Jak zmieniało się myślenie o programowaniu i jak powstała metodyka *BDD*.

Programowanie było jeszcze bardzo młodą dziedziną nauki, kiedy pod koniec lat 50ych XX wieku w Massachusetts Institute of Technology pojawiły się pierwsze próby wprowadzenia „obiektów" jako jej podstawowego budulca. Cyfrowy obiekt miał reprezentować nowy poziom abstrakcji, stanowić most pomiędzy ludzkim językiem i jego postrzeganiem świata, a pamięcią komputera, w której wszystko musi mieć swoje miejsce, wartość; a przy tym koncepcja ta miała być instynktownie prosta, jak pojęcie podmiotu w zdaniu. Stworzony w 1958 r. przez Johna McCarthy'ego (nagroda Turinga w 1971 r. za prace nad podstawami sztucznej inteligencji) język LISP oferował „atomy" – protoplastów dzisiejszych obiektów – i system ich atrybutów na długo przed tym, gdy dr Alan Kay (który w 2003 r. otrzymał nagrodę Turinga za wkład w prace nad językami programowania, jak również interfejsami graficznymi) w 1966 r. zaproponował termin object-oriented programming[1] (ang. programowanie zorientowane obiektowo – choć w języku polskim używa się prostszej nazwy „programowanie obiektowe"). Język LISP nie był dla Kaya jedyną inspiracją – na przełomie lat 1960-61 Ivan Edward Sutherland (laureat nagrody Turinga w 1988 r.) rozwijał swój autorski projekt Sketchpad, kładąc kamień węgielny pod całą rodzinę programów typu CAD (Computer-aided Design, ang. Projektowanie Wspomagane Komputerowo). Jednak ten pierwszy w historii program z interfejsem graficznym (którego Kay stał się natychmiast gorącym orędownikiem), napisany został w języku, który przeznaczony był tylko dla niego; poza tym miał inną wadę – działał na superkomputerze Lincoln TX-2, o którym dr Alan Kay powiedział: „Był to ostatni komputer w Ameryce, który miał własny dach"[2].

Pierwszy język programowania, w którym explicite zdefiniowano obiekty powstał jednak nie na terenie Stanów Zjednoczonych, ale w odległej Norwegii – był to język Simula I, autorstwa Kristena Nygaarda i Ole-Johana Dahla (nagroda Turinga przyznana obojgu w 2001 r. za pionierskie prace nad programowaniem obiektowym). Zgodnie z nazwą, rodzina języków Simula (później powstał jeszcze Simula 67) służyła do przeprowadzania symulacji – nie tylko wspomaganych komputerowo, ale także wykonywanych w całości w pamięci maszyny. Nowatorskie rozwiązania zapewniły norweskim informatykom sławę. W roku 1962 Nygaard został zaproszony do laboratoriów UNIVAC, producenta komputerów „biznesowych", czyli maszyn typu mainframe. W czasie swojego pobytu w USA prezentował rozwiązania programowania obiektowego, które sam stworzył. Wkrótce Simulę zaimplementowano w amerykańskim komputerze

1 http://userpage.fu-berlin.de/~ram/pub/pub_jf47ht81Ht/doc_kay_oop_en (stan z dnia 12.07.2011)

2 W 1987r, na potrzeby *University Video Communications*, dr Alan Kay przeprowadził prezentację pionierskich systemów komputerowych, które jako pierwsze w latach 60-tych i 70-tych przedstawiły światu interfejs użytkownika; w połowie piątej minuty filmu rozpoczyna się prezentacja programu *Sketchpad* – http://www.archive.org/details/AlanKeyD1987 (stan z dnia 13.07.2011)

Burroughs B5500, jak również i w radzieckim Ural-16.

