Peter Seibel

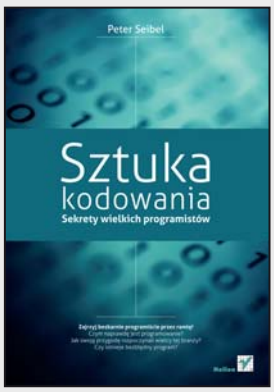
Sztuka kodowania

Sekrety wielkich programistów

Zajrzyj bezkarnie programiście przez ramię!
Czym naprawdę jest programowanie?
Jak swoją przygodę rozpoczynali wielcy tej branży?
Czy istnieje bezbłędny program?

Helion

Sztuka kodowania. Sekrety wielkich programistów



Autor: Peter Seibel
Tłumaczenie: Tomasz Walczak, Mikołaj Szczepaniak
ISBN: 978-83-246-2755-4
Tytuł oryginału: Coders at Work
Format: 168×237, stron: 416

Zajrzyj bezkarnie programiście przez ramię!

- Czym naprawdę jest programowanie?
- Jak swoją przygodę rozpoczynali wielcy tej branży?
- Czy istnieje bezbłędny program?

Programiści to tajemnicze osoby, które potrafią zmusić komputery do wykonywania karkołomnych zadań, wymyślanych przez zwykłych użytkowników. Spędzają cały dzień przed komputerem, a ich świat to monitor, klawiatura i hektolitry kawy. Czy to prawda? Na to pytanie odpowiada książka, którą trzymasz w rękach. Dzięki niej spojrzysz na ten zawód z zupełnie innego punktu widzenia. Jej autor przeprowadza wywiady z najbardziej rozpoznawalnymi osobami z tej branży. Mówią one o swoich pierwszych krokach w świecie programowania, opowiadają, jak nauczyły się swojego pierwszego języka oraz jak widzą tę gałąź wiedzy w przyszłości.

Czym naprawdę jest programowanie? Rzemiosłem, sztuką, a może nauką? Te pytania stawia autor we wprowadzeniu i ma nadzieję, że wypowiedzi jego gości choć trochę zbliżą go do znalezienia odpowiedzi. Ta unikalna książka pozwoli Ci poznać bliżej wybitne osoby, od lat związane z informatyką. Zagłębiając się w kolejne wypowiedzi, przekonasz się, jak często przypadek decyduje o sukcesie lub porażce.

W trakcie lektury będziesz mieć okazję zapoznać się z wypowiedziami takich sław, jak:

- Jamie Zawinski – wybitny programista Lisp, pracujący przy pierwszych wersjach przeglądarki Netscape
- Brad Fitzpatrick – najmłodsza osoba w gronie, programista „od zawsze"
- Douglas Crockford – starszy architekt JavaScript w Yahoo!; pomysłodawca formatu JSON
- Brendan Eich – twórca języka JavaScript
- Joshua Bloch – szef Java Architect w Google; w trakcie pracy w Sun Microsystem był kierownikiem zespołu projektującego i implementującego Java Collections Framework
- Joe Armstrong – autor języka programowania Erlang
- Simon Peyton Jones – rozpoczął projekt, którego efektem było powstanie języka Haskell
- Peter Norvig – dyrektor działu badań w Google, wcześniej pracujący dla NASA
- Guy Steele – znawca języków; Cobol, Fortran, PDP-10, Java, Haskell to tylko niektóre z jego repertuaru
- Dan Ingalls – współtwórca języka Smalltalk
- L. Peter Deutsch – programista od końca lat pięćdziesiątych; zaczynał w wieku jedenastu lat
- Ken Thompson – współtwórca systemu UNIX
- Fran Allen – przez czterdzieści pięć lat pracował dla firmy IBM; instruktor języka Fortran
- Bernie Cosell – współautor oprogramowania wykorzystywanego w pierwszych węzłach sieci ARPANET
- Donald Knuth – autor jedynego najprawdopodobniej bezbłędnego programu – TeX

Poznaj tajemny świat programistów, hakerów i wybitnych specjalistów!

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl

Spis treści

O autorze		**7**
Podziękowania		**9**
Wprowadzenie		**11**
Rozdział 1.	**Jamie Zawinski**	**13**
Rozdział 2.	**Brad Fitzpatrick**	**45**
Rozdział 3.	**Douglas Crockford**	**75**
Rozdział 4.	**Brendan Eich**	**101**
Rozdział 5.	**Joshua Bloch**	**123**
Rozdział 6.	**Joe Armstrong**	**147**
Rozdział 7.	**Simon Peyton Jones**	**169**
Rozdział 8.	**Peter Norvig**	**197**
Rozdział 9.	**Guy Steele**	**221**
Rozdział 10.	**Dan Ingalls**	**251**
Rozdział 11.	**L Peter Deutsch**	**277**
Rozdział 12.	**Ken Thompson**	**299**
Rozdział 13.	**Fran Allen**	**321**
Rozdział 14.	**Bernie Cosell**	**341**
Rozdział 15.	**Donald Knuth**	**369**
Dodatek	**Bibliografia**	**391**
Skorowidz		**395**

ROZDZIAŁ

8

Peter Norvig

Peter Norvig jest z natury człowiekiem o szerokich horyzontach i hakerem. Kiedyś napisał program do znajdowania w dziennikach wyszukiwania serwisu Google trzech kolejnych zapytań jednego użytkownika składających się na haiku (oto jedno z tych, które najbardziej utkwiły mi w pamięci: „java ECC / java elliptical curve / playboy faq").

Na swojej stronie Norvig udostępnia odnośniki do standardowych materiałów: napisanych przez siebie książek i prac, slajdów z wygłoszonych wykładów oraz różnych fragmentów kodu. Jednak oprócz tego znajdują się tam odsyłacze do tekstów opublikowanych w McSweeney's Quarterly Concern, dowcipny opis prac nad programem do generowania rekordowo długich palindromów i prezentacja Gettysburg Powerpoint Presentation *(jest to parodia programu PowerPoint Microsoft wspomniana przez Edwarda Tuftego i pojawiająca się na pierwszej stronie wyników po wpisaniu w wyszukiwarce Google hasła* PowerPoint*).*

Norvig jest obecnie dyrektorem do spraw badań w firmie Google. Wcześniej piastował stanowisko dyrektora do spraw jakości wyszukiwania. Przedtem kierował działem nauk obliczeniowych w ośrodku NASA Ames Research Center, a jeszcze wcześniej pracował w powstałym pod koniec lat dziewięćdziesiątych dwudziestego wieku internetowym startupie Junglee. W 2001 roku Norvig otrzymał nagrodę NASA Exceptional Achievement Award i jest członkiem organizacji American Association for Artificial Intelligence oraz Association for Computing Machinery.

Dzięki pracy w Google, NASA i Junglee Norvig ma doświadczenie w stosowaniu „hakerskich" i „inżynieryjnych" technik budowania oprogramowania, a w wywiadzie opowiada o zaletach oraz wadach obu tych podejść. Ponieważ jest byłym wykładowcą nauk komputerowych, a obecnie pracuje w jednej z największych na świecie firm zajmujących się oprogramowaniem, ma też ciekawy punkt widzenia na relacje między akademickimi naukami komputerowymi i praktyką ze świata przemysłu.

Oprócz tego, rozmawiamy, jak zmieniło się programowanie w ostatnich latach, dlaczego żadne techniki projektowania nie zastąpią wiedzy na temat wykonywanych zadań i dlaczego dla NASA korzystniejsze może być stosowanie bardziej zawodnego, ale tańszego oprogramowania.

Seibel: Kiedy nauczyłeś się programować?

Norvig: W szkole średniej. Szkoła miała komputer PDP-8, tak mi się wydaje, i zapisałem się na kurs. Zaczęliśmy od programowania w języku BASIC i to był mój początek.

Seibel: W którym to było roku?

Norvig: Skończyłem średnią szkołę 1974, dlatego musiał to być rok 1972 lub 1973. Pamiętam z tych czasów parę rzeczy. Przykładowo nauczycielkę, która próbowała nauczyć nas tasowania talii kart. Jej algorytm działał tak: użyj generatora liczb losowych do wyboru dwóch miejsc, a następnie zamień karty z tych pozycji miejscami; przechowuj wektor bitowy z informacją o przeniesionych kartach i kontynuuj proces do czasu zamiany wszystkich kart. Pamiętam, że pomyślałem wtedy: „To bez sensu. Musi to być najgłupsze rozwiązanie na świecie. Algorytm może działać w nieskończoność, ponieważ może znaleźć się jedna para, której nigdy nie wybierzemy". Nie wiedziałem wtedy, że wystarczy stwierdzić, iż złożoność to *n* kwadrat, choć powinna to być złożoność liniowa *n*. Wiedziałem jednak, że rozwiązanie jest po prostu złe. Potrafiłem wymyślić, jak mi się zdaje, algorytm Knutha, który zamieniał kartę zerową z pięćdziesiątą drugą, potem zerową z pięćdziesiątą pierwszą itd. — daje to złożoność liniową *n*. Pamiętam, że nauczycielka broniła swojego podejścia. Wtedy pomyślałem: „No cóż, może mam talent do programowania". Pomogło mi to także stwierdzić, że chyba nauczyciele nie wiedzą wszystkiego.

Seibel: Czy zaraz po tym, jak nauczycielka opisała algorytm, uznałeś, że jest błędny? A może najpierw przez pewien czas eksperymentowałeś z nim, a dopiero potem stwierdziłeś: „Jejku, to strasznie dużo operacji"?

Norvig: Chyba od razu zauważyłem problem. Trudno stwierdzić, co naprawdę myślałem, ale chyba od razu dostrzegłem, że algorytm może nie zakończyć pracy. Nie jestem pewien, czy równie dobrze zdawałem sobie sprawę z oczekiwanego czasu działania.

Pamiętam też, że znalazłem na strychu stare numery „Scientific American" ojca. Był w nich artykuł Christophera Stracheya na temat inżynierii oprogramowania. Strachey pisał, że zaczniemy używać języków wyższego poziomu. Wymyślił język, dla którego nigdy nie utworzono kompilatora. Był to język „papierowy". Strachey stwierdził, że napisze w tym języku program do gry w warcaby. Pamiętam, jak go czytałem. Był to pierwszy skomplikowany program, z którym się zapoznałem — w szkole uczyliśmy się tylko, jak tasować karty i robić podobne rzeczy. Niedawno ponownie przeczytałem artykuł i pierwszą rzeczą, jaką zauważyłem, było to, że znajduje się w nim błąd. To wspaniałe uczucie, ponieważ wiesz, że to Christopher Strachey i powinien wiedzieć, co robi. Dodatkowo to magazyn „Scientific American" — pracują nad nim redaktorzy i inni ludzie, którzy powinni wykryć błędy. W tekście autor twierdzi, że funkcja `make-move` przyjmuje pozycję na planszy i zwraca posunięcie. Kiedy zajrzysz do kodu, zobaczysz, że funkcja `make-move` przyjmuje pozycję na planszy i dodatkowy parametr. Najwidoczniej autor najpierw napisał tekst, a dopiero potem kod. Okazało się, że głębokość przeszukiwania nie może być nieskończona, dlatego Strachey dodał nowy parametr. Określa on głębokość przeszukiwania. Program przechodzi rekurencyjnie do określonego poziomu, a następnie kończy przeszukiwanie. Operację tę dodano później, ale autor nie poprawił dokumentacji.

Seibel: Był to więc pierwszy ciekawy kod, który przeczytałeś. Jaki pierwszy interesujący program napisałeś?

Norvig: Chyba był to program Game of Life. Napisałem go w ramach zadania domowego. Szybko je ukończyłem, a wtedy — oczywiście — nie mieliśmy dobrych wyświetlaczy. Nie posiadałem swojego trzydziestocalowego monitora — miałem dalekopis z żółtym papierem. Stwierdziłem, że szkoda drukować jedno małe pole (mieliśmy chyba użyć pola dziesięć na dziesięć kratek), a potem następne

i jeszcze następne. Dlatego stwierdziłem, że wydrukuję pięć pokoleń, jedno za drugim. Pamiętam, że w języku BASIC nie można było stosować tablic trójwymiarowych, a z jakiegoś powodu nie mogłem nawet użyć zestawu tablic dwuwymiarowych. Prowadziło to do wyczerpania pamięci lub czegoś w tym rodzaju. Dlatego musiałem wymyślić, jak utworzyć pięć lub sześć potrzebnych tablic dwuwymiarowych. Wtedy odkryłem pola bitowe.

Seibel: Z uwagi na ograniczenia pamięci utworzyłeś własną pamięć na dużą ilość danych. Czy ktoś nauczył cię używać tablic bitów i wymyśliłeś, jak je zastosować, czy może przeglądałeś podręcznik i stwierdziłeś: „O, popatrzcie, mamy tu instrukcje PEEK i POKE" lub coś w tym rodzaju?

