsku. Z 99% klientami z całego świata można dogadać się po angielsku i z 99% programistów można dogadać się po angielsku. 99% kodu źródłowego jest po angielsku (identyfikatory, komentarze). Nie przesadzam. Zdarzają się nieliczne wyjątki, ale to jest naprawdę margines programistycznego świata. Szukając odpowiedzi na różne programistyczne zagadnienia, zapytania w Google wpisuję już od razu po angielsku, bo w ten sposób szybciej jest trafić na wyniki (choć trzeba przyznać, że źródeł po polsku jest też bardzo dużo).

Jeśli jesteś dorosły i nie czujesz się jeszcze dobrze, komunikując się w tym języku, zacznij się uczyć już teraz, jak najszybciej. Jeśli masz dzieci, zadbaj o to, by uczyły się tego języka możliwe jak najwcześniej (to przyda się im niezależnie od tego, czy będą w przyszłości programować, czy nie). Moi rodzice zapisali mnie na prywatne lekcje angielskiego bodaj jeszcze przed moim pójściem do szkoły podstawowej i jestem im za to naprawdę wdzięczny.

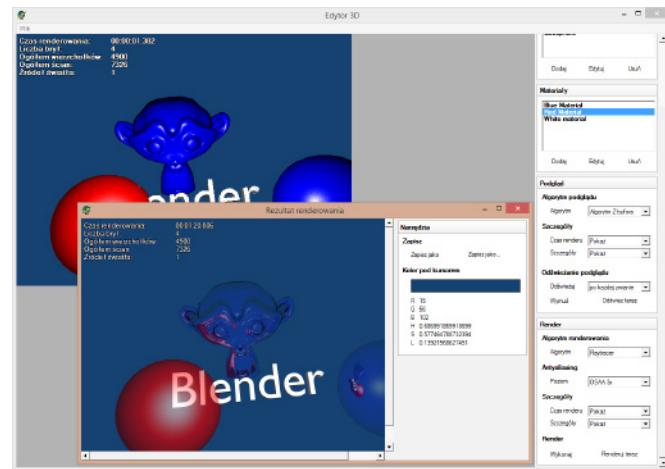
JEDYNA SŁUSZNA TECHNOLOGIA?

W liceum wpadła mi w ręce świetna książka, można by powiedzieć klasyka gatunku – „Algorytmy + struktury danych = programy” Nicolausa Wirtha, twórcy Pascala. Kiedy dotarłem do rozdziału traktującego o dynamicznych strukturach danych i udało mi się zaimplementować pierwszą listę jednokierunkową, przeżyłem swego rodzaju moment „wow” – otworzyły się przede mną zupełnie nowe możliwości. Przejście na programowanie obiektowe w Pascalu (Pascal w wersji 7.0 wspierał już obiekty w sposób zbliżony trochę do C++), a potem na Delphi było dla mnie kolejnym kamieniem milowym. Aktywnie uczestniczyłem również w życiu grupy dyskusyjnej pl.comp.lang.delphi, która była wówczas naprawdę na poziomie – można było z niej czerpać garściami merytoryczną wiedzę.

Dużym błędem, który jednak wówczas popełniłem – co uświadomiłem sobie dopiero po latach – było prawie fanatyczne przywiązanie do tej jednej technologii. W pewnym stopniu było to usprawiedliwione – Borlandowy (wówczas) VCL swoją prostotą rozkładał Microsoftowy MFC na łopatki, Delphi w pełni zasługiwał wówczas na miano RAD (Rapid Application Development). Dla początkującego programisty była to bajka, bo znacznie uprościła wejście w świat programowania okienkowego, jednak dużym błędem było ignorowanie innych języków i technologii. Moi koledzy przenieśli się w międzyczasie z Delphi na C++ i w tym języku pisali

aplikacje na olimpiady informatyczne, niektórzy zainteresowali się czystym C, co bardziej ambitni czytali o assemblerze, a ja tkwiłem w Delphi, zaciekle broniąc stanowiska, że w języku tym można napisać wszystko i nie ma sensu uczyć się niczego innego. W moim pokoju w domu rodzinnym na ścianie do dziś wisie pożółkła już, wydrukowana kartka „10 reasons, why Pascal is better than C”.

Sytuacja zaczęła zmieniać się dopiero na studiach. Najpierw musiałem nauczyć się C, ponieważ był to obowiązujący język na jednych z zajęć – przesiadka z wygodnego i bezpiecznego świata Pascala na mroczny i nie wybaczający błędów świat C była naprawdę bolesna (szczególnie że na zajęciach do dyspozycji mieliśmy Vim i Emacs, praktycznie bez jakiegokolwiek wsparcia obecnego we wszystkich współczesnych IDE). Moim drugim podejściem było napisanie jednego z programów zaliczeniowych – bodaj na Analizę Sygnału – w C#, ot, po prostu dlatego, że w przeciwnym wypadku musiałbym zwyczajnie bezmyślnie wklepać kolejny program w Delphi i miałem poczucie, że zmarnuję w ten sposób okazję do nauczenia się czegoś nowego. Aplikację do pracy magisterskiej napisałem jednak zachowawczo w Delphi.



Rysunek 2. Raytracer w Delphi – aplikacja powstała w trakcie pisania pracy magisterskiej

Przez długie lata z zaciękością godną Rambo bronilem Delphi we wszystkich flame-warach, które rozgrywały się na łamach pl.comp.lang.delphi, bronilem tego języka wśród znajomych, ba, nawet prowadziłem w moim liceum (oczywiście już po jego ukończeniu) kółko programistyczne właśnie w Delphi. Musiało upchnąć dużo czasu, żebym uświadomił sobie, że stwierdzenie, że „Pascal jest lepszy od C”, jest równie bezsensowne co „Śrubokręt jest lepszy od młotka”. Każdy język programowania, framework, biblioteka to tylko narzędzia, z których możemy skorzystać do rozwiązania danego problemu. Jeżeli ktoś tłucze gwoździe śrubokrętem, próbując je wbić w deskę, a robi to tylko po to, żeby wszystkim udowodnić, że śrubokręt też się do tego nadaje, zachowuje się po prostu głupio. Cóż, jak ja od liceum mniej więcej do końca studiów.

Na rynku jest w tej chwili zbyt dużo języków i technologii, które ze sobą w dużym stopniu współpracują, żeby móc skupić się tylko na jednym lub jednej. Piszesz w C#? Super. Ale niebawem będziesz musiał napisać stronę w ASP.NET i wtedy przyda się już wiedza o HTMLu, CSSie i JavaScript. A kiedy twój projekt zacznie się rozrastać, nagle dowiesz się, że CSSa trzeba będzie poskładać w Sassa albo Lessa, które pomogą zadbać o spójność definicji stylów, JavaScript zostanie zastąpiony TypeScriptem, który pomoże uporządkować kod frontendowy, kilka zapytań, które są nieoptymalnie generowane przez ORM, trzeba będzie napisać w czystym SQLu (razem

z kilkoma procedurami), a na koniec okaże się, że w celu dopasowania aplikacji do potrzeb kilku różnych klientów jej część trzeba będzie oskryptować – na przykład przy użyciu IronPythona. I tak dalej.

Jestem daleki od tego, by zachętać do uczenia się wszystkiego naraz. Tego się nie da zrobić, co więcej, zazwyczaj nie ma nawet takiej konieczności. Chodzi mi bardziej o otwarte podejście do technologii i języków. Piszesz aplikacje natywne? Nie krytykuj JIT (na przykład Javy lub .NETu). Piszesz w JIT? Nie dyskredytuj języków skryptowych. Myślę, że każdy język ma swoje zastosowanie w takim lub innym obszarze i mądry programista to nie taki, który próbuje użyć jednego do wszystkich obszarów, ale raczej dobiera go do konkretnego problemu, z którym w danym momencie jest skonfrontowany.

JA NAPISZĘ NOWE, LEPSZE

Na jednej z rozmów z moim teamleaderem podsumowujących rok pracy usłyszałem od niego: „Za dużo rzeczy robisz sam od nowa, mało korzystasz z gotowych rozwiązań”. Nawet nie próbowałem się kłócić, to po prostu cały ja.

Kiedy pisałem jeszcze w Delphi, wśród jego bibliotek standarodowych brakowało parsera XMLa. Wprawdzie w system Windows od którejś jego wersji jest wbudowany silnik MSXML, jednak jest on udostępniany poprzez COM, do którego miałem straszną awersję. Niewiele więc myśląc, zacząłem pisać od zera własny parser XMLa w czystym Delphi i dojechałem – zgodnie ze standardem (który jest większy i bardziej skomplikowany niż mogłoby się wiele wydawać) – do XMLowych przestrzeni nazw, w których to momencie mój zapał nieco ostygł i parser na tym etapie zamknąłem. Użyłem go nawet do jednego czy dwóch swoich projektów, potem udostępniłem za darmo grupowiczom pl.comp.lang.delphi i na dobrą sprawę na tym jego żywot się zakończył.

Jeszcze jednym przykładem przepisywania istniejących technologii było zaimplementowanie przez mnie sporej części definicji Microsoftowego Ribbona – czyli „wstążki”, interfejsu, który pojawił się w Microsoft Office 2007, a potem rozprzestrzenił do mnóstwa innych aplikacji. Delphi cierpiało na brak dobrej implementacji tego komponentu, a ponieważ pisanie komponentów – o czym być może wiedzą stali czytelnicy „Programisty” – jest jednym z moich koników, zabrałem się do pisania go od zera. Oczywiście implementacja kompletnego komponentu po godzinach zajęłaaby miesiące pracy, więc zatrzymałem się na etapie zakładek, tafl i przycisków, ale był on już na tyle funkcjonalny, że można było z niego swobodnie korzystać podczas pisania aplikacji. Ponieważ był to jeden z moich ostatnich projektów w Delphi, zdecydowałem się go zopensource'ować i przekazałem społeczności Lazarusa, który jest darmowym i otwartym odpowiednikiem Delphi. Kontrolka wciąż figuruje w repozytorium Lazarusa i jest rozwijana, można ją tam odnaleźć pod nazwą TSpkToolbar.



Rysunek 3. Ostatnia wersja TSpkToolbar przed udostępnieniem go społeczności Lazarusa

Dalej. Na studiach mieliśmy przedmiot o nazwie „Techniki komplikacji”, na którym poznaliśmy sposoby wydajnej analizy kodu źródłowego (parsersy, leksery). Jednym z zadań było skorzystanie z narzędzi Flex oraz Bison do wygenerowania kalkulatora operującą-

cego na prostych wyrażeniach matematycznych. Jednak Flex jest skonstruowany w taki sposób, że generuje cały program, gdy tymczasem – jak mi się wydawało – znacznie wygodniej byłoby, gdyby wygenerował pojedynczy moduł lub klasę parsera, którą można byłoby potem łatwo zintegrować z własnym programem. Niewiele więc myśląc siadłem i w kilka godzin napisałem własny odpowiednik Flexa, działający właśnie w opisany przeze mnie sposób. Powstałego wówczas SpkParserGeneratora wciąż używam, gdy mam potrzebę przeparsowania jakiegoś pliku, i generalnie robi swoją robotę, ale Flexa nie znam do dziś. Na studiach uratowało mnie nonkonformistyczne podejście – na zajęciach prowadzący był tak zaskoczony, że napisał własny generator parserów, iż pozwolił mi skorzystać z niego we wszystkich zadaniach normalnie wymagających Flexa.

Trzeba tu jednak na marginesie przyznać, że wyposażylem go też w kilka ciekawych rozwiązań, których – według mojej wiedzy – nie ma we Flex, na przykład możliwość zastosowania kilku automatów i przełączanie się między nimi, by częściową analizę gramatyczną wykonać już na etapie analizy składni (w ten sposób można na przykład odróżnić znak minusa będący binarnym operatorem odejmowania od tego samego minusa będącego unarnym operatorem ujemności). W ramach ciekawostki prezentuję poniżej plik wejściowy do mojego generatora, przykładowy program korzystający z wygenerowanego parsera i na końcu efekt jego działania.

Listing 3. Plik wejściowy do generatora parserów

```
1,Number->2=[0-9]+(\.[0-9]+)?
1,Variable->2=[a-zA-Z][a-zA-Z0-9]*
1,OpenParenthesis->1=\(
2,CloseParenthesis->2=\)
2,Operator->1=[\+\\-\/*\\/]
```

Listing 4. Przykładowy program korzystający z wygenerowanego parsera

```
#include <stdio.h>
#include <string>
#include "Calc.h"

int main(int argc, char * argv[])
{
    std::string expression = "2*(34.5+4*pi-(4+8))";
    CTokenizer MyTokenizer;

    const unsigned char * cursor = (unsigned char *)
        (expression.c_str());
    while (*cursor != 0)
    {
        CToken token = MyTokenizer.Process(cursor);

        if (token.tokenType == ttUnknown)
            return 1;

        printf("%s\t\t%s\n", CTokenizer::TokenTypeToStr(token.
            tokenType).c_str(), token.TokenStr.c_str());
    }

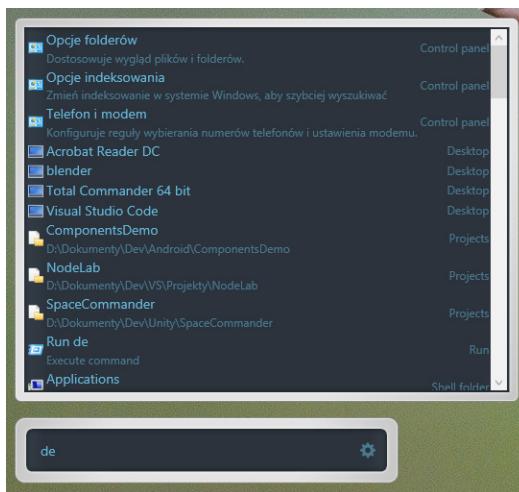
    getchar();
    return 0;
}
```

Listing 5. Efekt jego działania

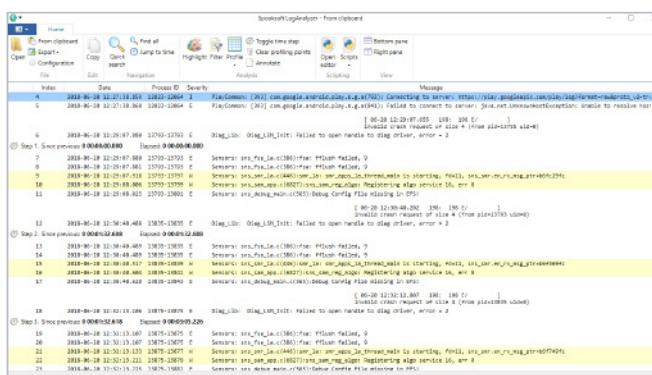
Number	2
Operator	*
OpenParenthesis	(
Number	34.5
Operator	+
Number	4
Operator	*
Variable	pi
Operator	-

```
OpenParenthesis (
Number      4
Operator    +
Number      8
CloseParenthesis )
CloseParenthesis )
```

Oprócz pchełek, których napisanie zabrało średnio kilkanaście, ewentualnie kilkadziesiąt godzin, mam też na koncie kilka aplikacji, które technicznie również powielają gotowe rozwiązania, ale mimo wszystko zdecydowałem się je napisać. Jestem więc autorem dosyć zaawansowanego programistycznego kalkulatora, który pomagał mi przy różnych matematycznych zagadnieniach pisanych przeze mnie aplikacji, na co dzień korzystam też z „Z” (chyba każdy musi napisać aplikację z oryginalną nazwą), który jest w zasadzie zamiennikiem dla menu Start – launcherem z wyszukiwarką, który pomaga szybko dostać się do różnych miejsc w systemie. Częściowo na potrzeby projektu, nad którym obecnie komercyjnie pracuję, napisałem też narzędzie wspomagające analizę plików logów – można je zobaczyć na Rysunku 4. Aplikację opublikuję na mojej stronie, gdy tylko znajdę chwilę czasu.



