Wybrane elementy praktyki projektowania oprogramowania
Wykład 01/15 - Wprowadzenie

Wiktor Zychla 2019/2020

Sprawy organizacyjne

Z przyjemnością witam Państwa na wykładzie Wybrane elementy praktyki projektowania oprogramowania, który będzie okazją do zapoznania się w sposób przekrojowy ze współczesnym warsztatem technologicznym w obszarze projektowania i wytwarzania oprogramowania.

Celem naszego wykładu jest dostarczenie wiedzy i umiejętności pozwalającej poruszać się w obszarach inżynierii oprogramowania, baz danych, projektowania obiektowego oraz wybranych bieżących implementacji tych obszarów.

W ramach zajęć zostanie zaprezentowany cykl 15 wykładów uzupełnionych spotkaniami w laboratorium, w trakcie którego studenci będą mogli zmierzyć się z szeregiem praktycznych zadań, związanych z materiałem wykładu.

Wykłady będą uzupełnione notatkami, które proszę systematycznie przeglądać i korzystać z gęsto zamieszczonych w nich odnośników, stanowiących zachętę do samodzielnego poszukiwania i poszerzania wiedzy. Listy zadań będą publikowane w formie osobnych dokumentów.

Technologie, języki

Kompletny warsztat wytwarzania oprogramowania obejmuje wiele obszarów, w tym:

- Inżynieria oprogramowania – znajomość podstaw metodyk zarządzania projektami i metodyk wytwarzania oprogramowania oraz przebiegu i organizacji samego procesu. W tym obszarze omówimy skrótowo obszar metodyk zarządzania, na chwilę zatrzymamy się w obszarze metodyk wytwarzania, gdzie omówimy elementy praktyki metodycznej projektowania obiektowego:
    - Zbieranie wymagań
    - Przypadki użycia
    - Analiza obiektowa

  oraz poznamy przemysłowe sposoby dokumentowania w/w artefaktów – czyli język UML wraz z towarzyszącym mu warsztatem technologicznym

- Bazy danych – technologie magazynowania danych relacyjnych i nierelacyjnych, w tym wybrane języki zapytań. W ramach wykładu poznamy podstawy technologii relacyjnych, w tym wybrane bazy danych PostgreSQL i SQL Server oraz język zapytań SQL

- Języki programowania i platformy technologiczne – spośród tych wymieńmy tylko wybrane, w tym duże przemysłowe platformy technologiczne:
    - Język C# i środowisko .NET
    - Język Java i środowisko Jakarta EE (dawniej: Java EE, J2EE)
    - Język Python i jego interpretery
    - JavaScript/TypeScript i środowisko node.js (więcej niżej)

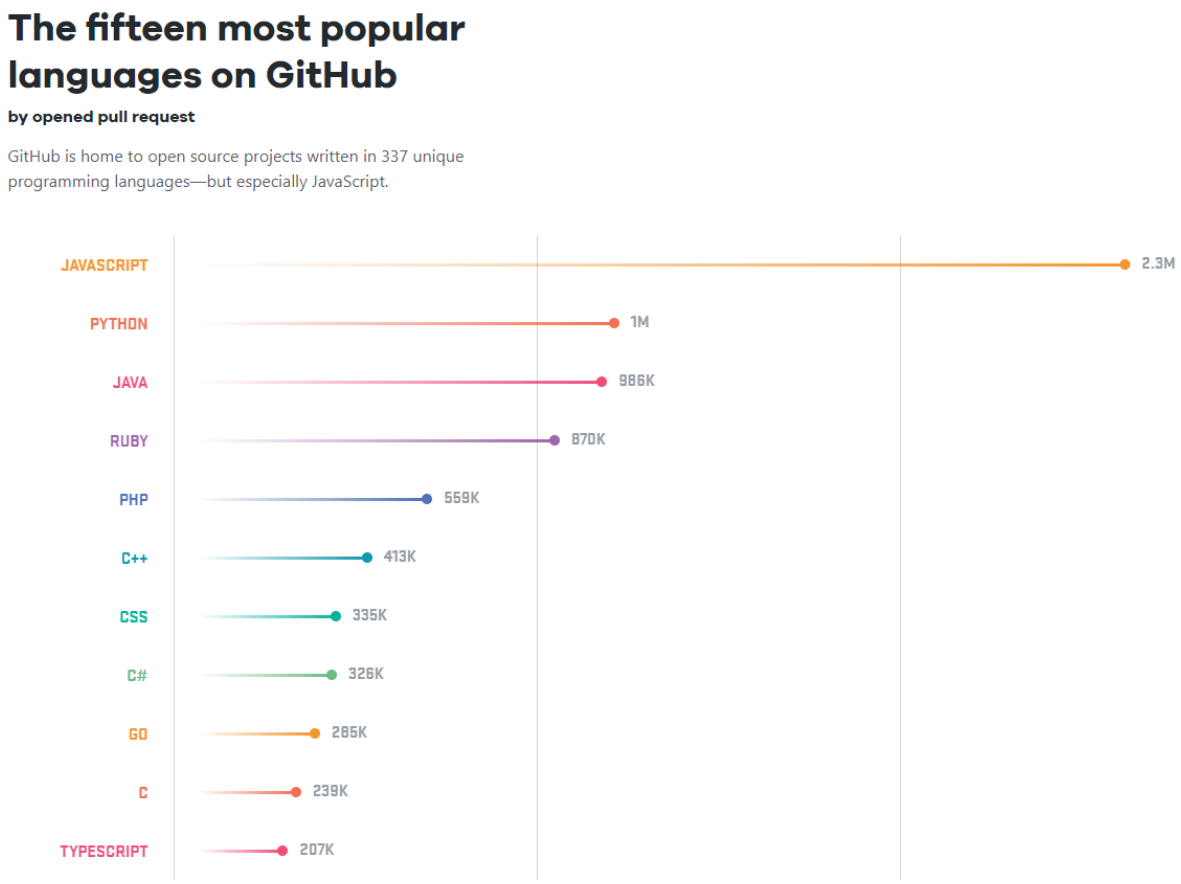
- Wzorce projektowe i wzorce architektury aplikacji – tymi zajmiemy się wyłącznie w wybranym zakresie, powiemy m.in. o wzorcach
    - Model-View-Controller
    - Repository i Unit-Of-Work