Norvig: No cóż, zapisywałem w każdej lokalizacji zero lub jedynkę, a gdzie indziej musiałem zapisać więcej danych. Stwierdziłem: „Och, zapiszę inne liczby tutaj". W zasadzie nawet nie pamiętam, czy użyłem pamięci bitowej. Możliwe, że stosowałem cyfry i to raczej dziesiętne niż dwójkowe, ponieważ nikt nie przedstawił nam systemu dwójkowego w ciekawy sposób. Potem dodałem różne możliwości, np. sprawdzanie, czy liczby się nie powtarzają, a jeśli tak, to w jakim cyklu. Nie można było tego zrobić przy przechowywaniu tylko jednego wcześniejszego pokolenia.

Seibel: Kiedy rozwijałeś się jako programista, czy robiłeś coś specjalnego, aby wzbogacić swoje umiejętności w tym obszarze, czy po prostu pisałeś programy?

Norvig: Wydaje mi się, że tylko pisałem programy. Przede wszystkim robiłem to, co sprawiało mi przyjemność. Działo się tak zwłaszcza na studiach, kiedy byłem mniej zależny od harmonogramu. Znajdowałem ciekawy problem i próbowałem go rozwiązać. Robiłem to dla przyjemności, a nie dlatego, że prowadził do postępów w pisaniu pracy dyplomowej.

Seibel: W college'u studiowałeś przedmioty komputerowe, ale nauki komputerowe nie były twoim głównym kierunkiem, prawda?

Norvig: Kiedy zaczynałem, kursy komputerowe były prowadzone na wydziale matematyki stosowanej. Zanim ukończyłem college, powstał wydział nauk komputerowych, ale moim głównym kierunkiem pozostała matematyka. Miałem wrażenie, że jeśli chcę ukończyć nauki komputerowe jako główny kierunek, muszę jako główny kierunek studiować IBM. Trzeba było poznać ich asembler, ich system operacyjny 360 itd. Nie uznałem tego za ciekawe. Niektóre kursy mi się podobały, dlatego się na nie zapisałem, ale nie chciałem brać udziału we wszystkich wymaganych zajęciach.

Po college'u przez dwa lata pracowałem w firmie programistycznej w Cambridge. Po tych dwóch latach stwierdziłem: „Szkoły zacząłem mieć dość po czterech latach, a pracy — po dwóch, może więc dwa razy bardziej lubię szkołę?".

Seibel: Co robiłeś w tej firmie?

Norvig: Jej głównym produktem był pakiet narzędzi do projektowania oprogramowania. Firma prowadziła też doradztwo z różnych obszarów oprogramowania. Założyciele pracowali w Draper Labs w Cambridge w ramach misji *Apollo* i nad podobnymi projektami. Mieli znajomości w lotnictwie i otrzymywali zlecenia od rządu. Posiadali swoją wizję projektowania oprogramowania. Nigdy w nią nie wierzyłem, ale była ciekawa.

Pamiętam, że jeden z projektów w tej firmie wymagał napisania narzędzia do rysowania diagramów przepływu. Pomysł polegał na tym, że narzędzie miało analizować program i generować dla niego diagram przepływu. Było to idealne rozwiązanie, ponieważ właśnie w ten sposób ludzie korzystają z takich diagramów. Należy je przygotować na początku, ale nikt tego nie robi — tworzymy je po napisaniu kodu. Narzędzie było pomysłowe, ponieważ miało specyficzną gramatykę częściową, dlatego działało dla programów niepoprawnych składniowo i ukrywało elementy, których nie potrafiło przetworzyć. Narzędzie musiało umieć przetwarzać instrukcje `IF`, ponieważ

składały się one na różne bloki, natomiast inne elementy miało jedynie umieszczać w blokach. Uzyskaliśmy kontrakt na opracowanie tego programu. Zleceniodawca określił, że narzędzie ma działać w systemie Unix. Pożyczyliśmy więc maszynę z uczelni MIT i użyliśmy do budowania kompilatora wszystkich uniksowych narzędzi — narzędzia yacc i innych. W ostatniej chwili zleceniodawca stwierdził, że chcą zainstalować program w systemie VMS. Nagle okazało się, że nie mamy dostępu do narzędzia yacc. Uznaliśmy jednak, że nie stanowi to problemu, ponieważ go nie potrzebujemy. Narzędzie to było potrzebne tylko do wygenerowania tablic, co już zrobiliśmy.

Seibel: Dopóki gramatyka się nie zmieni, wszystko będzie w porządku.

Norvig: To prawda. Udostępniliśmy więc produkt, a odbiorca był zadowolony. Jednak wtedy — oczywiście — gramatyka się zmieniła, a my nie mieliśmy już dostępu do żadnych maszyn z Uniksem. Ostatecznie musiałem poprawiać gramatykę przez domyślanie się, co oznaczają tablice. Odbywało się to tak: „Tu mamy przejście do innego stanu — w porządku, wymyślę nowy stan i przejdę w zamian właśnie do niego".

Seibel: Czy rzeczywiście było to odpowiednie rozwiązanie? Czy nie pomyślałeś o tym, aby po prostu napisać nowy parser?

Norvig: Prawdopodobnie powinienem był tak zrobić, ale chodziło tylko o tę jedną małą poprawkę.

Seibel: Czy nie wpadłeś przez to w pułapkę związaną z tym, że odbiorca co kilka tygodni zgłaszał nową zmianę w gramatyce?

Norvig: Wtedy poszedłem na studia doktoranckie. Ktoś inny musiał zmierzyć się z tym problemem i nie wiem, co się stało.

Seibel: Nie było to już twój problem. Uzyskałeś tytuł doktora. Czy zmieniłbyś coś w sposobie, w jaki uczyłeś się programowania?

Norvig: Ostatecznie trafiłem do środowiska komercyjnego, dlatego mogłem wcześniej robić więcej rzeczy z tego obszaru. Nauczyłem się ich, jednak dużo czasu spędziłem w college'u i na studiach doktoranckich. Sprawiało mi to wiele frajdy, dlatego niczego nie żałuję.

Seibel: Czego musiałeś nauczyć się o programowaniu w środowisku komercyjnym?

Norvig: Musiałem przestrzegać terminów i dbać o zadowolenie członków zespołu, klientów oraz menedżerów. Na studiach doktoranckich nie trzeba tego robić. Wystarczy od czasu do czasu spotkać się z opiekunem.

Chyba największą zmianą było przejście od samodzielnej pracy do działania w zespole i poznawanie interakcji między ludźmi. Z tym zwykle nie spotykamy się na studiach. Wydaje mi się, że niektóre uczelnie zaczynają wprowadzać do programu zajęcia z tego obszaru. Kiedy byłem na studiach, pracę zespołową nazywano oszukiwaniem.

Seibel: Jakie umiejętności, oprócz pisania kodu, powinny rozwijać osoby, które chcą pracować w branży informatycznej?

Norvig: Przede wszystkim komunikowanie się z innymi. Ważna jest umiejętność zrozumienia oczekiwań klienta — ustalenia, co trzeba utworzyć i czy przygotowane rozwiązanie jest poprawne. Trzeba umieć komunikować się z odbiorcami oraz innymi członkami zespołu. Istotne są też interakcje z osobami zajmującymi wyższe stanowiska w firmie i z klientami. Są to różne relacje społeczne wymagające odmiennych umiejętności.

Seibel: Czy obecnie programowanie stało się zadaniem bardziej społecznym niż kiedyś?

Norvig: Tak uważam. Etapy programowania były kiedyś bardziej rozdzielone od siebie. W dawnych czasach stosowano przede wszystkim przetwarzanie wsadowe, dlatego interfejs był znacznie prostszy. Można było używać modelu wodospadu i stwierdzić, że danymi wejściowymi ma być np. talia kart, a danymi wyjściowymi — przykładowo raport z daną liczbą w określonej kolumnie.

Prawdopodobnie nie był to najlepszy sposób tworzenia specyfikacji. Należało od początku częściej komunikować się z klientem. Jednak wydawało się, że poszczególne etapy są bardziej niezależne od siebie. Teraz wszystko wydaje się bardziej płynne i interaktywne, dlatego sensowniejsze jest stwierdzenie: „Zamiast od początku przygotowywać kompletną specyfikację, zbierzmy klientów i rozpocznijmy burzę mózgów".

Seibel: Czy pamiętasz konkretne momenty „olśnienia", w których zauważyłeś różnice między samodzielną pracą a wykonywaniem zadań w zespole?

Norvig: Nie wiem, czy były to konkretne momenty. Raczej zdałem sobie sprawę, że nie mogę zrobić wszystkiego sam. Uważam, że dużą część procesu programowania można wykonać w głowie, ale nie da się w ten sposób zrobić wszystkiego — przynajmniej ja tego nie potrafię. Później trzeba polegać na innych osobach mających odpowiednie pomysły i je wykorzystać. Zacząłem myśleć: „W jaki sposób jest to prawdopodobnie zrobione?" zamiast: „Wiem, jak jest to zrobione, ponieważ sam to napisałem". Jak zrobiłbym daną rzecz, gdybym otrzymał takie zadanie? Zakładam, że tak właśnie to działa, a jeśli jest inaczej, muszę ustalić, dlaczego tak jest, a następnie odkryć, jak z tego korzystać.

Seibel: Czy uważasz, że nauczenie się działania w zespole umożliwia ci pracę nad większymi zadaniami w pojedynkę, tak jakbyś był specyficznym zespołem rozproszonym w czasie?

Norvig: Chyba rzeczywiście tak jest. Z pewnością widzę to u młodych programistów, którzy obecnie zaczynają pracę. Inna różnica między przeszłością i teraźniejszością polega na tym, że obecnie programowanie bardziej przypomina składanie elementów niż pisanie wszystkiego od podstaw. Teraz kiedy ktoś wykonuje zadaną pracę, mówi: „Potrzebowałem witryny, dlatego użyłem Ruby on Rails do tego, systemu Drupal w tej części, następnie wykorzystałem ten skrypt w Pythonie, a potem pobrałem tę procedurę do obliczeń statystycznych". Wystarczy dodać skrypty łączące elementy — nie trzeba pisać wszystkiego od początku. Dlatego uważam, że zrozumienie interfejsów i współdziałania między nimi jest dużo ważniejsze niż opanowanie szczegółów działania poszczególnych pakietów.

Seibel: Czy uważasz, iż oznacza to, że obecnie inne osoby mogą odnosić sukcesy w programowaniu?

Norvig: Uważam, że prawdziwe sukcesy odnoszą takie same osoby — przynajmniej w moim środowisku. Jednak rzeczywiście, trochę ważniejsze jest szybkie niż kompletne zrozumienie potrzebnych elementów. Sądzę, że jest w tym trochę brawury, gotowość do powiedzenia: „Po prostu to zrobię", brak obaw przed stwierdzeniem: „Nie rozumiem wszystkiego, ale zajrzałem do dokumentacji i poznałem te trzy elementy; wypróbowałem rozwiązanie i zadziałało, dlatego idę dalej". Pozwala to dojść do pewnego poziomu, ale myślę, że to nie wystarczy do zostania naprawdę dobrym programistą. Do tego trzeba zrozumieć coś więcej i zapytać: „Czy to bezpieczne? Co jest przyczyną błędu? Rzeczywiście, raz spróbowałem i zadziałało, ale czy zawsze tak będzie? Jak mam napisać przypadki testowe, aby to pokazać i lepiej zrozumieć program? Kiedy już to zrobię, jak wyodrębnić wykonaną pracę i udostępnić nowe narzędzie przydatne innym osobom dlatego, że złożyłem elementy w określony sposób?".

Seibel: Jaki sposób pracy w zespole ci odpowiadał, kiedy byłeś programistą? Czy lepiej podzielić problem, tak aby każdy otrzymał jego fragment? A może wolisz model programowania ekstremalnego, kiedy wszystko jest pisane w parach, a kod jest własnością całego zespołu?

Norvig: Bardziej odpowiada mi podział pracy. Steve Yagge napisał kiedyś artykuł *Good Agile, Bad Agile*. Uważam, że ma rację. Przez 10% czasu naprawdę warto pracować razem, aby wszyscy rozumieli zadanie w ten sam sposób. Przez większość tego czasu programiści nie są w pełni efektywni.

Jeśli zespół składa się z dwóch dobrych programistów, lepiej, żeby pracowali osobno, a następnie debugowali nawzajem swój kod po zakończeniu zadania. Gorsze jest podejście: „Zgadzamy się na 50% spowolnienie pracy w zamian za dodatkową parę oczu".