Rysunek 4. Z – launcher dla Windows



Rysunek 5. LogAnalyzer – analizator plików logów

Przepisywanie od nowa istniejących rozwiązań to śliski temat. Pisanie na nowo parsera XMLa chyba nie było warte zainwestowania czasu. Zrobiłem to właściwie głównie pod wpływem dużej niechęci do poznawania nowej technologii i skończyłem z nieznaną jomością COM, MSXML i z kilkuset linijkami kodu, które właściwie utknęły w szufladzie (kawał dobrej, nikomu niepotrzebnej roboty). Generator parserów przydaje się od czasu do czasu, z perspektywy

akademickiej było to doświadczenie bardzo ciekawe, ale znów powieściło rozwiązań, które już istnieją na rynku. Przy SpkToolbarze nauczyłem się trochę, w jaki sposób integrować się ze środowiskiem programistycznym, ale z drugiej strony poświęciłem sporo czasu na zrobienie czegoś, co już istniało, i przy odrabinie samozaaparcia zapewne mógłbym z tego po prostu skorzystać.

Z aplikacjami było już trochę inaczej – przynajmniej w przypadku ostatnich dwóch zrobiłem solidny i uczciwy research i nie udało mi się znaleźć darmowego rozwiązania, które by mnie usatysfakcjonowało. W ten sposób powstał kalkulator, który jest jednocześnie bardzo lekki i bardzo funkcjonalny, launcher, który nie jest może bardzo efektywny, ale znaczco przypomina moją codzienną pracę, oraz analizator logów, który chyba jako jedyny na rynku pozwala na automatyzowanie analizy logów przy użyciu skryptów pisanych w Pythonie.

Czy warto poświęcać czas na tego typu projekty? Wydaje mi się, że moją odpowiedzią jest ostrożne „nie” – przynajmniej w kwestii wspomnianych trzech pierwszych. We współczesnym świecie programistycznym ogromna liczba frameworków i bibliotek jest dostępna całkowicie za darmo, a często również w postaci otwartego kodu. Gotowe rozwiązania mają też tę zaletę, że są rozwijane od dłuższego czasu, przetestowane, konserwowane – błędy często są naprawiane na bieżąco przez społeczność – dostajemy więc stabilny i funkcjonalny produkt, którego możemy od razu użyć bez obaw, że będzie w przyszłości powodował jakieś poważniejsze problemy. Napisanie własnego rozwiązania może wydawać się na początku atrakcyjne, ale tylko do momentu, w którym uświadomimy sobie, że skazujemy się na samotne jego testowanie, rozwijanie i naprawianie błędów, a to pochłania naprawdę bardzo dużo czasu. I nie ma co się czarować – na pewno nie osiągnie ono takiego funkcjonalnego poziomu, jak produkty rozwijane od dłuższego czasu przez społeczności programistyczne albo wręcz przez wielkie korporacje. Dlatego też czas zaplanowany na kodowanie takiego rozwiązania chyba lepiej przeznaczyć na poznanie jakiejś nowej technologii – prawdopodobnie przyniesie to znacznie więcej korzyści w stosunku do włożonego wysiłku.

WŁASNE PROJEKTY

Programista, lekarz i prawnik to trzy zawody, do których nie wolno przyznawać się na imprezach (podpowiedź: „A wiesz, ostatnio miałem problem z podłączeniem drukarki, może będziesz wiedział”). Mają one jednak jeszcze jedną wspólną cechę – trzeba się ich uczyć całe życie.

Spójrzmy choćby na rynek programistyczny. Na początku wybuchło zapotrzebowanie na aplikacje i gry desktopowe. Potem popularny zrobił się powoli web, więc pojawiły się rozwiązania backendowe (np. PHP) i frontendowe (JavaScript), które ewoluowały, by sprostać rosnącemu zapotrzebowaniu na nowe funkcjonalności. Potem nagle wybuchł świat mobilny – najpierw Windows CE i bardzo popularny Symbian, które jednak potem oddały pole Androidowi i iOSowi, a w tle równolegle dynamicznie rozwijało się programowanie wbudowane (embedded). Teraz na topie jest chmura i sztuczna inteligencja, ale w kolejce ustawa się już na przykład programowanie kwantowe. Razem z platformami zmieniło się też zapotrzebowanie na języki – C++, którego większość programistów znała od samego początku, okazał się w niektórych miejscach zbyt skomplikowany lub niewygodny, więc pojawił się PHP, Java, C# i cała mnogość innych, dopasowywanych często do potrzeb określonego, niejednokrotnie specyficznego ekosystemu.

W jaki sposób najłatwiej zmotywować się do poznawania nowych technologii? Moim zdaniem świetnym środowiskiem do tego są własne, prywatne projekty.

Od samego początku mojej programistycznej kariery bodaj cały czas miałem gdzieś w tle jakiś projekt, nad którym w danym momencie pracowałem. Często wszystko zaczynało się od jakiegoś niewinnego „Fajnie byłoby, gdyby istniał program, który robi...”, a potem siadałem do klawiatury i zaczynałem pisać.

Nie ukrywam oczywiście, że bardzo często w efekcie moich sesji radosnego szala twórczego powstawały prawdziwe potworki, jak na przykład edytor programistyczny, z poziomu którego można było wysyłać SMSy. Jednak każda linijka kodu to było zdobycie doświadczenia, każdy błąd, który popełniłem w projektach prywatnych, ustrzegł mnie przed popełnieniem go w projektach komercyjnych, każda poznana biblioteka wzbogacała moją programistyczną wiedzę. Może to brzmieć banalnie, ale z moich prywatnych projektów naprawdę wyciągnąłem dużo wiedzy i doświadczenia, które przydaje mi się teraz, podczas pracy komercyjnej.

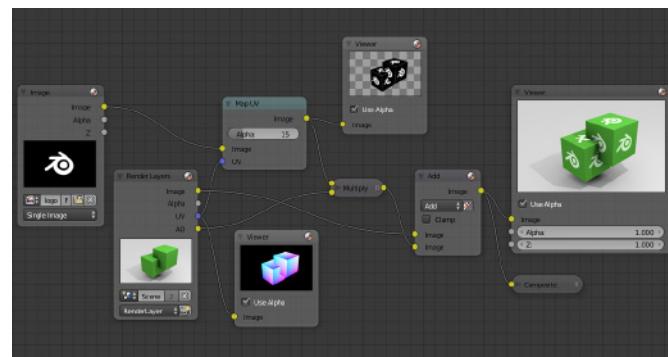
Projekt prywatny – poza oczywistą wadą w postaci konieczności poświęcenia mu własnego, wolnego czasu – ma bodaj same zalety. Nikt nie narzuca tematu, nie ma konkretnej technologii, z której trzeba korzystać, nie ma wymogów dotyczących formowania kodu. Co ważne, nie ma też wymuszonego SRP, SOLIDa i innych dobrych praktyk programowania. Uważam to za zaletę, bo można dzięki temu samodzielnie przekonać się o przydatności danej reguły (albo wręcz przeciwnie – znaleźć się w sytuacji, gdy powoduje ona więcej szkód niż pożytku, tak też się zdarza). Kiedy pierwszy raz usłyszałem Wujka Boba mówiącego, że „metoda może mieć maksymalnie 6-7 linijek”, moją reakcją było zdecydowanie, cytuje „chyba sobie jaja robisz, koleś”. Pisalem wtedy metody dłużości – czasem – kilkuset linii kodu i wszystko działało. Ale minął czas, zacząłem wracać do moich starych aplikacji, zgrzytałem zębami, nie mogąc przekopać się przez tony spaghetti, i zaczynałem sam – mniej lub bardziej świadomie – wprowadzać kolejne postulaty Clean Code. Oczywiście od strony kodu SOLID wypracowany przez mnie samodzielnie w prywatnych projektach nie różnił się niczym od SOLIDA, o którym mógłbym choćby przeczytać w Internecie, ale ta kluczowa różnica była we mnie: stosowałem reguły, bo doświadczylem ich korzyści, a nie ślepo podążając za kimś, kto powiedział mi, że mam tak robić. Poza tym stosowanie SOLIDA podczas projektowania architektury aplikacji niejako weszło mi w krew – do tego stopnia, że mam teraz wręcz trudności z napisaniem kodu, który stoi w sprzeczności z tymi regułami.

Projekty prywatne oddały mi też jeszcze jedną przysługę – otóż na rozmowę kwalifikacyjną wziąłem notebooka i zapytałem technicznego rekrutera, czy chce obejrzeć napisane przeze mnie aplikacje. Ponieważ był zdumiony, zapytałem, czy coś jest nie tak, i usłyszałem w odpowiedzi: „Nie, ale jest pan pierwszą osobą, która chce pokazać swoje projekty”. Nie wiem, w jakim stopniu moja prezentacja wpłynęła na pozytywny wynik rekrutacji, ale na zdrowy rozum, jeżeli zatrudnimy malarza, to pewnie bardziej niż referencje i ukończone przez niego uczelnie interesują nas namalowane przez niego obrazy. Na marginesie: możliwość opowiedzenia o zastosowanych rozwiązań minimalizuje szansę na otrzymanie krępującego pytania, na przykład: „Czy wiesz, czym jest polimorfizm?”.

CZASEM TRZEBA SIĘ PODDAĆ, ŻEBY WYGRAĆ

Jeżeli już jesteśmy przy prywatnych projektach, chciałbym opowiedzieć o dwóch, które wspominam w sposób szczególny.

Czy korzystaliście kiedyś z Blendera? Ma on wbudowany bardzo ciekawy mechanizm, który pozwala na projektowanie dynamicznych materiałów poprzez wizualne układanie na ekranie węzłów i wiązanie ich ze sobą. Edytor ten spodobał mi się w dużej mierze dlatego, że jest niesamowicie czytelny – nawet jeżeli dany graf ktoś ogląda po raz pierwszy, może bardzo szybko określić, co on robi. Również krzywa nauki jest bardzo łagodna – wystarczy tylko nauczyć się, jakich węzłów można użyć i jak działają, by móc zacząć projektować swoje pierwsze materiały. Zamarzyło mi się przygotować aplikację, która stanowiłaby taki grafowy edytor-framework, który można byłoby dostosować za pomocą pluginów do realizowania dowolnych zadań.



Rysunek 6. Edytor grafu postprocessingu w Blenderze (źródło: <https://docs.blender.org/manual/en/dev/compositing/introduction.html>)

Przeważnie w takich sytuacjach miałem już jakiś wstępny pomysł, jak co napisać, ale tym razem było inaczej – założyłem bardzo dużą swobodę w zakresie projektowania rozszerzeń, co w wielu miejscach mocno komplikowało architekturę aplikacji, a do tego doszły też kwestie techniczne (przygotowanie API, przez które pluginy mogłyby się łączyć z aplikacją), wizualizacyjne (musiałem przygotować dosyć duży komponent wyświetlający graf i zapewniający płynną interakcję z użytkownikiem) i na końcu też strukturalne (musiałem przygotować mechanizm do zapisu i odczytu przygotowanego grafu). Walczyłem z prototypami kilka tygodni bez większych sukcesów, aż w końcu doszedłem do momentu, w którym stwierdziłem: „To mnie przerasta”.

To nie było zmęczenie materiału – potrafię wykrzesać z siebie nawet tyle determinacji, żeby zrobić naprawdę dużą i nudną refaktoryzację, która poprawi architekturę moich aplikacji. Doszędłem jednak do wniosku, że zadanie przekracza mój bieżący zakres wiedzy i doświadczenia i po prostu nie jestem w stanie go sam zrealizować. Od czasu do czasu zdarzało mi się wracać do tematu – podałem chyba jeszcze dwie próby zrealizowania tego zadania, obie zupełnie na czysto, żeby nie powielać wcześniejszych błędów, i obie zakończone w ten sam sposób.

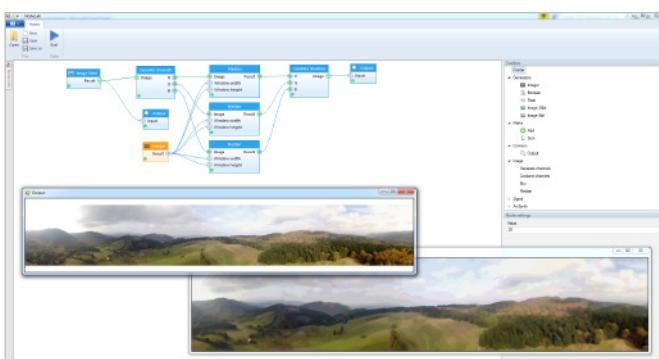
Podobnie było z innym projektem – klonem mało znanej, starej gry Critical Mass (z uwagi na popularną nazwę w Internecie będzie ją łatwiej odnaleźć po wpisaniu autora – Sean O’Connora). Tym razem chodziło o zagadnienie obliczeniowe – zaprojektowanie matematycznego modelu nawigacji statku kosmicznego w dwuwymiarowej przestrzeni. Większość czasu spędziłem przy kartce z ołówkiem w ręku, starając się opracować deterministyczny algorytm, który umiałby pokierować statkiem niezależnie od zastanego stanu (położenia, orientacji i prędkości). Również i tym razem musiałem pogodzić się z porażką – nijak nie udawało mi się osiągnąć zadowalających wyników.



Rysunek 7. Gra Critical Mass (źródło: <http://www.windowsgames.co.uk/critical.html>)

Z perspektywy czasu uważam, że moje decyzje o zawieszeniu projektów były bardzo dobre, ponieważ świadczyły o tym, że bardzo realnie podszedłem do moich ograniczeń. Uparte ciągnięcie ich dalej nie dałoby pewnie żadnych pozytywnych efektów, a tylko wzmo-głoby moją frustrację. Co natomiast zabawne, w późniejszym czasie udało mi się skończyć zarówno pierwszy, jak i drugi z nich.