W 1966 r. brytyjski informatyk Sir Charles Antony Richard Hoare (laureat nagrody Turinga w 1980 r.) przedstawił definicję „klasy" – formalnego opisu danego rodzaju obiektu, przy założeniu, że programista może stworzyć podklasę – nowy „gatunek" obiektu, który „dziedziczyć" będzie atrybuty z klasy z której się wywodzi, dokonując na nich własnych zmian, analogicznie jak podczas ewolucji czynią to gatunki potomne w przyrodzie. Rok później powstał *Simula 67* – był pierwszym językiem w historii, który wprowadził klasy i pierwszym wyposażonym w system garbage collection (ang. zbieranie śmieci) – proces działający w tle programu, „czyszczący" zwalniane komórki pamięci tak, by łatwo zająć je mogły inne programy. W maju tego roku, w Oslo, przyjęto formalną definicję tego języka, a już w czerwcu Ole-Johan Dahl postulował wprowadzenie paradygmatu, który miał traktować typy prymitywne (liczby, znaki alfanumeryczne, wartości logiczne) jako obiekty określonej klasy, w ten sposób unifikując wszystkie dane programu do postaci obiektu. Jego propozycja została jednak odrzucona.

Dr Alan Kay, zafascynowany biologią, widział przyszłość programistyki w kontynuowaniu prac nad „obiektem" – cyfrowym bytem, opisanym atrybutami, reagującym na komunikaty, wysyłane przez inne byty. Atomy *LISPa* były dosyć prymitywne, obejmowały tylko liczby i łańcuchy znaków (zwane symbolami); program *Sketchpad* traktował geometryczne wierzchołki, krawędzie, a również całe figury, jako obiekty, ale jakkolwiek innowacyjne rozwiązania wprowadzał (jest pierwszym programem wykorzystującym ideę „ekranu dotykowego", chociaż „dotyk" odbywał się tutaj za pomocą świetlnego rysika, więc system nie reagowałby na dotyk ludzkiej dłoni), w swojej implementacji ograniczony był do konkretnego rozwiązania – projektowania graficznego. *Simula I* i *67*, były krokiem w dobrym kierunku, ale choć języki te jako pierwsze wprowadziły w życie koncepcje, o których Kay mówił na konferencjach otwarcie już od roku, chociaż dawały programistom potężne narzędzie w postaci cyfrowego obiektu i jego klasy, to już w 1969r Kay (pracujący wtedy w ośrodku *Xerox* w Palo Alto) poprowadził zespół w pracach nad językiem nowej generacji.

W roku 1972 powstał język *Smalltalk* (wśród członków zespołu Kaya należy wymienić Daniela Ingallsa, który pierwszą wersję nowego języka zaimplementował w ciągu zaledwie kilku dni na podstawie jednej strony notatek, w efekcie wygrywając zakład z Kayem). Na przestrzeni lat 1972-80 powstało kilka wersji rozwojowych tego języka. Zaprezentowany w 1980 r. *Smalltalk-80* z niewielkimi zmianami przetrwał do dziś i wciąż jest wykorzystywany w praktyce. Podstawową ideą tego języka było „traktowanie wszystkiego jako obiekt" – tak, jak to w 1967 r. w Oslo postulował Ole-Johan Dahl. Ujednolicenie wszystkich „podmiotów" biorących udział w konstrukcji programu, poza spełnieniem osobistego marzenia dra Alana Kaya o języku prawdziwie „obiektowo-zorientowanym", miało bardzo konkretny cel – zmienić sposób myślenia programistów.

Pracując jeszcze w MIT, dr Kay miał okazję obserwować początki rewolucji w komunikacji człowieka z maszyną. Wśród prekursorów w tej dziedzinie należy wymienić rzecz jasna wspomnianego wcześniej Ivana Sutherlanda, ale przede wszystkim

Douglasa Carla Engelbarta (nagroda Turinga w 1997 r.) – wynalazcę myszki komputerowej i hipertekstu – systemu wzajemnie powiązanych dokumentów (leksji), które użytkownik nawiguje nie w ustalonej kolejności, a wg. własnego uznania, wybierając odpowiednie łącza – bez tego wynalazku internet, jakim go znamy, nigdy by nie powstał. Mimo wszystko te koncepcje całkowicie odmiennego podejścia do komunikacji z maszyną przez blisko 20 lat pozostawały tylko w sferze ciekawostek. Doświadczenia wyniesione przez dr Alana Kaya z tych obserwacji umocniły go w przekonaniu, że komputery zmieniły się wystarczająco i nadszedł czas by zmienić „drugą stronę" dialogu: użytkownika i jego podejście. Przyświecała mu przy tym myśl kanadyjskiego filozofa i teoretyka komunikacji Herberta Marshalla McLuhana (jako człowiek niezwiązany z amerykańskimi ośrodkami badań i rozwoju informatyki przewidział powstanie sieci komputerowej i unifikację mediów na 30 lat zanim został stworzony internet; jako pierwszy użył też powiedzenia, że świat dąży do tego, by stać się „globalną wioską"), który żartobliwie mówił, że „nie wiemy kto wymyślił wodę, ale na pewno nie były to ryby" – przekaz był dla Kaya jasny, rewolucji obsługi komputera nie można dokonać tylko w gronie naukowców, którzy od lat stykają się z najnowszymi technologiami. Potrzebni byli użytkownicy „z zewnątrz".