Szereg innych, bardzo interesujących rzeczy w naszym wykładzie się nie znajdzie, z braku miejsca, m.in.:

- Koncepcje Continuous Integration, Continuous Delivery i Continuous Deployment
- Praktyki refaktoryzacji
- Analiza czasochłonności
- Metryki jakościowe i ilościowe
- Itd.

Javascript – historia, aktualny status

Materiał wykładu przedstawiającego praktykę projektowania i wytwarzania oprogramowania musi ilustrować przedstawiane tezy w ramach wybranej, konkretnej technologii. Językiem wybranym na potrzeby niniejszego wykładu jest zdecydowanie najczęściej używany w praktyce język o największym zasięgu technologicznym - JavaScript.



Rysunek 1 Ranking popularności języków na platformie GitHub, według https://octoverse.github.com

Wbrew obiegowej opinii, wynikającej właśnie z dużej popularności, a co za tym idzie – z dużej ilości kodu różnej jakości (w tym niskiej!) jest to język o interesujących podstawach teoretycznych i niezwykłej uniwersalności i elastyczności, dzięki której współcześnie zdobył praktycznie wszystkie możliwe obszary technologiczne:

- Programowanie skryptów po stronie przeglądarki internetowej
- Programowanie aplikacji po stronie serwera – m.in. platforma node.js
- Programowanie aplikacji mobilnych – m.in. technologie NativeScript czy React Native

JavaScript narodził się w 1995 roku w startupie technologicznym Netscape Communications jako język skryptowy przeglądarki internetowej Mosaic. Zadanie zaprojektowania języka powierzono inżynierowi specjalizującemu się w projektowaniu języków, Brendanowi Eichowi. Język miał być odpowiedzią na język HyperTalk, umożliwiający tworzenie dynamicznych skryptów wspierających statyczne prezentacje tworzone w technologii HyperCard.

Rysunek 2 Brendan Eich

Zaprojektowany język, nazwany roboczo Mocha, był początkowo mocno inspirowany językiem funkcyjnym Scheme (dialekt Lispa), następnie w wyniku prac otrzymał składnię wzorowaną na Javie oraz elementy tzw. obiektowości prototypowej wzorowane na języku Self.

Jeden z uznanych specjalistów od JavaScript, Douglas Crockford, opowiada o tamtych czasach w trakcie prezentacji, którą warto prześledzić.