W przypadku pierwszego musiałem przegrywać się przez swoją nieposkromioną puchę („Dam sobie radę sam”), złapałem w firmowej kantynie znacznie bardziej doświadczonego ode mnie kolegę, wyłuszczyłem problem i spytałem, w jaki sposób by do niego podszedł. Ten praktycznie od razu podrzucił mi ciekawe rozwiązanie – żebatki w głowie zaczęły się kręcić, po powrocie do domu siedłem do środowiska i kilka dni później miałem już wstępny, ale działający prototyp.



Rysunek 8. NodeLab

Również i matematyczny problem udało mi się rozwiązać. Tym razem kluczem stało się między innymi uproszczenie założień, bo po dłuższym namyśle stwierdziłem, że przy założeniach, jakie przyjąłem na początku problemu, nie udałoby się rozwiązać go metodami deterministycznymi. W tym przypadku musiałbym raczej zaprząć do pracy algorytm PID albo jakiś heurystykę, zgadującą trochę, co w danym momencie zrobić. Poza tym również i tym razem otrzymałem pomoc od osoby trzeciej, a dokładniej na portalu math.stackexchange.com – jeden z piszących tam matematyków pomógł mi rozwiązać blokujące dalszy postęp zagadnienie, co pozwoliło mi pokonać ostatni problem, i projekt ruszył.

Myszę, że ważną cechą dobrego programisty jest świadomość swoich niedoskonałości. Bardzo często w komercyjnych projektach zdarza się trafić albo na błąd, albo na problem do rozwiązania, który ciągnie się godzinami lub dniami i nie udaje się go rozwiązać. W takim przypadku bardzo często najprostszą metodą rozwiązania go jest po prostu podejść do kogoś i poprosić o pomoc.

Wbrew pozorom, choć może być to zaskakujące, bardzo często programiści mają z tym naprawdę duży problem. Gdy prowadziłem kółko w moim liceum, młodzi adepti programistycznej sztuki tak bardzo bali się zadawać pytania, że od pewnego momentu zacząłem zmuszać ich do tego, by podnosili rękę nie ci, którzy mają pytania, ale ci, którzy ich *nie mają*. Z uwagi na to, że teraz zamiast skłamać pasywnie (poprzez niepodniesienie ręki) musieliby skłamać aktywnie (poprzez podniesienie ręki), wreszcie, chcąc nie chcąc, przyznawali się do niewiedzy i w końcu mogłem dotrzeć do tego, które zagadnienia były niejasne i trzeba je dopowiedzieć lub opisać w inny sposób. Na szczęście później nabrali trochę śmiałości i zaczęli pytać już z własnej inicjatywy. Również niektórzy praktykanci, których prowadziłem, woleli siedzieć godzinami i dojść do rozwiązania samodzielnie zamiast podejść i zapytać bardziej doświadczonych kolegów. Prosiłem i błagałem, ale dopiero po pewnym czasie udało im się przełamać jakąś psychiczną barierę i faktycznie zaczęli zadawać pytania.

Pracuję w firmie, która kładzie bardzo duży nacisk na dzielenie się wiedzą – oprócz często pojawiających się warsztatów i krótkich prezentacji (lightning-talk) panuje tu również swoista atmosfera komfortu – mam pewność, że nawet gdybym pojechał do naszego rzeszowskiego albo gdańskiego oddziału (gdzie mnie mało kto zna), mogę podejść tam do dowolnej osoby, która ma doświadczenie w kwestii, którą potrzebuję rozwiązać, poprosić o pomoc i tę pomoc otrzymam.

WYJAZDY I KONFERENCJE

Kiedy byłem jeszcze mocno zaangażowany w programowanie w Delphi, kilkukrotnie jeździłem na organizowane przez członków grupy dyskusyjnej pl.comp.lang.delphi Zloty Programistów Delphi. Pierwszy, w którym brałem udział, odbył się w Poznaniu, zaś pozostałe w Krakowie na Akademii Górnictwo-Hutniczej. Zloty miały już wtedy ukształtowaną formułę – w piątek było zawsze zakwaterowanie i integracja, sobota była dniem wykładów i warsztatów – wykłady odbywały się w dwóch ścieżkach, aby można było zawsze wybrać coś dla siebie ciekawego, zaś dzień kończył się „developer party” w jakimś pobliskim klubie. Niedziela była natomiast dniem zakończenia i podsumowania Zlotu oraz – oczywiście – powrotu do domów. Co było wyjątkowo wartościowe, to to, że Zlot odbywał się pod oficjalnym patronatem BSC Polska: polskiego przedstawiciela Borlanda, a potem Embarcadero. Dzięki temu jeden z warsztatów zawsze prowadzony był przez pracownika BSC – według mojej pamięci był to zazwyczaj Bogdan Polak, który przedstawiał nowości w środowisku i języku i kierunek rozwoju, w którym obecnie zmierza Delphi.

Wszystkie Zloty wspominam naprawdę fantastycznie. Umówmy się: jeden dzień wykładów to za mało, żeby nauczyć się czegoś zaawansowanego; warsztaty przeznaczone były przeważnie dla programistów początkujących i przybliżały im tylko podstawy programowania obiektowego i Delphi jako takiego. Jednak możliwość przedyskutowania różnych programistycznych kwestii z innymi developerami na żywo, wgląd w nowości, wiedza na temat różnych bibliotek i rozwiązań, które prezentowane były na wykładach – to było nie do przecenienia. Na sobotnich developer party

zwykle znajdowałem się w gronie maniaków, którzy wprawdzie opróżniali kufle soku pomarańczowego, a nie piwa, ale na tematy programistyczne dyskutowali z takim zacięciem, że następnego dnia ledwie mogli wydobyć z zachrypnietych gardeł głos. Napić się piwa można zawsze i wszędzie, natomiast znaleźć w gronie programistów-pasjonatów i omówić różne nurtujące mnie kwestie – takie coś nie zdarzało się zbyt często. Śmiałem się trochę, że to były jedynie trzy dni w roku, gdy wszyscy dookoła rozumieliby, o czym do nich mówię. Złoty dawały mi niesamowitego, energetycznego kopa – wyjeżdżałem stamtąd zawsze z głową wypchaną po brzegi pomysłami, czego nowego mogę się nauczyć i jakie nowe aplikacje mogę spróbować napisać. W 2009 roku odbył się ostatni Złoty, na którym byłem, a potem nastąpiła długa przerwa. Sprawdziłem przed chwilą, że Złoty zostały jednak wznowione i ostatni odbył się dosłownie na dniach – jeszcze w czasie, gdy piszę ten artykuł. Cieszę się, że społeczność Delphi znów zmobilizowała się, by znów zacząć się spotykać i wymieniać wiedzą.

Kiedy przeniosłem się na języki i środowiska wspierane bardziej przez Microsoft, zacząłem z kolei jeździć na Microsoft Technology Summit. Oczywiście imprezy te miały znacznie większą skalę niż Złoty – przyjeżdżały setki ludzi, ścieżek wykładowych było bodaj sześć czy osiem, poza tym na terenie konferencji wystawiały się różne firmy powiązane z technologiami MS. Konferencje te – głównie z uwagi na swój rozmiar – nie miały już takiego klimatu jak spotkania programistów Delphi, ale jeździłem na nie z równym zaangażowaniem i wyjeżdżałem równie pełny energii co po Złotach. Zdecydowanie było warto i trochę szkoda, że konferencje te nie są już organizowane; miałem możliwość wzięcia udziału bodaj w trzech ostatnich.

Ostatnimi czasy nie jeżdżę już tak często na konferencje programistyczne, ale uważam, że są to niesamowicie wartościowe wydarzenia, w których definitywnie warto brać udział – nawet tylko po to, by zaobserwować bieżące trendy, zmiany na rynku IT i po prostu poznać trochę nowości.

ZDROWA DAWKA UPORU

Pisałem kiedyś komercyjny projekt – z grubsza był to edytor pliku XML opisującego pewną szczególną strukturę danych. Oprócz standardowych zasad XML miał on wprowadzone jeszcze kilka dodatkowych reguł (na przykład możliwość określania zależności pomiędzy elementami), wykraczających nieco poza możliwości XSD (który dawał tu tylko powierzchniową możliwość weryfikacji struktury przetwarzanego pliku).

Zaproponowałem klientowi, że najpierw zbuduję bazową strukturę danych, która odzwierciedli i będzie w locie weryfikować wszystkie te rozszerzone reguły rządzące ich specyficznym formatem, potem dopiero na niej zbuduję kolejną strukturę, odzwierciedlającą tym razem konkretne dane, które w tej strukturze były przechowywane, a na końcu dopiero operującą na tej ostatniej edytor. Podałem również estymację, na którą klient zareagował: „A może po prostu napisz to wszystko w jednej strukturze albo operuj na samym XML?”. Doszczętnie jednak do wniosku, że spowoduje to usunięcie z architektury całego edytora warstwy, która daje mi dodatkową kontrolę nad edytowanym dokumentem, i pozostałem na stanowisku, że lepiej będzie zrobić to po mojemu. Na moje szczęście klient był programistą, więc w końcu zgodził się na moje rozwiązanie (nieco na zasadzie „No doooooobra”, ale jednak). Rozwój edytora trwał prawie rok i udało się go z powodzeniem ukończyć i wprowadzić na rynek.

Chyba miesiąc albo dwa później klient odezwał się znów z prośbą o dodanie nowej funkcjonalności. Gdy spytałem o szczegóły,

dowiedziałem się, że chodzi mu o możliwość cofania wprowadzonych zmian („Undo”).

Jeżeli, drogi Czytelniku, pisałeś kiedyś edytor czegokolwiek, masz pewnie świadomość, jak trudne jest zrobienie możliwości cofania wprowadzonych zmian *we właściwy sposób*. Efektywnie każda pojedyncza akcja podjęta przez użytkownika musi zostać zarejestrowana i jednocześnie konieczne jest dodanie mechanizmu, który potrafi taką akcję wycofać. Jeżeli ktoś nie pomyślał o przygotowaniu odpowiedniej funkcjonalności od samego początku, takie zadanie jest rodem z najgorszego koszmaru – wymaga przebudowania prawie całej aplikacji i jest niesamowicie błędogenne.

Napisany przeze mnie edytor nie był do tego przygotowany *per se*, ponieważ klient na samym początku *explicitely* powiedział mi, że funkcjonalność taka nie będzie potrzebna. Jednak pod spodem cały czas siedziała moja generyczna struktura, przez którą przechoǳiła każda pojedyncza operacja zrealizowana przez użytkownika. Dokładnie tak: *każda, pojedyncza*. W efekcie wystarczyło więc do tego mechanizmu dodać rejestrację takich operacji (w tym również operacji kompozytowych – takich par, które muszą zostać cofnięte razem), do tego dopisać prosty interfejs użytkownika wyliczający wszystkie zrealizowane przez niego operacje i pozwalający na ich wycofanie, i robota była skończona. Całość zajęła ledwie kilka dni. Gdybym zdecydował się na początku na uproszczenie struktury danych według sugestii klienta, prawdopodobnie skończyłbym z zadaniem na kilka tygodni albo – co bardziej prawdopodobne – nawet miesiąecy.

Moja historia nie ma być oczywiście zachętą do tego, żeby na siłę wciskać zawsze klientom swoje rozwiązanie jako najlepsze. Trzeba słuchać ludzi, zdarzało mi się też, że to właśnie klient zaproponował lepsze rozwiązanie od tego, które miałem już poukładane w głowie. Jeśli jednak wiesz, że twoje rozwiązanie jest *dobre* (mam na myśli na przykład zasady SOLID, elastyczność architektury, odporność na potencjalne błędy i tak dalej) i widzisz, że jego niezastosowanie może doprowadzić do poważnych konsekwencji, spróbuj o nie powalczyć. W końcu to Ty będziesz rozwijał i utrzymywał kod, który właśnie piszesz. Nie ma co pakować się samemu w bagno, jeśli jest możliwość, by klienta od tego odwieść.

TECHNOLOGIE SĄ DLA LUDZI

Pamiętacie moje przygody z COM? Moja niechęć spowodowana była tym, że technologia ta ma pewne specyficzne wymagania, których trzeba się trzymać. Interfejsy, IUnknown, GUIDy, zliczanie referencji, QueryInterface – to wszystko wydawało mi się niepotrzebnym komplikowaniem spraw, tym bardziej że w Delphi koncepcja interfejsów została wprowadzona właśnie głównie pod COM. Doszczętnie więc do wniosku, że z tym COMem jest za dużo zachodu i dałem sobie z tym spokój.

Z czasem moje podejście do nowych technologii zmieniło się. Uświadomiłem sobie bowiem, że technologie są dla ludzi – one nie mają być z założenia tak skomplikowane, że nie da się ich nauczyć, tylko wręcz przeciwnie – są przeważnie skomplikowane tylko tak bardzo, jak wymaga tego mechanizm, którego abstrakcję stanowią. COM zawiera wszystkie koncepcje, o których wspomniałem, nie dlatego, że jest niepotrzebnie skomplikowany, tylko dlatego, że stanowi binarny pomost standaryzujący komunikację pomiędzy różnymi aplikacjami napisanymi w różnych językach, więc wymagał opracowania spójnego i jednolitego interfejsu, z którego będzie można skorzystać w każdym z nich. Co więcej, jest on dosyć dobrze udokumentowany, istnieje w Internecie mnóstwo przykła-

dów, jak go użyć, więc wystarczyło poświęcić trochę czasu i mógłbym z niego swobodnie korzystać.

Jakiś czas później postawiono przede mną i kolegą zadanie zaimplementowania wizualizacji trójwymiarowej danych medycznych. Mieliśmy do wyboru OpenGL i DirectX; w pierwszym napisałem kilka „Hello-worldów”, drugiego nie znałem kompletnie – nie miałem z nim wcześniej do czynienia ani jeden raz. Z uwagi jednak na to, że projekt powstawał w WPFie, zdecydowaliśmy się na DirectX, licząc na lepsze wsparcie na etapie integracji naszego kodu do całej aplikacji.

To była naprawdę rozwijająca przygoda. Zaczęliśmy od napisania kilku *proof-of-conceptów*, trzeba było przekopać się przez różne tutoriale i dokumentację, a było to o tyle trudne, że korzystaliśmy ze świeżo wprowadzonego wówczas DirectX11, do którego nie było jeszcze w Internecie tak dużo przykładów, jak do jego poprzednich wersji. Kolejnym krokiem było napisanie prostego, ale dosyć elastycznego silnika, na którym budowaliśmy potem wizualizację i interakcję z użytkownikiem. Przyłożyliśmy się do niego naprawdę starannie: pierwszy raz w życiu z własnej woli usiadłem nad diagramami UML, żeby dobrze przemyśleć interakcje pomiędzy składowymi tego silnika i ocenić potencjalne możliwości jego późniejszego rozwoju. Opłaciło się, bo potem okazało się, że w trakcie implementowania naszego rozwiązania musielimy ten silnik nieco przerobić albo rozbudować i jego architektura za każdym razem się wybroniła – nie musielimy wszystkiego przepisywać później od nowa.