Uważam, że współpraca jest ważna przy ustalaniu planu pracy. Chodzi tu o analizy problemu do rozwiązania i dodawanych funkcji. Przed rozpoczęciem pracy nie wiadomo, czym ma być produkt. Naprawdę warto wtedy pracować razem. Następnie grupa dochodzi do etapu, kiedy można powiedzieć: „W porządku, wiemy, co chcemy zrobić. Jak podzielimy pracę?". Także to warto zrobić razem. Kiedy już wiadomo, co trzeba robić, uważam, że lepiej pracować samodzielnie. Informacje zwrotne są przydatne, dlatego myślę, że każdy fragment kodu powinien przejrzeć ktoś oprócz autora, ale nie musi się to odbywać w czasie rzeczywistym, w trakcie pisania kodu.

Pamiętam pomysł mistrza programisty promowany przez IBM. Wydało mi się to najgłupszą rzeczą, jaką kiedykolwiek usłyszałem. Dlaczego ktoś miałby pracować na rzecz jednego prawdziwego programisty?

Seibel: Dziwi mnie, że uważasz model mistrza programisty za tak zły pomysł. W eseju „Teach Yourself Programming in Ten Years" napisałeś, że programowanie to dziedzina, której prawdziwe opanowanie wymaga — podobnie jak inne umiejętności — około dziesięciu lat. W wielu dziedzinach obowiązuje hierarchia typu mistrz – czeladnik – uczeń. Możliwe, że nikt nie chce być uczniem, ale czy to takie dziwne, że osoba z dziesięcioletnim doświadczeniem ma robić coś innego niż ktoś świeżo po studiach?

Norvig: Uważam, że najlepsza w byciu uczniem jest możliwość podpatrywania mistrza. Chciałbym częściej widzieć takie podejście. Dlatego uważam, że jest to następne zastosowanie programowania w parach. Dla początkującego naprawdę korzystne może być podglądanie kogoś o znacznie większym doświadczeniu. Dotyczy to zwłaszcza zadań, których szkoły nie uczą, np. debugowania. Każdy może nauczyć się algorytmów i tym podobnych zagadnień, ale szkoły naprawdę nie uczą debugowania. Przydatne jest podglądanie kogoś, kiedy można powiedzieć: „Nigdy bym o tym nie pomyślał".

Sądzę, że związki mistrz – uczeń rozwinęły się po części z powodu niedostatku materiałów. Jubiler miał do dyspozycji tylko niewielką ilość złota. Chirurg operuje tylko jedno serce, dlatego pracować powinna tylko najlepsza osoba, a pozostałe mają służyć pomocą. Przy pisaniu kodu jest inaczej. Mamy dostęp do wielu terminali i klawiatur. Nie trzeba ich racjonować.

Seibel: Wspomniałeś o rzeczach, których szkoły nie uczą. Pracujesz na uczelni i w przemyśle. Czy uważasz, że akademickie nauki komputerowe i programowanie komercyjne są dopasowane do siebie?

Norvig: To poważne pytanie. Sądzę, że program nauk komputerowych zawiera mało nieprzydatnych elementów. Uważam, że studenci uczą się głównie bardzo wartościowych rzeczy. Moim zdaniem, studia są przydatne, ale nie wystarczą do odniesienia sukcesu w branży lub przy budowaniu systemów. Myślę, że program na wielu uczelniach zmienia się za wolno. Dotyczy to wielu obszarów. Szkoły nie uczą pracy w zespole. Nie pokazują także, jak łączyć różne elementy, ale młodzież w jakiś sposób się tego uczy, dlatego może nie stanowi to problemu. W Google interesują nas duże chmury obliczeniowe, przetwarzanie równoległe itd. Szkoły nie uczą tych zagadnień, choć — moim zdaniem — cieszą się one dużym zainteresowaniem. Dlatego uważam, że występuje tu pewne opóźnienie, natomiast uczelnie na pewno są przydatne.

Seibel: Czy istnieją obszary, w których naukowcy wyprzedzają przemysł? Czy firmy czasem ignorują wartościową wiedzę na temat budowania oprogramowania?

Norvig: W pewnym stopniu tak się zdarza. Prawdopodobnie najlepszy przykład to weryfikacja modelowa. Intel nie przywiązywał do tego dużego znaczenia, a następnie musiał wycofać produkt z rynku i stracił dużo pieniędzy, ponieważ w mnożeniu był błąd. Wtedy firma zaczęła zwracać uwagę na tę kwestię i zgłosiła się do naukowców z pytaniem: „Jak możecie pomóc?". Coś więc było na rzeczy. Teraz weryfikacja modelowa to integralna część procesu Intela, dlatego to dobry przykład. W pewnym stopniu dotyczy to także języków programowania. Trwają intensywne prace w tym obszarze, ale nie widać ich dużego wpływu na nowsze języki programowania. Naukowcy pracują też nad systemami operacyjnymi. Wspieramy kierowany przez Dave'a Pattersona ośrodek RAD Lab w Berkeley i inne jednostki. Jednak — oczywiście — to przemysł styka się z większymi problemami. Możliwe, że nie zna rozwiązań każdego z nich, ale firmy pracują nad nimi ciężej niż pracownicy uczelni.

Seibel: Uważasz więc, że w na uczelniach nie powstają pomysły pomijane przez przemysł z uwagi na opór przed pewnymi zmianami? Może generacja programistów, którzy sami nauczyli się PHP, nigdy nie zafascynuje się Haskellem, nawet jeśli jest to lepszy sposób na pisanie oprogramowania?

Norvig: Jestem dość sceptycznie nastawiony. Uważam, że jeśli jakaś technika ma rzeczywiste zalety, ludzie z niej skorzystają. Nie sądzę, że rynek jest optymalny i natychmiast znajduje najlepsze rozwiązanie, jednak niewiele do tego brakuje. Naukowcy mogą nie dostrzegać całego problemu, z którym zmaga się przemysł. Po części jest to kwestia systemu edukacji, ale jeśli w firmie pracuje wielu programistów, którzy nie wiedzą, czym jest monada, i nie ukończyli kursów z teorii kategorii, występuje luka.

Częścią problemu są starsze systemy, których nie można po prostu wyrzucić, dlatego przechodzenie odbywa się stopniowo. Jestem pewien, że w niektórych obszarach przemysł powinien być bardziej nastawiony na przyszłość zgodnie z podejściem: „Pewnie, nie możemy dokonać zmiany już dziś, ale musimy mieć plan określający, gdzie będziemy za dziesięć lat. Powinien on także informować, jak mamy się tam znaleźć".

Jednak firmy oczekują usprawnień w obszarach, które będą miały duże znaczenie. Uważam, że prace nad językami programowania odbywają się często na zbyt niskim poziomie, aby miało to tak duży wpływ na przemysł, jak tego oczekują projektanci. Twórca języka może powiedzieć: „Popatrzcie, w moim nowym wspaniałym języku tych sześć wierszy kodu można zapisać za pomocą dwóch wierszy". No cóż, świetnie, podejrzewam, że pozwala to zwiększyć produktywność programistów oraz uprościć debugowanie i konserwowanie kodu. Jednak możliwe, że napisany kod to tylko mała część całego systemu produkcyjnego, a prawdziwym problemem jest codzienne aktualizowanie danych, zbieranie informacji z internetu, zapisywanie ich w systemie i przekształcanie na odpowiedni format. Dlatego trzeba pamiętać, że rozwiązanie często dotyczy tylko bardzo małego fragmentu całego problemu. Oznacza to, że musi istnieć bardzo poważna przeszkoda do pokonania, aby warto było dokonać zmiany.

Seibel: Pomińmy teraz praktyczne badania nad językiem. Uważasz, że przebyliśmy długą drogę od czasu, kiedy nauki komputerowe przypominały kierunek „IBM"?

Norvig: Tak. Uważam, że program jest obecnie dobry. Smutne jest to, że tak mało studentów z tego korzysta. Liczba zapisów spada. Oczywiście, jest grupa osób, które kochają komputery lub projektowanie tak bardzo, że trafiają na studia informatyczne. Dbamy o tę grupę. Jednak wielu najlepszych i najinteligentniejszych studentów wybiera fizykę, biologię lub coś innego, ponieważ są to najpopularniejsze dziedziny. Jeszcze inna grupa mówi: „Tak, lubię komputery, ale nie mam przyszłości w tej branży, ponieważ wszystkie stanowiska i tak są przenoszone do Indii. Dlatego wybiorę

kursy przygotowujące do studiów prawniczych lub coś innego, abym mógł znaleźć pracę". Szkoda, że tak się dzieje. Uważam, że te osoby mają błędne informacje.

Seibel: Chcesz powiedzieć, że to szkoda, ponieważ licznym z tych osób programowanie spodobałoby się, czy dlatego, iż ich potrzebujemy?

Norvig: Oba powody są ważne. Wiele osób może czerpać przyjemność z rozmaitych zajęć. Jeśli daną osobę w równym stopniu interesują dwa kierunki, nie chcę twierdzić, że powinna wybrać nauki komputerowe. Jednak sądzę, że występuje brak dopasowania. Potrzebujemy większej liczby dobrych ludzi. Sądzę, że mogą oni mieć duży wpływ na świat. Jeśli to właśnie chcą osiągnąć, powinniśmy kierować większą liczbę najlepszych osób na nauki komputerowe.

Seibel: W jednej z prac Dijkstra opisuje, że nauki komputerowe to część matematyki, a studenci tego kierunku przez pierwszych *n* lat edukacji nie powinni nawet dotykać komputera. Zamiast tego mają nauczyć się manipulowania systemami formalnych symboli. Jak sądzisz, w jakim zakresie musi znać matematykę kompetentny programista?

Norvig: Nie uważam, że potrzebny jest poziom postulowany przez Dijkstrę. Poza tym koncentruje się on na specyficznym rodzaju matematyki — na matematyce dyskretnej, dowodach logicznych. Ja jestem związany z obszarem, gdzie ma to mniejsze znaczenie. Dziedzina ta jest bardziej probabilistyczna niż logiczna. Rzadko tworzę program, którego poprawność mogą udowodnić.

Czy wyszukiwarka Google działa poprawnie? No cóż, wpisz dowolne słowa, a otrzymasz dziesięć stron. Jeśli wystąpi awaria, działanie wyszukiwarki będzie nieprawidłowe, jeżeli jednak otrzymasz te dziesięć odnośników zamiast dziesięciu innych, nie będziesz mógł określić, które wyniki są poprawne. Możesz mieć swoje zdanie na temat tego, które są lepsze, ale nie stwierdzisz nic więcej. Sądzę, że Dijkstrze nie o to chodziło. Uważam, że kiedy zacznie się rozwiązywać tego typu problemy lub kwestię poruszania się robota kierowcy po mieście bez wjechania w kogokolwiek, próba przeprowadzenia logicznego dowodu jest skazana na niepowodzenie.

Seibel: Czy istnieje kluczowa umiejętność potrzebna dobrym programistom? W różnych dziedzinach obowiązują — oczywiście — inne wymagania, czy jednak występują pewne punkty wspólne związane z pisaniem kodu?

Norvig: Musisz robić postępy, a następnie wprowadzać poprawki. To wszystko, co musisz umieć w życiu. Potrzebujesz jakiegoś pomysłu, aby powiedzieć: „Oto kierunek, w którym mam zmierzać". Następnie musisz stwierdzić: „Teraz trzeba to poprawić". Oto możliwe przyczyny usprawnień: „Nie zrobiłem tego dobrze — zapomniałem o obsłudze pewnych warunków" albo „Teraz, kiedy lepiej to rozumiem, utworzę bardziej abstrakcyjne narzędzie, aby ułatwić sobie pisanie podobnych systemów w przyszłości". Sądzę, że potrzebna jest do tego introspekcja związana z pytaniami: „Dokąd chciałem dojść? Jak to zrobiłem? Czy istnieje lepszy sposób na wykonanie zadania?".

Seibel: Sądzisz więc, że ta umiejętność — czyli tworzenie rozwiązania, usuwanie błędów i powtarzanie tego procesu — to rodzaj myślenia, którego należy uczyć wiele osób, także tych, które nie chcą zostać programistami? Czy gdybyś tworzył program nauczania dla szkoły średniej lub uczelni, chciałbyś, aby wszyscy uczyli się programowania w ten sposób? A może jest to zbyt wyspecjalizowana umiejętność?

Norvig: Myślę, że jest to wyspecjalizowana umiejętność. Uważam, że programowanie to tylko jeden przykład opisanego sposobu myślenia. Byłbym zadowolony także z wprowadzenia innych przykładów, takich jak problemy mechaniczne: „Oto zbiór części. Jak sprawić, aby woda przepłynęła stąd dotąd i znalazła się w kubku?". Nie musi chodzić o manipulowanie wierszami kodu. Może to być praca z różnego rodzaju elementami i obserwowanie ich współdziałania.

