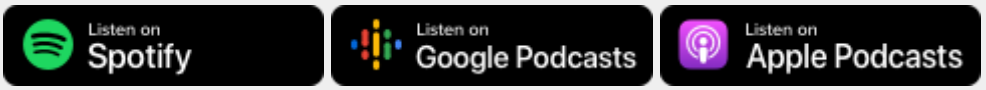
Pierwsze kroki w IT

PODCAST



Jak zawód programisty zmieniał się przez 15 lat?

Gość: Patryk Jar

Wszystkie polecane materiały i linki znajdziesz na stronie odcinka:

https://devmentor.pl/b/jak-zawod-programisty-zmienial-sie-przez-15-lat

Dziś moim gościem jest Patryk Jar. Patryk opowie nam o zmianach, jakie zaszły w ostatnich 15 latach pracy programisty. Patryku, dziękuję, że przyjąłeś moje zaproszenie na rozmowę.

Dziękuję bardzo za zaproszenie i bardzo miło mi tu być.

Zaczniemy od przedstawienia Twojej osoby – powiedz nam proszę co łączy cię z branżą IT i z jakimi technologiami miałeś do czynienia podczas swojej kariery?

Z branżą IT rozumianą dosyć szeroko, bo niekoniecznie mówimy tutaj o samej pracy zawodowej, ale o takiej działalności edukatorsko-hobbistycznej, jestem związany od ponad 21 lat. Moje początki to nasze wspólne działania, o czym może jeszcze dziś opowiemy, a aktualnie jestem liderem technicznym zespołu innowacji w firmie Kainos. Kainos to globalna firma IT, zatrudniająca ponad 3000 ludzi na trzech kontynentach, oferująca end-to-end software dla przeróżnych klientów. Jeśli chodzi o to, jak po kolei wyglądała moja ścieżka (bo nie można nazwać tego na początku karierą), to gdzieś w gimnazjum, ponad 20 lat temu, zainteresowałem się programowaniem: zestawem technologii prawdopodobnie całkowicie typowym dla tak młodych ludzi w tamtym czasie − Turbo Pascalem z jakimś Delphi, czyli środowiskiem pozwalającym budować Windowsową aplikację okienkową, HTML-em. Prawdopodobnie to wszystko zaczęło się w ogóle od HTML-a i tutaj pozdrowienia dla mojego nauczyciela pana Ireneusza Cierniaka, który był fantastycznym nauczycielem informatyki. Prawdopodobnie w dużej mierze zacząłem interesować się tym dzięki niemu.

Uczyłem się również na kursie Pawła Wimmera na Helionie, który dzisiaj prawdopodobnie jest już całkowicie zapomniany, ale wtedy był niezwykle popularny − wielu ludzi w IT, tak wiekowych, jak ja, może pamiętać właśnie ten kurs. Oczywiście był też CSS. Ciekawe jest to, że CSS-a nauczyłem się ściągając jakiś template i zmieniając to jedną, to drugą

wartość, patrząc co się tam wtedy dzieje. Potem oczywiście pojawiły się fantastyczne książki Erica Meyera (nie wiem na ile są one aktualne dzisiaj − pewnie się zdezaktualizowały, ale były w swoim czasie wybitne).

Może dodam, że też mam u siebie na półce książkę pod tytułem „Kuloodporne strony internetowe" czy coś takiego.

Tak, „Kuloodporne strony", ale nie tylko. Połowa mojej półki to Eric Meyer i nawet jeszcze dzisiaj mam taką specjalną sentymentalną półeczkę z książkami, po które może niekoniecznie sięgam, ale były one dla mnie kluczowe.

Potem pojawił się JavaScript, ale był to JavaScript niezwykle prosty (żeby nie powiedzieć prostacki), bo jednak były to dużo inne czasy niż dzisiaj. Później, w pewnym momencie, trochę zainspirowany moim nauczycielem informatyki z gimnazjum, napisałem kurs tworzenia stron www. Był rok 2003 (nie 23, tylko 2003) i pamiętam, że właśnie wtedy, dość typową rzeczą (która dla dzisiejszych użytkowników internetu może być niewiarygodna) było to, że taki kurs miał przycisk Zapisz kopię offline/Pobierz kopię offline i można było pobrać sobie kurs offline. Internet był strasznie wolny, strasznie drogi, bo wdzwaniało się modemem i tego typu rzeczy były potrzebne. Były też wtedy fora internetowe − dzisiaj tęsknię za forami internetowymi: przestrzenią moderowaną, gdzie bezsensowne wpisy są kasowane, gdzie ludzie dbają o swoje dobre imię, bo budują sobie markę. Wtedy każdy w internecie miał jakiś nickname, a nie swoje imię i nazwisko i z jakiegoś powodu bardziej dbano o to niż o swoje prawdziwe imię i nazwisko (jest to dla mnie zdumiewający aspekt społeczny).

Później, tak jak już wspomniałem, byłem redaktorem WebMate.org (tu się pewnie trochę uśmiechniesz), Youthcoders.net i yarpo.pl. Potem przez jakieś 10 lat miałem swój własny blog dla programistów, który od kilku lat już nie istnieje, bo uznałem, że już może jednak wystarczy i nie potrzebowałem go kontynuować.

To może dopowiem, bo nie wiem, czy słuchacze w ogóle zdają sobie sprawę, że właśnie WebMate.org to była taka przestrzeń, w której uczyliśmy się. Ja i ty publikowaliśmy artykuły (było tam jeszcze parę osób), żeby pokazać jak tworzyć strony internetowe. Byliśmy webmasterami − tak to się kiedyś nazywało. W zasadzie tak się poznaliśmy. Nie widzieliśmy się w życiu. Nie jestem pewien, czy jak zgraliśmy się tydzień, czy dwa tygodnie temu, to nie był pierwszy raz, kiedy mieliśmy okazję zobaczyć się na żywo. Jest to też ciekawa historia, że ludzie, którzy w ogóle się nie widzieli, tworzyli wspólną przestrzeń, w której dzielili się wiadomościami i teraz jesteśmy tu, gdzie jesteśmy: ty cały czas pracujesz jako programista, ja uczę ludzi programować i cały czas jesteśmy związani z tą samą rzeczą.

Webmastering to jest w ogóle bardzo ciekawa sprawa, bo już chyba nikt nigdzie nie używa takiego sformułowania. Ono jakoś naturalnie umarło, ale chodziło o kogoś, kto tworzy strony internetowe pokrywając właściwie wszystko od back endowca, front endowca, UX-a, Content Designera, DevOps-a, Delivery Managera i Scrum Mastera. To wszystko robił Webmaster. Jeden człowiek, który musiał zrobić wszystko i np. trzeba było sobie postawić serwer samodzielnie, czyli np. LAMP-a: Linux, Apache MySQL i PHP albo WAMP-a, gdzie Linux był zastąpiony

Windowsem. Były oczywiście całe paczki, np. serwer Krasnal, w którym instalowało się jedną paczką i wszystko działo się samo „automagicznie”. Było to niezwykle przydatne dla kogoś, kto był bardzo mało doświadczony w takich sprawach. Patrząc choćby na to: został wykonany olbrzymi krok − dzisiaj ściąga się jakimś npm-em odpowiednią paczkę, potem npm-start i działa.

Może od razu wejdę ci tu w słowo, bo o tym jeszcze będziemy mówić, ale warto zwrócić uwagę na to, że mówi się, że kiedyś było łatwiej, ale patrząc na to, o czym mówimy, to wcale tak nie było. Pamiętam, że aby w ogóle móc ruszyć np. z zainstalowaniem takiego serwera i żeby ten PHP czy MySQL zaczął działać, to trzeba było się mocno namęczyć i nie było gdzie szukać rozwiązania. Człowiek siedział godzinami, dniami nad jedną rzeczą i nie wiedział jak sobie z tym poradzić. Coś się udało i nawet nie zawsze było wiadomo jak, tylko po prostu coś się pozmieniało i działało.

Ja w ogóle tylko podpowiem, że jak się używało np. Linuxa (bo dawniej był to wyznacznik bycia superhakerem − siedziało się na Linuxie, jeszcze najlepiej tylko z wiersza poleceń) to bardzo często nie wchodziło się w Google, aby coś wyszukać, tylko np. wpisywało się w wierszu poleceń „man 3” i jakaś komenda i polegało to na tym, że chcę sprawdzić manual do tej komendy, czy do tej funkcji w C++, czy jakiejś powłoki Linuxowej.

Czyli dokumentację albo coś w tym stylu?

Tak. Ona była też cashowana lokalnie − to nie było coś takiego, że miało się od razu komentarze tak jak dzisiaj na Stack Overflow: mamy

wybrane najlepsze komentarze, informacje czy komentarz rozwiązał problem, czy nie. Wtedy wymagało to naprawdę ekstremalnie dużo więcej wysiłku i więcej samozaparcia, żeby spróbować coś zrobić, bo nie do końca było gdzie się spytać: siedziało się na problemie 5 godzin, a dzisiaj rozwiązałoby się go dosłownie w 30 sekund, bo czasami chodziło o średnik na końcu, czy coś podobnego, ale jak było się młodym, to można było się tego nie dowiedzieć.

W ten sposób doszliśmy do moich studiów (skoro już tak szybko przechodzimy, a przynajmniej próbuję szybko przejść po mojej karierze) i na początku studiów był oczywiście C/C++. Doceniam, że uczyłem się takich technologii, choć nigdy w nich nie pracowałem, ale ważna jest świadomość tego, że istnieje pamięć i trzeba nią zarządzać i trzeba te komórki pamięci jakoś obsługiwać. W C/C++ żeby to rozumieć wymogi są dużo większe niż np. w Java. W Java czy w JavaScripcie mówisz po prostu „new” coś tam i działa − nie zastanawiasz się, co się stało, a potem tego nie używasz i to znika, nie zastanawiasz się jak. W C/C++ tego nie było. Być może jest to przydatne jako pewna ciekawostka, natomiast tak czysto praktycznie, zawodowo, niekoniecznie jest to najważniejsze, ale może warto kiedyś spróbować.

Ja pamiętam jak na pierwszym roku studiów męczyli nas koszmarnie zadaniami na SPOJ. To była taka strona internetowa, na której kod wklejało się w formularz, klikało się Wyślij i on się kompilował i wykonywał zestawy zadań. Musiał wykonać to w odpowiednim czasie i z odpowiednim wynikiem. Ostatnio zauważyłem, że są takie fajne stronki (nie mam żadnego zysku, nie jestem właścicielem, jestem po prostu zwykłym użytkownikiem, który się czasem trochę bawi) np.

codingame.com, ale jest bardzo wiele podobnych stron: można się tam za darmo zarejestrować, nawet kontem githubowym. Jest to taka gamifikacja dla programowania − jak ktoś już ogarnął mniej więcej if-y, for-y i tego typu rzeczy tak, że to nie sprawia mu/jej problemu, aby napisać krótki kod 10-15 linijkowy tego typu, to dołącza się do jakiejś grupy i dostaje się jedno zadanie, które rozwiązujemy na czas, kto szybciej. Jest to bardzo fajne. Gdy znalazłem sobie tę stronkę, to przypomniały mi się czasy studiów i właśnie tego SPOJA i wrzucania kodu do przeglądarki. Tu jest to ładnie i fajnie.

Warto dodać, że są tam zadania z większości istniejących technologii, nie tylko z C czy C++.

Ja nie używałem C++, ale chyba jest. Natomiast jest tam oczywiście Java, Python, JavaScript − bardzo wiele technologii. Nie wiem jak długa jest ta lista, ale wydaje się bardzo długa i naprawdę jak ktoś szuka sobie takich dodatkowych zadań to jest to świetna okazja, z jednej strony trochę w formie rozrywkowej, bo przy okazji buduje się tam swoje konto, nabija XP-a, − taki trochę RPG.

Myślę, że jeżeli ma się już jakieś osiągnięcia to spokojnie można je też wrzucić do CV, żeby pokazać na jakim jest się poziomie. Działa to chyba trochę tak jak na forach, o których wspominałeś− że było tam napisane ile postów napisałeś, jaki masz level. Tutaj jest bardzo podobnie.

Coś w tym było. Natomiast tutaj też tak jest. Ogólnie, jeśli już mówimy trochę o tym jak się dzisiaj promować, myślę, że rozsądne wykorzystywanie LinkedIna to jest na pewno dobry pomysł. Przez

rozsądne mam na myśli przykładowo: robię jakiś fajny projekcik u siebie na GitHubie. Napisałem go, skończyłem, on w miarę działa: wrzucam zrzut ekranu, linki do tego GitHuba, opowiadam jak to się stało, dlaczego to robiłem. Jeśli było to coś mojego, jakaś zajawka, opowiadam skąd wzięła się ta zajawka − daję się trochę poznać. To może być tak, że wcale wam się to nie przyda − ten projekt nie będzie waszym źródłem zarobku, ani nie załatwi wam pracy, ale może być kiedyś tak, że w rekrutacji będziecie wy i jeszcze jeden kandydat, czy kandydatka i idziecie łeb w łeb, wszystko dokładnie tak samo. Ktoś wchodzi na wasz profil i widzi, że wy żyjecie tym programowaniem, a ta druga osoba nie opublikowała nawet zdjęcia i nie ma o niej absolutnie żadnych informacji i będzie to taki remis ze wskazaniem, gdzie wam się akurat uda. Nie mówiąc o tym, że zawsze można złapać fajny feedback, zawsze można złapać kogoś, kto może chciałby dołączyć z tego projektu. W tym CodinGame może będzie tak, że zbierzecie się lokalnie gdzieś w kawiarni i zrobicie sobie zawody lokalne w pięć osób.

A może ktoś was nawet zauważy, bo jest też chyba tak (przynajmniej tak mi się wydawało, nie wiem, czy to jeszcze jest), że takie super rozwiązania wrzucane są na listę, że jesteś np. w top 3 najlepszych rozwiązań i ludzie to lajkują albo wskazują ci, co jest nie tak, więc jest to społeczność, w której możesz też znaleźć miejsce dla siebie i dla swojej oferty, gdy chcesz pracować u kogoś.

Tak, tak, zdecydowanie – polecam, a potem polecam też mądre, racjonalne i takie sympatyczne dzielenie się tym, co robi się w tym obszarze, szczególnie kiedy chce się budować swoje CV.