Potem na silniku zaczęliśmy budować samą wizualizację i nieśmiało zaczęły pojawiać się pierwsze efekty. I po kilkunastu tygodniach pracy nagle okazało się, że udało nam się osiągnąć cel – nasza wizualizacja została zintegrowana z aplikacją i do końca mojej pracy nad tym projektem nie został zgłoszony do niej żaden poważniejszy błąd.

Pamiętajmy, że żyjemy w czasach gwałtownego rozwoju świata IT. Nowe technologie to nie potwory, które przyjdą i nas zjadą, gdy tylko spróbujemy się nimi zająć. Wszystko jest robione dla ludzi – mało, ostatnio zauważam bardzo pozytywny trend, by API dostarczane programistom były jak najbardziej wygodne w użytkowaniu. Często działają one nawet na zasadzie dobrze przygotowanego środowiska domyślnego, które można potem – w miarę coraz głębszego poznawania danej technologii – coraz bardziej dopasowywać do własnych potrzeb, zastępując domyślne implementacje własnymi, skrojonymi na miarę.

SZEROKIE ZAINTERESOWANIA

Kiedy byłem jeszcze w liceum, zainteresowałem się trochę assemblerem. Kupiłem ze dwie książki i zacząłem czytać, a potem siedłem do pisania kodu. Wyniki mojej pracy były zdecydowanie mało efektowne, bo szczytem moich osiągnięć był bodaj programik, który wyświetlał w DOSowej konsoli liczby o 1 do 10. Całość trwała może dwa tygodnie, może trochę więcej, w każdym razie temat zarzuciłem i wróciłem do Pascala.

Wiele lat później, podczas rozwijania aplikacji medycznej, zespół, do którego należalem, stanął przed trudnym zadaniem zoptymalizowania całego procesu przetwarzania danych do bardzo krótkiego czasu – rzędu trzech sekund. Walczyliśmy zaciekle o każdy ułamek sekundy, zarówno na wysokim poziomie – na przykład przez zastosowanie wysoce zoptymalizowanych bibliotek IPP (Intel Performance Primitives) – jak i na poziomie samej implementacji, na przykład przez zrównoleglanie fragmentów kodu, które się do tego nadawały, albo upraszczaniu niektórych obliczeń.

Nadziałem się tam na prostą funkcję, której zadaniem było znalezienie liczby będącej potęgą dwójką nie mniejszą od zadanej

liczby. Rozwiążanie zaimplementowane było prostą pętlą z warunkiem, która testowała coraz to większe potęgi dwójką, aż do spełnienia wymaganego warunku. Coś mnie jednak tknęło i zacząłem wertować specyfikację assemblera procesorów Intel, aż znalazłem polecenie BSR, czyli Bit Scan Reverse (<https://goo.gl/mszZWP>, strona 3-110). Sposób jego działania opisany był następująco:

Listing 6. Pseudokod instrukcji BSR (źródło: dokumentacja architektury procesorów Intel)

```
IF SRC = 0 THEN
    ZF ← 1;
    DEST is undefined;
ELSE
    ZF ← 0;
    temp ← OperandSize - 1;
    WHILE Bit(SRC, temp) = 0 DO
        temp ← temp - 1;
    OD;
    DEST ← temp;
FI;
```

Bingo – to było to, czego potrzebowaliśmy. Do sprawdzenia pozostała oczywiście przypadek brzegowy, w którym liczba była już potęgą dwójką, ale również i to można było napisać w czystym assemblerze przy użyciu instrukcji POPCNT, która wyznacza liczbę zapalonych bitów w zadanym operandzie (4-392 w tej samej dokumentacji) – liczby będące potęgami dwójką mają zapalone tylko jeden bit.

Z czystej ciekawości zaimplementowałem moje rozwiązanie – w Delphi jest to bajecznie proste, bo wystarczy zastąpić słowo klużowe begin rozpoczynające implementację metody słowem asm, by móc bezpośrednio pisać w assemblerze. Wynik był powalający – fragment kodu, który zmodyfikowałem, zaczął działać kilkanaście razy szybciej. Wynikało to między innymi z tego, że metoda, którą zmodyfikowałem, sama była wywoływana wielokrotnie w pętli, a jej wykonanie (z uwagi na wewnętrzną pętlę) zajmowało znacznie więcej cykli procesora niż kilka napisanych przeze mnie instrukcji.

Trzeba uczciwie dodać, że moje rozwiązanie nie przyspieszyło jakoś znacząco całego procesu, bo była to tylko niewielka część całego algorytmu, ale jednak zyskaliśmy kolejne ułamki sekund na drodze do zadanych trzech sekund – a więc warto było spróbować.

Podejrzewam, że gdybym w liceum nie poświęcił chwili czasu na poznanie podstaw assemblera, nawet przez myśl by mi nie przeszła próba rozwiązania problemu w ten właśnie sposób. A wystarczyła tak naprawdę elementarna wiedza o komendach, rejestrach procesora i flagach – nic bardzo zaawansowanego, to jest tematyka z pierwszych rozdziałów książek, które kupiłem.

Pomimo tego, że współcześnie ogromną część rynku zajmują języki interpretowane (skryptowe) oraz takie jak Java czy C#, które bardzo mocno odcinają programistę od niskopoziomowego programowania, stoję na stanowisku, że każdy szanujący się programista powinien znać przynajmniej elementarne podstawy assemblera i tego, w jaki sposób działa procesor. Jest to taki rodzaj wiedzy, która często potrafi podrzuścić zupełnie nową perspektywę dla rozwiązywanego problemu albo nawet pomaga domyśleć się, z czego może wynikać dany problem.

Żeby było zabawniej, assembler – wbrew pozorom – jest językiem przeraźliwie prostym. Składa się po prostu ze zbioru instrukcji, z których każda robi coś konkretnego. Żadnych klas, metod wirtualnych, ref-parametrów czy dziedziczenia. Kiedyś z ciekawością napisałem dwa proste, równoważne funkcjonalnie programy w C++ i C# i oglądałem ich disassembly w Visual Studio – bardzo

fajne ćwiczenie. Pomaga zobaczyć, czym tak naprawdę różni się pisanie w języku niższego i wyższego poziomu.

Trochę skupiłem się w tej sekcji na assemblerze, ale tak naprawdę chodziło mi o to, że warto dbać o szerokie horyzonty programistycznej wiedzy. Czasami wystarczy wiedzieć tylko w przybliżeniu, jak coś działa, żeby móc wyciągnąć pomocne wnioski. Myślę, że warto poświęcić trochę czasu na przykład na poznanie języków funkcyjnych czy logicznych, kupienie taniego klonu Arduino Uno (w granicach 25 PLN) i napisanie programu działającego na mikrokontrolerze, spróbowanie swoich sił w pisaniu programów na platformy mobilne, które rządzą się w niektórych aspektach swoimi prawami, czy napisanie compute shadera, który wykona obliczenia na karcie graficznej. Nawet podstawowa wiedza na dany temat może naprawdę dużo przynieść w zupełnie nieoczekiwany momencie.

Oczywiście nie da się uczyć wszystkiego naraz. Mój sposób polegał na tym, że stawiałem sobie za zadanie zrobienie czegoś, co zawierało dla mnie jakąś niewiadomą, coś, czego jeszcze nie umiałem. Na przykład nieco wiedzy na temat programowania mikrokontrolerów zdobyłem, projektując maszynę do timelapseów (<https://majsterkowo.pl/maszyna-do-timelapseow/>). W tej chwili na biurku obok leży STM32F469NIH6 (<https://botland.com.pl/stm32/7778-stm32f469i-disco-discovery-stm32f469nih6-ekran-dotykowy-4-.html>), który dostałem od żony jako prezent urodzinowy (dziękuję, kochanie!), który jednocześnie jest nowym wyzwaniem, bo programowanie 32-bitowych mikrokontrolerów to już zupełnie inna bajka niż proste i przyjazne 8-bitowe Arduino.

SAMODZIELNE MYŚLENIE

W któryś piątek, kiedy będąc na studiach, wracałem pociągiem do rodzinnego domu na weekend, byłem mimowolnym świadkiem rozmowy jakiegoś gościa z siedzącą obok dziewczyną. Tłumaczył jej zasadę działania licznika rowerowego – wyłowiłem fragment rozmowy: „No i na szprychę roweru zaczepia się taki mikrochip i ten mikrochip wysyła sygnał do odbiornika, który jest zamocowany na widelcu.”

Pal licho ten nieszczęsny mikrochip, będący tak naprawdę zwykłym magnesem, który przesuwając się obok „odbiornika”, czyli cewki, indukuje w niej prąd. Nie każdy musi znać dokładną zasadę działania każdego mechanizmu. Ale wystarczy sobie zadać bardzo proste pytanie: czym ten „mikrochip” miałby być zasilany, jeżeli jest zamocowany na szprysze bez jakiegokolwiek przewodu czy baterii? A kiedy brakuje odpowiedzi, to już jest jakaś wskazówka, że być może rozwiązanie jest zupełnie inne – i być może stanie się to okazją do poczytania gdzieś o tym i poszerzenia swojej wiedzy.

Żeby dopowiedzieć, oczywiście da się zasilać układy elektroniczne bezprzewodowo – tak na przykład działają piórka w telefonach i tabletach Samsunga oraz w tabletach graficznych Wacom (Samsung korzysta właśnie z komponentów od Wacom). Jeśli ktoś jest zainteresowany – niech to właśnie będzie taka okazja do poszerzenia wiedzy.

Samodzielne myślenie to nie tylko umiejętność wyszukania rozwiązania danego problemu w sieci. Czasami można przeprowadzić taki „heurystyczny” ciąg myślowy, który pozwala choćby i na oszacowanie, usprawiedliwione zgadnięcie, w której części aplikacji coś poszło nie tak. Kiedyś śledziłem błąd w module, który pisałem, bo właśnie tam aplikacja się wywalała i kończyła działanie z nieobsłużonym wyjątkiem. Wyjątek leciał bezpośrednio w destruktorze `std::vector`, był to jakiś access-violation, który niewiele mi mówił. Ale logika podpowiadała, że w tym miejscu raczej nie ma prawa polecić wyjątek, bo to by oznaczało, że Micro-

soft dostarczył błędna implementację biblioteki standardowej, a to jest mało prawdopodobne. Znacznie bardziej prawdopodobne było to, że pamięć w jakiś sposób została w międzyczasie uszkodzona i że wcale niekoniecznie musiało się to stać w kodzie, który napisałem. Ja wprawdzie wtedy poległem – po bodaj półtora tygodnia poszukiwań poddałem się i zadanie przejęli moi koledzy z zespołu, ale na końcu faktycznie okazało się, że w zupełnie innym module ktoś próbował dostać się do elementu spoza zakresu tablicy i rozmyazywał znajdujące się zaraz za nią bardzo istotne dla późniejszego działania programu pole (tablica miała stałą liczbę elementów). Błąd propagował się przez aplikację i manifestował dopiero w module, za który byłem odpowiedzialny.

Czasami ciąg myślowy może być bardzo ogólny. Choćby i „gdybym był autorem tej biblioteki, prawdopodobnie zaimplementowałbym to w taki sposób, więc problem może leżeć w tym miejscu”. Oczywiście zgadywanie pozostaje zgadywaniem. Jednak zgadywanie poparte wiedzą, doświadczeniem i – co najważniejsze – samodzielny, logicznym myśleniem ma znacznie większą szansę na powodzenie.

UCZ SIĘ OD LEPSZYCH

Jakiś czas temu pisałem aplikację do transkrypcji utworów muzycznych (czyli taki reverse-engineering, odzyskiwanie zapisu nutowego z utworu w postaci pliku .mp3). Stanąłem wtedy przed koniecznością napisania komponentu wyświetlającego wykres fali dźwiękowej. Byłem już wtedy sporą po okresie pisania wszystkiego od zera bez opamiętania, więc najpierw zrobiłem rozpoznanie rynku, ale nie znalazłem żadnego darmowego rozwiązania, które by mnie usatysfakcjonowało, a wszystkie płatne grubo przekraczały budżet domowego projektu (swoją drogą szkoda, że tak mało firm myśli o hobbystach i nie oferują licencji personalnych do stosowań niekomercyjnych). Ponieważ nie chciałem wynajdywać koła od nowa, zastanowiłem się, gdzie takie rozwiązanie jest już zaimplementowane, i do głowy przyszła mi bodaj najpopularniejsza otwarta aplikacja do edycji i miksuowania dźwięku, czyli Audacity.

Sklonowałem jej źródła, ale samodzielne odnalezienie odpowiedniego fragmentu kodu w gąszczu plików było ponad moje siły, więc napisałem krótki email do jednego z głównych developerów tego projektu. I niedługo potem otrzymałem odpowiedź, w której nie tylko wskazywał mi on odpowiednie miejsce w kodzie, ale nawet z grubsza opisał zastosowany tam algorytm, co pozwoliło mi już na bezproblemowe, samodzielne zaimplementowanie podobnego rozwiązania w mojej aplikacji. Takich sytuacji miałem znacznie więcej.

Takie podejście do sprawy musiałem w sobie w czasie wykształcić, ponieważ będąc znacznie bardziej zaawansowanym w zakresie programowania od większości moich rówieśników z liceum, obrosłem w piórka i wykształcilem w sobie przekonanie, że ze wszystkim z pewnością dam sobie radę sam. Rzeczywistość nauczyła mnie pokory – kilku projektów nie dałbym rady napisać, gdyby nie pomóc kogoś znacznie bardziej ode mnie doświadczonego.

Pokora niestety nie jest wartością propagowaną wspólnie przez świat. Bohaterowie współczesnego kina i – niestety – czasem również pierwszych stron gazet to ludzie pewni siebie, wiedzący wszystko najlepiej i nierzadko gardzący innymi. To nie jest cecha pomagająca stać się dobrym programistą. Oprócz tego, że tak naprawdę nikt nie lubi pyszałków i często są to ludzie po prostu bardzo samotni, traktowanie drugiego człowieka z góry czasem może obrócić się przeciwko nam. Praktykant, którego w jego projekcie ze studiów prowadziłem w poprzedniej pracy, jest w tej chwili project-managerem i zarządza programistami takimi jak ja. Inny praktykant, którego

również wprowadzałem w arkana programowania w ASP.NET, w tej chwili jest w innym projekcie tech-leadem i podejrzewam, że wie na temat programowania webowego znacznie więcej ode mnie.

WARSZTAT

Programowanie w dużej mierze jest rzemiosłem. Dla wielu problemów albo bardziej – klas problemów – istnieją już gotowe i dobre rozwiązania, które warto stosować. Z czasem programiści wypracowali takie zestawy reguł jak na przykład SOLID, które – jeżeli tylko komuś zależy na jakości pisanej przez siebie kodu – nie są już luksusem, tylko absolutną koniecznością.