Smalltalk posłużył w serii eksperymentów w drugiej połowie lat 70ych, w których wzięło udział kilkaset dzieci w wieku lat 12-13 z okolicznych szkół w Palo Alto. W ciągu trwających dwa szkolne semestry zajęć nauczono młodych programistów obsługi komend *Smalltalk* poczynając od tych prostych, odziedziczonych po języku *Logo* – czyli tworzenia grafiki na ekranie za pomocą funkcji – aż po naukę pisania własnych programów. Najważniejszym elementem eksperymentu były zajęcia prowadzone dla uczniów przez uczniów ze starszych klas. Okazało się, że dzieci, które nie miały wcześniej żadnej styczności z informatyką, nie będące „zakładnikami" starych koncepcji programowania, były w stanie pisać kompletne programy użytkowe, jak edytory grafiki i animacji, a ich kod był wyjątkowo koherentny w porównaniu z tym jak wyglądałby on w języku proceduralnym. Zespół *Xeroxa* z Palo Alto dowiódł tym samym, że paradygmat „obiektu" jest naturalny, instynktownie rozumiany nawet przez dzieci, mające zaledwie kilkumiesięczne doświadczenie w pracy z komputerem.

Nie ma przesady w mówieniu, że programowanie obiektowe zrewolucjonizowało informatykę, chociaż na prawdziwe efekty należało czekać do lat 80tych, kiedy komputery osobiste na dobre trafiły do domów indywidualnych użytkowników, a programowaniem mogli zająć się ludzie nie związani z uczelnianymi, bądź wojskowymi ośrodkami badań. Do dziś jedyną alternatywą dla programowania proceduralnego jest programowanie obiektowe, a najpopularniejsze języki[3] w użyciu są właśnie obiektowymi. Wyjątkiem jest język C, któremu historia wyznaczyła specjalnie miejsce: jako język niemal natywny dla tworzonych procesorów obsługuje ich najbardziej podstawowe funkcje, jest przez to najszybciej wykonywanym ze wszystkich języków wysokiego poziomu (choć często mówi

3 Wykresy popularności języków programowania używanych w projektach http://www.langpop.com/ (stan z dnia 12.07.2011)

się o nim, jako o języku poziomu pośredniego), stanowi przez to jedyny słuszny wybór, gdy liczy się szybkość, a nie wygoda wynikająca z użycia obiektów.

Rok 1958 był bardzo ważny w historii informatyki i telekomunikacji nie tylko ze względu na pionierskie prace nad rozwojem programowania w instytucie MIT. Amerykanie od początku lat 50ych opracowywali narzędzie ułatwiające komunikację i współpracę cywilnych i wojskowych ośrodków badawczych rozsianych po terenie całych Stanów. Pomysł wykorzystania sieci komputerowej, przesyłającej dane podzielone na „pakiety" pochodził od Paula Barana, informatyka polskiego pochodzenia (ur. w 1926 r, w Grodnie – obecnie Białoruś) ówczesnego członka RAND Corporation, jednego z pierwszych think-tanków w historii, i stanowił odpowiedź na postawione przez Departament wymaganie, aby system taki był w stanie przetrwać atak nuklearny z użyciem wielu głowic, co oznaczało, że musiał być zdecentralizowany. Do roku 1957 prace na ten temat były czysto teoretyczne, gdy nieoczekiwanie świat obiegła wiadomość, że Związek Radziecki umieścił na ziemskiej orbicie pierwszego satelitę w historii – *Sputnika 1*. Podjęto decyzję o natychmiastowym przyspieszeniu badań i już rok później, w 1958 r., administracja prezydenta Dwighta Eisenhowera powołała agencję ARPA (Advanced Research Projects Agency ang. Agencja Zaawansowanych Projektów Badawczych).