Ja jako młody człowiek (może to był błąd) nigdy nie uczestniczyłem w hackathonach, nie było to wtedy tak popularne, przynajmniej z mojej perspektywy, a może moja bańka taka była i po prostu nie docierały do mnie te informacje. Teraz jest ich bardzo dużo. Polecałbym je nie dlatego, że są tam jakieś firmy i szukają perełek, ale w ten sposób budujecie sobie sieć ludzi, których znacie, budujecie sobie dobrą opinię fajnego człowieka, z którym się pracuje. Nieważne jak w przyszłości będzie wyglądał rynek, jak będzie wyglądała praca ludzi w IT (o czym będziemy jeszcze później mówić) – ważna jest umiejętność pracy z innymi ludźmi, umiejętność pracy w zespole, bycie fajnym ziomkiem, z którym się przyjemnie siedzi i programuje 24 godziny. Jak jesteś z kimś w stanie spędzić 24 godziny, nie pokłócić się z nim, a na koniec jeszcze zbijacie sobie piątaka i macie dobre wspomnienia, to znaczy, że macie dobry potencjał na to, żeby być fajnym członkiem zespołu.

Zdecydowanie. To może jeszcze odnośnie do tych hackathonów, to dodałbym, że nie wiem, czy one w ogóle były, bo ja pamiętam, że jak ja byłem na studiach, to stowarzyszenie informatyków, w którym byłem, organizowało pierwszą taką nockę w czasie moich studiów, więc chyba po prostu to dopiero zaczynało się dziać (teraz jest chyba piętnasta albo dziesiąta noc informatyka w Krakowie, więc zrobiło się to dość popularne i odbywa się cyklicznie).

Bardzo możliwe. Myślę, że ogólnie wszystkich tego typu wydarzeń jest teraz po prostu ogrom. I to też jest ta różnica pomiędzy kiedyś a dziś. Natomiast to nie musi być dla każdego. Nie proponowałbym tego komuś kto tego nie czuje, aby tam chodzić i się męczyć, ale dajcie sobie jedną szansę. Zobaczcie, czy się wam spodoba. To jest naprawdę fajne. Teraz

już jako mentor, jako osoba w firmie, która odpowiedzialna jest za współpracę ze środowiskiem start-upowo- akademickim, często zdarza mi się być choćby chwilowo na jakichś hackathonach. To jest często naprawdę fajna impreza.

Ja potwierdzam. Byłem tam w zeszłym roku i w tym roku też planuję być i zabrać nawet swoją żonę i córkę, bo wiem, że nie będą się nudzić, przynajmniej przez jakiś czas, bo jak pokazywałem córce filmiki, to bardzo jej się podobały, mówiła: *tata weź mnie, bo ja też chcę być*. Naprawdę fajna sprawa.

Tak, więc trzeba spróbować. Bo niekoniecznie musi się to wam spodobać, ale spróbujcie − czemu nie?

Wracając do tego jak przebiegała moja kariera − całkowicie normalna ścieżka chyba każdego, kto studiował w tym czasie co ja, czyli dorabianie robiąc jakieś proste stronki czy pracując nad różnymi innymi rozwiązaniami informatycznymi i praktycznie ten techniczny zbiór technologii: PHP, JavaScript, MySQL, jakiś FTP czyli File Transfer Protocol − nie było tak, że wrzuca się coś na git-a i git automatycznie daje znać jakiejś chmurce, która to ładnie zaciąga, deployuje wam ładnie − to wszystko było ręcznie. Do tego wszystkiego trzeba było się gdzieś zalogować, gdzieś wejść, gdzieś przeciągnąć jeden katalog z drugim i to naprawdę tak się robiło. Z jednej strony to było prostsze, bo wyglądało to trochę jak praca na folderach na swoim komputerze, ale z drugiej strony była to praca w więcej niż jedną osobę, albo w jedną osobę, ale nad większym projektem. To było jak chodzenie po linie.

Tak, nadpisywanie sobie: ktoś wrzucił swoje zmiany rano, a ja godzinę później wrzuciłem swoje, nie pytając o co chodzi i nagle się wszystko rozjechało. Nie zapomnę jak był też taki problem, że jak się miało źle skonfigurowane znaki polskie, to nagle wrzucałeś, a tu miałeś krzaczki i nie wiadomo było o co chodzi. W MySQL trzeba było pewnie poustawiać. Historie były naprawdę różne.

Owszem. Poziom problemów, które dzisiaj w ogóle nie istnieją.

Moja praca była wtedy „zdalna”, w dużym cudzysłowie, bo była zleceniem, a nie taką regularną pracą. Natomiast miałem tam kilka fajnych projektów, które dobrze wspominam. Jednym z nich było śledzenie TIR-ów, gdzie trzeba było odczytywać dane z GPS-a i pokazywać je na mapie z sensorów. To naprawdę był bardzo ciekawy projekt. Pracowałem tam kilka miesięcy przez wakacje. Jak zacząłem studia uznałem, że jednak chcę się skupić na studiach i nie dam rady ciągnąć jednocześnie tego i tego.

Pracowałem też nad nieistniejącą już stroną trafnadiagnoza.pl, gdzie, bez sztucznej inteligencji, zaznaczało się, jakie się ma objawy i z bazy danych po dopasowaniu listy objawów pokazywało jaka to potencjalnie choroba. Nie wiem, czy był to najskuteczniejszy sposób, ale może dlatego już nie istnieje.

Natomiast z ciekawostek − ja poniekąd sam siebie uznawałem za programistę JavaScript nim ktokolwiek uznawał JavaScript za język programowania, bo były to czasy, kiedy JavaScript uznawano za to coś, czego używa się do tego, żeby na stronie padał śnieg albo, gdy klikam i wychodzi mi alert, czy wychodzę ze strony i dostaję alert, *nie, nie*

wychodź, zostań z nami. Były to głównie przeszkadzajki i rzeczy, które irytowały użytkowników, a nie pomagały im.

Później natomiast nastąpił moment przełomowy − pojawianie się AJAX-a i bardziej dojrzałych frameworków czy bibliotek, gdzie JavaScript zaczął być traktowany już coraz poważniej i dzisiaj chyba już nikt nie podważa statusu JavaScriptu. Ewentualnie ktoś może powiedzieć, że lepszy jest TypeScript i ja mogę się z tym nawet zgodzić, chociaż nie będę za to ginął. W dużym projekcie statyczne typowanie jest łatwiejsze do refaktoringu, lepiej wspiera autouzupełnianie kodu IDE itd. − z czysto praktycznego punktu widzenia. Natomiast moim zdaniem JavaScript jako język sam w sobie jest bardzo przyjemny, fajny, nietypowy − bardzo go lubię.

Jak tak jeszcze o tym wspominamy, to może warto powiedzieć o bibliotekach, czyli jQuery, które były wtedy mega popularne i mega doceniane. Może warto też zwrócić uwagę na to, że wtedy przeglądarki nie wspierały JavaScriptu tak dobrze i w ogóle nie były one ujednolicone, więc trzeba było sprawdzać, czy wszystko działa na każdej przeglądarce, a jQuery miał to mocno ułatwiać − tym pewnie też jeszcze wspomnimy trochę później.

Tak. Wtedy trzeba było pisać wersję kodu dla przynajmniej dwóch przeglądarek, a czasami dla większej liczby przeglądarek − Internet Explorer 6 szczególnie, chociaż 7 też nie była idealna.

Przypomnijmy, że jest to przeglądarka od Microsoftu, bo musimy pamiętać, że nie wszyscy mogą sobie z tego zdawać sprawę.

To prawda. I wtedy w ogóle było tak, że Internet Explorer, jakby chciało mu się zlecić za dużo (np. wersji Internet Explorer 6), to on potrafił się po prostu zawiesić, więc dlatego było to też duże ograniczenie dla rozwoju aplikacji webowych: bo jeśli 50-60% rynku to byli ludzie używający tak przestarzałej przeglądarki (miała ona chyba 5 lat i praktycznie nie miała żadnych update'ów), to trudno było na to coś pisać i myślę, że to dlatego biblioteki te były tak popularne, bo nakładały one na to warstwę abstrakcji. Czyli ogólnie coś, co bardzo lubię − zamiast pracować nad implementacją, pracowałeś na interfejsie.

Tutaj mam wielką radę dla każdego juniora (choć jak się o tym pierwszy raz słyszy to nie jest banalne, ale z każdym kolejnym miesiącem i rokiem pracy pewnie będziecie coraz łatwiej przyswajać tego typu rzeczy), żeby programując, skupiać się na tym, aby tworzyć odpowiednie interfejsy. Nie mówię tu o interfejsie graficznym tylko o interfejsie waszego obiektu czy czegokolwiek np. metody − żeby ona była łatwa do wykorzystania i zrozumiała: że czytam nazwę, widzę parametry, wiem co się dzieje. I wtedy naprawdę ucieczka, *że Okej, to ja chcę mieć to − piszę. W środku chciałbym wykonać to i to.* I niemalże powinno się to czytać jak książkę: „jeśli*"* i zamiast: wtedy zmienna taka i nie taka i taka lub taka, to zróbcie z tego sobie prywatną metodę. Niech ona powie, co ten „if*"* znaczy: co ten warunek pięciu różnych zmiennych znaczy, że np. to znaczy, że nie zostało jeszcze zapisane w bazie danych, albo nie istnieje w ogóle. I np. czytam i nie muszę kompilować sobie tego w głowie i robić całej logiki w głowie, tylko wiem, że „to" oznacza „to" − czytając ten kod błyskawicznie i szybko dostaję informacje, która jest mi bardzo przydatna. jQuery był tutaj przykładem tego, że zamiast pięćdziesięciu „ifów" na każdą przeglądarkę, ja używałem selektora podobnego do tego, który potem

jest używany w CSS. jQuery rozpropagował w dużej mierze tego typu podejście i to było niezwykle wygodne w porównaniu z wcześniejszymi Get By Id, albo Get By Tag Name. To było akurat naprawdę fajne.

Potem zaczęło pojawiać się już coraz więcej frameworków JavaScriptu, choćby Dojo, Ext, szybko zmarły GWT, czyli Google Web Toolkit, o którym też będziemy pewnie za chwilkę mówić. Pamiętam, że pracowałem wtedy przez pewien czas na Politechnice, bo szukałem pracy magisterskiej i znalazłem pracę, która dotyczyła JavaScriptu: *Porównywanie frameworków JavaScriptu i wybór odpowiedniego frameworka JavaScriptu do konkretnego narzędzia (..)*. Poszedłem porozmawiać z promotorem i okazało się, że moja wiedza na ten temat jest całkiem ciekawa i dużo już wiem i umiem. Czytałem wtedy dużo książek czy źródeł anglojęzycznych, bo w Polsce tych źródeł było wtedy niezwykle mało. Pojawiały się już pierwsze, np. słynny artykuł *Falsy value*, który tłumaczył na czym polegają wartości true i false w JavaScripcie i dlaczego należy używać potrójnego równa się. Przez pewien czas artykuł ten był wręcz kultowy. Natomiast w tym projekcie, jeśli chodzi o back end, była już Java, był front end, czyli podział klient-serwer. Nie był to już jeden wielki spaghetti code. Były już frameworki JavaScriptu. Pojawił się tam SVN.

To może powiedzmy czym jest SVN?

To taka wcześniejsza opcja, która została skutecznie zastąpiona Gitem. W folderze .git lokalnie mamy tak naprawdę całe repozytorium, wszystkie skopiowane rzeczy i możemy zobaczyć, jak duży jest dany folder w dużym projekcie. W SVN-ie było jedno centralne repozytorium, do którego wrzucało się rzeczy i ściągało się z niego, ale u siebie

lokalnie miało się tylko i wyłącznie jedną wersję plików. Czyli w razie upadku centralnego repozytorium wszystko zostało stracone – mamy tylko to, co skopiowaliśmy u siebie. Natomiast sama zasada działania była dość podobna: commit i pull, niekoniecznie był tam push. Chyba od razu commitowało się tam do centralnego repozytorium (tak mi się wydaje − już dawno nie korzystałem).

Pojawiła się też wtedy wirtualizacja komputerów. Mieliśmy serwer i stawiało się na nim wirtualne maszyny − były to pierwsze początki chmury (chmura to jeszcze za dużo powiedziane, ale pojawiało się coś takiego). Warto pamiętać, że każda kolejna technologia jest pomostem do następnej. Jakby ktoś dzisiaj usiadł i miałby od razu wymyślić chmurę, to mógłby mieć z tym wielki problem, ale dzięki temu, że najpierw był jakiś serwer pod łóżkiem podłączony do Internetu, potem mieliśmy serwer wykupywany w jakiejś dużej serwerowni, którym zarządzała firma i mogliśmy się tam zalogować, instalować na nim różne rzeczy. Potem zaczęliśmy instalować wirtualne maszyny, następnie dokery itd. Wtedy rozumiemy skąd się to wszystko wzięło i dlaczego.

Potem, po pracy na Politechnice zacząłem pracę w Kainosie. Czyli w firmie, w której pracuję dzisiaj jako programista. Miałem wtedy okazję pojechać do Wielkiej Brytanii i pracować przy kilku projektach rządowych przez kilka kolejnych lat. W pierwszym projekcie (akurat dość dziwacznie jak na naszą firmę, ale było to dosyć dawno temu i klient podjął decyzję, na którą nie mieliśmy dużego wpływu) pracowaliśmy w PHP. Ponieważ był to projekt, który przede wszystkim miał być w pełni funkcjonalny w każdej stacji testującej samochody, to musiał być odporny na słabe łącze internetowe. Te strony musiały być dość lekkie, dość proste w budowie,

ale jednocześnie funkcjonalne, żeby nikt nie mylił się klikając różne rzeczy: solidność ponad piękno − tak bym to nazwał.

Właśnie tam było coś takiego jak Progressive Enhancement − nie wiem, czy ktokolwiek używa tego jeszcze dzisiaj. Chodziło o to, że sprawdzam na jak wiele pozwala mi przeglądarka, bo np. wspomniany Internet Explorer nie był w stanie obsłużyć połowy rzeczy, które chciałbyś zrobić, więc trzeba było sprawdzić: *czy jestem na Internet Explorerze? Jestem. OK − to w ogóle nic nie będzie. Będziesz miał tylko Submit, strona ci się będzie odczytywać, przeładowywać i tyle. Jeśli jesteś na Operze czy na Firefoxie to wtedy jest już wspierane, zrobimy ci jakieś ładne przejście, gdzieś w tle będzie request AJAX-owy, nie będziesz widział tego.* I tego typu rzeczy. Progressive Enhancement polegał właśnie na tym, żeby kroczek po kroczku robić jak najlepszą stronę, ale gdyby się okazało, że na którymś poziomie twoja przeglądarka tego nie wspiera, to wciąż powinieneś móc używać danej strony internetowej. W tym projekcie zabawne dla mnie było to, że zacząłem tam poniekąd jako front endowiec, bo miałem dużo doświadczenia z PHP, z MySQL − z typowego zestawu narzędzi front endowych i dodatkowo byłem dobrym programistą JavaScriptu, co było przydatne przy implementowanym tam Progressive Enhancement.