Jeżeli ktoś uważa, że umiejętności pisania dobrego, czytelnego kodu nie jest wcale tak ważna, bo ważniejsze jest kryterium, czy kod działa czy nie, zapraszam do zajęcia się utrzymywaniem jakiegoś (działającego) programu, który u podstaw był na przykład pisany przez studentów. Drodzy studenci, nie obraźcie się. Ja też byłem studentem i przyznam uczciwie, że obecnie nie miałbym najmniejszej ochoty utrzymywać i rozwijać kodu, który sam wtedy napisałem. Naprawdę sporo czasu i wysiłku wymaga wypracowanie w sobie nawyku pisania *dobrego* kodu – takiego, który działa, ale jednocześnie jest łatwy do przeanalizowania, naprawiania oraz rozwoju. Ostatnio przeczytałem gdzieś w Internecie: „Pisz kod w taki sposób, jak gdyby ktoś, kto będzie go testował lub utrzymywał, był psychopatycznym mordercą, który zna twój adres”. I jeszcze gdzieś indziej: „Każdy głupek potrafi napisać kod, który rozumie komputer. Dobrzy programiści piszą kod, który potrafi zrozumieć człowiek”. Te dwa zdania zwięzle opisują naprawdę ważny aspekt programistycznego rzemiosła.

W jaki sposób wypracować dobry warsztat programistyczny? Na pewno pisząc dużo programów. Bardzo dobrym doświadczeniem jest utrzymywanie i rozwijanie własnego kodu. Widać wtedy bardzo dobrze, które obszary można byłoby napisać lepiej, gdzie można byłoby przeprojektować architekturę, by uczynić ją bardziej elastyczną lub odporną na błędy. Ostatnio potrzebowałem dopisać do mojego kalkulatora obsługę nowego operatora – i naprawdę kilka razy zazgrzytałem zębami. Po pierwsze, niepotrzebnie napisałem go w C++ – okazało się później, że w niektórych aspektach C# jest równie szybki albo – w mojej obecnej implementacji – wręcz szybszy. Po drugie, zupełnie inaczej przygotowałem teraz jego architekturę – poskracał metody i nieco inaczej zaprojektował klasy, co znacznie ułatwiały późniejszy rozwój aplikacji. Aplikacje, które napisałem niedawno – na przykład Z albo LogAnalyzer – rozwija mi się i utrzymuje naprawdę znacznie łatwiej.

Dobrą metodą jest oczywiście zainteresowanie się SOLIDem, wzorcami projektowymi, koncepcją nazewnictwa identyfikatorów i tak dalej. Ja jestem trudnym przypadkiem, nie wystarczyło mi wiedzieć, jak trzeba coś robić, musiałem jeszcze wiedzieć, dlaczego – przez co nauczenie się pisania dobrego kodu zajęło mi znacznie więcej czasu. Ale za to daje mi to dużo silniejszą motywację do trzymania się tych reguł niż „bo tak trzeba robić”.

POPEŁNIAJ BŁĘDY

Odnoszę wrażenie, że niektórzy praktykanci przychodzą na praktyki z przeświadczeniem, że jeśli popełnią błąd, to na pewno ich nie przyjmiemy na staż i potem do pracy. Drodzy praktykanci, gdyby programiści nie popełniali błędów, to niepotrzebni byliby testerzy oprogramowania. Popełnianie błędów jest po prostu wpisane w nasz zawód – możemy co najwyżej zadbać o to, żeby z czasem

popełniać ich coraz mniej i coraz mniej istotnych. Mnie też zdarzało się położyć produkcję. Trzeba wtedy odtworzyć poprzednią jej wersję, poprawić błąd, ponownie publikację na serwer i żyć dalej. Jest takie bardzo mądre powiedzenie: „Tylko ten nie popełnia błędów, kto nic nie robi”.

Poza tym doświadczylem wielokrotnie prawdziwości oklepanego stwierdzenia: „Człowiek uczy się na błędach”. Bo gdy napisałem kawałek kodu, który nie zadziałał, często w procesie naprawiania go musiałem poczytać dokumentację, poszperać w Internecie i znaleźć kilka osób, które miały podobny problem i rozwiązały go na kilka sposobów, obejrzeć tutoriale, może rozrysować sobie coś na kartce i na koniec okazywało się, że dzięki temu błędowi nauczyłem się naprawdę dużo ciekawych rzeczy. Słaby programista to nie ten, który popełnia błędy. To raczej ten, który się na nich nie uczy i popełnia je potem ponownie.

WIEDZA TAJEMNA?

Kiedy mówię ludziom, że jestem programistą, często spotykam się z odpowiedzią: „Ja tam nie rozumiem nic z tych waszych cyferek”. Mało ludzi wie tak naprawdę, na czym polega praca programisty, zawód ten wciąż owiany jest mgiełką tajemnic, wynikającą zapewne właśnie z niezrozumienia. Programowanie nie jest zawodem dla wybranych, zabrać się za niego może każdy, kto umie logicznie myśleć, jest wytrwały w rozwiązywaniu problemów i chętny wciąż uczyć się i rozwijać.

Opisana przeze mnie ścieżka – od Basica, Pascal, Delphi do C# i innych języków – wcale nie jest jedyną słuszną. Można również dobrze zacząć od pisania aplikacji webowych w C# albo od sterowania robotami za pomocą aplikacji w C++ – programowanie to głównie sposób myślenia, a nie znajomość technologii, a tego można uczyć się na każdej platformie i w każdym języku.

NA KONIEC

Nasz zawód jest bardzo wymagający i pociąga za sobą konieczność ciągłego rozwoju – zarówno na płaszczyźnie technologicznej, jak również i personalnej. Nie da się w pewnym momencie stwierdzić: „Teraz już jestem developerem – specjalistą”, bo za chwilę pojawi się nowy język, framework, paradygmat programowania, który będzie różnił się znaczco od już obecnych i znów trzeba będzie usiąść, zakasać rękawy i zabrać się do nauki i dalszego doskonalenia umiejętności. Skończyły się też czasy jednoosobowych projektów, musimy umieć pracować ze sobą, dzielić się wiedzą, pomagać w trudnych sytuacjach, uczyć i wspierać. Ale jasna strona jest taka, że mamy przed sobą możliwości, jakich ludzie nie mieli przez grube tysiąclecia. Pod opuszczami naszych palców drzemie ogromna potęga, za pomocą której można uczynić mnóstwo dobrego – i jest to coś, co nieustannie motywuje mnie do dalszego rozwoju.

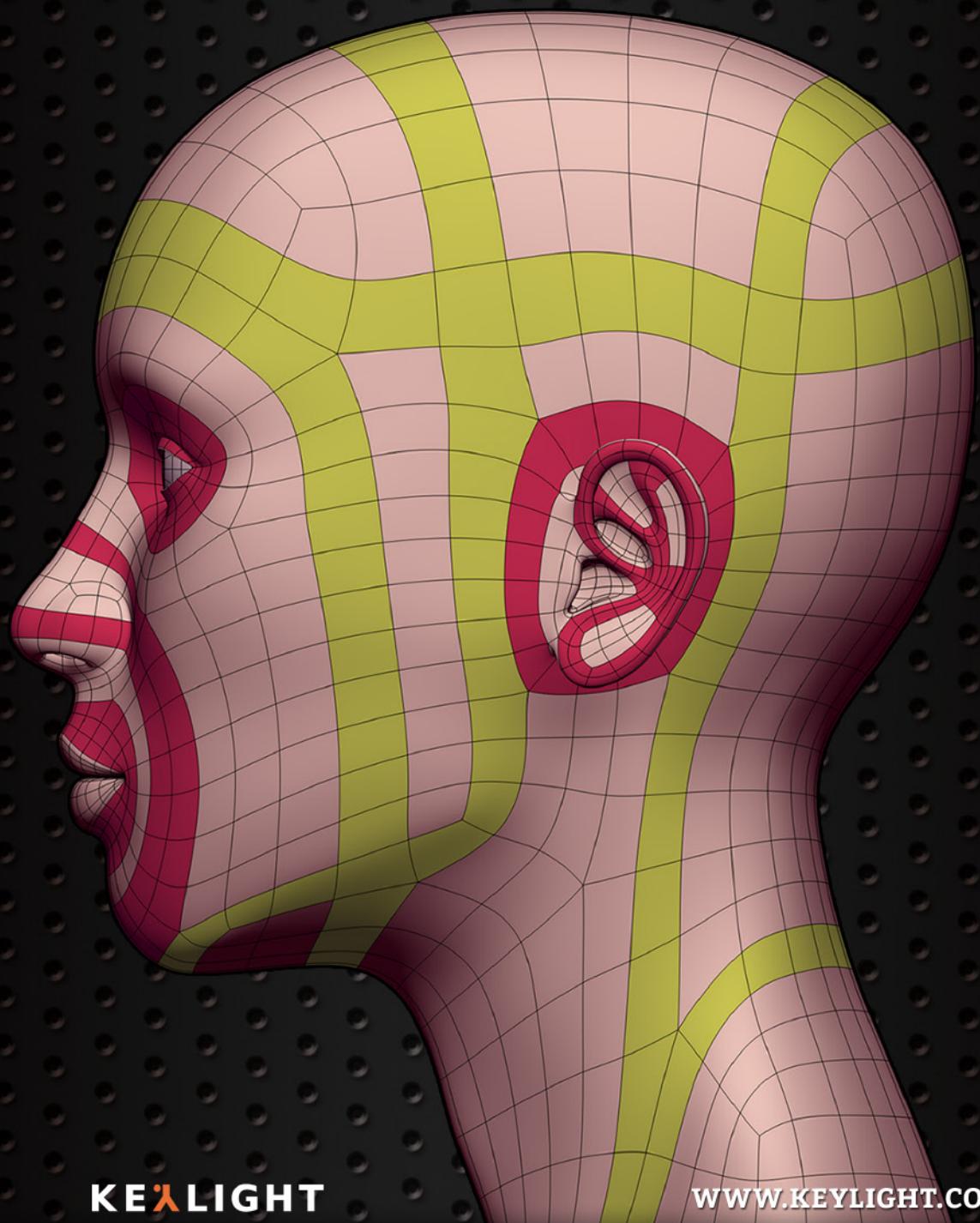
Artykuł dedykuję pewnej sympatycznej parze narzeczonych, zgłębiającej wspólnie tajniki programowania na jednej z wrocławskich uczelni.



WOJCIECH SURA

wojciechsura@gmail.com

Programuje od przeszło dziesięciu lat w Delphi, C++ i C#, prowadząc również prywatne projekty. Obecnie pracuje w polskiej firmie PGS Software S.A., zajmującej się tworzeniem oprogramowania i aplikacji mobilnych dla klientów z całego świata.



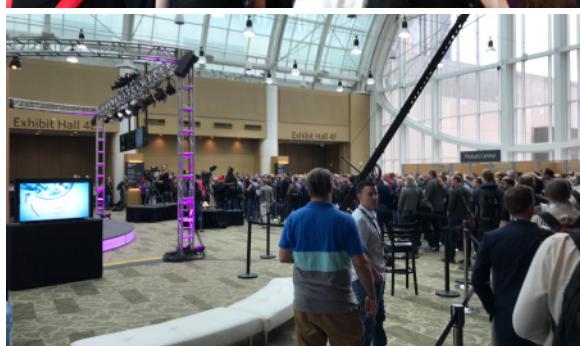
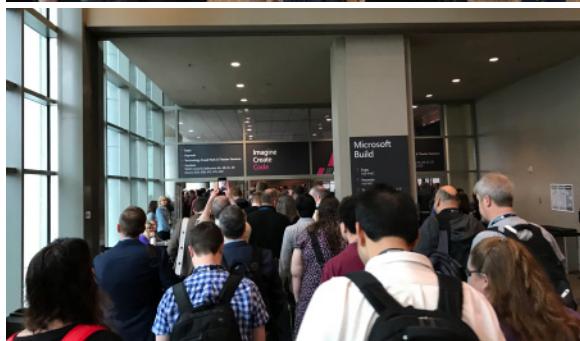
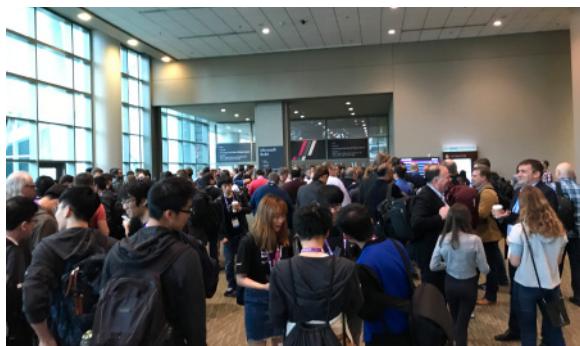
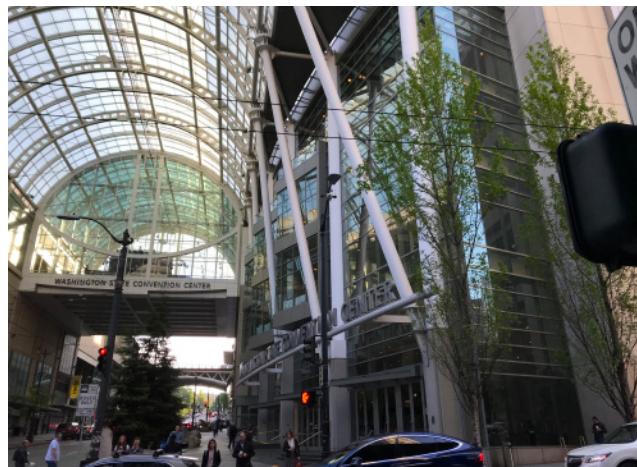
KEYLIGHT

WWW.KEYLIGHT.COM.PL

Microsoft Build 2018. Relacja z konferencji

Microsoft Build to coroczna konferencja dla developerów. W tym roku odbyła się już jej ósma edycja, a omawiane zagadnienia koncentrowały się głównie na nowościach w technologiach chmurowych, sztucznej inteligencji, Internet of Things oraz narzędziach wspierających produktywność programistów. W tym artykule opiszę wybrane nowości prezentowane na Microsoft Build 2018.

Konferencja Microsoft Build 2018 odbyła się w dniach 7-9 maja w Seattle w stanie Washington. Podobnie jak w poprzednim roku, miała miejsce w centrum konferencyjnym Washington State Convention Center (górnego zdjęcia na Rysunku 1) i zgromadziła kilka tysięcy programistów z całego świata. Choć tuż po konferencji wszystkie referaty są dostępne on-line, to osobiste w niej uczestnictwo jest ekscytującym doświadczeniem, umożliwia bowiem bezpośredni kontakt z pracownikami Microsoftu, którzy rozwijają daną technologię, a ponadto chętnie i wyjątkowo kompetentnie odpowiadają na wszelkie pytania.