Seibel: Do jak niskiego poziomu powinni schodzić programiści? W eseju „Teach Yourself Programming in Ten Years" piszesz o tym, że trzeba wiedzieć, jak dużo czasu zajmuje wykonanie instrukcji w porównaniu z odczytem danych z dysku itd. Czy nadal musimy uczyć się asemblera?

Norvig: Nie wiem. Knuth twierdził, że powinniśmy wszystko pisać w asemblerze, ponieważ kod w języku C jest zbyt niewydajny. Nie zgadzam się z tym. Uważam, że warto wiedzieć, które instrukcje są niewydajne, jednak obecnie nie dotyczy to poziomu pojedynczych instrukcji. Nie chodzi o porównywanie sekwencji składających się z dwóch i trzech instrukcji. Ważne jest, czy wystąpił błąd strony lub nieudanego odczytu danych z pamięci podręcznej. Myślę, że nie musimy znać asemblera. Potrzebna jest wiedza o architekturze. Należy zrozumieć, czym jest asembler, a także wiedzieć, że istnieje hierarchia pamięci, a przenoszenie danych z jednego poziomu hierarchii na inny związane jest z dużym spadkiem wydajności. Jednak nadal uważam, że należy zrozumieć ten system na poziomie abstrakcyjnym.

Seibel: Czy są jakieś książki, które — twoim zdaniem — powinien przeczytać każdy programista?

Norvig: Myślę, że wybór jest duży. Nie ma jednej, jedynej ścieżki. Trzeba przeczytać jedną z książek o algorytmach. Nie wystarczy wycinać i wklejać algorytmów. Może to być książka Knutha lub Cormena, Leisersona i Rivesta. Są też inni autorzy, np. Sally Goldman, która napisała nową książkę z bardziej praktycznym ujęciem algorytmów. Moim zdaniem, jest to całkiem ciekawa pozycja. Należy więc przeczytać coś o algorytmach. Trzeba też zapoznać się z książką o abstrakcji. Lubię pozycję Abelsona i Sussmana, ale są też inne książki.

Trzeba dobrze poznać używany język programowania. Przeczytać podręcznik. Zapoznać się z książkami opisującymi zarówno mechanizmy języka, jak i cały proces debugowania oraz testowania — coś w rodzaju *Code Complete*. Myślę jednak, że jest wiele różnych ścieżek. Nie twierdzę, iż trzeba przeczytać jeden zestaw książek.

Seibel: Wprawdzie twoja praca nie wymaga dużej ilości programowania, ale nadal piszesz programy na potrzeby esejów z witryny. Jak podchodzisz do tworzenia tych małych programów?

Norvig: Uważam, że jedną z najważniejszych kwestii jest umiejętność jednoczesnego utrzymywania wszystkich informacji w głowie. Jeśli to potrafisz, masz dużo większe prawdopodobieństwo powodzenia. Wtedy tworzenie małych programów jest łatwiejsze. Przy większych programach potrzebne są do tego dodatkowe narzędzia.

Ważne jest też, aby wiedzieć, co się robi. Kiedy pisałem program do rozwiązywania sudoku, niektórzy blogerzy pisali komentarze na ten temat. Zwracali uwagę na różnice między programem do sudoku Norviga, a podejściem innego programisty, którego nazwiska zapomniałem — jest on jednym z guru z obszaru projektowania opartego na testach. Programista ten zaczął pracę i napisał: „Zamierzam zająć się sudoku. Potrzebna będzie ta klasa, a pierwszą rzeczą, jaką zrobię, będzie napisanie zestawu testów". Jednak nie udało mu się nic osiągnąć. Zamieścił pięć wpisów na blogu, w każdym z nich dodał fragment kodu i napisał mnóstwo testów, ale nie utworzył działającego programu, ponieważ nie wiedział, jak rozwiązać problem.

Ja, znając sztuczną inteligencję, wiedziałem, że jest coś takiego jak propagacja ograniczeń i że to działa. Wiedziałem, że jest coś takiego jak wyszukiwanie rekurencyjne i że to działa. Potrafiłem od razu dostrzec, że można połączyć te dwie techniki i rozwiązać sudoku. Wspomniany programista tego nie wiedział, dlatego w pewnym sensie błądził w ciemnościach, choć jego kod „działał", ponieważ przygotował przypadki testowe.

Blogerzy zawzięcie spierali się, czego to dowodzi. Moim zdaniem, nie dowodzi niczego. Uważam, że projektowanie sterowane testami to świetna technika. Obecnie stosuję ją znacznie częściej niż

kiedyś. Jeśli jednak nawet przetestujesz wszystkie elementy, a nie wiesz, jak podejść do problemu, nie uzyskasz rozwiązania.

Seibel: Można zadać pytanie, skąd inny programista miał wiedzieć to, co ty? Czy powinien zrobić doktorat i specjalizować się w sztucznej inteligencji? Nikt nie zna wszystkich algorytmów. Obecnie można skorzystać z wyszukiwarki Google, jednak znalezienie odpowiedniego podejścia do problemu różni się nieco od wyszukania platformy do tworzenia aplikacji sieciowych.

Norvig: Chodzi ci o to, skąd wiemy, czego nie wiemy?

Seibel: Właśnie.

Norvig: Ważne są dwie kwestie. Po pierwsze, trzeba założyć, że może istnieć znane rozwiązanie problemu. Można powiedzieć: „No cóż, na pewno nikt nie wie, jak to zrobić, dlatego szukanie rozwiązania na chybił trafił to równie dobry pomysł jak każdy inny". To jedna możliwość. Oto inne podejście: „No cóż, prawdopodobnie ktoś wie, jak to zrobić. Nie wiem, jak nazywa się potrzebny algorytm, dlatego muszę to ustalić". Moim zdaniem, po części wynika to z intuicji i podejścia typu: „Wydaje mi się, że tego rodzaju rozwiązania powinny być częścią wiedzy z obszaru sztucznej inteligencji". Następnie trzeba ustalić, jak znaleźć rozwiązanie. Wspomniany programista prawdopodobnie mógł poszukać informacji o sudoku i znaleźć potrzebne metody. Może myślał, że to oszukiwanie. Nie wiem.

Seibel: Załóżmy, że *jest* to oszukiwanie. Przyjmijmy, że byłeś pierwszą osobą, która próbuje rozwiązać sudoku. Techniki, które zastosowałeś, cały czas istniały i czekały na zastosowanie.

Norvig: Powiedzmy, że chcę rozwiązać pewien problem z obszaru biologii. Nie wiem, które algorytmy najlepiej nadają się do sekwencjonowania genów lub wykonywania innych zadań. Zdaję sobie jednak sprawę, że takie algorytmy istnieją. Wtedy rozpoczynam poszukiwania. Na innym poziomie niektóre kwestie są podstawowe — jeśli nie wiesz, czym jest programowanie dynamiczne, znajdujesz się w bardzo trudnej sytuacji. Problem ten będzie cały czas wracał, jeśli nie znasz ogólnej idei poszukiwań, jeżeli nie wiesz, że można dokonać wyboru i cofnąć się, kiedy dany element okaże się niepotrzebny. Wszystkie te pomysły pochodzą z lat sześćdziesiątych dwudziestego wieku. Ludzie odkryli te rozwiązania w pierwszych latach po powstaniu dziedziny związanej z programowaniem. Wydaje mi się, że wszyscy powinni znać takie pomysły. Natomiast niektóre rozwiązania, np. wymyślone w zeszłym roku, nie muszą być znane każdemu.

Seibel: Zatem programiści powinni czytać stare prace?

Norvig: Nie, ponieważ zdarza się wiele nietrafionych pomysłów. Ponadto występują „fuzje", kiedy to w dwóch różnych dziedzinach rozwijane są zupełnie niezależne technologie i terminologie, po czym naukowcy odkrywają, że robili to samo. Uważam, iż należy patrzeć raczej ze współczesnej perspektywy, niż dokładnie śledzić rozwój pomysłów. Należy jednak je znać. Nie wiem, które książki będą tu najlepsze, ponieważ sam uczyłem się w trudniejszy sposób — po kawałku.

Seibel: Wróćmy do projektowania oprogramowania. Co możesz powiedzieć o pracy nad większymi programami, kiedy nie można zapamiętać, jak współdziałają wszystkie fragmenty kodu? Jak projektujesz takie rozwiązania?

Norvig: Uważam, że warto mieć dobrą dokumentację na poziomie projektu całego systemu. Co można uzyskać dzięki temu i w jaki sposób? Dokumentowanie każdej metody jest zwykle zbyt żmudne. Przeważnie polega to na powielaniu informacji, które można wyczytać z nazw funkcji i parametrów. Jednak naprawdę ważne jest, aby na początku opracować ogólny projekt tego, co mają robić poszczególne elementy. Musi to być opis, który wszyscy rozumieją. Musi to być także

prawidłowe rozwiązanie. Jedną z najważniejszych rzeczy w pracy nad udanymi projektami jest to, aby ludzie mieli doświadczenie wystarczające do zbudowania odpowiedniego programu. Jeżeli tego brakuje, a rozwijasz coś, czego wcześniej nie tworzyłeś, i nie wiesz, jak to zrobić, następnym najlepszym podejściem jest zachowanie elastyczności, tak aby można wprowadzić zmiany po popełnieniu błędu.

Seibel: Jak długo możesz zastanawiać się, jak coś powinno działać, jeśli wcześniej nie budowałeś podobnego rozwiązania? Czy musisz zacząć pisać kod, aby naprawdę zrozumieć problem?

Norvig: Jednym ze sposobów myślenia o rozwiązaniu jest cofanie się. Chcesz osiągnąć końcowy stan, czyli dobry produkt, a w przypadku niektórych problemów istnieje tylko jedno dobre rozwiązanie. W innych sytuacjach są ich miliony i można wybrać wiele różnych kierunków, a każdy z nich będzie porównywalny z innymi. Dlatego uważam, że praca wygląda inaczej, w zależności od rodzaju problemu.

Następnie warto zastanowić się nad trudnymi i prostymi wyborami. Co doprowadzi do prawdziwych kłopotów, jeśli dokonasz niewłaściwego wyboru w obszarze architektury? Jeżeli natrafisz na wbudowane ograniczenia lub po prostu utworzysz złe rozwiązanie? W Google natrafiamy na problemy każdego rodzaju. Cały czas powraca problem skalowalności. Jeśli spojrzymy na obecny stan rzeczy i stwierdzimy, że zbudujemy coś, co potrafi obsłużyć dziesięciokrotnie większe obciążenie, za kilka lat wzrost przekroczy szacunki. Wtedy trzeba będzie wyrzucić rozwiązanie i zacząć wszystko od nowa. Należy jednak dokonać prawidłowego wyboru przynajmniej w kontekście wybranych warunków działania. System ma działać dla przedziału od miliarda do dziesięciu miliardów stron internetowych. Co to oznacza w kategoriach rozpowszechniania danych do wielu maszyn? Jakiego rodzaju dane będą przesyłane w obu kierunkach? Trzeba przekonująco opisać system na tym poziomie. Na niektóre pytania można odpowiedzieć na podstawie obliczeń wykonanych na serwetce, w innych przypadkach trzeba przeprowadzić symulacje, a w niektórych sytuacjach — przewidzieć przyszłość.

Seibel: Wydaje mi się, że na tego rodzaju pytania znacznie łatwiej odpowiedzieć poprawnie na podstawie obliczeń na serwetce lub symulacji niż przez pisanie kodu.

Norvig: Tak, też tak uważam. W takich sytuacjach obliczenia są prawdopodobnie lepszym podejściem. Pojawiają się też inne kwestie. Producent informuje, że w przyszłym roku udostępni przełącznik, który obsłuży dziesięciokrotnie większy ruch. Czy zaprojektujesz program pod tym kątem? Czy wierzysz producentowi? A może zaprojektujesz program na podstawie dzisiejszych możliwości? Wymaga to uwzględnienia wielu plusów i minusów.

Występują też problemy związane z interfejsem użytkownika, które pojawiają się dopiero po utworzeniu produktu. Myślisz, że interakcja będzie przebiegać doskonale, ale kiedy pokazujesz program użytkownikom, połowa z nich nie potrafi z niego korzystać. Wtedy trzeba zrobić krok wstecz i wymyślić coś nowego.

Seibel: Pomińmy teraz projektowanie interakcji z użytkownikami. Kiedy warto tworzyć prototypy, zamiast tylko wyobrażać sobie, jak coś będzie działać?