Natomiast finalnie okazało się, że wylądowałem w zespole do migracji danych, ponieważ moją cechą (tak zakładam, patrząc po moich późniejszych krokach w karierze) było to, że dość łatwo znajdowałem porozumienie między zespołami: byłem nienajgorszym człowiekiem do tego, żeby wysłać mnie do kogoś, aby pogadać, że może nawet nie pokłócę się z tą osobą, a może się z tym kimś polubię, dogadamy się,

ustalimy coś − wszyscy będą zadowoleni. Ponieważ było tam kilku dostawców dla tego rozwiązania, całkowicie naturalnie w którymś momencie zacząłem pełnić taką funkcję, w pewnym momencie w ogóle formalnie. Migrowaliśmy dane z poprzedniej bazy danych − 2,5 mld rekordów. Wiem od znajomych, którzy mieli w Wielkiej Brytanii samochody i musieli się tym zajmować, że wszystko działa. Był to więc wielki sukces, ale w tamtym czasie technologicznie pojawia się już chmura, Git, unit testy, testy funkcjonalne czy też fitness, czyli testy API, które polegają na tym, że uruchamia się API i uderza się w endpoints z konkretnymi danymi, sprawdza się, czy dla tych danych aplikacja zwraca to co trzeba. Pojawiło się już Selenium, czyli testowanie interfejsów graficznych gdzie end to end, zarówno horyzontalnie, jak i pionowo − klikam coś na stronie i po wszystkich warstwach wchodzi albo przeklikuję całą stronę i robię całe user journey.

Kolejnym narzędziem był Cucumber, był to już chyba BBD, czyli Behaviour Driven Development, gdzie opisywało się jakiś scenariusz i on miał być wykonany. Od strony jakości kodu zaczęło się tu już robić naprawdę solidniej.

I warto może dodać, że to było około roku 2015.

Tak, 2013-2015, to nie było tak, że wcześniej nikt tego nigdzie nie robił, tylko ja dopiero wtedy miałem okazję to wykorzystywać. Wcześniej mogłem próbować samodzielnie robić jakieś unit testy, zobaczyć czy coś działa, czy nie. Warto tu zauważyć, że wszystkie te narzędzia, których wtedy używaliśmy zachęcały lub wręcz wymuszały budowanie aplikacji w ten sposób, żeby łatwo ją było testować. Podział na klient-serwer również sprawia, że łatwo jest testować osobno logikę po stronie API i

można testować sobie front end np. mockując API. Jednocześnie używanie jakiegoś frameworka po stronie back endu sprawia, że masz Controller, Model, View − jesteś w stanie przetestować każdy z tych elementów osobno w unit testach.

Wracając do jQuery, on nie miał takiej funkcji: być może da się pisać jQuery tak, aby był on modularnym jQuery i można było go sobie łatwo testować, ale to nie jest w jego rdzeniu. Jego założeniem nie jest to, żeby łatwo było przetestować to co napisaliśmy. Jego założeniem jest „załaduj mnie i używaj" − ma być szybko i łatwo. Możemy to nazwać takim old schoolowym podejściem: robię coś, trochę na łapu capu, ale działa. Przeklikałem, działa, wrzucam na serwer, wszyscy zadowoleni, klient szczęśliwy. Natomiast w dużych projektach takie podejście po prostu się nie sprawdzi – jest to niewykonalne. Nie chodzi o to, że zajmowałoby to więcej czy mniej czasu. To byłoby nieosiągalne. Przeklikanie projektu, który ma ileś dziesiąt tysięcy linijek kodu: jak zmienia się jeden moduł, czy on nie wpłynął na inne rzeczy. 600 możliwych ścieżek, które tu istnieją, po prostu jest to niewykonalne w skończonym czasie.

Dlatego właśnie narzędzia, nazwijmy to dojrzalsze czy bardziej profesjonalne, z założenia same wymuszają na tobie taki podział. Zobaczmy jak wygląda dzisiaj np. React. On z założenia ma folder, w tym folderze masz jeden plik, drugi, trzeci. Nawet nazwa: wymuszaj, który będzie testem, który będzie czym. Pod tym względem uważam, że po prostu sama technologia widzi, że jest to niezbędne. Jednocześnie jednak próg wejścia jest trudniejszy dla osoby początkującej. Ja tworzyłem plik index.html, wrzucałem go przez FTP na serwer − działało.

A dzisiaj trzeba jednak złączyć ze sobą dużo więcej poszczególnych bloczków.

Później, gdy z tego projektu przeszedłem do innego, już w Londynie, zacząłem być Javowcem. Był tam mały front end w Angularze − operowaliśmy głównie w Javie. Używaliśmy tam OpenDJ, OpenAM. OpenDJ to taki LDAP, w dużym (może za dużym) skrócie, taka baza danych. OpenAM to był taki system do logowania, żeby móc robić single sign-on. Wtedy, mimo że mieliśmy już Docekra, YAML-a, Ansibla, była już chmura, to ciągle sami definiowaliśmy na HaProxy, który serwer będzie głównym serwerem, który pomocniczym itd. Tutaj znowu był kroczek do przodu. Bardzo dobrze wspominam tamten projekt, bo już jako senior mogłem uczyć się bardzo dużo od wybitnych specjalistów, byli to naprawdę świetni ludzie. Życzyłbym każdemu, żeby od czasu do czasu trafił do takich projektów, aby miał okazję się uczyć tyle ile ja wtedy. Widzimy, że był to już projekt. Mimo że był to rok 2017-2018, widzimy, że wyglądało to już całkiem podobnie do dzisiejszych projektów. Jakby ktoś cofnął się do tamtych czasów, to nie czułby szoku kulturowego.

Następnie byłem przez moment BA w jednym projekcie, bo był to projekt bardzo techniczny, gdzie potrzebna była świadomość różnych aspektów czysto programistyczno bazo-danowych. Pracowaliśmy nad biblioteczką, która miała być potem wykorzystywana przez zespoły back endowe, więc jeszcze niżej niż back end. Wtedy znowu, jako członek zespołu, byłem otwarty na rozmowę, byłem chętny do tego, żeby coś proaktywnie zrobić np. dostaję jakąś wiadomość: a*, jesteś zablokowany? No dobra, to ja nic nie mogę zrobić, to trzy dni nic nie robię.* Nie − *Jesteś*

zablokowany? To od razu działam: *Czy mamy Skypa, czy mamy Teams, czy mamy Discorda, czy cokolwiek tam innego używacie?* Jeśli pracujemy zdalnie − od razu robię pokój z trzema osobami, które powinny o tym wiedzieć: *słuchajcie, jest ten problem, jest to tu udokumentowane w Jirze, musimy zrobić to i to. Ja nie mam tu dostępów, wiem, że ty masz dostępy. Proszę zrób to, bo inaczej trzy osoby są zablokowane.* Tak proste rzeczy potrafią ekstremalnie przyspieszyć pracę zespołu.

Nie chciałbym tutaj mocno krytykować systemu edukacji, ale chciałbym zauważyć pewną rzecz w polskim systemie edukacji. Dostajesz zadanie domowe (teraz może nie, ale mniejsza o to). Jak nie zrobiło się zadania domowego, siedziało się cicho i nikt nie złapał, to następnego dnia był nowy dzień i wszystko było zapomniane. W zespole projektowym tak nie będzie, bo jeśli dzisiaj tego nie zrobisz, to jutro nadal nie jest to zrobione i udawanie, że nie ma problemu, nie jest żadnym rozwiązaniem. Dlatego znowu taka rada od trochę bardziej doświadczonego kolegi: szczerze i otwarcie mówcie − *słuchajcie, ja mam problem, ja tego i tego nie umiem, nie wiem jak to zrobić, niech ktoś mi pomoże*. Jeśli wy tego teraz nie zrobicie i sami nie wykonacie tego w ciągu np. pięciu dni, to tydzień został zmarnowany. Jednocześnie być może dwa czy trzy inne zadania, które miały być odblokowane przez wasze zadanie, będą zablokowane i w tym momencie wszyscy będą zablokowani. Możliwe, że to nie chodzi o to, że wy nie umiecie tego zrobić, tylko mogliście się już tak bardzo zakręcić na jakimś jednym punkcie, że gdyby tylko ktoś z wami usiadł i przeprowadzilibyście go przez swój kod, tłumacząc co robicie, to byście sami powiedzieli, *już wiem, dobra, idź sobie*. Pozwólcie sami wykorzystać drugą połowę mózgu, bo programując, czytając kod

potrzebna jest logika, a próbujcie to opowiedzieć − jest chociażby metoda gumowej kaczki, która polega na tym, że bierzesz gumową kaczkę i opowiadasz gumowej kaczce to, co twój kod robi. Przynosi to zaskakująco dobre efekty, bo wymaga od ciebie opowiedzenia o tym, co twój kod ma zrobić.

Później znowu wróciłem do projektu dla Ministerstwa Sprawiedliwości Wielkiej Brytanii, znowu Java, tym razem akurat Spring Boot. Wcześniej był Drop Wizard − nie widzę większej różnicy. Pamiętajcie o tym, że technologia to technologia. Jest to łopata/szpadel: potrzebujecie skopać ogródek − bierzecie szpadel, potrzebujecie przerzucić węgiel − weźmiecie łopatę. Nie można się aż tak strasznie fiksować. Moim zdaniem olbrzymią wartością (promowaną w firmie, w której pracuję, ale pewnie nie tylko u nas) jest to, żeby mieć dobre podstawy inżynierskie, wiedzę o tym jak wytwarza się programowanie, jak to powinno działać. Na to trzeba nałożyć umiejętności programowania w tej konkretnej, dowolnej już technologii czy to Java, JavaScript, React, specjalisty od bardzo wysublimowanych baz danych konkretnej wersji. Natomiast nadal jest ta warstwa gruba, całej otoczki programistycznej, czyli wszystkiego co musi robić inżynier oprogramowania, a nie jest konkretnie programowaniem. Moim zdaniem jest to naprawdę bardzo istotne. Dlatego mówiłem wcześniej o tym, żeby np. spróbować kiedyś poprogramować w C++, w JavaScripcie, w Javie, może w Pythonie, żeby zobaczyć, że jak nauczyło się już programować w jednym języku, to potem to jest tylko *syntax i sugar*. Różnica jest tylko między tym, czy masz na końcu klamrę, czy dwukropek. Oczywiście nie mówię, że będziesz od razu specjalistą w tej drugiej dziedzinie, bo potem są tam różne smaczki i jak ktoś używa np. w JavaScriptu i nie wie, jak *this*

będzie się zmieniało w różnym kontekście, to będzie bolało, ale ogólnie jest to po prostu technologia.

Nie bójcie się innych technologii. Bądźcie specjalistami w jakiejś jednej konkretnej technologii, ale nie róbcie potem tak, że *ja znam tylko C++, a mam zrobić stronę, no nie wiem, to tam CSS, to ja C++'em będę to pisał, to bez sensu*. Dajcie sobie szansę nauczyć się czegoś nowego.

W moich ostatnich projektach korzystałem z technologii całkowicie nowoczesnej: Azure, Kubernetes, mikroserwisy, Node.js, porządna warstwa testowania na wszystkich poziomach, pipeline'y. To nie jest tak, że pipeline'y pojawiły się nagle − korzystaliśmy już wtedy z chmur najnowszego rodzaju, czyli np. Azure, ale AWS dostarcza to samo. Pewnie Google i inne wielkie firmy również − nie różnią się pewnie od siebie drastycznie. Znowu − ta technologia sama z siebie zakłada, że ty będziesz tego potrzebował i poniekąd trochę to od ciebie wymusza. Chcesz, chociażby zdeployować prostą stronkę w Amplify'u, to od razu masz pipeline'a, chcesz czy nie chcesz. Będzie pipeline, zdefiniujemy dla ciebie defaultowy plik YAML, o którym nawet nie musisz wiedzieć, ale on tam jest. Jak będziesz chciał coś zmienić, to sobie wejdziesz i zmienisz.

To pokazuje jak ta technologia nadąża za potrzebami rynku, inżynierii i oprogramowania, nie tylko programowania jako takiego − coraz mniejszą wartością jest samo pisanie kodów, w sensie akt dotykania klawiatury, żeby powstawały jakieś tam znaczki i z nich zmienne funkcje 3 klasy. To nadal jest istotne i nadal trzeba to robić dobrze, ale to nie wystarczy. To jest ta główna wielka zmiana.

Jeszcze sobie do tego przejdziemy. Mieliśmy już całą historię jak to wyglądało, jak to się zmieniało, teraz przejdziemy do szczegółów − na początek chciałbym zapytać o etap nauki. Wiemy już, kiedy to było, ale jak w twojej perspektywie wyglądała wtedy branża i sam proces zdobywania wiedzy? Troszkę już o tym wspomniałeś, ale może chciałbyś coś dodać?

Moim zdaniem taką podstawową różnicą było pewnie to, że w IT, w programowaniu (wtedy jeszcze bardziej programowaniu, bo dużo więcej się programowało, a mniej było wszystkich dodatkowych rzeczy) było więcej takich pasjonatów, hobbistów. To byli ludzie, zresztą sam byłem takim człowiekiem, który, nawet gdyby mi nikt za to nie płacił, to i tak bym siedział i programował. To było moje hobby − uwielbiałem ten motyw, że mam klawiaturę i mogę coś stworzyć z niczego, z takiej czystej abstrakcji: coś powstaje, coś tam działa, coś się wyświetla. To było dla mnie po prostu niesamowite i dawało mi bardzo wiele frajdy i przyjemności, więc nawet gdybym nie miał później z tego pracy zawodowej, to pewnie nadal od czasu do czasu bym sobie gdzieś siadał i poprogramował, tak jak niektórzy ludzie układają puzzle czy pasjansa. Ja po prostu lubiłem programować.

Natomiast było dużo mniej materiałów. Miało to wady i zalety, bo z jednej strony, jak był jeden kurs Pawła Wimmera, to wszyscy wiedzieli, że wchodzisz w ten kurs i czytasz. Czy był on optymalny dla nauki? Pewnie nie, ale w ogóle był i był pierwszym albo jednym z pierwszych kursów Polsce. Jednak wiele osób się z niego uczyło, więc pewnie był niezły. Nie było tak wielu różnych źródeł, na YouTubie, bo w ogóle YouTube'a

nie było w takiej formie jak dzisiaj. Wtedy nie czuliśmy się tak przytłoczeniu, że *o nie, to pewnie muszę jeszcze obejrzeć to, to i to, bo pewnie nie umiem − to jak nie umiem, to nie mogę w ogóle programować, kto to widział*? Z jednej strony, ten brak materiałów był problemem, ale ponieważ ich było tak mało, to jeśli ktoś chciał, to po prostu brał to co było i robił. I robił tak, jak się dało. To było dosyć ciekawe.