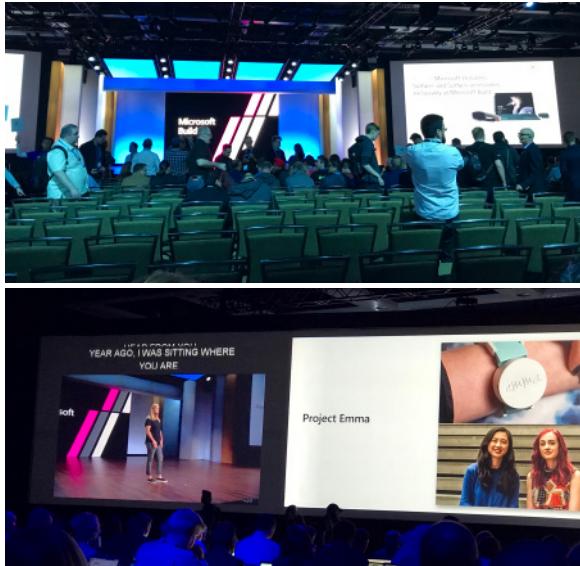


Rysunek 1. Miejsce konferencji, identyfikator oraz koszulka konferencyjna

Udział w konferencji rozpoczęłem od rejestracji i odebrania identyfikatora oraz koszulki konferencyjnej (dolina część Rysunku 1), po czym udało się w kierunku głównej sali, w której odbywały się wykłady plenarne. Zdjęcia z Rysunku 2 ilustrują liczbę zgromadzonych osób oczekujących na wykład. Tuż przy głównej sali zlokalizowane były również stanowiska Channel 9 Live oraz podcastów, m.in. Merge Conflict prowadzonego przez Jamesa Montemagno.

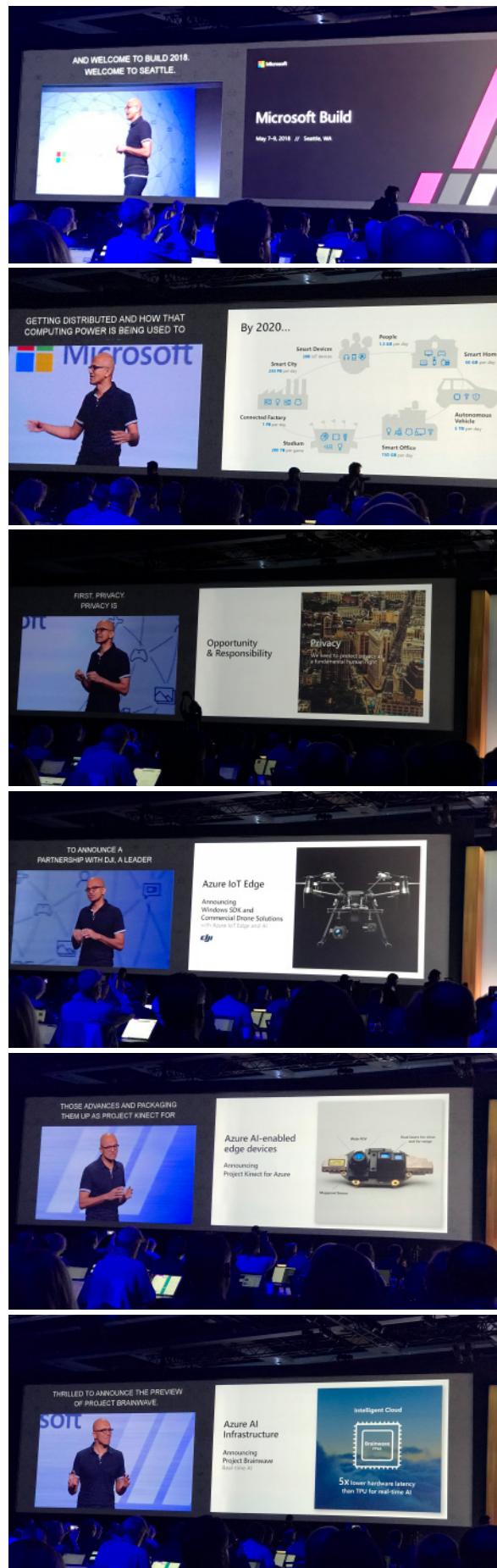
Rysunek 2. Oczekiwanie na wykład plenarny, kolejka do sali wykładowej i stanowisko Channel 9 Live (dolny wiersz)

Pierwszy wykład plenarny rozpoczął się od krótkiego podsumowania poprzedniej konferencji, a w szczególności tych aspektów, które w zeszłym roku wywołyły największe owacje. Najciekawszym zagadnieniem był projekt Emma (Rysunek 3). Ma on na celu utworzenie urządzenia ubieralnego, które pomaga ludziom z chorobą Parkinsona. Urządzenie ubieralne jest wyposażone w zestaw programowalnych silników, które wywołują drgania, mające na celu kompensowanie mimowolnych drgań dloni. Dzięki temu chorzy na Parkinsona mogą wykonywać czynności, które wymagają stabilnych ruchów dloni, takich jak np. pisanie czy rysowanie.



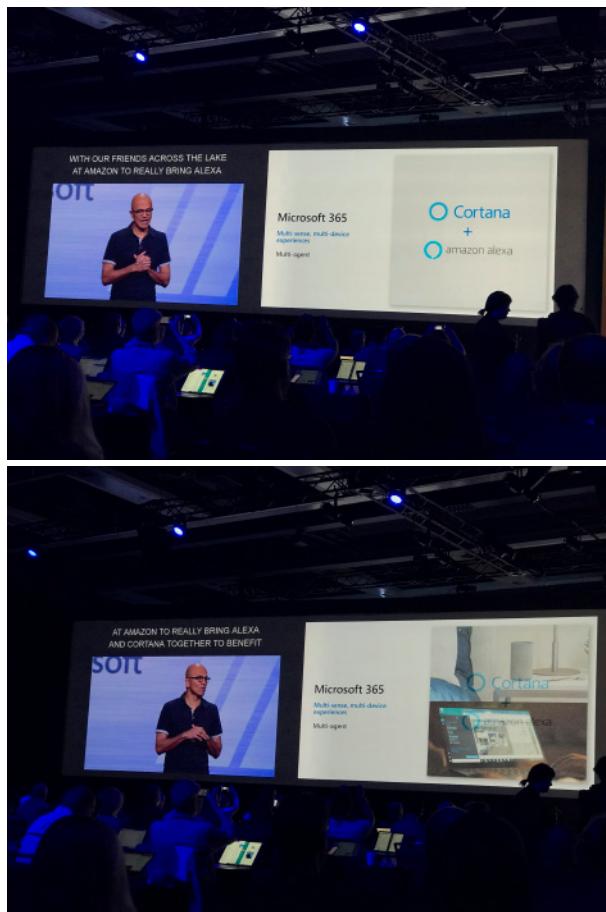
Rysunek 3. Oczekiwanie na wykład plenarny i podsumowanie projektu Emma

Po krótkim wprowadzeniu swoje wystąpienie rozpoczęł CEO Microsoftu, Satya Nadella (Rysunek 4). Podczas referatu omawiał on możliwości stojące przed developerami w zakresie tworzenia rozwiązań chmurowych i IoT wspieranych przez sztuczną inteligencję oraz nowe urządzenia i infrastrukturę sprzętową. Wszystkie te wysiłki mają na celu wspieranie rozwiązań Intelligent Cloud Intelligent Edge (ICIE), w których analizy AI wykonywane są nie tylko po stronie chmury, ale również po stronie intelligentnego urządzenia krańcowego. Sztuczna inteligencja jest tu potrzebna do analizy coraz to większych zbiorów danych, generowanych przez sensory wbudowane w autonomiczne samochody, intelligentne fabryki, domy czy urządzenia mobilne, które nosimy przy sobie przez cały dzień. Podejście ICIE ma bowiem na celu ułatwienie wdrażania intelligentnych rozwiązań IoT w sytuacjach, gdy nie mamy stałego połączenia z Internetem lub gdy wymagane są krótkie czasy odpowiedzi. W tym miejscu warto przytoczyć chociażby projekt Brainwave. Jego celem jest wykorzystanie układów FPGA do równoległego prowadzenia obliczeń na potrzeby sztucznej inteligencji. Ciekawym rozwiązaniem jest również wykorzystanie AI do analizy obrazów w dronach, produkowanych przez firmę DJI. Zgodnie z zapowiedziami mają być one kontrolowane przez Windows 10 i łatwo programowalne przez odpowiednie SDK. Bardzo duże zainteresowanie wywołało również ogłoszenie projektu Kinect dla Azure. Dostarcza on gotowe urządzenia w postaci OEM, które pozwala analizować głębie sceny w oparciu o pomiar czasu przelotu światła (ang. *time-of-flight sensor*). Ta funkcja w połączeniu z kamerą RGB i akcelerometrem oraz technikami sztucznej inteligencji ma znaleźć swoje zastosowania w automatyzacji wielu gałęzi gospodarki, np. na potrzeby intelligentnych fabryk, dronów czy robotów.



Rysunek 4. Zdjęcia z wykładu plenarnego CEO Microsoftu

Podczas swojego wystąpienia Satya Nadella podkreślał również, że okazje stojące przed programistami wymagają odpowiedzialności. Ta odpowiedzialność dotyczy głównie ochrony danych użytkowników oraz unikania wykluczenia cyfrowego, aby technologia była dostępna dla wszystkich, bez względu na wiek, płeć czy pochodzenie etniczne.



Rysunek 5. Prezentacja współpracy Alexy i Cortany

Wizja Microsoftu zaprezentowana przez Satya Nadella była następnie uzupełniona przez wystąpienie partnerów z firmy Amazon. Ta prezentacja pokazywała połączenie interfejsów Alexy i Cortany, aby ze sobą współpracowały (Rysunek 5). Ma to na celu tworzenie rozwiązań skoncentrowanych na użytkowniku i niezależnych od konkretnych urządzeń czy ich dostawców (projekt Rome). Opowieść ta była następnie uzupełniona o Microsoft 365, który stanoi połączenie Office 365, Windows 10 oraz technik bezpieczeństwa (Enterprise Mobility and Security). Prezentacja była poparta przykładami wykorzystania Microsoft 365 na potrzeby współpracy w nowoczesnych środowiskach pracy, zgodnie z główną wizją Microsoftu: „to empower every person and every organization on the planet to achieve more”.

W dalszej części przyszła kolej na omówienie nowości w Microsoft Azure (Rysunek 6). Tę część prowadził już Scott Guthrie. Najpierw omówił główne motywacje dalszego rozwoju usług Azure. Następnie pokazał listę największych klientów Azure, po czym omówił nowości z zakresu:

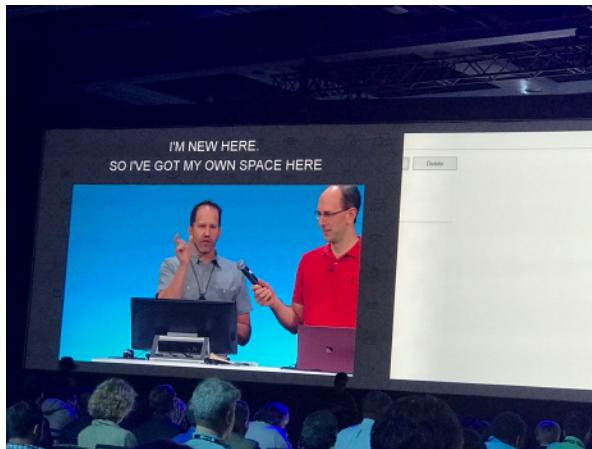
- » Narzędzi dla developerów,
- » Kontenerów i architektury bezserwerowej,
- » Internet of Things,
- » Analizy danych oraz sztucznej inteligencji.



Rysunek 6. Zdjęcia z wystąpienia Scotta Guthrie

Z powyższej listy najciekawszą wyglądała część dotycząca narzędzi dla developerów. Tutaj duże wrażenie wywarło na mnie Visual Studio Live Share. Jest to narzędzie, które umożliwia zespołowe wprowadzanie, debugowanie i śledzenie zmian w kodzie źródłowym bez względu na platformę (działa w Visual Studio i Visual Studio Code) i język programowania. Podczas konferencji zaprezentowano praktyczne użycie Visual Studio Live Share w typowej sytuacji. Mamy wykonać release za dwie godziny, a aplikacja jest w „rozspę”. Developer, korzystający z Visual Studio Code, poprosił o pomoc

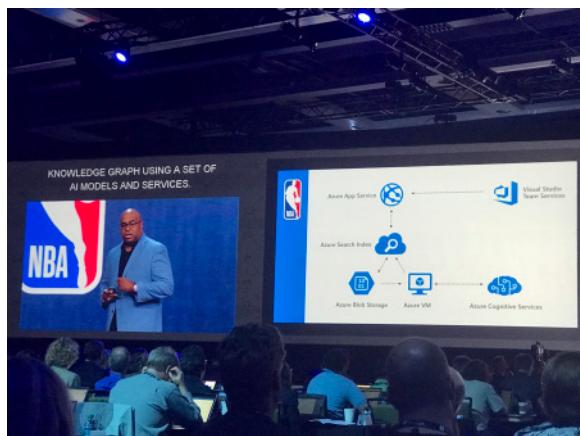
programistkę pracującą w Visual Studio. Ta w czasie rzeczywistym wykonała korekty w kodzie źródłowym oraz zdebugowała konkretny błąd, czyniąc release możliwym do wykonania.



Rysunek 7. Scott Guthrie pełni funkcję „stojaka” pod mikrofon dla Scotta Hanselmana

W trakcie prezentacji o Microsoft Azure, gdy na scenie znajdował się Scott Hanselman, nastąpiła awaria jego mikroportu. W zamian otrzymał zwykły mikrofon, ale jego trzymanie uniemożliwiało mu programowanie. Całą sytuację obrócił w żart i przytrzymywał mikrofon swoją brodą. Ten problem rozwiązał Scott Guthrie, który wrócił na scenę, aby przytrzymywać mikrofon dla Scotta Hanselmana (Rysunek 7). Obaj podsumowali tę sytuację w taki sposób, iż zawsze chcieli razem wystąpić na scenie podczas konferencji Microsoft Build.

W dalszej części kilku klientów zaprezentowało scenariusze użycia usług Microsoft Azure w ich codziennej pracy. Swoje wystąpienia mieli przedstawiciele NBA i Starbucks. W szczególności pierwszy z nich omawiał, w jaki sposób NBA wykorzystuje metody sztucznej inteligencji (Azure Cognitive Services) do automatycznego tagowania zdjęć i filmów. Informacje te są następnie wykorzystywane do sortowania i kategoryzacji multimedialnych dostępnych w witrynie NBA.



Rysunek 8. Przedstawiciele NBA i Starbucks podczas omawiania ich scenariuszy użycia technologii Microsoftu

Podczas konferencji, oprócz referatów, można było również odwiedzić wiele stoisk reprezentowanych przez partnerów lub samych pracowników Microsoftu (Rysunek 9). Było to doskonała okazja do zadawania mniej formalnych pytań i uzyskania informacji z „pierwszej ręki”.