Norvig: Myślę, że warto wyobrazić sobie rozwiązanie, aby stwierdzić, czy będzie działać. Warto określić, czy wygląda na wygodne. Przydatny jest zestaw narzędzi pomagający zbudować potrzebne rozwiązanie, a także ułatwiający modyfikowanie systemu w przyszłości. Jeśli zaczniesz tworzyć prototyp i nagle okaże się, że jest niewygodny, może przygotowałeś zły zestaw podstawowych elementów. Warto dowiedzieć się o tym tak wcześnie, jak to możliwe.

Seibel: Jak oceniasz pomysł sterowania projektowaniem za pomocą testów?

Norvig: Traktuję testy bardziej jak technikę poprawiania błędów niż projektowania. Ekstremalne podejście działa mniej więcej tak: „Najpierw piszemy test, który na końcu pracy poinformuje, że mamy poprawne rozwiązanie". Potem uruchamiamy test. Kończy się niepowodzeniem, a wtedy pytamy: „Czego teraz potrzebuję?". Według mnie, nie jest to odpowiednia metoda projektowania.

Wydaje mi się, że technika ta miałaby sens tylko wtedy, gdyby istniało z góry określone rozwiązanie. Uważam, że najpierw trzeba pomyśleć o problemie. Trzeba zadać sobie pytania: „Jakie elementy występują? Jak napiszę testy dla elementów, skoro niektórych nie znam?". Następnie, po przejściu tego etapu warto utworzyć testy dla każdego elementu i dobrze zrozumieć interakcje między nimi, przypadki brzegowe itd. Testy należy przygotować dla wszystkich elementów. Nie sądzę jednak, że cały proces projektowania powinien być sterowany stwierdzeniem: „Test zakończył się niepowodzeniem".

Inną rzeczą, która mi nie odpowiada, jest to, że wiele problemów, na które natrafiamy w firmie Google, nie wpasowuje się w prosty dwuwartościowy model testowania. Kiedy przyjrzysz się zestawom testów, zobaczysz asercje `assertEqual`, `assertNotEqual`, `assertTrue` itd. Jest to przydatne, ale chcemy też używać asercji w rodzaju `assertAsFastAsPossible` i zastosować ją do dużej bazy danych możliwych zapytań. Uzyskujemy precyzję na określonym poziomie przy wykrywalności na danym poziomie i chcielibyśmy to zoptymalizować. Pakiety testów nie obejmują tego rodzaju statystycznych lub ciągłych wartości, które próbujesz zoptymalizować, a tylko dwuwartościowe testy informujące, czy rozwiązanie jest prawidłowe, czy błędne.

Seibel: Jednak ostatecznie wszystko można przekształcić na logikę dwuwartościową. Wystarczy uruchomić zestaw zapytań, zapisać wszystkie wartości i sprawdzić, czy znajdują się w odpowiednim przedziale.

Norvig: Jest to możliwe. Jednak na podstawie metod dostępnych w pakietach testów można stwierdzić, że nie są one dostosowane do wykonywania takich zadań. Jestem zaskoczony tym, jak bardzo opisane przeze mnie podejście jest akceptowane w Google. Kiedy pracowałem w Junglee, pamiętam, że musiałem nauczyć go ludzi z działu jakości. Pracowaliśmy nad wyszukiwarką cen i powiedziałem: „Potrzebuję testu. To zapytanie ma dawać 80% prawidłowych odpowiedzi". Dział jakości odpowiedział: „W porządku! A więc zła odpowiedź oznacza błąd, prawda?". Powiedziałem: „Nie, jedna zła odpowiedź jest dopuszczalna, jeśli nie ma ich więcej niż 80%". Na to dział jakości stwierdził: „A więc zła odpowiedź nie jest błędem?". Traktowali to tak, jakby były tylko dwie możliwości. Nie wpadli na pomysł, że chodzi bardziej o pewien kompromis.

Seibel: Jesteś jednak zwolennikiem testów jednostkowych. W jaki sposób programiści powinni traktować testy?

Norvig: Należy pisać wiele testów. Programiści powinni myśleć o różnych warunkach. Uważam, że obok testów jednostkowych warto używać bardziej złożonych testów regresyjnych. Trzeba też zastanowić się nad symptomami błędów. Pamiętam jedną ze znakomitych lekcji na temat programowania. Zjawiłem się na lotnisku Heathrow, a akurat miała miejsce awaria prądu i żaden z komputerów nie działał. Mimo to, mój samolot wystartował zgodnie z planem.

Obsługa znalazła gdzieś wydruki dotyczące wszystkich lotów. Nie wiem, skąd je wzięli — jeden z komputerów musiał działać poza lotniskiem. Nie wiem, czy obsługa wydrukowała dane tylko tego dnia, czy robią to zawsze poprzedniego wieczoru, a jeśli nie wystąpi awaria, po prostu wyrzucają wydruki do śmieci. Jednak dane były dostępne i obsługa przy bramkach znała procedurę korzystania z papierowego rozwiązania awaryjnego zamiast systemu komputerów.

Uznałem to za doskonałą lekcję z dziedziny projektowania oprogramowania. Sądzę, że większość programistów nie myśli o tym, jak program zadziała, kiedy zabraknie prądu.

Seibel: Jak działa Google bez zasilania?

Norvig: Google nie funkcjonuje zbyt dobrze bez prądu. Jednak mamy awaryjne generatory i kilka centrów danych. Ponadto myślimy, jak będzie działał dany fragment, jeśli serwer będzie nieaktywny lub wystąpią inne awarie. Jeżeli program działa na tysiącu maszyn, zastanawiamy się, co się stanie, kiedy jedna z nich przestanie pracować. W jaki sposób obliczenia zostaną wznowione w innym miejscu?

Seibel: Knuth w eseju na temat rozwijania programu TeX napisał o włączaniu destrukcyjnej osobowości do kontroli jakości i dokładaniu wszelkich starań w celu wywołania awarii w swoim kodzie. Jak myślisz, czy większość programistów dobrze sobie z tym radzi?

Norvig: Nie. Mam przykład w postaci mojego narzędzia do sprawdzania pisowni. Popełniłem błąd w kodzie, który mierzył, jak dobrze sobie radzę, a w tym samym czasie wprowadziłem drobne zmiany w kodzie programu. Uruchomiłem aplikację i uzyskałem o wiele lepszy wynik. I uwierzyłem w ten wynik! Gdyby był znacznie gorszy, nigdy bym nie pomyślał, że drobna zmiana w funkcji to spowodowała. Jednak chciałem wierzyć, że drobna modyfikacja znacznie poprawiła wynik. A powinienem być sceptyczny i pomyśleć: „Nie, to nie mogło mieć tak dużego wpływu. Coś musi być nie tak".

Seibel: W jaki sposób unikasz nadmiernego uogólniania i tworzenia większej ilości kodu, niż to potrzebne, oraz marnowania w ten sposób zasobów?

Norvig: To ciągła walka i wiele bitew. Prawdopodobnie nie jestem najlepszą osobą, aby odpowiedzieć na to pytanie, ponieważ wolę rozwiązania eleganckie od praktycznych. Dlatego muszę walczyć ze sobą i powtarzać, że w czasie pracy nie mogę pozwolić sobie na tego rodzaju myślenie. Muszę powiedzieć: „Jesteśmy tu, aby udostępnić najsensowniejsze rozwiązanie. Jeśli istnieje rozwiązanie doskonałe, prawdopodobnie nie stać nas na jego przygotowanie". Musimy zrezygnować i stwierdzić: „Zrobimy to, co jest w tej chwili najważniejsze". Muszę przypominać o tym sobie i osobom, z którymi pracuję. W języku niemieckim jest powiedzenie, że lepsze jest wrogiem dobrego. Zapomniałem, skąd dokładnie pochodzi. Każdy praktyczny inżynier musi nauczyć się tej lekcji.

Seibel: Dlaczego takie kuszące jest rozwiązywanie problemów, które tak naprawdę nie istnieją?

Norvig: Chcesz poczuć się mądry i zamknąć pewien etap — ukończyć coś i przejść do innych zadań. Uważam, że ludzie są zbudowani tak, iż potrafią zajmować się tylko określoną ilością rzeczy. Dlatego chcesz powiedzieć: „To jest już gotowe. Mogę o tym zapomnieć i iść dalej". Jednak musisz policzyć, jaki będzie zwrot z inwestycji w przygotowanie całego rozwiązania. Ilustruje go zawsze krzywa w kształcie litery S, dlatego po ukończeniu prac na poziomie 80 – 90% zyski są coraz mniejsze. W dolnej części krzywej znajduje się setka innych rzeczy, które przyniosą większy zwrot. W pewnym momencie trzeba powiedzieć: „Wystarczy. Zakończmy to i zabierzmy się za coś, co da lepsze wyniki".

Seibel: W jaki sposób programiści mogą nauczyć się lepiej określać miejsce na krzywej, w którym obecnie się znajdują?

Norvig: Uważam, że potrzebne jest odpowiednie, nastawione na wyniki środowisko. Myślę, że ludzie potrafią się sami tego nauczyć. Chcesz zoptymalizować efekty, ale jeśli zostaniesz pozostawiony sam sobie, zadbasz w własne poczucie komfortu. Jednak wyniki należy optymalizować pod innym kątem. Według niektórych chodzi o zwrot z inwestycji dla firmy, według innych — o zadowolenie klientów. Trzeba pomyśleć, jakie będą zyski dla klienta, jeśli rozbuduję daną funkcję z 95 do 100%, zamiast pracować nad dziesięcioma innymi funkcjami, których poziom rozbudowy wynosi 0%.

W Google jest to proste, ponieważ pracujemy zgodnie z filozofią „udostępniaj wcześnie i często". Wynika to także z wielu cech tej firmy. Po pierwsze, za większość produktów nie pobieramy opłat, dlatego łatwo powiedzieć: „No cóż, udostępnijmy to. Jak bardzo użytkownicy mogą się skarżyć?". Po drugie, nie tłoczymy płyt CD i nie umieszczamy ich w pudełkach, dlatego jeśli coś nie jest gotowe na dziś i nawet jeżeli wystąpi błąd, nie prowadzi to do katastrofy. Większość oprogramowania znajduje się na serwerach, dlatego możemy jutro wprowadzić poprawki, a wszyscy natychmiast otrzymają aktualizacje. Nie miewamy koszmarów o instalowaniu aktualizacji. Dlatego łatwiej nam powiedzieć: „Udostępnimy to, uzyskamy informacje zwrotne od użytkowników, a potem poprawimy, co trzeba, i nie będziemy przejmować się innymi elementami".

Seibel: Jakich narzędzi używasz, kiedy projektujesz duży system? Czy siedzisz nad arkuszem papieru technicznego, czy używasz narzędzia do rysowania diagramów UML?

Norvig: Nigdy nie lubiłem narzędzi w rodzaju UML-a. Zawsze uważałem, że jeśli nie można czegoś zrobić w samym języku, jest to jego wada. Myślę, że wiele rzeczy rozpatrujemy na wyższym poziomie. W Google często się zastanawiamy, jak podzielić pewne zadania i wykonywać je równolegle. Kod trzeba uruchamiać na wielu maszynach, jednak mamy tak wielu użytkowników i — w przypadku licznych aplikacji — tak dużo danych, że nie wiadomo, jak ma działać. Dlatego częściej myślimy na poziomie maszyn i szafek z komputerami niż na poziomie funkcji i interakcji. Kiedy ogólne kwestie zostaną ustalone, można zająć się poszczególnymi funkcjami i metodami.

Seibel: Czyli na tym poziomie opisu używacie po prostu języka naturalnego?

Norvig: Tak, przede wszystkim. Niektóre osoby przygotowują rysunki i mówią: „Ten serwer będzie obsługiwał żądania tego rodzaju i będzie połączony z tym serwerem. Następnie użyjemy tych narzędzi do przechowywania danych oraz wykorzystamy dużą rozproszoną tablicę haszującą i inne mechanizmy. Zastosujemy te trzy gotowe narzędzia, a następnie zastanowimy się, czy musimy budować nowe. Pomyślimy, które z istniejących rozwiązań się sprawdzają i czy potrzebujemy czegoś innego".

Seibel: Jak przebiega ocena projektów tego rodzaju?

Norvig: Trzeba pokazać je osobom, które wcześniej wykonywały podobne zadania. Te osoby mogą powiedzieć: „Wygląda na to, że tu potrzebna będzie pamięć podręczna. Sam system będzie za wolny, ale wiele żądań powinno się powtarzać, dlatego bardzo pomocne będzie, jeśli zainstalujecie tu pamięć podręczną o tym rozmiarze". Należy przeprowadzić przegląd projektu, w czasie którego ludzie przyglądają się projektowi i oceniają, czy ma sens. Następnie można rozpocząć tworzenie i testowanie rozwiązania.