Przyszło mi też do głowy, że nie było tylu rozpraszaczy. Siadałeś i robiłeś swoje, a nie tak, że wchodzisz na YouTuba, chcesz obejrzeć, jak się coś robi, a nagle pięć różnych filmików o czymś innym i zamiast robić co masz robić, to sobie przyklikujesz.

Dokładnie. Nie mówiąc o tym, że nie było social mediów. Fora internetowe były zupełnie inne. Taka ciekawostka: Twitter, dzisiaj X, ale wtedy jeszcze Twitter, miał ograniczenie do 140 znaków. Pewnie niektórzy nie uwierzą, ale na forach dawniej bywały wtyczki, które sprawiały, że jak napisało się posta krótszego niż 200 znaków, to nie dało rady go zapisać, bo uznawano, że post, który ma tylko 200 znaków nie może mieć wartości − naprawdę były takie wtyczki do for internetowych. Nie zawsze tak było, ale można było zrobić coś takiego. Świat zmienił się więc chociażby pod tym względem.

Nie było sytuacji, że *o, dostałem tutaj maila, o, tu mnie ktoś zagadał, o, tutaj sobie obejrzę coś fajnego*. Internet był tak wolny − jak miałem kilkanaście lat, to wdzwaniałem się przez modem i to było zawsze na zasadzie: *ok, to mam, 10 minut, pół godziny to byłoby już super czas*. Poza tym to łączenie było strasznie wolne, w ogóle nie było możliwości

robienia tych rzeczy, które można robić dzisiaj. Przeglądarki były inne, więc nawet nie dałoby rady zbudować takich aplikacji, które są dzisiaj.

Natomiast było mniej materiałów, było mniej rozpraszaczy, więc ludzie po prostu siedzieli, uczyli się i kodowali. Możliwe, że kodowanie było wtedy bardziej atrakcyjne, bo było mniej jeszcze atrakcyjniejszych rzeczy, więc pewnie tu było trochę łatwiej.

Kolejna rzecz: nie było choćby takich podcastów jak twój, nie było videotutoriali. Wyobraźmy sobie, że czytasz tekst i ktoś ominął jakiś fragment o tym, że gdzieś kliknął, coś zmienił. W videotutorialu nie ominął tego, tylko ewentualnie nie mówi w tym czasie, ale mówi *no i teraz tutaj tam kompilujemy*, ale zaznaczył jakiś tam checkbox i widzisz, że zaznaczył. Później np. przez dwa dni szukało się rozwiązania: *Kurde, czemu nie działa? Co tu się stało? No przecież jest napisane, że powinno działać, a nie działa*. To jest znowu, taki miecz obusieczny: on z jednej strony bardzo pomaga, z drugiej strony, może trochę rozprasza, przytłacza, że tyle tego wszystkiego jest.

Teoretycznie mogłoby się wydawać, że trzeba było poświęcić mniej czasu, żeby się nauczyć, bo objętościowo było wszystkiego mniej, ale jeżeli właśnie pojawiło się dużo takich problemów, o których mówisz, to nie jest tak, że przeczytało się coś i się to zrobiło, tylko siedziało się trzy dni i szukało się tego dodatkowego przycisku.

Dokładnie. I to naprawdę był bardzo frustrujący czas. Ja chyba dzisiaj bym nie umiał − jestem już tak przyzwyczajony do tego, że mam problem, to wpisuję go sobie w Google, w ChatGPT. Tak, używam ChatGPT, chociaż z dużą dozą braku zaufania dla wyników. Natomiast

nie widzę w tym niczego złego, tak długo jak to, co wytwarzam uznaję osobiście, etycznie, w głowie, w moim sumieniu − że to jest moje, bo używałem narzędzia. Gdy w jakimś IDE generuje sobie automatycznie getery, setery, to też nie uważam, żeby to przestało być moje z tego powodu. Ja dokładnie wiem co robię, po co i jak to robię, oszczędzam tylko czas na samo klepanie w klawiaturę. Jeśli np. mam jakiś problem i chciałbym się czegoś dowiedzieć − używam ChatGPT i on np. proponuje mi takie czy takie rozwiązanie, to stwierdzam: *no to mi się bardziej podoba niż to i sobie to użyję*. Nie widzę w tym problemu. Moim zdaniem nie jest to oszustwo, ale ja nadal wiem co ten kod robi, dokładnie to rozumiem i jestem w stanie też taki kod napisać, tylko dla przykładu usprawniło mi to pracę o 5 minut − 5 minut tu, 5 minut tam i okazuje się, że w tygodniu uzyskujesz pół dnia i można go jakoś ciekawie spożytkować.

Myślę też, że dziś jest więcej firm IT i wiele z nich (nie wszystkie niestety) aktywnie uczestniczy w jakimś środowisku programistów czy środowisku IT czy organizuje meetupy. W przyszłą środę, 25 kwietnia (więc już i tak nie może być to reklama, ponieważ release tego podcastu będzie później) będziemy mieli spotkanie w Gdańsku – stacjonarnie, u nas, 80 osób zarejestrowanych − które będzie dotyczyło Accessibility i testowania Accessibility. Wczoraj ustawiałem krzesła na Event Space, żeby zobaczyć czy na pewno wszyscy się zmieszczą. Chyba się zmieszczą, trzymam kciuki. Tutaj zobacz − ja mówię Event Space, w biurze mamy coś takiego. Dawniej nie mieliśmy czegoś takiego, bo ta branża się tak zmieniła, że na tyle często mamy jakieś wydarzenia, jakieś prezentacje, że ta przestrzeń stała się bardzo przydatna, potrzebna i dlatego ją dzisiaj mamy.

Tutaj kolejna zmiana: programista dzisiaj musi umieć prezentować. Może niekoniecznie junior, ale senior na pewno będzie co najmniej raz w miesiącu coś prezentował: dla klienta, dla potencjalnego klienta w ramach zespołu, będzie prezentował w ramach firmy np. nową wersję czegoś tam i że co się zmieniło albo, że wykryliśmy jakiś bug vulnerability w Cyber Security i że jak się teraz przed tym zabezpieczyć: 10-minutowy call dla 50 osób z firmy. To jest dziś bardzo istotny skill, który kiedyś nie był tak istotny. Kiedyś dużo bardziej liczyły się po prostu twoje twarde umiejętności. Dzisiaj jest dużo większy nacisk na pracę w zespole, prezentacje, umiejętności miękkie. Choćby w sytuacji gdzie robisz komuś code review (czegoś takiego też kiedyś nie było) to w jaki sposób to piszesz? Czy piszesz w taki sposób, że ten ktoś sobie myśli: *kurde, mądry ziomek, ale fajnie mi podpisał: to dobre, to dobre,* czy piszesz tak, że sobie myśli: *ale buc, nie chcę z nim gadać*. W obu wypadkach możesz mieć rację − treść będzie ta sama, ale chodzi tu o formę. Na pewno ten obszar jest dziś skrajnie różny niż był kiedyś.

Kiedyś było mniej firm i, siłą rzeczy, mniej wydarzeń. Organizowanie wydarzeń było w ogóle dużo mniej popularne. Branża była dużo mniej dojrzała, było mniej automatyzacji, mniej zarządzania jakością. Chociaż tutaj zauważmy jak dużo jest dziś ofert pracy dla testerów − czy to ręcznych, czy automatycznych w porównaniu do ilości ofert kiedyś.

Wydaje mi się, że w ogóle nie było kursów, szkoleń, ani niczego takiego, co pozwalałoby się w ogóle nauczyć. Już pomijając to, że za moich czasów studia nie do końca przygotowywały do bycia programistą. Ja mam raczej takie wyobrażenie, że był profesor,

który uczył się 20 lat temu i nie aktualizował wiedzy i my uczyliśmy się tego samego co 20 lat temu.

Rzeczywiście często mogło tak być. Natomiast jako lider zespołu innowacji, jednym z obszarów moich działań jest właśnie współpraca z uczelniami. I muszę powiedzieć, że widzę niezwykły postęp w porównaniu do okresu, gdy ja studiowałem i potem przez krótki czas pracowałem na uczelni jako programista, nie jako naukowiec. Idziemy w dobrym kierunku: współpraca uczelnia - biznes działa naprawdę fajnie i moim zdaniem jest coraz lepiej. Nie wiem, jak to się przekłada na konkretne zajęcia. Być może nadal nie zawsze są idealne, ale jestem tutaj optymistą.

Mamy za sobą naukę. Teraz czas na pierwsze prace − powiedz nam proszę, jakie technologie cieszyły się popularnością? Trochę już o tym wspomniałeś, ale powiedz czego w ogóle wymagało się od kandydata?

Tak. Rozdzielę tutaj może pierwszą pracę jako zlecenie: udzielałem się gdzieś na forach, wyrobiłem sobie markę i ktoś się do mnie odzywał, *słuchaj mam stronkę do zrobienia − Ok, spoko, tu jest taka cena, tu jest to, na kiedy i tak dalej*. Wtedy było się po prostu jednoosobową firmą, która wszystko miała zrobić.

Webmasterem.

Webmasterem, dokładnie. Natomiast jak już się poszło do firmy, nawet na rozmowie kwalifikacyjnej, dużo więcej pytano i dużo bardziej istotne

było właśnie to, czy umiesz napisać jakiś fragment kodu, który rozwiąże jakiś fragment problemu. Może ja trafiałem na takie firmy na rozmowach kwalifikacyjnych, ale jak rozmawiałem ze znajomymi to mieli podobne odczucia i mam wrażenie, że wtedy to był taki trend, że programista był programistą i dostawał zadania programistyczne.

Dziś, choćby u nas w procesie rekrutacji (nie jestem w pełni świadomy, jak inni prowadzą procesy rekrutacji, ale spodziewam się, że też jednak jest ta zmiana w tym kierunku), jest dużo więcej pytań w stylu: *dobrze, to opisz, że miałeś jakiś tam problem i jak go rozwiązałeś,* a nie: *napisz kod, który rozwiąże ten problem* − zakładamy, że umiesz programować, przynajmniej w wystarczającym stopniu. Ważniejsze jest właśnie to wszystko inne: *Poszedłeś porozmawiać z Product Ownerem? Poszedłeś porozmawiać z seniorem? A może wpisałeś pytanie do ChatGPT?* To może być dobry pomysł albo nie − zależy od sytuacji. Sama rozmowa kwalifikacyjna wyglądała wtedy inaczej.

Ja dostałem pracę jako ten webmaster czy web developer (chyba tak formalnie nazywała się moja pierwsza praca na umowie) i tam na pewno było pytanie: *czy dobrze programuję w tym? Czy dobrze programuję w tym? Czy rozwiążę np. jakiś CSS-owy selektor i powiem, co mi tak naprawdę złapie na tej stronie taki selektor?* Sprawdzili mi angielski − dali mi do przeczytania jakąś dokumentację po angielsku i miałem powiedzieć co dana funkcja robi. Z mojej perspektywy na rozmowach kwalifikacyjnych były wtedy tego typu rzeczy plus trzeba było rozwiązać jakieś tam zadanko programistyczne albo nawet kilka takich zadanek. Wyglądało to inaczej niż dzisiaj, przynajmniej inaczej niż ja to teraz widzę z drugiej strony jako interwiewer, wiedząc jakie my zadajemy

pytania. Być może moja firma podchodzi do sprawy troszkę inaczej, bo my stawiamy bardziej na to, żeby być inżynierem oprogramowania, niż np. inżynierem Java. Tutaj być może jesteśmy trochę inni od rynku, ale niekoniecznie. Wydaje mi się, że jest to coraz częstsze.

Myślę, że może to też bardzo zależeć od wielkości firmy − jak masz dużą firmę, która pracuje w różnych technologiach, to dla nich istotne jest, żeby człowiek był elastyczny, żeby można go było wrzucić do różnych projektów. Natomiast jeżeli firma skupia się na jednej konkretnej technologii, to bardziej jej zależy na tej konkretnej technologii, a nie na wszystkim.

Bardzo możliwe, że to tak wygląda. I u nas też zdarzały się czasami sytuacje, gdzie rzeczywiście zatrudniano jakiegoś konkretnego człowieka, nawet jako kontraktora, który np. miał zespół nauczyć tej konkretnej technologii, bo gdzieś tam było przeczucie, że może warto byłoby trochę podskillować ludzi. Ogólnie sądzę, że podstawową różnicą było to, że dawniej zdecydowanie większy nacisk był na umiejętności twarde. Dziś nie są one nieistotne, żeby ktoś nie stwierdził, że nie muszę już umieć nawet programować. Nie, nadal musisz umieć programować, nadal musisz umieć to zrobić. Dostałeś fragment kodu i masz powiedzieć gdzie jest błąd: średnik na końcu to raczej nie, ale np., że powinno być większe równe, a nie większe gdzieś tam w IF-ie czy coś tego typu.

Natomiast myślę, że oprócz tego, dzisiaj zdecydowanie stawia się bardzo mocno na pracę w zespole. Choćby dlatego, że, nawet w małej firmie, bardzo rzadko będziesz miał jednego człowieka na projekt. Dzisiaj to już jest więcej niż jeden człowiek i umiejętność współpracy z

tym człowiekiem, podziału zadań, komunikacja między wami jest kluczowa do tego, żeby ten projekt się udał. Czy to jest wielka firma, 100 programistów tworzy jeden wielki system, czy mały projekcik, 2-3 osoby, to nadal jest zespół i nadal ten zespół musi umieć pracować i myślę, że jednak firmy mocno zwracają na to dzisiaj uwagę.

Jak już wywołałeś ten temat to chciałbym zapytać o to jak wyglądała sama forma pracy? Czy już wtedy mieliśmy metodyki zwinne, typu Scrum, system kontroli wersji, o czym też już trochę wspomniałeś oraz czy normalnością była praca zdalna?

Zabawne w sumie jest to, że moja pierwsza praca, kiedy rzeczywiście pracowałem w jakimś projekcie, który nie był czysto zleceniem, że mam zrobić jakąś jedną prostą stronkę, tylko taka praca na dość nieograniczony czas: *pracujmy, zobaczmy ile będziemy pracować nad tym projektem, coś tam stwórzmy.* To było to śledzenie tirów na Mapach Google z informacją ile paliwa jeszcze mają, czy im się coś nie zepsuło − były tam różne sensory i trzeba było to wyświetlić, żeby ktoś z centrali mógł sobie sprawdzać, gdzie jest cała jego flota samochodów w Europie. To było zabawne, że ta moja pierwsza praca była w 100% zdalnie. Byłem w Wielkiej Brytanii, bo pojechałem na wakacje i pracowałem w tej firmie, której siedziba była w Irlandii, więc była to dość zabawna koincydencja.