Rysunek 9. Wystawa Hub oraz panel Microsoft Build



DAWID BORYCKI

Naukowiec, programista, autor wielu książek, amerykańskich patentów, artykułów i videotutoriali o programowaniu w różnych technologiach. Pierwszy polski autor w Microsoft Press i MSDN Magazine. Z wykształcenia doktor fizyki teoretycznej. Obecnie pracuje w Polskiej Akademii Nauk, gdzie zajmuje się rozwojem nowoczesnych, nieinwazyjnych urządzeń do analizowania mikroprzepływów krwi w mózgu i obrazowania przez ośrodkie nieprzezroczyste optycznie.

Letnie nerda przemyślenia

Lato. To pora roku jednoznacznie kojarząca się z upałem, urlopem i ogólnie wypoczynkiem. Choć ostatnimi latami aura nam się zmienia ze znanego (i przez wielu lubianego) klimatu umiarkowanego na coś zbliżonego do śródziemnomorskiego, to nastawienie pozostaje bez zmian. Ogólnie to czas, gdy prawie nic się nie chce. Projekty wloką się niemiłosiernie (tak jak dni), dominuje wrażenie drastycznego spadku wydajności. Wszystko jest „rozlazłe” i najczęściej bardziej zniechęcające niż intrygujące i dodające skrzydeł.

Zresztą prosty przykład. Pewien błąd, który nie dawał spokoju przez ostatnie pół roku. W końcu przyszedł czas na niego, wiecie, ten stan desperacji „albo go rozgryzę i poprawię, albo projekt do kosza” (tak, jasne, do kosza, yhy, ale chyba każdy programista dochodzi kiedyś do takiego momentu, że ma ochotę rzucić ten projekt, nad którym siedzi, w diabły i coraz głośniej i częściej gdzieś z tytułu głowy dzwoni dzwon z napisem „zaczni od nowa”). Trzy dni zabawy z debuggerem i voila – błąd naprawiony, wszystko działa ślicznie, jednak brakuje tego „czegoś”, co prawdopodobnie byłoby w jakąkolwiek inną porę roku. Po prostu ten upał przyćmiewa wszystko, kto to widział, by u nas na dworze w słońcu było 46 stopni?! Toč to jak na pustyni, do tego wilgoć ogromna i jest przytaczające wrażenie ciągłego płynania w zupie – hej, kucharzu zmniejsz troszkę gaz, bo się zaraz przypali wszystko.

Ale ten felieton nie ma być o pogodzie i zniechęceniu. On ma być ogólnie o przemyśleniach dotyczących kilku tematów około programistycznych i historyjce pokazująccej, że jak się chce – można wiele. Zaczynę od przemyśleń. A dokładnie od sensowności podążania za trendami i nowościami. Tak, wiem, jest w tym numerze troszkę o tym, ale nic nie stoi na przeszkodzie, bym dorzucił coś od siebie jako uzupełnienie lub kontrę tamtych tez. Michał Lewandowski przedstawia hipsterskie metody wyboru technologii, Wojtek Sura opisuje między innymi swoje zaparcie z jednym językiem/technologią ciągnące się na przestrzeni lat (nie zamierzam tutaj robić streszczenia ich tekstów – nie o to chodzi). Ja w kilku słowach opowiem o nieudanych próbach wprowadzenia sporą zmian w jednym z moich projektów. Czemu nieudanych? Bo w pewnym momencie eksperymentowania przyszło otrzeźwienie „to nie ma najmniejszego sensu”. Ale po kolei. Jeden z moich projektów składa się z kilku elementów składowych: część serwerowa, część frontendowa i część działająca na dedykowanym urządzeniu (ot, taka specyfika rozwiązania, przy czym część serwerowa może być rozzielona na kilka maszyn – w ogóle może kiedyś opisać go jako przykład, jak z prostego tematu zrobić bardzo rozbudowane „coś”? Zobaczmy). Budowa pierwszej i ostatniej, oraz użyte w nich technologie, nie mają tutaj najmniejszego znaczenia – ważne jest

to, że wszystkie trzy się komunikują między sobą w ustandyzowany sposób. Środkowy element jest zwykłym interfejsem webowym opartym na standardowych technologiach znanych zdecydowanej większości z nas, czyli PHP w tle, front napędzany Ulkit'em w wersji 2.x – możecie zapytać, czemu w poprzedniej wersji? Niedługo będzie to zrozumiałe w pełni. Dodatkowo na froncie używam DataTables ze zmodyfikowanym wyglądem, tak by pasował pod Ulkit'a, oraz troszkę dodatkowych bibliotek dopasowanych wyglądem do reszty, sam Ulkit nie ustrzegł się też sporej porcji modyfikacji – wszystko po to, aby interfejs był przejrzysty, spójny i po prostu się podobał. Dodatkowo interfejs jest bardzo zmodularyzowany, a za składanie wszystkiego w całość odpowiada Smarty. Ot, taki miszmasz, który jednak ma swój sens, widoczny dopiero po zapoznaniu się z dokładną budową całości. Spotkamy tutaj specyficzne klocki do budowy specyficznego interfejsu w dużej mierze złożonego podobnie do Matrioszek – klocek w klocku umieszczony w innym klocku. Mówiąc krótko, podstawowy interfejs składa się ze 120 powiązanych ze sobą plików szablonów Smarty. Tak, 120... każdy element składowy interfejsu tworzący sensowną całość jest traktowany jako „komponent” i siedzi w osobnym pliku. Ma to swoje wady i zalety. Niewątpliwie zaletą jest to, że zmiana układu/funkcjonalności jednego z elementów widoczna jest w całym systemie, minusem natomiast jest problematyczność składania całości w jedno. A największa wada ujawniła się przy zaanonsowaniu wersji 3 Ulkit i udostępnieniu pierwszej bety. Domyślacie się? Wersja 3 jest niezgodna z 2 – procedury migracji są oczywiście opisane, ale pociąchanie całości w ten, a nie inny sposób odbija się gigantyczną czawką przy jakiekolwiek próbie jej rozpoczęcia. Mniej więcej przy 30 pliku pojawia się pierwszy kryzys, drugi w okolicach 40 – wtedy człowiek zaczyna zastanawiać się, po cho^H^Ho to tak ciachał, przy 50 (nadal praktycznie brak działającego prawidłowo czegokolwiek) miałem dość. Pojawia się myśl, że chyba trzeba to przepisać od nowa, mniej pociąchać, bardziej sensownie. Ale to nie takie proste – pluginy bazują na częściach z głównego szablonu, odpowiedzi – przesypane przez tak bardzo uwielbiany przez niektórych programistów AJAX – też mają kawałki kodu generowane

za pomocą tych „skrawków” szablonów. Ogólnie całość bez tych drobnych fragmencików templatek wymagałaby całkowitego refaktoringu, a czasu na to brak.

Co gorsza, próby przerobienia całości na coś innego, typu Angular, Vue.js, też spełzyły na niczym, czasowy koszt przeanalizowania całości, naniesienia wymaganych zmian, dostosowania wszystkiego pod swoją wizję wyglądu, przygotowanie tak, by całość działała bezproblemowo dalej, wraz z pluginami mającymi swoje szablony, jest dużo wyższy niż koszt napisania całego projektu od nowa – ale znacie to uczucie? Poświęcony gigantyczny zasób czasu, działająca wersja, fajnie wszystko pociachane w wygodny (dla mnie) sposób. Prostota budowy UI dla pluginów, szybkość ograniczenia budowy ekranów składowych całego interfejsu i ich sensowna i wygodna spójność bronią użycie archaicznej już biblioteki i niektórych rozwiązań wystarczająco skutecznie, by odciągnąć od szalonego pomysłu migracji na wystarczająco długi czas. Można nazwać to rozsądkiem, można nazwać też lenistwem (przecież miałeś człowieku ponad 50 plików – jeszcze kilka i zaczynasz lecieć z ilością tylko w dół! – w końcu przepiszesz/zmigrujesz wszystkie i będzie spokój!) i na kilka innych sposobów. Jednak sprawdza się to praktycznie za każdym razem – kiedyś może bym się rzucił w wir przepisywania, zmigrował w końcu na coś nowszego dla swojego dobrego samopoczucia. Teraz wolę odejść od komputera, niż gnąć przy nim bez widocznych szybko efektów.

Bo czym uzasadnić pogoń za numerkami? Nowością? Lepszością? No, z tym drugim nie zawsze można się zgodzić. Często zmiana na nową wersję generuje mnóstwo nowych i dodatkowych błędów, i albo czekamy, jak autorzy uaktualnionej biblioteki je poprawią (z lenistwa, braku czasu, by się tym samemu zająć), albo zaczynamy tracić czas na poszukiwanie źródła problemu, a nie dalsze rozwijanie projektu. Co jest lepsze? Czy klient poczeka, aż znajdziesz przyczynę, czemu wykłada się tworzone dla niego narzędzie, bo tak po prawdzie to zachciało się ci zmienić numerkę wersji głównej biblioteki, a jeszcze tydzień temu 99% całości działało poprawnie? Czy opóźnienie ze swoim prywatnym projektem przesuwające termin wdrożenia w nieokreślona przyszłość, powodujące kolejne oddalenie płatności jest czymś, na co możesz sobie pozwolić? Po co w sumie zmieniać coś, co tyle czasu się sprawdzało? Przecież to wywoływanie wilka z lasu – i to tego złego, warczącego i kąsającego rękę, która go karmi, w tym wypadku programistę, który go adoptuje do swoich potrzeb. Zgodzić się z wymienionymi wyżej autorami muszę w kwestii tego zbędnego podążania za modą, nowościami i całą otoczką towarzyszącą tym zagadnieniom. Nie macie, drodzy programiści, wrażenia, że mimo pędu technologii naprzód, mimo tego coraz prostszego „niby” tworzenia oprogramowania, tak naprawdę drepczemy w miejscu, a wręcz często łapiemy swój własny ogon i męczymy go i siebie coraz bardziej?

Ostatnie podejście do tematu zmiany Ulkit'a na coś uwspółczesnionego i dużo „wygodniejszego” zakończyło się znacznie szybciej

niż wcześniejsze. Nie było to po prostu migrowanie kodu na nowe „cudo”. To było zrobienie od nowa, bez dzielenia na poszczególne klocuszki, standardowego widoku, jaki jest po zalogowaniu się. Już w połowie prac okazało się, że to, co zapowiadało się jako super hiper i fajnie i co w teorii uprości zabawę, bo ma dwa widgety, które bardzo się przydadzą i kilka potencjalnych do wykorzystania, sprawia dość spory problem, by zachować choć zbliżony funkcjonowaniem widok. Czemu zbliżony? By można byłoby przynajmniej zostawić pomoc w spokoju. Nie tracić czasu na ponowne zrzuty, poprawki opisów w tych miejscach, które uległy zmianie itd. itp. Dziś, z perspektywy czasu, śmieję się sam z siebie. Ot, dałem się w jakimś stopniu nabrać na papkę marketingową reklamującą „super produkt”. Ale na szczęście przyszło wspomniane otrzeźwienie i projekt rozwija się dalej swoim tempem, bez utknięcia w marljwym punkcie tylko po to, by mieć troszkę bardziej „nowoczesne” biblioteki na liście użytych technologii. To nie tedy droga – czasem lepiej opłaca się pozostać przy czymś starszym, ale doskonale znanym, niż gonić uciekającego króliczka w zupełnie nieznany terenie – mniej boli ;)

Tyle przemyśleń. Może to wpływ pory roku, co rozleniwa strasznie, może to wpływ doświadczenia, albo wieku – ale bardzo cieszy mnie to otrzeźwienie – to taki sygnał, że nabralo się w jakimś stopniu odporności na marketingową papkę wychwalającą dane rozwiązanie jako ultra hiper wygodne, wydajne i łatwe do wykorzystania. Może i takie jest, ale w ściśle określonych przypadkach, które tak po prawdzie leżą u źródeł powstania danego rozwiązania. Nie dajmy się porwać niepotrzebnie wizji „lepszego jutra”, często obudzimy się w realnym świecie „przeklętego piekła”...

Na początku tego tekstu wspomniałem o pewnej historyjce mającej potwierdzić, że jak się chce, to można wiele. Jest to historia oparta w stu procentach na prawdzie o tym, jak zwykły szeregowy pracownik jednej firmy, mający pewne subtelne predyspozycje, w końcu otworzył szeroko oczy, wziął się za naukę i... no właśnie, i co?

Rzecz miała miejsce jakiś czas temu. Ot, zwykłe spotkanie dwóch kolei w jednej pracy, jeden pracuje jako admin, hobby-stycznie i częściowo zawodowo dodatkowo jako koder. Czego nie pisał w życiu, to sam nie wie. Ale w tamtym czasie dużo styczności miał z frontendem. Wiecie, jak to jest, faceci jakoś szybko w swoim gronie nawiązują znajomości z osobnikami o podobnym „wypaczeniu” hobby-stycznym. Nie wiem czemu – w większym gronie raźniej? Tu było podobnie, sporo czasu spędzał na różnych dyskusjach *stricte* technicznych. I ten admin po pewnym już czasie przyjaźni wypalił drugiemu, że się marnuje na stanowisku, jakie obecnie zajmuje, że widać, co go naprawdę interesuje, że widać pewne cechy, które świadczą o tym, że jakby przysiadł, to by nauczył się wszystkiego dość szybko. Ot tak, po prostu impuls w jakimś odpowiednim momencie. I impuls ten zasiał ziarenko, ziarenko, które zaczęło kiełkować i rosnąć. Ich dyskusje zdominował webdev,

używane narzędzia, co jest potrzebne, co przydatne, co można na początku odpuścić, co należy bezwzględnie umieć. Typowa gadka, jaką nieraz odbywa się z kimś z branży. W końcu chłopak ten, nazwijmy go „J”, wciągnął w całą sprawę swojego drugiego bliskiego znajomka, zawodowego kodera z wieloletnim doświadczeniem. To był prawdopodobnie najbardziej przełomowy moment. Bo nim zaczęło się u J przyspieszone odbywanie edukacji branżowej. Tutoriale, kursy video, proste zadania otrzymane do wykonania, ocena kodu, obrona własnego kodu i tak dalej, i tak dalej. Po około pół roku chłopak potrafił zrobić samodzielnie, bez niczyjej pomocy, prawie bezbłędnie, określone UI, którego mock-up dostał dzień wcześniej. Potrafił nawet sensownie uzasadnić pewną zmianę, jaką wprowadził w jednym miejscu do projektu.