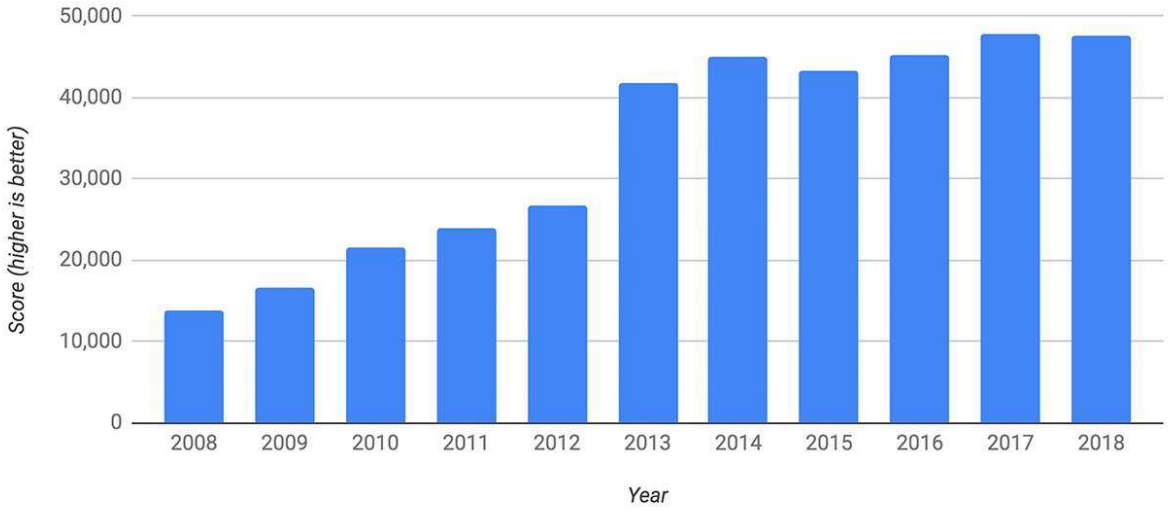
Po początkowym burzliwym rozwoju (wersja 2 w roku 1998, wersja 3 w roku 1999) nastąpiło wyraźnie spowolnienie, wynikające m.in. z braku porozumienia między kluczowymi dostawcami technologii. Kolejne wersje ukazywały się wolniej (wersja 5 w roku 2009, wersja 6 w 2015) i dopiero od 2015 można mówić o powrocie języka na właściwą ścieżkę rozwoju.

Edition	Date published	Changes from prior edition	Editor
1	June 1997	First edition	Guy L. Steele Jr.
2	June 1998	Editorial changes to keep the specification fully aligned with ISO/IEC 16262 international standard	Mike Cowlishaw
3	December 1999	Added regular expressions, better string handling, new control statements, try/catch exception handling, tighter definition of errors, formatting for numeric output and other enhancements	Mike Cowlishaw
4	*Abandoned*	Fourth Edition was abandoned, due to political differences concerning language complexity. Many features proposed for the Fourth Edition have been completely dropped; some were incorporated into the sixth edition.	

5	December 2009	Adds "strict mode," a subset intended to provide more thorough error checking and avoid error-prone constructs. Clarifies many ambiguities in the 3rd edition specification, and accommodates behaviour of real-world implementations that differed consistently from that specification. Adds some new features, such as getters and setters, library support for JSON, and more complete reflection on object properties.[9]	Pratap Lakshman, Allen Wirfs-Brock
5.1	June 2011	This edition 5.1 of the ECMAScript standard is fully aligned with third edition of the international standard ISO/IEC 16262:2011.	Pratap Lakshman, Allen Wirfs-Brock
6	June 2015[10]	The sixth edition, initially known as ECMAScript 6 (ES6) and later renamed to ECMAScript 2015 (ES2015)[10] adds significant new syntax for writing complex applications, including classes and modules, but defines them semantically in the same terms as ECMAScript 5 strict mode. Other new features include iterators and for/of loops, Python-style generators and generator expressions, arrow functions, binary data, typed arrays, collections (maps, sets and weak maps), promises, number and math enhancements, reflection, and proxies (metaprogramming for virtual objects and wrappers). As the first "ECMAScript Harmony" specification, it is also known as "ES6 Harmony."	Allen Wirfs-Brock
7	June 2016[11]	ECMAScript 2016 (ES2016)[11], the seventh edition, intended to continue the themes of language reform, code isolation, control of effects and library/tool enabling from ES2015, includes two new features: the exponentiation operator (`**`) and Array.prototype.includes.	Brian Terlson
8	June 2017[12]	ECMAScript 2017 (ES2017), the eighth edition, includes features for concurrency and atomics, syntactic integration with promises (async/await).[13][12]	Brian Terlson
9	June 2018[8]	ECMAScript 2018 (ES2018), the ninth edition, includes features for asynchronous iteration and generators, new regular expression features and rest/spread parameters.[8]	Brian Terlson

W 2008 roku nastąpiło jedno z przełomowych zdarzeń w rozwoju technologii – Google ogłosiło własną przeglądarkę, Chrome, wraz z silnikiem uruchomieniowym JavaScript, który został nazwany V8. W 2009 roku silnik zaadaptowano na potrzeby wysokowydajnego przetwarzania po stronie serwera, w ten sposób narodziło się środowisko node.js.