Natomiast później po tej pracy (pracowałem w niej 4 czy 5 miesięcy), zdecydowanie wszystkie kolejne prace „biurowe”, nie zlecenia, były jednak na zasadzie przychodzisz do pracy, logujesz się, więc

zaczynamy ci liczyć czas, że już jesteś w pracy (były też takie firmy) i nawet do tego stopnia, że nie, że przychodzę ze służbowym laptopem i sobie otwieram i pracuję i potem zamykam, biorę do domu i wieczorem jeszcze sobie siądę i skończę dwa unit testy. Tylko miałem stacjonarny komputer, jeden monitor, 15-17-calowy, a może 19 (nie pamiętam) i po prostu przychodziłem do mojego stanowiska pracy: zawsze to samo biurko, ten sam komputer i wtedy pracowałem. Wychodziłem z pracy, więc wychodziłem z pracy − nie było szansy, żebym cokolwiek zrobił, bo nie miałem żadnego dostępu do niczego. Miało to swoje wady i zalety. Wadą było to, że gdybym wieczorem wpadł na rozwiązanie: *o to tak muszę rozwiązać ten bug. Już teraz wiem* − co przecież często się zdarza − to nie mogłem tego zrobić. Z drugiej strony nie mogłem tego zrobić i to było zaletą, bo jak wychodziłem do domu, to wychodziłem do domu.

A jeśli chodzi o formę pracy typu Scrum?

W tych kilku firmach, w których pracowałem na samym początku swojej kariery w Gdańsku, mam wrażenie, że niektórzy próbowali twierdzić, że używanie Scruma czy Agile było raczej wymówką na to, że próbują w jakiś sposób ogarnąć chaos. Jednak czysto z punktu widzenia managementu, zarządzanie projektami, procesem, zarządzanie ludźmi, oczekiwaniami itd. raczkowało. Pracowałem wtedy w niemałych firmach w Gdańsku, które były zauważalne na rynku (niekoniecznie chciałbym podać ich nazwy). Zatrudniały one po kilkadziesiąt czy może nawet ponad sto ludzi i dostarczały dosyć ciekawe i dojrzałe narzędzia dla klientów. Z tego co wiem, teraz robią to już lepiej – na pewno wtedy się tego uczyli, a skoro utrzymali się na rynku to musieli się rozwinąć.

Natomiast wydaje mi się, że wtedy Scrum był jeszcze nowością i ludzie podchodzili do tego niechętnie. Poza tym nastąpiła też zmiana pokoleniowa, zmiana nastawienia, w ogóle mentalność zmieniła się bardzo przez ostatnie 20 lat. Tak teoretycznie mieliśmy Scruma i Waterfalla. Waterfall wyglądał tak, że najpierw naprodukujemy tej dokumentacji, potem ją przerzucimy przez mur do programistów, niech oni tam pół roku pracują i przyjdziemy spytać − *no gdzie mój program?* Okazało się, że to jest nieskuteczne, bo nawet jeśli wiesz dokładnie co chcesz osiągnąć, jeśli byś wiedział jaki program chcesz zbudować, to jest niezwykle trudno tak go opisać w dokumentacji, żeby ktoś, kto z tobą nie rozmawiał, albo nie rozmawia na co dzień, zbudował go tak jak ty chcesz. A to jest ekstremalnie optymistyczny scenariusz, że ty wiesz jaki program chcesz mieć. Bo najczęściej nie wiesz. Tak naprawdę.

I klient nie wie.

Tak, tzn. mówię do ciebie jako do klienta − gdybyś był moim klientem, to pewnie nie wiedziałbyś tak naprawdę, tobie się wydaje, że ty chcesz to i to, ale po tygodniu czy dniu rozmów na ten temat, okaże się, że miałeś na myśli zupełnie co innego. Ja miałem sytuację, gdzie robiliśmy jakiś projekt we współpracy z wieloma uczelniami i rozmawiałem z jedną osobą, rozmawiałem z drugą, trzecią osobą i miałem wrażenie, że z każdym się dogadałem, wszyscy wiedzą, po czym przyszliśmy razem pogadać i każdy myślał, że budujemy co innego. I to były bardzo kompetentne osoby, które były zaangażowane, które były chętne, żeby to robić − a nie zawsze tak jest w projekcie. Stąd Scrum, Agile czy Kanban − tam, gdzie angażujemy klienta na co dzień − ma tę wielką korzyść, że klient uczy się z nami, rozumie dlaczego coś się stało. Tak

jak jest z tym pomostem: z jednego punktu do drugiego, wiemy dlaczego jesteśmy tu i dlaczego jesteśmy tu. Wytłumaczenie klientowi dwóch lat projektu, wyjaśnienie dlaczego jesteśmy w danym miejscu, byłoby niewykonalne. A jeśli klient jest jednak regularnie z nami, to sam widzi dlaczego jesteśmy akurat tu, że po prostu jest to najbardziej racjonalne co mogliśmy zrobić.

SVN wtedy bywał − było różnie, czasami zmieniało się też jeszcze pliki na produkcji jakimś WinSCP. SVN zaczął być powszechny − widać było mocny trend, że: *projekty gdzie tego nie robimy są już przestarzałe − niech te projekty sobie umrą*, ale już we wszystkich nowych wszyscy go używali, bo widzieli już, że nie da się pracować w taki sposób. To było niewykonalne: skoro nie było SVN-a to nie było pull requestu, nie było przeglądu kodu, więc to od razu wpływało na jakość kodu. Jeśli wrzucało się coś ręcznie, to nie było żadnego pipeline'a, nie było automatyzacji, to zajmowało dużo czasu, więc albo release był raz na jakiś czas i był straszną męką: wchodziło się, kopiowało, instalowało, zawsze coś się zepsuło, coś nie działało, albo był ad hoc, ale tak bardzo ad hoc − wchodziło się i podmieniało się jeden plik za jeden, licząc na to, że nic się nie zepsuło. To były straszne rzeczy, ale pamiętajmy co nas wtedy otaczało. Dzisiaj siedzimy przy komputerze: *o, masz aktualizację: Zainstalować? OK - Zainstaluj*. A kiedyś trzeba było kupić jakieś pisemko, jakiś *Chip* czy *Komputer Świat,* czy coś innego, gdzie była łatka do Windowsa np. XP i można było sobie ją zainstalować, bo była umieszczona akurat na CD. Kiedyś była zupełnie inna bajka.

To ja może dodam, że widzę co najmniej jeden plus tego typu rozwiązania, bo przynajmniej trzeba było dopracować produkt, żeby

mógł on działać − nie dało się go łatwo zaktualizować to trzeba było go dopracować. Teraz wypuszcza się grę, która w ogóle nie działa, no ale mamy przecież fajnych klientów, którzy nam za nią zapłacą, a jeszcze przy okazji ją przetestują.

Myślę, że to jest prawda − wymuszało to w jakiś sposób większą troskę o to, co wypuszczamy. Jednak czasami też się okazuje, że coś akurat umknęło i wtedy był większy problem w załataniu tego, a jednocześnie np. ktoś, kto tego nie załatał, *bo mi dobrze pasuje, nie będę nic instalował* − mógł mieć np. poważne błędy bezpieczeństwa, więc mimo wszystko myślę, że dzisiejsze rozwiązanie jest trochę lepsze nawet z punktu widzenia użytkownika.

Idźmy dalej. Jakie narzędzia lub technologie już wtedy uznawano za przestarzałe, a jakie perspektywiczne? Jakie przewidywania się sprawdziły, a jakie były chybione?

Myślę, że już wtedy przestarzały był pewnie Turbo Pascal, choć był on popularny, bo dostępne były książki na ten temat, a możliwe, że nauczyciele na swoich studiach często uczyli się o Turbo Pascalu. Z jakiegoś powodu był on uznawany za język prosty. Nie wiem dlaczego Turbo Pascal miałby być szczególnie prostszy od np. Java czy nawet JavaScript.

Może był on łatwiejszy względem Assemblera

Na pewno − i dziurkowania kart performowanych.

Bo my to porównujemy do innych rzeczy, a pewnie ci co mówili o Turbo Pascalu to porównywali go do czegoś innego.

Z pewnością. Natomiast tutaj jeszcze coś wspomnę o Aplet Java. Nie wiem, czy pamiętasz coś takiego?

Pamiętam, pamiętam.

No właśnie, ale pewnie wielu słuchaczy nie pamięta i dobrze, że nie pamięta − gratuluję wam, bo to nie było coś, co jakoś szczególnie warto pamiętać. To były aplikacje Java, które po prostu były zagnieżdżone już na stronie, taki prostokącik. Był tam Aplet Java. To było z czasów, kiedy JavaScript był tak słaby, tak niewydajny we wszelkich pracach, plus przeglądarki były tak różne i tak niestandardowe, że po prostu czasem łatwiej było zrobić Aplet Java i go załadować. Jednak pisało się w jednym języku, załączało się to jako jakaś dziwna forma wtyczki na stronie, ale już bardzo długo tego nie widziałem. Nawet przygotowując się do naszego wywiadu, jak mówiłeś, że porozmawiamy o tym jakie technologie były dobre i złe, to sobie przypomniałem, że przecież były Aplety Javy. Wyparłem to.

Wydawało się, że Flash będzie perspektywiczny.

Może powiedzmy co to było, bo podejrzewałem, że większość osób w ogóle nie kojarzy.

To znowu były takie animacje, czyli wchodziło się na stronę i ona była tak naprawdę taką trochę prezentacją PowerPointa na wypasie. To były takie filmiki.

Interaktywny filmik − tak bym to chyba określił. Myślisz, że można?

To nie byłoby pewnie błędne. To były często grafiki wektorowe, więc one się nieźle zachowywały na różnych rozdzielczościach. Natomiast było to koszmarne do SEO, fatalne do Accessibility, o którym wtedy nikt nie myślał. W ogóle była to technologia firmy Adobe, ale jakoś umarła śmiercią naturalną i chyba mało kto za nią tęskni. Natomiast był tam ActionScript, który też był implementacją ECMAScriptu, czyli był poniekąd takim bratem czy siostrą JavaScriptu.

Dawno temu był też DHTML, jako Dynamiczny HTML, czyli HTML i JavaScript. Wydawał się on perspektywiczny i poniekąd to się sprawdziło, choć w trochę innej formie niż przewidywano.

XML, jak ktoś jeszcze kojarzy skrót AJAX − to był pierwotnie skrót. Potem stał się rzeczownikiem, bo wszyscy chcieli ukryć, że w nazwie tej technologii jest XML, bo to jest Asynchroniczna JavaScript i XML, a powinniśmy mówić raczej AJAJ, tj. asynchroniczny JavaScript i JSON, bo chyba dzisiaj głównie JSON zwracamy, jeśli ktoś w ogóle tego używa. Wtedy były akurat takie czasy, że XML był tak popularny, że wszystkie technologie musiały mieć w nazwie XML. Tak jak dzisiaj wszystkie technologie, nawet jeśli niekoniecznie mają AI, to są AI.

A myślisz, że można powiedzieć, że HTML to jest właśnie szczególny przypadek XML-a?

XHTML 1.1 był aplikacją XML-a, a XHTML 2.0, który się absolutnie nie uczył i był całkowitą ślepą uliczką, ponieważ wymagał tego, by kod był w pełni semantyczny, poniekąd tak jak kompilacja programu: jak w kodzie nie zamknęło się gdziekolwiek tagu czy cokolwiek innego się źle zrobiło, to on się po prostu wysypywał i nie chciał nic wyświetlić. I okazało się, że

biznes tego nie chce, że biznes dość mocno ignoruje fakt, że nie domknęło się jakiegoś tagu <p> czy bo co to za różnica? Klient kliknął, klient kupił, klient jest zadowolony − *co mnie interesuje, że tag był niedomknięty?*

Można by powiedzieć, że HTML całkowicie semantyczny mógłby być aplikacją XML-a − w XML-u były tagi, które można było samemu definiować. Ja troszkę będę bronił XML-a, bo tam były fajne rzeczy jak Schema i inne takie rzeczy, transformacje XML-a. To naprawdę było ciekawe, ale okazało się, że to jest bardzo ciekawe teoretycznie, a w praktyce nikt tego nie potrzebuje i są lepsze rzeczy.

Chciałbym się tutaj zatrzymać na chwilę, bo myślę, że na tym etapie, ludzie, którzy nas słuchają, pomyślą sobie, że jednak chyba wtedy nie było tak prosto, bo teraz wszyscy myślą, że kiedyś to był właśnie CSS, HTML, JavaScript. Wystarczyło się go nauczyć, dostawałeś pracę, ale jak teraz przeszliśmy przez te wszystkie technologie, to wcale nie było tego tak mało tylko po prostu pamiętamy już o tym końcu, a nie wiemy ile działo się po drodze.

Tak. Myślę, że to może być bardzo słuszne, a przecież my jeszcze nie poruszyliśmy tutaj wielu technologii, choćby takich, które były chybione: wspomniałem XHTML 2.0, Flash też się okazał chybiony, ale nie aż tak jak XHTML 2.0, ale mieliśmy również Google Web Toolkit. I kto pamięta o tym, że się pisało kod Javy, który był kompilowany do JavaScriptu? Był to zauważalny trend, a potem z tego trendu wyszło coś bardzo fajnego czyli TypeScript, gdzie TypeScript jest później również kompilowany do JavaScriptu − bo kompilacja to tylko tłumaczenie z jednego języka na drugi: czyli kompilacja np. C++ jest kompilacją do Assemblera i

Assembler jest już potem uruchamiany jako kod bitowy na procesorze. Nie musimy mieć tej kompilacji tylko i wyłącznie jako takiej ostatecznej, do języka zrozumianego przez procesor komputera.

Google Web Toolkit, choć był ślepą uliczką sam z siebie, to dla większej grupy programistów otwierał koncept tego, aby JavaScript został Assemblerem przeglądarki. I tutaj troszkę się pochwalę, bo w mojej pracy magisterskiej, w pierwszych dwóch rozdziałach, które wydałem, dziś nazwałbym to ebookiem, ale wtedy był to po prostu PDF (nazywał się *JavaScript na poważnie*) i w ostatnim podrozdziale mam „Assemblaryzacja JavaScript" gdzie przewiduję to, co stało się z JavaScriptem dzisiaj. To nie było tak strasznie trudne do przewidzenia, ale w 2011 roku mogło to nie być tak oczywiste.