Seibel: Czy przeprowadzacie formalne przeglądy projektów? Pracowałeś w NASA, gdzie przeglądy projektów były bardzo formalne.

Norvig: Nie stosujemy żadnego formalnego podejścia, inaczej niż NASA. Stawka jest niższa, ponieważ łatwo można przywrócić stan systemu po wystąpieniu awarii. W NASA zwykle pierwsza awaria jest krytyczna, dlatego potrzebna jest znacznie dalej posunięta ostrożność. W Google nie przejmujemy się tym tak bardzo. W większym stopniu przeprowadzamy konsultacje niż przeglądy.

Niektóre osoby oficjalnie czytają dokumenty projektowe i oceniają je. Należy przejść ten etap, a projekt zostaje zatwierdzony. Jednak jest to znacznie mniej formalne niż w NASA. Taki przegląd ma miejsce przed uruchomieniem projektu. W trakcie jego trwania odbywają się okresowe przeglądy, jednak nie obejmują one dogłębnej analizy kodu. Bardziej chodzi o pytania: „Jak wam idzie? Czy wyprzedzacie harmonogram, czy może macie opóźnienie? Jakie duże problemy wystąpiły?" — i inne z tego poziomu.

Najbardziej formalny jest proces udostępniania oprogramowania. Używamy listy kontrolnej, która jest bardzo formalna w zakresie bezpieczeństwa. Jeśli udostępnimy rozwiązanie, czy ktoś będzie mógł się włamać i zastosować technikę XSS, aby przejąć inny komputer? Zasady są dość ścisłe.

Seibel: Pewnego razu powiedziałeś mi, że kiedy Guido van Rossum przyszedł do firmy, musiał przejść test z Pythona, a Ken Thompson — test z języka C, aby potwierdzić, że spełniają bardzo ściśle określone standardy kodowania. Czy macie równie ściśle określone standardy projektowania?

Norvig: Nie. Niektóre standardy kodowania dotyczą kwestii projektowych, natomiast w zakresie projektu swoboda jest dużo większa. Jednak określone są pewne zasady, dlatego programista musi otrzymać certyfikat, zanim zacznie przesyłać kod do repozytorium. Każdy przesyłany kod musi zostać przejrzany przez kogoś innego i zweryfikowany.

Seibel: Czy każdy kod przesyłany do repozytorium p4 depot jest przeglądany?

Norvig: Można samodzielnie tworzyć eksperymentalne rozwiązania. Ponadto występują wyjątki, kiedy można przesłać kod, który zostanie przejrzany w przyszłości. Jednak programiści powinni jak najrzadziej stosować to podejście.

Seibel: W zasadzie sprowadza się to do klasycznej analizy kodu. Pokazujesz kod, ktoś go czyta i mówi: „Tak, to jest poprawne".

Norvig: Tak. Tego dotyczył pierwszy projekt Guida. Używaliśmy standardowych narzędzi diff do sprawdzania kodu, co było dość niewygodne. Dlatego Guido napisał rozproszony system z bardziej wymyślnym wyglądem i kolorowaniem kodu, co usprawniło przeglądanie przesyłanych fragmentów.

Seibel: Firmy twierdzą, że należy przeprowadzać przeglądy, jednak bardzo rzadko to robią. Na pewnym poziomie trzeba wyszkolić ludzi do wykonywania tego zadania.

Norvig: Uważam, że zawsze tak było, dlatego ludzie to akceptują. No cóż, może nie całkowicie. Niektóre osoby potrzebują czasu, aby się przyzwyczaić. Jeden z typowych problemów ma miejsce, kiedy przychodzi nowy pracownik, który nie jest przyzwyczajony do danego sposobu pracy. Taka osoba tworzy eksperymentalne odgałęzienie i przechowuje w nim cały kod. Inni pracownicy mówią mu: „Słuchaj, nie przesłałeś jeszcze żadnego kodu". Taka osoba odpowiada: „Tak, tak, właśnie go porządkuję. Prześlę kod jutro". Mija następny tydzień, potem jeszcze następny, aż wreszcie nowy pracownik przesyła gigantyczny pakiet kodu. Oznacza to problemy. Minęło zbyt dużo czasu, trudno ocenić cały kod na raz, a niektóre rzeczy uległy zmianie. Wtedy nowa osoba widzi kłopoty, do jakich doprowadziła, i uczy się tego nie robić.

Seibel: To dotyczy pisania kodu. Czy recenzenci rozwijają jakieś umiejętności?

Norvig: Oczywiście, są osoby uznawane za lepszych recenzentów od innych. Przy przeglądach trzeba uwzględnić plusy i minusy dwóch podejść — możesz pójść do kogoś, od kogo uzyskasz wiele cennych informacji zwrotnych, lub wybrać osobę, która jak najszybciej zatwierdzi kod.

Seibel: Co sprawia, że niektórzy recenzenci są lepsi?

Norvig: Dostrzegają więcej rzeczy. Niektóre z nich to banalne kwestie, np. zła liczba odstępów we wcięciu itp., jednak czasem można usłyszeć: „Uważam, że projekt będzie bardziej przejrzysty, jeśli przeniesiesz to tutaj". Dlatego niektóre osoby częściej przeprowadzają przeglądy, a inne się tym nie zajmują.

Seibel: Następne pytanie po części jest z tym związane. Czy każdy dobry programista zostaje dobrym architektem? A może niektóre osoby są doskonałymi koderami, ale tylko na pewnym poziomie, dlatego nie należy pozwalać im rozwijać większych projektów?

Norvig: Uważam, że poszczególne osoby mają różne umiejętności. Jeden z naszych najlepszych ludzi zajmujących się wyszukiwaniem z pewnością nie jest najlepszym programistą, jeśli chodzi o wygląd kodu. Jednak kiedy spytasz: „Ten nowy czynnik — no wiesz, liczba kliknięć na stronie po zrobieniu tego i tego — jak włączyć go w wyniki wyszukiwania?", on odpowie: „Och, w wierszu czterysta dwudziestym siódmym znajduje się zmienna `alpha`. Musisz wziąć nowy czynnik, podnieść go do kwadratu, pomnożyć to przez półtora i dodać do tej zmiennej". Eksperymentujesz przez kilka miesięcy i wypróbowujesz różne rozwiązanie, po czym odkrywasz, że twój rozmówca miał rację, tylko należy użyć wartości jeden i trzy dziesiąte zamiast jeden i pięć.

Seibel: Chcesz więc powiedzieć, że ta osoba ma bardzo dobry model rozumowy działania oprogramowania?

Norvig: Perfekcyjnie rozumie kod. Inne osoby potrafią lepiej pisać kod, ale on rozumie wszystkie skutki różnych rozwiązań.

Seibel: Czy uważasz, że te kwestie są powiązane? Często mam wrażenie, że osoby, które piszą najmniej zrozumiały kod, potrafią utrzymywać w głowie najwięcej informacji. To tylko dzięki temu mogą pisać kod tego rodzaju.

Norvig: Tak, myślę, że rzeczywiście może tak być.

Seibel: Tak więc przeglądy w Google są mniej formalne niż w NASA. Jakie inne różnice występują między etosem „inżyniera" i „hakera" w najlepszym tych słów znaczeniu?

Norvig: Poważną różnicą jest struktura organizacji i sposób akceptowania oprogramowania. Google założono jako firmę związaną z oprogramowaniem, a następnie zatrudniono dyrektora wykonawczego z tytułem doktora nauk komputerowych z Berkeley i wiceprezesa do spraw sprzedaży z doświadczeniem w inżynierii komputerowej. Takie sytuacje mają miejsce w całej firmie. W NASA pracują naukowcy od rakiet! Nie znają się na oprogramowaniu. Mówią: „Oprogramowanie to zło konieczne. Prosty wiersz kodu potrafię jeszcze zrozumieć. Jeśli występuje pętla, sprawy się komplikują. Jeżeli wewnątrz pętli znajduje się rozgałęzienie, no cóż, wtedy odbiegamy od tego, co mogę rozwiązać za pomocą równania różniczkowego z teorii kontroli". Dlatego nie mają zaufania do kodu.

Seibel: Choć powinni!

Norvig: To prawda, powinni. Nie mają też zaufania do innowacji. Możesz powiedzieć: „Popatrz, jaki świetny prototyp przygotowałem". Usłyszysz odpowiedź: „To fantastyczne. Chciałbym polecieć tym na misję, kiedy zostanie już sprawdzony w dwóch innych lotach". Możesz pójść do dowolnej innej osoby i usłyszysz to samo.

Dan Goldin objął stanowisko administratora w NASA i powiedział: „Musimy pracować lepiej, szybciej i taniej. Misje kosmiczne kosztują za dużo. Lepiej przeprowadzić więcej misji. Niektóre z nich zakończą się niepowodzeniem, jednak ogólnie zrobimy więcej przy takich samych kosztach". Była to niezaprzeczalna prawda. Niestety, było to niepoprawne politycznie. Utrata statku kosmicznego jest niedozwolona, ponieważ opinia publiczna bardzo dobrze rozumie wtedy, że NASA utraciła maszynę. Ludzie nie wiedzą jednak, że istnieje różnica między statkiem kosmicznym za sto milionów dolarów i za miliard. Przy tym nie musi dochodzić do dziesięciokrotnej utraty stu milionów zamiast jednego miliarda. Dlatego przyjmowane podejście nigdy nie było do końca słuszne.

Seibel: Jaki był najgorszy błąd, który musiałeś wykryć?

Norvig: Chyba najpoważniejsze konsekwencje miały błędy, których nie popełniłem sam, ale które musiałem naprawić. Chodziło o usterki w programie Mars w 1998 roku. Jeden błąd polegał na użyciu funtostóp zamiast newtonów. Inny, choć tego nie jestem w 100% pewien, dotyczył przedwczesnego wyłączania silników z uwagi na problem z oprogramowaniem.

Seibel: Czytałem jeden z raportów na temat maszyny Mars Climate Orbiter dotyczących problemu z użyciem funtostóp zamiast newtonów. Byłeś jedynym przedstawicielem nauk komputerowych w grupie. Czy uczestniczyłeś w rozmowach z informatykami, aby ustalić, w czym tkwi problem?

Norvig: Po fakcie rozwiązanie problemu było całkiem proste, ponieważ informatycy znali symptomy błędu. Na tej podstawie mogli zlikwidować skutki i szybko odkryli przyczynę. Następnie miały miejsce analizy, dlaczego doszło do pomyłki. Uważam, że wynikało to z połączenia różnych czynników. Jednym z nich był outsourcing. Prace były prowadzone wspólnie przez firmy JPL z Pasadeny i Lockheed-Martin z Kolorado. Uczestniczyły w tym dwie osoby z dwóch różnych zespołów, które nie pracowały wspólnie i nie spotykały się przy lunchu. Jestem przekonany, że gdyby było inaczej, informatycy rozwiązaliby problem. Jednak zamiast tego jeden z nich wysłał e-mail: „Coś jest nie tak z tymi pomiarami. Wygląda na to, że są trochę niedokładne. Nie jest to duża rozbieżność, prawdopodobnie wszystko jest w porządku, ale…".

Seibel: Czy miało to miejsce w czasie lotu?

Norvig: Tak. W czasie lotu było wiele okazji do wykrycia problemu. Informatycy wiedzieli, że coś jest nie tak, i wysłali wspomniany e-mail, ale nie wprowadzili informacji do systemu śledzenia błędów. NASA ma bardzo dobrą kontrolę w zakresie śledzenia błędów, dlatego na późniejszym etapie lotu ktoś musiałby zatwierdzić tę informację. Zamiast tego pojawił się tylko nieformalny e-mail, który pozostał bez odpowiedzi. W JPL uznano: „Pewnie w Lockheed-Martin rozwiązali ten problem". W Lockheed stwierdzono: „No tak, JPL nie zadaje więcej pytań — na pewno już ich to nie martwi".

Wystąpił więc problem z komunikacją. Inny kłopot był związany z powtórnym wykorzystaniem oprogramowania. Firma miała bardzo dobre testy elementów krytycznych dla misji, a w poprzedniej misji dane rejestrowane w funtostopach nie były krytyczne — znajdowały się w dzienniku, którego nie używano do nawigacji. Dlatego kod sklasyfikowano jako niekrytyczny dla misji. W nowej misji ponownie wykorzystano większość kodu, jednak zmodyfikowano sposób nawigowania. Dane z pliku dziennika stały się obecnie danymi wejściowymi dla systemu nawigacji.

Seibel: Problem polegał więc na tym, że po jednej stronie generowano plik z danymi w funtostopach przekazywany do oprogramowania, które obliczało dane wejściowe dla systemu nawigacji i oczekiwało wartości w newtonach?