To był etap jego edukacji. Po nim, jak to najczęściej bywa, nadszedł etap praktyki zawodowej. Złożył CV do kilku firm, w jednej usłyszał, że na chwilę obecną mają pełen zespół, ale bardzo możliwe, że po zamknięciu aktualnego projektu będą musieli zatrudnić dodatkowe osoby na etaty i jeśli tak się stanie – odezwą się do niego. Wyobraźcie sobie, że faktycznie zadzwonili po około trzech miesiącach. Zaprosili na rozmowę, podobno przebiegała w bardzo luźnej atmosferze i nie była czysto techniczna (aplikowała na junior developera). Po rozmowie okazało się, że rekrutację przeprowadził sam właściciel firmy, będący też programistą, i to czynnym. Jakiś czas później telefon – J, zapraszamy od tego i tego do pracy, dasz radę zamknąć wszystkie zaległe sprawy? I tak się zaczęła jego przygoda z zawodowym programowaniem. To było już jakiś czas temu. Dziś nadal pracuje w tej firmie, prawdopodobnie wkrótce może awansować na seniorkę, gdyż coraz częściej pełni rolę „lidero-koordynatora” projektu. Ba, doszło do tak kuriozalnej sytuacji, że jak poszedł na urlop, to reszta zespołu nie wiedziała, co ma robić, i przez czas jego nieobecności nie pchnięto zupełnie nic do przodu w kwestii projektu.

Można zadać sobie pytanie, czy faktycznie ktoś, kto nie miał wcześniej styczności z tym, bo nie został ukierunkowany przez nikogo, bo nikt nie zauważał pewnych dominujących cech u niego bardziej sprawdzających się w zawodach wymagających „myślenia algorytmicznego”, może zmienić tak drastycznie branżę i być szczęśliwym? Może. J opowiada o swoich zadaniach z takim entuzjazmem, że to się udziela wszystkim, którzy słuchają go w danym momencie – potrafi w osobach piszących od lat obudzić to magiczne „jeszcze linijkę, jeszcze jedną metodę, jeszcze malutką klaskę”, a tu przecież za oknem już księżyć w ekran zagląda. Chłopak czuje w końcu, że robi to, co powinien robić od dawna. Sprawia mu to ogromną przyjemność i daje poczucie bycia potrzebnym, jakże ważne w każdej pracy. Mimo to nie uderzyła mu sodówka do głowy, nadal jest taki, jaki był, i mimo już pewnego swojego

zawodowego stażu nie stał się przemądrzałym kolesem, co uważa się za guru w każdym możliwym programistycznym temacie, jak to się często dzieje z osobnikami wyprodukowanymi przez studia, określmy to, „branżowe”.

Teza została rzucona, przykład ją potwierdzający opowiedziany, choć to nie jest jedyny przypadek, po którym widać, że wykonuje nie do końca swój zawód, a wybitnie nadaje się do programowania/security/analytiki. Pamiętam, że za moich czasów, w liceum, miała miejsce rozmowa z pedagogiem, który miał jeden ścisłe określony cel: pomóc w wyborze przyszłej szkoły policealnej, czy to studium, czy uczelnia wyższa, czy coś bliżej, czy dalej miejsca zamieszkania. Rozmowa ta w jakimś stopniu umożliwiała uniknięcie „porażki” życiowej, jaką jest wykonywanie pracy w zawodzie, który „nie jest nasz”. Nie wiem, co się pozmieniało w naszym szkolnictwie przez te lata, ale obecnie czegoś takiego się nie praktykuje (przynajmniej kilka osób potwierdziło ten stan „rzeczy”). Stąd biorą się potem osoby, co intuicyjnie mogą programować, rozwiązywać nieświadomie w sposób algorytmiczny postawione przed nimi problemy, a czynności przez nich wykonywane są tak przemyślano i zoptymalizowane, że aż miło popatrzeć, a na co dzień wykonują bezmyślną pracę. Pracę przysłowiowego świstaka. Wiecie, tę prawie automatyczną, niewymagającą myślenia, niewymagającą dużego skupienia, ot, zwykłe ręka tu, ręka tam, długopis tu, długopis tam, podpis i pa pa. Ile takich osób może być? Dziesiątki? Setki? Tysiace? Talenty, które zostały zmarnowane, źle ukierunkowane, nierożpoznane. Osoby, które mogłyby być jednymi z lepszych w branży, obdarzone intuicją, nonstop analizującą powierzone zadania, opracowującą swoją ścieżkę postępowania, często zupełnie nieświadomie. Najczęściej zniechęcone do czegokolwiek, z niską samooceną, bo wszystko robią pod przymusem krzyczącym wewnątrz nich „muszę!” i nie wierząc w to, że są w stanie o 180 stopni zmienić swoje życie, często bez ponoszenia większego ryzyka i kapitału...

I wiecie, co jest najgorsze? Nawet jak im udowodnisz, że masz rację, że jest tak jak mówisz, a nie inaczej, i tak uwierzą w to, czego zostały nauczone przez swoje dotychczasowe życie, bo to, co przed nimi się otwiera, wygląda zbyt wspaniale i idealnie. Wręcz jak jakiś obrazek ze snów i marzeń. Czytelniku, jeśli czujesz, że robisz coś, co Cię nie cieszy, robisz to pod przymusem, może nadszedł czas na zmiany? Po co tkwić w czymś, co jest kłamstwem i obłudą? Po co tracić czas? Życie jest tylko jedno, przeżywajmy je najlepiej jak potrafimy, bo pewnego dnia obudzimy się, gdy już będzie za późno...

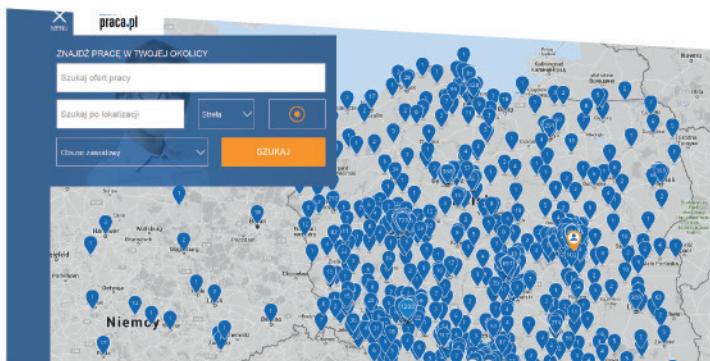
Mariusz „maryush” Witkowski

ps. J, pozdrowienia dla całej rodzinki. A – uwierz w końcu w siebie!



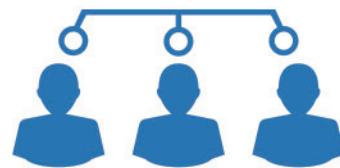
Chcesz dobrze zarobić?

Na Praca.pl codziennie znajdziesz ponad 3 000 ofert pracy
z obszaru IT i nowe technologie



Znajdź pracę w Twojej okolicy

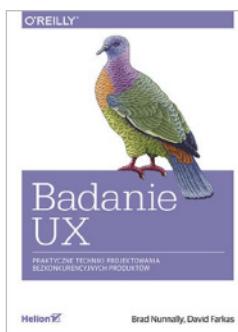
Lokalna.praca.pl



Poleć znajomego do pracy
i zgarnij 1 000 zł

Praca.pl/rekomendacje.html

Badanie UX. Praktyczne techniki projektowania bezkonkurencyjnych produktów



Wydana w 2017 roku, w Polsce pojawiła się na początku 2018. Ten drobny poślizg nie ujmie nic z aktualności książki, która skupia się na zarysowaniu praktycznych technik badań UX.

Zaczynając od rysu historycznego, autorzy szybko przechodzą do konkretów, czyli: jakie są metody badawcze ilościowe i jakościowe. W przypadku tych pierwszych koncentrują się na:

- » analizie danych systemowych (trudne określenie jakże standardowego analizowania statystyk strony),

- » ankietach masowych i ich opracowywaniu, a także analizowaniu efektów,
- » opisie testu drzewa (jest to ewaluacja nawigacji i jej zrozumienia przez użytkownika),
- » badaniach eyetrackingowych (czyli śledzenia wzroku badanej osoby, takie testy przeprowadzone na dużych próbach umożliwiają podejście statystyczne do wyników),
- » testach A/B (badanie dwóch wersji danego elementu/strony internetowej i statystyczne analizowanie, która wersja była efektywna).

Z kolei metody badawcze jakościowe, które są opisane w publikacji, to:

- » analiza krajobrazu rynkowego (czyli: zapoznanie się z daną branżą, sprawdzenie, jak działa konkurencja, jakie ma produkty, gdzie są słabe strony, a gdzie mocne),
- » przeglądy heurystyczne, badania kontekstowe (inny poziom badań nakierowanych na wgląd w zastaną sytuację: jest to ewaluacja istniejącego rozwiązania czy produktu w firmie, szukanie miejsc do ulepszenia w istniejących już sytuacjach),
- » projektowanie partycypacyjne (angażowanie wszystkich działów firmy w proces projektowania nowego produktu, np. nie tylko inżynierowie myślą nad nową wersją strony internetowej, ale też przedstawiciele działu sprzedaży czy obsługi klienta – ich głosy pomagają stworzyć lepszą wersję finalną),
- » testowanie i weryfikacja produktu (podczas sesji z realnymi użytkownikami końcowymi, tu wykorzystuje się np. prototypy, bądź jeszcze nieukończone wersje produktu, gdzie jest jeszcze szansa na wprowadzanie zmian).

Brad Nunnally i David Farkas skupiają się nie tylko na samych technikach, co na tym, które metody wybrać i dlaczego. Dużo miejsca w publikacji poświęcają rozważaniom, jak dopasować technikę badań UX do potrzeb: interesariuszy, zespołu oraz użytkownika końcowego.

W dalszej części książki czytelnik znajdzie także opis przygotowania, przeprowadzenia i podsumowania badań UX. Autorzy przygotowali praktyczne porady, jak zorganizować sesje badawcze, jakie są potrzebne dokumenty, jakie są kwestie honorariów i rekrutacji oraz cała logistyka z tym związana. Podpowiadają, jak badania zacząć: czyli w jaki sposób opracować pytania i scenariusze, jak rozmawiać z respondentami. Następnie przechodzą do kompetencji miękkich. Sporo miejsca poświęconego jest kwestiom interakcji, sposobu prowadzenia rozmowy, improwizacji, a nawet negocjacji. Autorzy poruszają także tak niby drobne sprawy jak kwestia ubioru.

Zakończenie książki to trzy rozdziały o analizie i raportowaniu wyników – to bardzo cenna część tej pozycji, zarysująca większy temat. I tu w sumie autorzy wracają do początku – pokazują, że całe przygotowanie badań to jedno, ważniejsze jest: po co się je robiło i co się faktycznie osiągnęło. Zaczynają od sekcji „porządkowanie chaosu”. Jak sama nazwa sugeruje, opisują, jak zebrać wszystkie efekty badania w sposób umożliwiający analizę. Kolejny etap – to komunikowanie wniosków. To także warte uwagi, zwykle badacze pomijają ten ważny aspekt – czytelnego (idealnie: wizualnego) prezentowania efektów badań, dzięki czemu osoby nieuczestniczące w procesie od początku mają szansę zrozumieć zarówno cel badania, jak i jego wynik. Ostatni etap to maksymalne wykorzystanie wniosków, czyli garść porad, jak uniknąć sytuacji, gdy wyniki badań trafiają do szufiady.

Podsumowując: książka ciekawa, zwłaszcza dla początkujących, którzy chcą poznać temat badań UX. Zaawansowani również znajdą interesujące fragmenty opisów tzw. „głosów z terenu”, czyli praktyków z różnych firm. Warto zajrzeć.

Katarzyna Małecka

Tytuł:	<i>Badanie UX. Praktyczne techniki projektowania bezkonkurencyjnych produktów</i>
Autor:	Brad Nunnally, David Farkas
Stron:	216
Wydawnictwo:	O'Reilly, Helion
Data wydania:	2018-03-09

redakcja

Zamów prenumeratę magazynu Programista
przez formularz na stronie:

<http://programistamag.pl/typy-prenumeraty/>

lub zrealizuj ją na podstawie faktury Pro-forma. W spawie faktur Pro-forma prosimy kontaktować się z nami drogą mailową:
redakcja@programistamag.pl.

Prenumerata realizowana jest także przez RUCH S.A.

Zamówienia można składać bezpośrednio na stronie: www.prenumerata.ruch.com.pl

Pytania prosimy kierować na adres e-mail: prenumerata@ruch.com.pl

lub kontaktując się telefonicznie z numerem:

801 800 803 lub 22 717 59 59, godz. 7:00 – 18:00 (koszt połączenia wg taryfy operatora).

Magazyn Programista wydawany jest przez Dom Wydawniczy Anna Adamczyk

Wydawca/Redaktor naczelny: Anna Adamczyk (annaadamczyk@programistamag.pl).

Redaktor prowadzący: Mariusz „maryush” Witkowski (mariuszwitkowski@programistamag.pl).

Korekta: Tomasz Łopuszański. **Kierownik produkcji:** Havok. **DTP:** Havok.

Dział reklamy: reklama@programistamag.pl, tel. +48 663 220 102, tel. +48 604 312 716.

Prenumerata: prenumerata@programistamag.pl.

Współpraca: Michał Bartylek, Mariusz Sierakiewicz, Dawid Kaliszewski, Marek Sawerwain, Łukasz Mazur, Łukasz Łopuszański, Jacek Matulewski, Sławomir Sobótka, Dawid Borycki, Gynael Coldwind, Bartosz Chrabski, Rafał Kociś, Michał Sajdak, Michał Bentkowski, Paweł „KraZQ” Zakrzewski, Radek Smilgin.

Adres wydawcy: Dereniowa 4/47, 02-776 Warszawa.

Druk: <http://www.moduss.waw.pl/>, Nakład: 4500 egz.

Nota prawa

Redakcja zastrzega sobie prawo do skrótów i opracowania tekstu oraz do zmiany planów wydawniczych, tj. zmian w zapowiadanych tematach artykułów i terminach publikacji, a także nakładzie i objętości czasopisma.

O ile nie zaznaczono inaczej, wszelkie prawa do materiałów i znaków towarowych/firmowych zamieszczanych na łamach magazynu Programista są zastrzeżone. Kopiowanie i rozpowszechnianie ich bez zezwolenia jest zabronione.

Redakcja magazynu Programista nie ponosi odpowiedzialności za szkody bezpośrednie i pośrednie, jak również za inne straty i wydatki poniesione w związku z wykorzystaniem informacji prezentowanych na łamach magazynu Programista.



Szkolenia i warsztaty eksperckie - **bo umysł to Twoje najważniejsze narzędzie**



DDD



ARCH



TEST&CRAFT



AGILE&SOFT



JAVA



.NET



C&CPP



WEB



BAZY



MOBILNE



EIP

SPRAWDŹ **200**
AUTORSKICH
PROGRAMÓW
SZKOLEŃ



Dołącz do Ericsson i mają wpływ na to,
w którym kierunku zmierza postęp
technologiczny. Nasze zespoły biorą udział
w prestiżowych projektach ICT, które
pozwalały nam wszystkim swobodniej
pracować, studiować i żyć w zrównoważonych
społecznościach na całym świecie.

Obecnie ponad 40% światowego ruchu
mobilnego przechodzi przez sieci firmy
Ericsson, z których korzysta ponad
2,5 miliarda abonentów.

Aplikuj już dziś!

[ericsson.com
/careerspoland](http://ericsson.com/careerspoland)



Zmieniaj
z nami
świat