Jeśli chodzi o te technologie, myślę, że to wszystko. Warto tu zauważyć, że wygrywały te technologie, które były dość łatwe do nauczenia się: czyli np. XML − on sam w sobie miał naprawdę fajne funkcje, ale był dość trudny tj. nie XML sam w sobie, ale wszystko dookoła. Te Schema, te − to nie było tak banalne i tak łatwe do zrozumienia jak JSON, który wysyłasz za pomocą AJAX-a i potem jQuery i sobie coś tam z nim robisz. Choć jak tak spojrzy na to jakiś naukowiec: *że jak to, a tutaj w ogóle nie walidujesz tego schematu, raz wyślesz to, raz wyślesz tamto.* Jednak zauważmy, że z jednej strony były i nadal są języki, gdzie typowanie jest statyczne i dynamiczne, jak JavaScript, ale JavaScript poszła w stronę Typescriptu, gdzie jednak definiujesz już typy i jednocześnie np. w Java też jest teraz słówko kluczowe var i nie interesuje cię jaka tam jest zmienna, bo ona będzie ci potrzebna tu i po

prostu w trakcie kompilacji będzie Przypisz typ (jaki uważasz za najlepszy).

Te technologie zaczynają się więc coraz bardziej schodzić do siebie. Był JavaScript, który miał domknięcia lambdy − to był język funkcyjny, ale on poszedł w stronę języka obiektowego, a jednocześnie też Java wprowadziła lambdy tj. funkcje, które sobie definiujesz, które coś robią, zamiast robić cały obiekt, klasę. Widać, że te technologie podpatrują od siebie nawzajem i adoptują te najfajniejsze rzeczy, bo czemu nie? Ja pamiętam, ile przecież było kiedyś walk o to, że JavaScript nie ma klas, po czym któraś wersja chyba JavaScript 6, nagle ciach, jednak są klasy. Przecież dla ewangelistów tego czystego JavaScriptu to musiał być nóż w plecy − do dzisiaj muszą się z tego zbierać.

Znowu − należy się przyzwyczaić, że technologia jest narzędziem i technologii używamy po to, żeby rozwiązywać problemy. Technologia nie jest celem samym w sobie i należy podchodzić do niej pragmatycznie. Jeśli jakaś technologia jest lepsza − używajmy tej lepszej, a jeśli jeszcze w innej technologii jest coś fajnego, to może twórcy tej technologii, której my używamy, powinni w następnej wersji po prostu wprowadzić parę fajnych usprawnień. Java wiele rzeczy sobie wprowadza z innych języków. Inne języki, jak np. JavaScript, idą w kierunku TypeScripta, czy też nawet taki zwykły JavaScript, one coraz bardziej idą w kierunku takich rzeczy. Tutaj znowu − każda z tych rzeczy wymusza to, aby nie tylko łatwiej programować, ale łatwiej zarządzać całym kodem – modularyzacja. To nie jest jeden wielki spaghetti code, tylko moduły. Ten moduł łatwo przetestować i niemalże wręcz sam silnik wymusza na tobie, że w tym miejscu będziesz miał testy.

To jeszcze wracając do tych technologii, nie wspomnieliśmy o jQuery. Dzisiaj mówiliśmy już o tym, ale trzeba wspomnieć, że to było bardzo doceniana biblioteka i wszyscy jej używali (tak mi się przynajmniej wydaje). Oczywiście na tle późniejszych lat, to już nie było takie istotne, ale warto dodać, że ostatnio wyszedł release. Nie sprawdzałem, co tam jest, ale może będzie to takie odświeżenie jQuery. Może w ogóle warto zajrzeć tam choćby po to, żeby zobaczyć jak to kiedyś wyglądało.

Z sentymentu, a dla osób młodych, które zajęły się tym niedawno i nigdy nie słyszały jQuery, żeby zobaczyć, jak świat wyglądał dawno, dawno temu. jQuery było niezwykłym krokiem, bo odpowiadało na wielkie zapotrzebowanie nałożenia interfejsu na wstrętną implementację JavaScriptową. W olbrzymim skrócie: co potrzebujesz? Show/hide, fade in/fade out i jeszcze parę innych rzeczy. To były najczęściej używane rzeczy. Potem jakoś banalnie można było robić własne wtyczki, że ciach-ciach − dodaj do tego elementu takie działanie.

To może właśnie dodajmy − żeby w JavaScripcie napisać ukrywanie i pokazywanie danego elementu, to to nie była jedna linijka − jak się chciało wesprzeć każdą przeglądarkę, która była popularna − to było kilkanaście, jak nie kilkadziesiąt linii kodu, więc jQuery to ułatwiał, bo pozwalał to zrobić w jednej linii. To naprawdę było bardzo dużo ułatwienie.

Tak, tu się w pełni zgadzam.

Zapytam tutaj od razu o największe przełomowe wydarzenia w ciągu każdych kolejnych pięciu lat twojej pracy w branży. Jakie to były wydarzenia twoim zdaniem?

To jest moja subiektywna lista i prawdopodobnie im później, tym trzeba by było mieć coraz mniejszy okres czasu. Myślę, że około 2005 roku było to upowszechnienie się AJAX-a: choćby Gmail, Facebook. Internet, który znamy dzisiaj istnieje dlatego, że istnieje AJAX. Czyli to, że klikam i strona nie musi mi się odświeżyć cała, żeby coś się na niej zmieniło − bo kiedyś każda strona odświeżała się zawsze cała. Sprawiało to dużo problemów, przenosiło do innego punktu, było nieprzydatne, niepotrzebne, generowało dużo ruchu.

Ja może tylko dodam, bo to chyba będzie dość istotne, że kiedyś jak kupowało się serwer to istoty były transfer i miałeś ograniczony transfer. Teraz chyba w ogóle nikt o tym nie myśli, ale wtedy było tak, że liczyło się to, żeby przypadkiem nie przekroczyć tego transferu.

Dokładnie. Dzisiaj akurat to się liczy z trochę innych powodów, bo ten transfer, czy ilość danych składowanych gdzieś tam na dyskach, czy też ilość użytego CPU i RAM-u przelicza się na emisję CO2. Polska, mimo fajnej lokalizacji geograficznej (bo wokół nas są duże i ludne narody i pewnie chętnie by u nas mogli coś mieć) − z uwagi na wysokoemisyjną energetykę − może tracić. Natomiast wraca to trochę. I cały ten trend nazywa się ogólnie Green Software i w AI-u (bo AI też używa olbrzymich ilości energii i olbrzymie ilości procesorów i olbrzymie ilości wody do schładzania wszystkiego) to się coraz mocniej bierze pod uwagę, ale z innego powodu: właśnie z takiego problemu gdzieś na wyższym

poziomie, a nie na tym takim moim konkretnym poziomie, że *ja mam swój tutaj hosting i będę musiał zapłacić 5 złotych więcej*.

Powiedzieliśmy o AJAX – potem był 2010 rok i tutaj powiedziałbym, że takie dwie największe przełomowe rzeczy, z mojej perspektywy, to frameworki JavaScriptu i jakiś wstęp do pracy zespołowej w stylu SVN/Git.

Około 2015: chmura − wprowadzanie już powszechne, upowszechnianie się świadomości, że to w ogóle istnieje, unit testy, inne testy, Selenium, automatyzacja całego procesu, CI-CD pipeliny, buildy, pull requesty – wszystko, co rozumiemy dzisiaj jako dobry QA i dobre zarządzanie plus Cloud.

W 2020 roku to była praca zdalna, bo był COVID. I myślę, że to był taki wielki boom, który sprawił, że okazało się wtedy, że praca zdalna to jest rozsądne, wykonalne i wyniki ekonomiczne są niezłe. Choć akurat chciałbym powiedzieć, bo pewnie słuchają nas często osoby, które zaczynają swoją karierę lub chcą zacząć swoją karierę − moja rada i to taka naprawdę szczera: idźcie do biura, poznajcie tych ludzi, uściśnijcie dłoń, wypijcie razem kawę.

To ja może tylko dopowiem: zorientujcie się, jak wygląda sytuacja, bo już nie raz słyszałem, że takie osoby nowo zatrudnione szły do biura, okazało się, że w pracy wszyscy seniorzy i midzi i są poza pracą. Co za różnica, że jesteś na miejscu, jeśli wszyscy inni są poza.

To prawda, ale mimo wszystko zawsze jest to jakaś szansa. Natomiast uważam, że jak się ma dużo juniorów (to jest taka moja prywatna opinia)

to praca seniora nie polega na tym, że senior będzie dowoził 10 czy 15 punktów na sprint − nieważne ile, ileś tam zadań na sprint. Oczywiście on ma dowozić, to jest prawda, ale senior jest od tego, żeby juniorzy umieli dowozić − to, że senior sobie dowiózł 10 punktów, a 3 czy 2 juniorów czy mid (ktokolwiek inny w zespole) nie był w stanie dowieźć swojego zadania, to to jest pytanie do seniora: a gdzie byłeś? Nie rozmawiamy dzisiaj o roli seniora w zespole, ale można tak napomknąć: dobry senior to nie jest ktoś, kto po prostu pisze bardzo dużo kodu. On ma też naparzać kod (tak kolokwialnie mówiąc), on ma to umieć. Jak trzeba, to ma być komandosem, że *o kurde, na produkcji się coś stanie − dobra, z nikim nie rozmawiam, daj ten komputer, przynieś mi coś do picia i jedziemy* − 3 godziny i naprawione. I on ma to rozumieć i umieć to zrobić.

To jest właśnie ta zmiana, która nastąpiła: senior ma być jednak człowiekiem, którego dużą częścią pracy jest to, że reszta mniej doświadczonego zespołu jest w stanie pracować. Dobry senior sprawi, że on nie będzie dostarczał 10 punktów co sprint, tylko że w pierwszym sprincie on dostarczy 2 punkty, ale 3 inne osoby w zespole też dostarczą po 2 punkty. W drugim sprincie on dostarczy 3 punkty, a oni dostarczą po 5 punktów (albo dwóch 5 punktów, a jeden 3 punkty). A w trzecim sprincie on dostarczy 5, tamci też 5. W czwartym sprincie on dostarczy 10, a tamci po 8. I to jest zadanie dla seniora. To jest rola seniora.

Jeśli ktoś jest w firmie, w której w ogóle nie czuje takiego wsparcia od seniorów (ja rozumiem, że sytuacja dzisiaj jest trudna i że pewnie nikt nie będzie aż tak bardzo kaprysił) to byłaby dla mnie taka żółta lampka, że coś jest nie tak. Wtedy dałbym np. taki przyjazny feedback, nie od

razu jakiś mocny, że *ooo, olaboga, krzywdzą mnie*, ale można porozmawiać ze swoim Team Leadem, Tech Leadem: *słuchajcie, ja bym chciał się rozwijać, jestem początkującym, wiecie, że jestem początkującym, bo zatrudniliście mnie jako trainee − może ustalmy, że raz w tygodniu chociaż jeden z seniorów będzie w biurze, usiądźmy razem, jak miałbym pytanie, żebym mógł je zadać.* Słyszałem np. takie coś, że firmy się tym chwaliły (ja nie mam na to opinii, ale mam pewne przeczucia), że one zwiększyły wydajność od kiedy junior, zanim spyta seniora, musi zadać to samo pytanie ChatGPT. To nie jest głupie rozwiązanie, ale gdzie tu jest team building? Gdzie tu jest ta wiedza ekspercka seniora, którą on miałby przekazać? Na czym ma polegać ta „seniorowatość" seniora? Ten senior naprawdę powinien być mentorem, tutorem, kolegą, który jest fajny, sympatyczny, z kim się dobrze pracuje. Czasami może być tak, że zwróci ci uwagę, że *no słuchaj może już wystarczy tych piłkarzyków, tak? Chodź idziemy do pracy, tak? Trzeba dostarczyć.* I oczywiście jest czas na dostarczanie, ważny moment, dzień przed releasem i jest czas na naukę − trzeba to jakoś wyważyć.

Naprawdę uważam, że to jest jedna z takich dużych zmian, które nastąpiły, że trochę inaczej definiuje się role, bo potem już TA, Solution Architecture jeszcze wyżej − oni znowu mają inną rolę. To nie jest tak, że masz po prostu być jeszcze lepszy, jeszcze więcej kodu dostarczyć. To nie może być tak, że jesteś juniorem i sobie tam pracujesz. Jesteś midem i sobie pracujesz już trochę lepiej, sprawniej, bez pomocy. Jesteś seniorem, to po prostu naparzasz ile się da, a potem nagle jesteś TA i nagle masz być „team working" i w ogóle − te twoje umiejętności muszą rosnąć razem z tobą.

To jak już wywołałeś temat zmian, to chciałbym cię zapytać o to, w jakich aspektach w ogóle cała branża IT zmieniła się najbardziej w przeciągu tych 15 lat, a gdzie nic się nie zmieniło?

Myślę, że najbardziej zmieniło się chyba to, że zespoły są dużo bardziej różnorodne − dawniej było po prostu np. czterech programistów i to był zespół. Teraz mamy czterech programistów, BA, QA, UX-a, Scrum Mastera (czy Agile Coacha − zwał jak zwał). Czasem może się pojawić kilka innych osób, bo okazało się, że aby dostarczać coś o odpowiednim poziomie jakości, jeśli nie jest to tak strasznie powtarzalne − jak stawiasz stronki na WordPressie, to nie trzeba, to jest inna bajka − ale jak budujesz zupełnie nowy system, który ma być potem wykorzystywany przez setki tysięcy czy miliony ludzi, to się okazuje, że przyda się jakiś DevOps, który zajmie się tym, aby to było od początku dobrze zrobione na poziomie chmury, buildów, automatyzacji itd. Przyda się jakiś UX, żeby ten front end wyglądał – nie, żeby był już zaprogramowany, ale żeby w ogóle był pomysł jak on ma wyglądać, jak to ma działać. Potem potrzebni są programiści, którzy to oprogramują. Programista, czy też inżynier oprogramowania jest jednym z wielu „klocków” w tej całej układance, aby było to zakończone sukcesem.

Nawet gdybyś miał najlepszych programistów, jakich możesz sobie wyobrazić, i byłby projekt z 20 programistami, ale nie byłoby BA, nie byłoby testera, nie byłoby UX-a, to sądzisz, że w rezultacie byłby to dobrej jakości produkt? Któryś z nich finalnie musiałby się stać QA-em albo któryś z nich musiałby się stać UX-em, żeby to jednak szło w jednym kierunku: że ktoś zaplanował jak to ma działać, a nie, że siada

dobry programista i myśli: *aha, to ma być button, no to ja dam zielony button po lewej, a drugi, o, ma być button, to ja dam czarny button po prawej, nie?* Tak proste rzeczy. Ktoś to musi zrobić, to się samo nie zrobi.

Zdecydowanie tak, jak opowiadałeś o tym, to wyobraziłem sobie to, jak kiedyś Steve Jobs prezentował swoje pierwsze produkty – to też wyglądało całkiem inaczej: ludzie sami lutowali sobie komputery, sami ogarniali wszystko w terminalu i można to w tą stronę zrobić i pewnie ci programiści by tak zrobili, ale to nie byłoby użyteczne dla wszystkich.