W trakcie kolejnych lat V8 otrzymał wiele usprawnień, włączanych do kolejnych wersji node.js. Silnik od samego początku stawiał na kompilację typu JIT (Just-In-Time) po stronie klienta, która zapewnia bardzo dużą wydajność uruchamianego kodu. Należy mimo to podkreślić, że w ciągu kolejnych lat, dzięki niekiedy przełomowym rozwiązaniom, wydajność uruchamiania kodu wzrosła kilkukrotnie:



Rysunek 3 Porównanie wydajności kompilacji JIT w silniku V8

W trakcie kolejnych wykładów skupimy się na języku oraz tych jego zastosowaniach które dotyczą aplikacji przeglądarkowych, zarówno po stronie klienta (przeglądarka) jak i serwera. Do przestudiowania we własnym zakresie pozostawimy inne, bardzo interesujące zastosowania technologii. Poniżej propozycje materiałów do przejrzenia „na zachętę" (stąd dość subiektywny wybór, skupiający się na spektakularnych efektach):

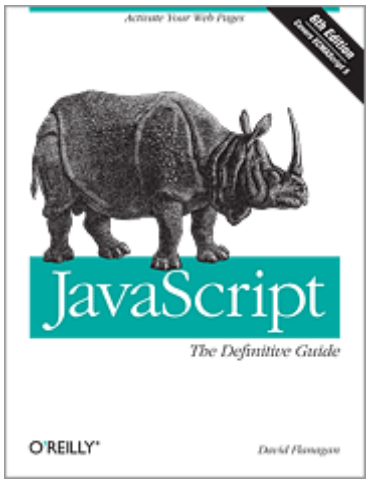
- WebGL – standard API graficznego 2D i 3D implementowanego przez przeglądarki
  - witryna Chrome Experiments
- Phaser - silnik graficzny 2D
- three.js i BabylonJS – silniki 3D
- Emscripten – kompilator C++ do JavaScript
- Lista kompilatorów innych języków do JavaScript (w tym dialekty JavaScript: TypeScript, CofeeScript)
- Jądro Linuxa skompilowane do JavaScript, uruchamiające się w przeglądarce
- ClassicReload – archiwum starego oprogramowania uruchamianego bezpośrednio w przeglądarce, w tym np.:
  - Windows 3.11
  - Windows 95
- jsDosBox – emulator DOS uruchamiający się w przeglądarce
- Emulatory różnych urządzeń i architektur, uruchamiające się w przeglądarce
- Konkursy programistyczne

- js1k – program zajmujący co najwyżej 1kb
- js13k – gra zajmująca co najwyżej 13kb

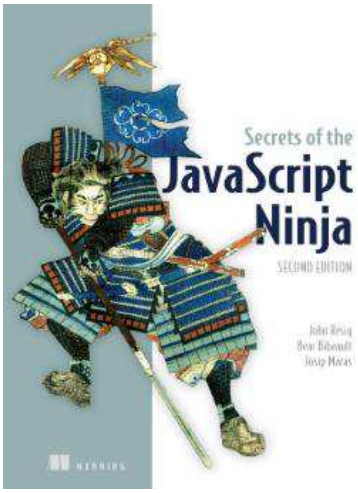
Literatura

Zachęcam do samodzielnego studiowania materiału. Poniżej propozycje źródeł:

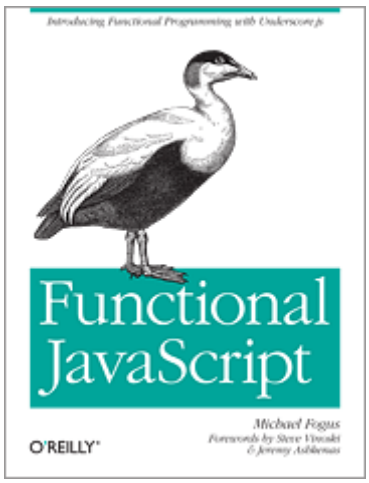
- Referencyjna dokumentacja języka utrzymywana przez fundację Mozilla
- Specyfikacja ogłaszana w ramach organizacji standaryzacyjnej ECMA
- David Flanagan, Javascript: The Definitive Guide



- Resig, Bibeault, Maras – Secrects of the JavaScript Ninja



- Fogus – Functional JavaScript



- Stefanov – JavaScript Patterns

- Hahn – Express in Action