Norvig: To prawda. Inną podstawową przyczyną problemu był za duży wiatr słoneczny. Pojazd kosmiczny jest asymetryczny i posiada panele słoneczne. Wiatr słoneczny nieco przekręca pojazd, dlatego trzeba odpalić rakiety, aby przywrócić jego pozycję. Nowy pracownik w Lockheed skontaktował się z producentem rakiet, a w ich specyfikacji używano funtostóp. Dlatego pracownik uznał, że wykorzysta te jednostki, i rejestrował dane przy ich użyciu. Nie wiedział, że NASA oczekuje danych w systemie metrycznym.

Seibel: W czasie lektury raportu byłem zaskoczony podejściem NASA: „No cóż, problem wynikał z błędu w oprogramowaniu, ale mieliśmy wiele okazji do tego, aby zauważyć, że statek znajduje się w nieoczekiwanym miejscu. Powinniśmy to dostrzec. Powinniśmy poprawić pozycję, mimo iż liczby, z których korzystaliśmy, były całkowicie niepoprawne z powodu głupiej usterki oprogramowania". Bardzo spodobało mi się to nastawienie.

Norvig: To prawda, w NASA zwracali uwagę na proces pracy.

Seibel: Czy często spotykane są tak poważne błędy w oprogramowaniu, o których nigdy się nie dowiemy, ponieważ inne procesy zapewniają działanie systemu?

Norvig: Tak sądzę. Pomyśl o błędach w oprogramowaniu komputera. Są ich miliony, a przecież zwykle nie dochodzi do awarii.

Seibel: Jednak oprogramowanie dla pojazdów kosmicznych kosztuje podobno około półtora tysiąca dolarów za wiersz z uwagi na staranność przy pisaniu kodu i brak błędów. Czy są to zwykłe kłamstwa?

Norvig: Nie, prawdopodobnie jest to prawda. Jednak nie wiem, czy jest to optymalne podejście. Uważam, że lepsze efekty można by uzyskać przez pisanie mniej doskonałego oprogramowania.

Seibel: Tańsze oprogramowanie i lepsze operacje?

Norvig: Tak, będzie to efektywniejsze z uwagi na zakres szkoleń, które przechodzą astronauci, żeby poradzić sobie z zadaniami nieobsługiwanymi przez oprogramowanie. NASA wsadza astronautów do symulatorów i stawia ich w różnych sytuacjach. Kiedy wystąpi problem, pojawia się ekran i przesuwają się po nim dane. Nie można ich zatrzymać, nie można się cofnąć, nie można wyświetlić podsumowania ważnych informacji. Astronautów trzeba wyszkolić tak, aby wiedzieli, że kiedy zobaczą pewne informacje, oznacza to określone zdarzenia. Pojawia się setka kolejnych komunikatów z informacją o awarii części elektrycznej, a astronauci muszą się nauczyć, że oznacza to awarię jednego elementu i kaskadowe zgłoszenie problemów we wszystkich pozostałych. Dlaczego nie można zrobić tego za pomocą oprogramowania? NASA nie próbuje zastosować tego podejścia, bo nie chce mieć problemów z oprogramowaniem.

Seibel: Przejdźmy do innej kwestii. Jakie techniki i narzędzia diagnostyczne preferujesz? Instrukcje `print`, formalne dowody, a może debuggery symboliczne?

Norvig: Używam różnych technik, w zależności od tego, w jakim środowisku pracuję. Czasem stosuję środowisko IDE z rozbudowanymi możliwościami w zakresie śledzenia, a czasem korzystam z edytora Emacs i nie mam takich narzędzi. Oczywiście, także śledzenie i instrukcje `print`. I myślenie. Piszę małe przypadki testowe i obserwuję ich działanie. Dzielę też mechanizmy na części, aby zobaczyć, w którym miejscu przypadek testowy powoduje błąd. Muszę przyznać, że często ostatecznie piszę kod od nowa. Czasem robię to nawet wtedy, kiedy nie znalazłem błędu. Dochodzę do momentu, kiedy po prostu czuję, że usterka znajduje się w danym miejscu. Nie podoba mi się określony fragment. Coś jest z nim nie tak. Zamiast wprowadzać drobne poprawki, po prostu usuwam kilkaset wierszy kodu, piszę ten fragment od początku i często błąd zostaje zlikwidowany.

Czasem w takiej sytuacji mam poczucie winy. Czy oznacza to moją porażkę? Nie zrozumiałem, w czym tkwił błąd. Nie znalazłem go. Po prostu zrzuciłem bombę na dom, wysadziłem wszystkie usterki w powietrze i zbudowałem dom od nowa. W pewnym sensie błąd mnie zwiódł. Skoro jednak doszedłem do właściwego rozwiązania, może jest to dobre podejście. Ukończyłem pracę szybciej, niż gdybym znalazł błąd.

Seibel: Co myślisz o asercjach lub niezmiennikach? Jak bardzo formalnie w trakcie pisania kodu traktujesz tego rodzaju elementy?

Norvig: Stosuję raczej nieformalne podejście. Nie używałem języków, gdzie takie elementy wymagają bardziej formalnych mechanizmów niż pisanie deklaracji. Weźmy np. niezmienniki pętli. Zawsze uważałem, że więcej z nimi kłopotów niż z nich pożytku. Czasem natrafiam na problem nieskończonej pętli, jednak zdarza się to rzadko, a mam wrażenie, że formalne mechanizmy spo-

walniają pracę. Jeśli wystąpi problem, debugger poinformuje o tym, w której pętli program utknął. Sądzę, że jeśli piszę krytyczne oprogramowanie działające w maszynach, w których bezawaryjność jest bardzo ważna, warto dowodzić poprawności wszystkich elementów. Jeśli jednak mam uzyskać pierwszą działającą wersję programu lub debugować go, wolę szybko poruszać się naprzód, zamiast przejmować potrzebną później formalną specyfikacją.

Seibel: Czy kiedykolwiek zrobiłeś coś, aby bezpośrednio wyciągnąć naukę z popełnionych błędów?

Norvig: Tak. Uważam, że jest to dość ciekawe, i żałuję, iż nie robiłem tego częściej. Prowadzę obecnie rozmowy na temat tego, czy mogę przeprowadzić w firmie — a może i na całym świecie — eksperymenty, aby lepiej zrozumieć pewne kwestie z tego obszaru. Chcę się dowiedzieć, jak klasyfikujemy błędy, ale też poznać czynniki związane z produktywnością. Skąd wiadomo, że ktoś jest produktywny? Czy są to osoby określonego typu? Jakie cechy tych osób sprawiają, że są bardziej produktywne? Ponadto uważam, że ciekawsze jest poznanie kontrolowalnych czynników, które pozwalają poprawić wyniki pracowników. Jeśli udostępnienie ludziom większych monitorów zwiększa produktywność o określony procent, prawdopodobnie należy im je zapewnić.

Seibel: Ludzie znienawidzą cię, jeśli odkryjesz, że małe monitory zwiększają produktywność.

Norvig: To prawda. Jeśli zapewnienie ciszy jest ważne, prawdopodobnie należy to zrobić, jeżeli jednak istotna jest komunikacja między członkami zespołu, także trzeba ją umożliwić. Jak pogodzić te dwa czynniki?

Zacząłem się zastanawiać, jak się tego dowiedzieć. Jak zaplanować eksperyment? Co śledzić? Czy istnieją już dane, do których możemy dotrzeć za pomocą kwestionariuszy? Czy w ogóle musimy prowadzić eksperymenty?

Seibel: Często można spotkać się z twierdzeniem, że różnice w produktywności między programistami sięgają rzędów wielkości. Czytałem krytyczny tekst na ten temat. Według niego, badania, w których uzyskano te wyniki, przeprowadzono dość dawno. Od tego czasu w programowaniu zmieniło się wiele rzeczy, które mogły wpływać na różnice. Niektóre osoby w trakcie badań nadal stosowały techniki przetwarzania wsadowego, natomiast inne używały wielozadaniowych środowisk programistycznych.

Norvig: Nie sądzę, aby różnice wynikały tylko z tego. Różnice występują także w ramach organizacji, gdzie używane są mniej więcej te same narzędzia. Pamiętam też krytyczne opinie na temat badań, w których wykryto korelację, ale nie określono przyczyn i skutków. Jeśli odkryjemy, że programiści w dużych narożnych biurach są bardziej produktywni, to czy wynika to z nagradzania dobrych programistów gabinetami, czy może to praca w gabinecie zwiększa wydajność? Nie można tego jednoznacznie stwierdzić.

Seibel: Czy programowanie nadal sprawia ci taką samą przyjemność jak wtedy, kiedy zaczynałeś?

Norvig: Tak, ale frustrujące jest to, że nie wiem wszystkiego. Nie programuję zbyt wiele, dlatego zapominam niektóre rzeczy. Pojawiają się też nowinki. Naprawdę powinienem przeprojektować swoją witrynę i zastosować skrypty JavaScript po stronie klienta. Powinienem użyć języka PHP lub czegoś podobnego, ale nie zdobyłem się na to, by nauczyć się tych wszystkich technologii i poprawić witrynę.

Seibel: Czy uważasz, że programowanie to zabawa dla młodych ludzi?

Norvig: Myślę, że pod pewnymi względami młody wiek pomaga. Oczywiście, zatrudniamy wielu wyjątkowo zdolnych ludzi na różnych stanowiskach i w różnym wieku. Uważam, że przewaga młodych osób wynika z dużego znaczenia obejmowania rozumem całego programu, całego problemu.

Związane jest to z umiejętnością koncentracji. Sądzę, że młodym ludziom przychodzi to łatwiej. Ich mózgi lepiej sobie z tym radzą, a może takie osoby są po prostu mniej zaabsorbowane innymi sprawami? Jeśli ktoś ma dzieci, rodzinę itd., nie może poświęcać na pracę równie wielu godzin, co inni ludzie. To jednak tylko część obrazu sytuacji. Z drugiej strony, starsze osoby mają większe doświadczenie, dlatego pod niektórymi względami mogą nadrobić braki. Potrafią wykonać więcej zadań, ponieważ wiedzą, jak to zrobić.

Seibel: Jednym z aspektów współczesnego stylu programowania jest — jak mówiłeś — konieczność szybkiego przyswajania informacji przez programistów. Jak radzisz sobie z koniecznością zrozumienia dużego bloku kodu, którego wcześniej nie widziałeś?

Norvig: Odbywa się to po części statycznie, a po części dynamicznie. Zaczynam od przeczytania kodu i staram się go zrozumieć. Następnie śledzę, które funkcje wywołują inne funkcje, gdzie program spędza najwięcej czasu i jak wygląda przepływ sterowania. Później próbuję coś zrobić, np. wprowadzić drobną zmianę lub zajrzeć do bazy danych z problemami i przyjrzeć się jednemu z nich. Aby to zrobić, muszę poznać mały fragment kodu. To tylko jedna niewielka część, ale poznaję ją i mogę przejść do następnej.

Seibel: Czy kiedykolwiek stosowałeś programowanie piśmienne w stylu Knutha?

Norvig: Nigdy nie używałem jego narzędzi w pierwotnej postaci. Oczywiście, pisałem makra itd. Ponadto używałem narzędzia Javadoc i podobnych rozwiązań. Pod wieloma względami programowanie w Lispie skłania do rozwijania swojego systemu, dlatego ostatecznie stosuję coś na kształt programowania piśmiennego. Wymyślam własne makra na potrzeby programowania danej aplikacji. Częścią rozwiązania jest dokumentacja, częścią dane, a częścią kod. Oczywiście, stosowałem to podejście. Od niedawna, niezależnie od tego, jakiego języka używam — czy jest to Java, czy Python, czy coś innego — starannie piszę przypadki testowe i na nich opieram dokumentację.

Knuth w *Literate Programming* tak naprawdę chciał powiedzieć, jaki jest najlepszy porządek pisania książki. Zakładał przy tym, że ktoś zamierza przeczytać całą książkę i chce, aby odbywało się to w logicznej kolejności. Ludzie już tego nie robią. Nie chcą czytać książek. Oczekują indeksu, aby określić: „Jaki najmniejszy fragment książki muszę przeczytać? Chcę tylko znaleźć trzy akapity, których potrzebuję. Pokaż mi je, a potem przejdę dalej". Uważam, że to istotna zmiana.

Seibel: Zastanawiam się, czy nie można stosować programowania piśmiennego we współczesnym stylu. Narzędzia Knutha zapewniają indeks i fantastyczne odwołania. Może współczesne podejście do programowania piśmiennego będzie polegać na innym uporządkowaniu książki — zarówno całego programu, jak i fragmentów, które można poznawać pojedynczo?