Prawda jest taka, że Inclusive Design, jaki mamy w IT sprawia, że po prostu poszerza nam się grupa klientów. Teraz nie chciałbym kłamać, ale na jednej konferencji, na której niedawno byłem, na jednej z prezentacji było podane, że globalnie, rozporządzalny budżet dla ludzi z niepełnosprawnościami − to mogą być różne niepełnosprawności, słaby wzrok albo w ogóle ktoś nie widzi, może ktoś nie ma dłoni, albo ma złamaną rękę, to też jest jakaś tymczasowa niepełnosprawność itd. − to chyba 13 bilionów dolarów − to są kosmiczne pieniądze w skali globu. Co by się stało, jeśli uznalibyśmy, że *nie, to my będziemy robić tak jak każdy sobie chce*?

Pomyśl sobie, że gdybyśmy nie brali pod uwagę potrzeb klienta w IT, to dzisiaj Tinder byłby przez linię poleceń. Wyobraź sobie Tindera przez linię poleceń – niewykonalne, ale takie aplikacje jak Tinder, jak jakieś tam zamawianie jedzenia, szachy, że masz na komórce szachy, możesz sobie pograć, istnieją dlatego, że potrzeby klienta, potrzeby użytkownika stawiano bardzo wysoko i niekoniecznie były to potrzeby programisty.

Programista najchętniej pokodowałby sobie coś, żeby np. było 7 parametrów, a jak nie podasz jednego, to w ogóle usunie ci wszystko z Dysku C.

To może czas powiedzieć, co się nie zmieniło. Czy jest coś, co się nie zmieniło?

O, to jest bardzo dobre pytanie. Chyba wszystko się trochę zmieniło, mniej lub bardziej. Nadal mamy jakieś języki programowania i mimo ewolucji nie ma tam rewolucji − gdybyśmy wzięli to, co było 15 lat temu i porównali do tego, co jest dzisiaj, to to byłaby forma rewolucji. Jeśli jednak prześledziliśmy sobie wszystko kroczek po kroczku, to jednak czujemy, że każda ta kolejna technologia była pomostem z wczoraj do jutra np. wirtualizacja − to była droga z hostingu do chmury. Potem w chmurze mamy Serverless i w ogóle okazuje się, że jest jeszcze łatwiej.

Najpierw JavaScript pisany inline, gdzie popadło, potem jednak jakoś zbierany w ramach jQuery, potem jakiś inny cały framework, który ci buduje całą stronę i jest to już podzielone na moduły i łatwo jest to wyekstrachować, podzielić na dwa projekty, że w jednym i w drugim projekcie używasz np. tego samego walidatora itd. Nie widziałbym tutaj wielkich rewolucji − z jednego punktu do drugiego była rewolucja, ale w tak długiej perspektywie to była rewolucja, ale nadal ciągle mamy tam pod spodem technologię i to jest ciągle jakiś kod i ten kod można ciągle porównać. Ktoś, kto nauczył się programować 15 lat temu, spojrzałby na dzisiejszy kod, na jakąś konkretną jedną funkcję i mógłby uznać, że ją rozumie. Spojrzałby na cały projekt i przeraziłby się, bo to się zmieniło

ekstremalnie. Myślę jednak, że ten kod, jak spojrzymy na niego niskopoziomowo, to on nie zmienił się tak drastycznie. Mamy tam: *.forEach* zamiast *for*, ale to nie jest jakoś tak strasznie nie do ogarnięcia, żeby nie zrozumiał, co się dzieje.

W takim razie patrząc na to wszystko co sobie powiedzieliśmy − czy twoim zdaniem można uznać, że to, co było, jest łatwiejsze niż to, co jest teraz? Czy obecnie próg wejścia do branży jest dużo bardziej wyśrubowany?

Myślę, że dzisiaj jest dużo więcej rzeczy, ale też z uwagi na to, że jest tak duży podział, to można się np. wyspecjalizować w jednej rzeczy. Osobiście uważam, że jeśli ktoś chciałby spoza branży wejść do branży IT, to chyba dobrą opcją jest opcja testera, a szczególnie testera ręcznego (chociaż jest już chyba bardzo mało ofert dla juniorów testerów ręcznych), ale opcja testera byłaby prawdopodobnie jedną z łatwiejszych opcji. Natomiast, można się wyspecjalizować w Accessibility, można się wyspecjalizować w UX-ie, można się wyspecjalizować np. w aspektach chmurowych i być specjalistą od tego. Mimo że to spektrum jest dzisiaj wielkie, to już rzadko oczekuje się, że będziesz ekspertem czy guru w każdej kwestii, tzn. powinno się coś tam rozumieć w każdej kwestii: masz już kod, wchodzisz i wiesz, że zmienię coś tu czy tu, to zmieni się to i to. Jestem w stanie ogarnąć istniejący kod, dodać coś nowego, ale niekoniecznie muszę być w tym ekspertem. Natomiast trzeba to jakoś rozróżnić, ale tu mówimy już o jakimś seniorze czy architekcie.

Dla początkujących − myślę, że po prostu trzeba bardziej przeanalizować swoje mocne i słabe strony: czy chcę być back endem, czy bardziej na front endzie, czy na front endzie interesuje mnie bardziej design czy może jednak programowanie, ale np. design, ale jednak będę programował, bo to jest wtedy duża wartość: mam fajnego designera/designerkę − on fajnie to robi, testuje z klientami, bo jest sympatycznym człowiekiem, ale też profesjonalnym i kompetentnym (samo bycie sympatycznym nic nie daje − to jest podstawa do wszystkiego innego). Te umiejętności są przydatne, natomiast dodatkowo jest jeszcze w stanie przerobić swój projekt na jakieś partiale czy widgety i programiści dostają już jakiś fragment kodu, który jest całkiem przyzwoity, poprawią go w 5 minut czy dorzucą kilka rzeczy i mają gotowca − to byłoby ciekawe. Zobaczmy, że tutaj nie musisz mieć wiedzy o chmurach, o cyber security (o którym jeszcze prawie nie powiedzieliśmy), o inżynierii danych, bazach danych itd., więc można sobie wybrać wąski pasek w swojej specjalizacji.

Mamy np. T-shape i uważam, że to jest bardzo słuszne, żeby pójść trochę szerzej − będę jednak wiedział trochę więcej niż tylko i wyłącznie UX: UX i Accessibility i może np. aspekty związane z Green Software. Tu polecam choćby darmowy kurs na Linuxie: *Green Software for Practitioners*. Potem pochwalcie się, że zrobiliście coś takiego na LinkedInie. Przeczytajcie też książkę o Sustainability Software, którą niedawno napisał mój kolega z zespołu, Marc Nevin wraz z całą grupą ludzi z Kainosa. Książka opowiada o tym, jak to robić i co to w ogóle oznacza. Można szukać swojej niszy nawet w takim obszarze − jestem np. Designerem, ale umiem trochę programować, a jeszcze oprócz tego jestem specjalistą od rozwiązań, które będą korzystniejsze dla planety i

sobie dasz taki listek na stronie, że jesteś super eko − to ma dzisiaj jakąś wartość. Nie mówiąc o tym, że to jest akurat słuszny kierunek rozwoju technologii, bo ta energetyka zaczyna być bardzo kluczowa również dla nas.

To ja bym może jeszcze tylko dopowiedział do tego co powiedziałeś: często jest tak, że nawet jeżeli zajmujemy się jakąś technologią, to często specjalizujemy się w jakiejś tam działce. To nie jest tak, że wszystko będziemy umieć − ktoś zajmuje się streamingiem, ktoś zajmuje się czymś innym i on będzie się na tym znał. Ktoś zada mu pytanie z trochę innej działki i on już nie będzie wiedział, bo nie da się tego wszystkiego ogarnąć. Mam wrażenie, że często osoby początkujące myślą w ten sposób, że *ja muszę wiedzieć wszystko, żeby w ogóle móc coś robić*.

Pytałeś też o to, czy dziś jest łatwiej, czy trudniej − myślę, że łatwiejsze jest to, że dziś po jakichś kursach czy po skończonej uczelni można pójść do jakiejś firmy, która na początek organizuje ci swoje własne szkolenia i taki bootcamp. Mniej fair, moim zdaniem, jest to, kiedy ktoś bierze 30 osób i mówi: *bijcie się*. Ja mam co do tego wątpliwości etyczne. My robimy to trochę inaczej − jeśli już ktoś zakwalifikuje się na naszą akademię Kainosa (jest to trudniejsze, żeby się zakwalifikować) to jest już naszym pracownikiem od pierwszego dnia. I ma uczyć się współpracować od pierwszego dnia, bo to jest kluczowe, a nie to czy będzie on pisał kod troszkę szybciej od kogoś innego i nie będzie komuś pomagał, bo *broń Boże ten ktoś będzie wtedy lepszy ode mnie*. Uważam, że to jest strasznie toksyczny pomysł, żeby robić coś takiego, że ludzie współzawodniczą ze sobą i mają tak spędzić razem miesiąc.

Nie wiem jak ci ludzie to psychicznie znoszą. Myślę, że to musi być naprawdę trudne.

Natomiast pod tym względem byłoby może łatwiej, bo idziesz już do jakiejś firmy i oni ci już pokazują dokładnie co od ciebie oczekują, więc na dwoje babka wróżyła. Ja bym proponował, żeby się rozwijać, żeby robić to − najfajniej jak ktoś to lubi i jak ma do tego trochę talentu, to naprawdę samo będzie wychodzić.

Nie bójcie się na samym początku robić rzeczy nawet trochę za darmo, np. ciocia ma wypożyczalnię wózków widłowych − zróbcie jej stronę internetową za darmo. Będziecie mogli wpisać coś sobie w CV i będziecie mieli okazję z tym klientem trochę porozmawiać, porozumieć się, pomyśleć jak to zrobić. To jest naprawdę wartościowe.

I czekać na materiały cały miesiąc i to wcale nie jest takie hop-siup.

Dokładnie, to jest właśnie to. Myślę, żeby nie negować wartości takich drobnych w cudzysłowie zleceń, bo to nawet nie musi być za pieniądze.

A na końcu usłyszeć: ***aaa, bo wiem, że dla ciebie Mateusz to jest 5 sekund, to ja bym chciał jednak ten element, jednak w jakimś innym miejscu. Może już teraz możesz mi to zrobić?***

Dokładnie, tak też jest. Natomiast to można nadal robić. Chociaż dzisiaj też jest mniej tego typu zleceń, bo ktoś sobie po prostu robi stronę na Facebooku czy Instagramie. Dzisiaj świat jest trochę inny, ale myślę, żeby nie bać się próbować takich rzeczy. Po prostu próbować, robić, uczyć się, czerpać z tego frajdę, bo pamiętajcie też o tym: jeśli robienie takich rzeczy, póki nie jesteście zatrudnieni, to dla was męka i nie

chcecie tego robić, to rozważcie czy pewno IT jest dla was kierunkiem, bo nie będzie łatwiej. Musicie czuć z tego zajawkę i przyjemność.

Jeszcze odnośnie do tego, co powiedziałeś o tym, że teraz ktoś ma stronę na Facebooku czy na Instagramie i nie potrzebuje już innej − zauważyłem ostatnio taką ciekawostkę, nawet nie wiedziałem o tym, że w OBS-ie można ustawiać pozycje, wygląd różnych elementów przy pomocy CSS-a i np. nie oferuj stron, ale możesz oferować templatki pod OBS-a, bo on jest teraz popularny. Może są więc miejsca, o których nie wiemy, bo nie bawimy się tam, ale można też oferować usługi wykorzystując technologię, którą znamy.

Tak sądzę. Nie chciałbym też tutaj robić z siebie specjalisty od tego, jak zaczynać w branży IT. Ja po prostu proponowałbym przyjść do Kainosa. Akurat fajnie szkolimy początkujących. Natomiast na pewno ważna jest każda forma aktywności i proaktywności i to, żeby ludzie widzieli, że chcecie, lubicie i to jest coś, co daje wam radość i przyjemność, to zawsze będzie w cenie i to zawsze będzie dobre. Bardzo ważna jest jedna rzecz − nieważne jak w przyszłości będzie wyglądać programowanie i branża, jeśli w ogóle będzie jakakolwiek praca dla ludzi, to praca zespołowa będzie podstawą.

Jak już wywołałeś ten temat to, podsumowując całą naszą rozmowę − jaką przyszłość przewidujesz dla branży IT?

Dobrą − wydaje mi się, że aktualnie rzeczywiście sytuacja na rynku jest, jaka jest. Natomiast pamiętajmy o tym, że nawet jeśli Gen AI będzie na

takim poziomie, na którym wiele rzeczy będzie automatyzowanych i tworzonych wprost, to jednak nadal trzeba odpowiednio zadać pytanie temu Gen AI i to pytanie musi być potem zweryfikowane, sprawdzone. Ja wierzę, że to będzie działało jak taki super framework, gdzie 90% rzeczy jest robionych błyskawicznie, ale 10% rzeczy − przez to, że jest to skonstruowane tak, a nie inaczej − będzie bardzo trudnych do zrobienia, bardziej skomplikowanych, wymagających podejścia. Ktoś będzie jednak musiał działać z tym klientem, ktoś będzie musiał to przetestować, ktoś będzie musiał się tym jakoś opiekować, zająć.

Dodatkowo, jednak klienci nie przychodzą do firm informatycznych i nie mówią: *słuchajcie, mam tutaj tyle pieniędzy, zróbmy to*. Gdy zostało zrobione to co klient chciał zrobić, w danym czasie i budżecie jaki sobie określili i on mówi: *dobra, ja już więcej nic nie chcę do tej aplikacji − to jest dokładnie to, czego chciałem*. Nie, bo apetyt rośnie w miarę jedzenia. Spójrzmy na to, jakie robiliśmy aplikacje 10 czy 15 lat temu przy tamtych technologiach, a jakie aplikacje robimy dzisiaj, o ile więcej jest tam funkcji. Gdy po prostu będzie jeszcze łatwiej dostarczać te funkcje, to klient powie: *to ja jeszcze chcę to, to, to i to, bo ja miałem ten backlog jeszcze 100 zadań, tylko nie starczyło mi budżetu* − bo częściej kończy się budżet, a nie backlog.

Myślę, że z pewnością będą duże oczekiwania do tego, żeby programiści czy też ludzie wytwarzający oprogramowanie (nie umiem powiedzieć czy za 15 lat to nadal ludzie będą pisać kod − może nie, a może tak, może częściowo) zwiększyli jakość i wydajność, bo te narzędzia są coraz lepsze, lepiej wspierające, lepiej integrujące się ze sobą i po prostu będzie oczekiwanie, że coś, co kiedyś mogło zająć 2

tygodnie, będzie zrobione w 1 dzień, ale wtedy: jutro robisz kolejne duże zadanie, pojutrze kolejne duże zadanie i po prostu dostarczasz dużo większy system, który będzie działał. Tutaj znowu wracamy do tego problemu, że ten dużo większy system będzie zużywał dużo więcej energii i musimy dbać o to, żeby nasza energetyka była lepsza, abyśmy nie wypadli z tego rynku i o to, żebyśmy my jako programiści myśleli o tym, że pewne rzeczy są trochę bardziej optymalne i zróbmy je w ten energetycznie optymalny sposób, bo bardzo możliwe, że wtedy te takty procesora będą po prostu główną walutą dla aplikacji. Skoro już wytworzenie aplikacji będzie tańsze, to jej uruchomienie i wytrenowanie pewnych modeli, czy tam cokolwiek będzie w tej aplikacji, będzie tym głównym kosztem dla aplikacji.

Jak opowiadałeś o tym koszcie wytworzenia w CO2, to zastanawiam się, czy może nie będzie tak, że będziemy mieli ograniczony czas korzystania np. właśnie z ChatGPT − tak jak wcześniej, gdy były modemy i mieliśmy jakieś limity to tutaj też tak będzie?

Bardzo ciekawa koncepcja. Myślę, że tak nie będzie, ale myślę, że ważny jest obszar Green AI, który też istnieje − i tutaj akurat zrobię reklamę wydarzenia, które będzie w Gdańsku. Będzie ono już po opublikowaniu tego podcastu, więc to będzie prawdziwa reklama: w Gdańskim Parku Naukowo-Technologicznym 16 maja w czwartek odbędzie się Eco-ON (łatwo pewnie znaleźć na LinkedInie czy w Google). Polecam, bo będzie tam omawiany aspekt Green AI, aspekty energetyczne i o tym, jak robić offset, więc wszystkie rzeczy, o których właśnie powiedziałaś, będą tam częściowo pokryte. Ja będę miał

przyjemność poprowadzić tam panel dyskusyjny − jakby ktoś chciał się ze mną spotkać, zbić piątaka to zapraszam. Będzie mi bardzo przyjemnie.

To ja jeszcze dopytam tylko o dwie rzeczy: czy w takim razie dla juniorów będzie miejsce w IT? Druga rzecz to pytanie o to, czy trend może się odwrócić − może się okazać, że faktycznie sztuczna inteligencja nie jest jednak taka idealna i to zapotrzebowanie na programistów jest większe, niż firmy wnioskują teraz: nagle sytuacja całkowicie się odwróci, znowu będzie potrzeba dużo większej ilości programistów niż jest na rynku i znowu rekruterzy będą ciągle dzwonić, pytać, bo coś się dzieje na rynku?

Przewidywanie przyszłości jest łatwe, robienie tego poprawnie jest trudną częścią, więc ja mogę tylko dać jakieś swoje pomysły i za 10 lat zobaczymy czy się sprawdziły. Sądzę, że jednak jakieś miejsce dla juniorów będzie choćby dlatego, że dochodzimy do czasu, gdzie wiele osób, które dzisiaj są seniorami zaczyna mieć już wiek nawet gdzieś pod 50-tkę i w tym momencie te osoby mogą stwierdzić z przyczyn finansowych, że są już samodzielne finansowo, stabilne i nie muszą w ogóle pracować, im się nie chce, albo będą awansować i po prostu już nie będzie osób, które do tej pory wykonywały te zadania. Siłą rzeczy, wydaje mi się, że będą osoby, których będzie ciągnąć do tych kolejnych ról.

Rozumiem, że seniorzy na drzewie nie rosną.

Seniorzy na drzewie nie rosną, dokładnie. Co więcej, ja bym tutaj widział potencjalną szansę, jak wspomniałeś tego Steve'a Jobsa − to pokolenie

programistów, którzy byli w pełni przystosowani do tego, że oni piszą kod i ten kod musi się uruchomić. Zobaczmy, jak bardzo rośnie teraz low-code, no-code. Ja nie oceniam, czy to jest dobre czy złe. Nie znam się na tym, ale widzę, że bardzo rośnie ten rynek i może się okazać, że wykorzystanie low-code, no-code i Gen AI w perspektywie 10 lat sprawi, że będziemy mieć dwa razy tylu ludzi zatrudnionych w wytwarzaniu oprogramowania, mimo, że 10% tych ludzi tak naprawdę schodzi pod maskę i pisze kod.

Zobaczmy, że dzisiaj, każdy kto chce być inżynierem AI prawdopodobnie myśli o tym, że chciałby trenować te modele, rozwijać to, trochę opracować to naukowo wręcz itd. Jednak, jak mamy inżyniera Reacta, czy też programistę Reacta, czy JavaScriptu, to nie znaczy, że ten ktoś rozwija JavaScript czy Reacta. On dostał narzędzie i korzysta z tego narzędzia. Moim zdaniem w przyszłości programiści, będą korzystać z modeli AI w bardzo podobny sposób, jak dzisiaj z różnych bibliotek czy frameworków: mam coś do zrobienia, biorę model AI, importuję, konfiguruję sobie w jakimś YAML-u i po prostu korzystam z BlackBoxa. Wrzucam coś i otrzymuję efekt. Gdzie np. mogłoby się to przydać? Na formularzu mamy dzisiaj tylko walidację, że wprowadziłeś dane czy nie, ale moglibyśmy np. mieć walidację tego czy twój komentarz jest konstruktywny, czy nie jest hejterski, czy nie promujesz nieprawdy, bo np. rozsiewasz fake newsy. To jest nieosiągalne za pomocą klasycznego programowania, a z wykorzystaniem mocy AI robisz zwykły modulik, który już jest wytrenowany. On jest małym modelem, który dokładnie rozumie co i jak. Ewentualnie czasem robi się update, żeby ogarnął nowe promowane fake newsy. Byłby on pierwszą warstwą zabezpieczenia przed tym, aby social media nie służyły do polaryzacji

społeczeństwa. Pamiętajmy, że dezinformacja i polaryzacja społeczeństwa według World Economic Forum w perspektywie najbliższych dwóch lat − top 3: to globalnie jest większe ryzyko dla świata, niż wojna kinetyczna. Widzimy, że naprawdę to są takie obszary, gdzie ktoś może nie być programistą per se w naszym rozumieniu, a może zmieniać w przyszłości świat składając klocki razem do siebie. Myślę, że będzie coraz więcej automatyzacji, coraz większe oderwanie się od warstwy niskopoziomowej (ale sami właśnie mówiliśmy, że się oderwaliśmy już przy jQuery). To jest po prostu kolejny, kolejny, kolejny krok. I tych kroków będziemy mieli coraz więcej, ale pewnie ciągle będzie trochę programistów, którzy będą ten kod pisać albo poprawiać choćby po Gen AI. Natomiast być może to jest wręcz szansa dla juniorów, a nie zagrożenie, bo oni mogą być lepsi, bo nie będą mieli tej naleciałości, którą mają dzisiaj seniorzy. Natomiast to jest pewne wróżenie z fusów. Ja wolałbym być optymistą, bo jeśli mamy być pesymistami, to tylko siądźmy i płaczmy.

No dobrze, to nie jesteśmy załamani − jest szansa, że jeszcze coś podziałamy i będziemy mogli nacieszyć się swoimi wynikami stukania w klawiaturę. Podsumowując, na sam koniec, może jesteś w stanie polecić książkę dla osób, które chcą lepiej poznać branżę IT?

Ja bym zaczął od takich rzeczy, które uznaję za biblię i za najbardziej wartościowe. One niekoniecznie będą czymś dla osób na ekstremalnie początkowym poziomie. Jak ktoś już np. czuje, że jest juniorem − nie, że zaczyna swoją karierę, chciałby być juniorem, ale że już jest juniorem −

to może mu się to przydać. Jak ktoś czuje, że jest np. całkiem solidnym midem, to powinien już te książki w miarę rozumieć. Chodzi o *Czysty kod. Podręcznik dobrego programisty*. Był moment, gdzie widziałem takie wymaganie w ogłoszeniach o pracę, że rozumiesz, o co chodzi w *Czystym kodzie*.

Ta książka już wielokrotnie się pojawiała.

Druga książka, do której mam sentyment to *JavaScript, The Good Parts,* którą napisał Doug Crockford. Pamiętam, że czytałem ją jako ten w miarę dobry programista JavaScriptu. Niektóre rozdziały musiałem przeczytać dwa czy trzy razy, żeby naprawdę zrozumieć co on pisze, ale to jest takie samo mięcho – książka jest rewelacyjna. Większość języków programowania to dziś Object Oriented Programming i warto sięgnąć po tę książkę choćby po to, żeby zobaczyć, że są w ogóle inne rzeczy. Natomiast jest to książka trudna, ale daje dużo przyjemności.

I dwie książki, które nie są wprost do programowania, ale technologia rozwinęła się tak bardzo, że coraz częściej aspekty etyczne, moralne zaczynają być naprawdę istotne w wytwarzaniu oprogramowania. Jedna z nich (niekonieczne jest ona o etyce): *Wielka wojna o chipy* − niezwykle wciągająca książka. Jakby ktoś mi powiedział, że książka o tym, jak się produkuje procesory i kto, kiedy i gdzie je produkował może być wciągająca jak dobry kryminał, to bym go wyśmiał, ale ja naprawdę nie wiem, co by się stało z moim życiem, jakbym taką książkę przeczytał w wieku 15 lat, bo może zostałbym elektronikiem. Ten świat jest niesamowity, a książka jest fantastyczna.

A druga książka to: *W trybach chaosu. Jak media społecznościowe przeprogramowały nasze umysły i nasz świat*. Tak jak tutaj pokazywaliśmy dobre przykłady tego pomostu z dzisiaj do jutro, to w tej książce pokazane jest, w jaki sposób, przypadkiem, sytuacja po sytuacji, social media skręcały w kierunku, w którym są dzisiaj, gdzie bardzo często jest to nastawione na polaryzację, na konflikt, na negatywne emocje, zamiast na pozytywne, bo te emocje sprawiają, że bardziej angażujesz się i spędzasz więcej czasu na platformie Niestety to jest biznes i ten biznes chce zarobić, a zarabia z reklam, więc smutna konstatacja: jak nie płacisz za usługę, za produkt, to ty jesteś produktem. Czyli np. w radiu, za które nie płacisz, produktem nie jest audycja, tylko produktem jesteś ty słuchając tej audycji, bo dzięki tobie oni sprzedają reklamy. I podobnie jest na social mediach. Nie chodzi to, że mam coś przeciwko social mediom. Jak się ich mądrze używa to są bardzo fajne, przydatne, rozwijające. Myślę, że warto przeczytać tę książkę, żeby uświadomić sobie pewne rzeczy. Choć jeśli ktoś jest wielkim fanem Trumpa, to może mu się trudno czytać tę książkę, bo widać ewidentnie, że autor go nie znosi (momentami nawet mi to przeszkadzało, mimo że mam dosyć neutralne poglądy, choć dobrze, że akurat jak był konflikt na Ukrainie, był ktoś, kto miał jednoznaczne stanowisko, co było ważne dla Polski).

To już tak na sam koniec. Gdzie możemy cię znaleźć w sieci, jeżeli ktoś będzie chciał o coś podpytać?

Zapraszam na LinkedIna. Tam najłatwiej mnie znaleźć. Jestem dość aktywny na LinkedInie − co najmniej raz w tygodniu coś tam wrzucam. Oprócz tego, jak ktoś jest w Gdańsku, to można mnie złapać w różnych

miejscach, szczególnie jak ktoś jest młodym studentem, który chciałby porobić coś fajnego na studiach i szuka fajnych projektów − mamy kilka różnych fajnych rzeczy, które robimy razem z uczelniami, czy też nawet niekoniecznie my robimy, ale np. ja wiem, że ta uczelnia coś takiego robi i wiem, że szuka fajnych ludzi − bo nie zawsze jakaś informacja się tak łatwo rozchodzi, jeśli niektóre uczelnie mają 20 000 studentów, 3 000 pracowników.

A jakby ktoś jeszcze chciał mnie może spotkać, to zapraszam na Quantum Summer School. Jeśli dla kogoś AI to już za mało, to porozmawiajmy o kwantach i technologiach kwantowych – pod koniec sierpnia będzie otwarta rejestracja dla osób młodych, początkujących na studiach czy uczniów ostatnich lat ogólniaka czy technikum. Jakby ktoś zajrzał do mnie na LinkedIna, to niedługo będziemy o tym pisać. 10 maja będzie już o tym jakaś informacja u mnie na wallu, na LinkedInie. Jeśli ktoś odezwie się do mnie na LinkedInie, to może znajdziemy też inne okazje do spotkania.

To może ja jeszcze tylko przypomnę, że jest jeszcze ten Eco-ON, 16 maja − dobrze pamiętam?

Tak, bardzo dziękuję. Oczywiście Eco-ON: Gdański Park Naukowo-Technologiczny. Myślę, że naprawdę bardzo fajne wydarzenie. Bardzo mocna ekipa mówców i, moim zdaniem, naprawdę coraz ważniejsze tematy. Technologia zaczyna być tak potężna, że tutaj parafrazując tego naukowca z Jurassic Parku: wszyscy nie mogą być dzisiaj tak bardzo zajęci, tym co możemy zrobić, bo możemy naprawdę bardzo dużo. Powinniśmy zastanowić się czasem czy powinniśmy coś robić i to naprawdę zaczyna być coraz poważniejsze zagadnienie w

technologii. Mamy dzisiaj taką moc, to jest taka armata, że trzeba nad nią panować.

To ja tylko tak podsumuję, że jak to powiedziałaś, to od razu skojarzyła mi się wypowiedź Billa Gatesa na temat AI.

A co on powiedział?

On też mówił, żebyśmy troszkę uważali, bo zaczyna się to robić niebezpieczne.

Nie on jeden, nie on jeden. Miejmy nadzieję, że nigdy nie stanie się to niebezpieczne i miejmy nadzieję, że będziemy umieli tego używać w sposób, który naprawdę jest korzystny dla ludzkości.

I że film, który pewnie większość z nas oglądała za młodu, przynajmniej nasze pokolenie, się nie sprawdzi.

Terminator?

Tak jest.

Tak, tak, dokładnie.

No dobrze, Patryku, to ja bardzo ci dziękuję za naszą dzisiejszą rozmowę i podzielenie się z nami swoimi doświadczeniami.

Wielkie dzięki, dziękuję za zaproszenie. Mam nadzieję, że nasi słuchacze (twoi słuchacze) również mieli z tej rozmowy przyjemność. Było mi tutaj bardzo przyjemnie. Dziękuję.

Mnie również. Dzięki, trzymaj się.