Norvig: Nie jestem pewien. Uważam, że Knuth rozwiązywał problem, który w dużym stopniu już nie istnieje. Po części wynikało to z tego, że chciał ułożyć elementy w porządku liniowym, a nie sieciowym lub opartym na wyszukiwaniu. Częściowo wynikało to z ograniczeń. Knuth początkowo używał chyba Pascala. Język ten jest dość ścisły ze względu na to, co trzeba zadeklarować na początku. Nie zawsze odbywa się to w pożądanej kolejności. Współczesne języki zapewniają większą swobodę w tym zakresie, dlatego uważam, że problem jest mniejszy.

Seibel: Wspomniałeś, że czytałeś w „Scientific American" kod programu Stracheya do gry w warcaby. W eseju „Teach Yourself Programming in Ten Years" piszesz o znaczeniu czytania kodu. Jaki kod czytałeś w czasie nauki programowania?

Norvig: Czytałem dużo kodu firmy Symbolisc, ponieważ to właśnie było dostępne, kiedy studiowałem w Berkeley.

Seibel: Czy czytałeś ten kod dlatego, że był dostępny i interesujący? A może próbowałeś zrozumieć pewne zachowania programów, które obserwowałeś?

Norvig: Obie przyczyny były istotne. Czasem po prostu próbowałem dowiedzieć się, jak coś działa, a czasem potrzebowałem informacji do rozwiązania problemu.

Seibel: Jak podchodzisz do czytania kodu w ramach ogólnego rozwoju?

Norvig: Zwykle jest to oparte na zainteresowaniach. „Popatrzcie, ten system plików umożliwia odczyt plików przez sieć za pomocą protokołu używanego lokalnie. Ciekawe, jak to działa?". Potem mówisz: „Może chodzi o funkcję `open`?". Zaglądasz do tej funkcji i stwierdzasz: „Aha, funkcja wywołuje ten inny kod". Patrzysz na ten kod i mówisz: „Już wiem, jak to działa".

Seibel: Czy czytałeś któryś z piśmiennych programów Knutha w formie książkowej?

Norvig: Oczywiście, sięgnąłem po te książki i przekartkowałem. Można powiedzieć, że zajrzałem do nich, ale ich nie studiowałem.

Seibel: A co z serią *The Art of Computer Programming*? Niektóre osoby, z którymi rozmawiałem na jej temat, przeczytały ją od deski do deski. Część osób ma ją na półce i korzysta z niej jak z podręcznika. Inni po prostu mają ją na półce.

Norvig: Przez pewien czas używałem jej jako podstawki pod monitor, ponieważ była największą serią książek, jaką posiadałem, i miała odpowiednią wysokość. Było to wygodne, ponieważ zawsze miałem te książki pod ręką. Podejrzewam, że byłem bardziej skłonny korzystać z nich jak z podręcznika, ponieważ zawsze miałem je przed sobą.

Seibel: Jednak za każdym razem, kiedy chciałeś do nich zajrzeć, musiałeś podnosić monitor?

Norvig: Nie, miałem wydanie w pudełku. Musiałem mocno pociągnąć, ale mogłem wyjąć jedną książkę z pudełka. Teraz rzadziej korzystam z książek jak z podręczników — zwykle używam wyszukiwarki.

Seibel: Tylko dlatego, że jest to wygodniejsze?

Norvig: Jest to wygodne. Myślę, że także dlatego, iż prawdopodobnie jestem bardziej nastawiony na realizację celów. Knuth jest dobry, jeśli chcesz wiedzieć wszystko na dany temat. Jednak ja zwykle chcę wiedzieć, czy A jest lepsze od B, lub określić złożoność asymptotyczną danego rozwiązania. Kiedy już to ustalę, nie potrzebuję wszystkich szczegółów.

Seibel: Jesteś programistą. Uważasz, że jesteś naukowcem, inżynierem, artystą czy rzemieślnikiem?

Norvig: No cóż, po zapoznaniu się z tytułami różnych książek i innych tekstów zawsze uważałem, że prawidłowa odpowiedź to „rzemieślnik". Myślałem, że nazwa „artysta" jest nieco pretensjonalna, ponieważ sztuka ma być piękna lub wywierać emocjonalny wpływ, a ja nie próbuję uzyskać niczego podobnego. Oczywiście, chcę, aby programy były pod niektórymi względami ładne. Czasem mam wrażenie, że poświęcam na to zbyt wiele czasu. Znalazłem się w sytuacji, kiedy mogłem sobie pozwolić na stwierdzenie: „Wspaniale, mam czas, aby wrócić do kodu i trochę go uatrakcyjnić". Kiedy piszesz kod przeznaczony do publikacji, poświęcasz na jego upiększanie więcej czasu, niż gdybyś robił to ze względu na rozwój zawodowy.

Jednak nie myślę o tym jak o sztuce. Uważam, że *rzemiosło* to właściwe słowo. Możesz zrobić krzesło, które dobrze wygląda, ale przede wszystkim ma być funkcjonalne — to tylko krzesło.

Seibel: W jaki sposób stwierdzasz, że ktoś jest dobrym programistą, zwłaszcza przy naborze? Zatrudniliście wielu programistów i z pewnością staracie się, aby byli naprawdę dobrzy. Jak to zrobić?

Norvig: Nadal nie wiemy.

Seibel: Firma Google jest znana z zadawania zagadek w trakcie rozmów rekrutacyjnych. Czy uważasz, że jest to dobre podejście?

Norvig: Moim zdaniem, nie jest ważne, czy ludzie potrafią rozwiązywać zagadki, czy nie. Nie lubię podchwytliwych pytań. Uważam, że ważne jest, aby postawić kandydata w sytuacji technicznej, a nie tylko porozmawiać z nim i uznać, iż jest miłym człowiekiem. Choć — oczywiście — ważne jest, aby można było się z nim dogadać. Jednak przede wszystkim trzeba zobaczyć, czy w kontekście technicznym kandydat potrafi zrobić to, co twierdzi, że potrafi. Jest wiele sposobów, aby to sprawdzić. Często można ustalić to na podstawie życiorysu. Dla nas chyba najlepszą wskazówką jest to, czy dana osoba współpracowała wcześniej z jednym z naszych pracowników, a ten pracownik może to poświadczyć. Jednak — mimo to — staramy się sprawdzić tę osobę w czasie rozmowy rekrutacyjnej w siedzibie firmy. Chodzi głównie o zobaczenie, jak kandydat myśli, jak współpracuje z innymi i czy zna podstawowe zagadnienia. Czy potrafi powiedzieć: „No cóż, aby to rozwiązać, muszę znać A, B i C", po czym zaczyna łączyć elementy. Kandydat może to zademonstrować, nawet jeśli nie rozwiąże zagadki. Może powiedzieć: „Spróbuję rozwiązać zagadkę w taki sposób. Najpierw pomyślę o tym, potem zrobię to, a następnie — hmm, ale tę część nie do końca rozumiem". Niektóre osoby „łapią" te drobiazgi, a inne nie. Kandydat może uzyskać dobry wynik bez „załapania", o ile zademonstruje podstawowe umiejętności i sprawne myślenie. Bardzo przydatne jest też poproszenie kandydata — jeśli ma być programistą — o napisanie kodu na tablicy. Jest tak, ponieważ niektóre osoby zapomniały pewne rzeczy lub ich nie wiedzą. Można to szybko dostrzec.

Seibel: Jest to więc tylko negatywny wskaźnik? Kiedy ktoś nie potrafi napisać sensownego kodu, jest to zły znak. Jeśli jednak sobie poradzi, trudno powiedzieć, czy będzie pisał naprawdę dobry kod w ogólniejszym kontekście.

Norvig: To prawda. Do pewnego stopnia można to stwierdzić, ale na innych poziomach jest to niemożliwe. Analizowaliśmy to dość dokładnie, ponieważ otrzymujemy wiele podań i przyglądamy się im na dwóch poziomach. Po pierwszy, czy rozmawiamy z właściwymi osobami spośród wszystkich, które przesłały życiorys? Po drugie, czy na podstawie przeprowadzonych rozmów rekrutacyjnych zatrudniamy odpowiednie osoby?

Seibel: Jak to mierzycie? Nie wiecie nic o osobach, z którymi nie rozmawialiście lub których nie zatrudniliście.

Norvig: Rzeczywiście, to trudne zadanie. Na obu poziomach mamy dostęp tylko do części grupy, dlatego występuje problem niepełnych danych. Nasz sposób myślenia sprowadza się do tego: „Jak wyglądały życiorysy dobrych pracowników, z którymi rozmawialiśmy?". Następnie próbujemy znaleźć więcej podobnych osób. Czy określona liczba lat doświadczenia ma znaczenie? Czy ważna jest praca nad projektami o otwartym dostępie do kodu źródłowego? Jak ma się to do wygrania konkursu programistycznego?

Seibel: Czy rzeczywiście umieszczacie wszystkie te informacje bazie danych?

Norvig: Tak robimy. Kiedy prowadzimy rekrutację, mamy wyniki, które informują: „Wskaźnik z życiorysu wynosi tyle a tyle, a wskaźnik z rozmowy rekrutacyjnej wynosi tyle a tyle". Nie traktujemy tych danych jak wyroczni, ale jako jedną z danych wejściowych obok wszystkich innych informacji zwrotnych.

Seibel: Czy osoby prowadzące rozmowy rekrutacyjne otrzymują wcześniej te dane?

Norvig: Nie, przeglądamy je tylko na spotkaniu komitetu rekrutacyjnego po zebraniu wszystkich informacji zwrotnych. Odkryliśmy ciekawą rzecz, próbując przewidzieć, jak dobrze zatrudniona

osoba sprawdzi się po roku lub po dwóch. Jednym z najlepszych wskaźników sukcesu w firmie było uzyskanie najgorszego możliwego wyniku na jednej z rozmów. Oceniamy ludzi w skali od jeden do cztery. Jeśli ktoś otrzyma jedynkę na jednej z rozmów, jest to naprawdę dobrym predyktorem sukcesu.

Seibel: Jednak taka osoba musiała wypaść wyjątkowo dobrze w innym obszarze, skoro została zatrudniona?

Norvig: To prawda, o to chodzi; 99% osób, które otrzymały jedynkę na jednej z rozmów, nie zostaje zatrudnionych. Jednak w przypadku pozostałych, aby zostały one przyjęte, ktoś musiał z taką pasją wypowiadać się na ich temat, że walił pięścią w stół i mówił: „Muszę zatrudnić tego człowieka, ponieważ widzę w nim coś wielkiego. Ten, kto źle go ocenił, nie ma racji. Muszę stanąć w obronie tego człowieka i stawiam na szalę swoją reputację".

Seibel: W Google jesteś otoczony przez programistów najwyższej klasy. Komputery i oprogramowanie są wszechobecne w naszym społeczeństwie. Czy uważasz, że wszyscy muszą znać podstawy programowania, aby funkcjonować w świecie, w którym żyją, i go rozumieć?

Norvig: Prawdopodobnie warto, aby wykształcona osoba rozumiała powstawanie oprogramowania na podobnym poziomie, na jakim rozumie budowanie samochodów. Inną ciekawą kwestią jest to, w jakim stopniu dobrze poinformowany obywatel musi być programistą. Przeciętna osoba z pewnością potrafi używać edytora tekstu, a wielu ludzi umie korzystać z arkuszy kalkulacyjnych. Jeśli masz trochę doświadczenia w używaniu arkuszy, zaczynasz być programistą.

Wiele prób z obszaru programowania przez użytkowników i programowania dla każdego nie zakończyło się sukcesem. Nie wiem, jak proste jest programowanie. Może jedne osoby cechują się pewnym sposobem myślenia, który sprawia, że potrafią łatwo nauczyć się programowania, natomiast dla innych jest to trudne? A może wybraliśmy zły model nauczania? Może istnieje prosty model, który mógłby wpłynąć na programowanie, gdybyśmy go wymyślili?

Seibel: Wielu ludzi, z którymi rozmawiałem w czasie pisania tej książki i w innych okolicznościach, zajęło się informatyką, ponieważ sprawiało im to przyjemność i uważali, że może ona zmienić świat. Niektóre osoby, z którymi przeprowadziłem wywiad, miały takie podejście, a teraz są zawiedzione tym, w jak niewielkim stopniu świat się zmienił. Co myślisz na ten temat?

Norvig: Uważam, że znajduję się w odpowiednim miejscu. Mamy setki milionów użytkowników i możemy coś dla nich zmienić. Możemy szybko udostępniać im nowe usługi. Uważam, że to wspaniałe. Nie mogę sobie wyobrazić nic innego, dzięki czemu mógłbym wywierać tak duży wpływ na ludzi.