Zadanie stworzenia komputerowej sieci było tylko jednym z wielu, jakie rząd zlecił nowo powstałej agencji, ale mimo wszystko na pierwsze efekty trzeba było czekać zaskakująco długo, bo aż jedenaście lat. Projekt rozwijany był jednocześnie w kilku różnych ośrodkach badawczych, a najlepiej rokujące pomysły tworzyły pewnego rodzaju rdzeń dalszych prac. Pierwsze połączenie dwóch niezależnych ośrodków nastąpiło w Kalifornii w 1969 r. pomiędzy Instytutem Badawczym Stanforda, a kampusem Uniwersytetu Kalifornijskiego w Los Angeles[4] – jest to dystans około 500 kilometrów. Jeszcze w grudniu tego roku w sieci nazywanej ARPANETem połączone były cztery ośrodki; trzy w Kalifornii, jeden w Utah. To właśnie możliwość włączenia istniejących, wewnętrznych sieci jako podsieci do większego systemu teleinformatycznego, zapewniła ARPANETowi sukces.

W połowie lat 70ych na całym świecie operowało kilka niezależnych sieci komputerowych, jak francuski CYCLADES i brytyjska sieć Mark I, a w samych Stanach Zjednoczonych: Tymnet w Kaliforni, czy Merit Network w stanie Michigan. Zrodził się problem jak połączyć istniejące sieci całego świata w jeden wydajny układ. W 1974r w efekcie współpracy Roberta Kahna z DARPA (Agencja została przemianowana na Defense Advanced Research Projects Agency w 1972r) i Vintona Cerfa z Uniwersytetu Stanforda, powstała „specyfikacja RFC-675[5]", w tytule której po raz pierwszy pada nazwa „internet". Najważniejszym punktem dokumentu było postulowanie o umniejszenie roli samej sieci do zaledwie medium, przenoszącego pakiety, całą resztę pracy miały wykonywać przede wszystkim serwery i w mniejszym stopniu maszyny klienckie; był to pomysł zaczerpnięty

4 http://www.netvalley.com/cgi-bin/intval/net_history.pl (stan z dnia 17.07.2011)

5 http://tools.ietf.org/html/rfc675 (stan z dnia 17.07.2011)

od twórcy CYCLADESa – Louisa Pouzina. W roku 1982 formalnie zdefiniowano i opatentowano protokół TCP/IP.

Jeszcze do końca lat 80tych na terenie USA powstawały odrębne sieci naukowe jak CSNET (Computer Science Network) i NSFNET *(National* Science Foundation), ale z początkiem lat 90tych stanowiły wraz z ARPANETem jeden nierozróżnialny system, pozbawiony elementów centralnych, tak jak przewidywały to wymagania Departamentu Obrony sprzed 40 lat. Termin „internet" który wcześniej był zaledwie skrótem od wyrażenia internetworking (ang. praca wewnątrz sieci) – zaczął być używany jako nazwa własna nowego medium teleinformatycznego.

Ośrodek CERN w Europie kuszony perspektywami połączenia z uczelniami zza oceanu zaadaptował rozwiązania TCP/IP na własne potrzeby. To właśnie tam już w 1989 Sir Timothy Berners-Lee przedstawił propozycję utworzenia systemu, który umożliwi magazynowanie i przeglądanie danych wewnątrz sieci, wykorzystując do tego programy zwane „przeglądarkami", działające za zasadzie obsługi hipertkestu – pomysłu Douglasa Engelbarta sprzed trzydziestu lat, który jeszcze nie doczekał się realizacji na skalę światową. Rok później do Bernersa-Lee dołączył Belg Robert Cailliau; razem dopracowali projekt, stworzyli pierwszą przeglądarkę, pierwsze strony hipertekstowe i serwer na stacji roboczej *NeXT*, do którego można było się połączyć i czytać znajdujące się na nim dokumenty. Swój prototypowy system nazwali *World Wide Web* (ang. Sieć Ogólnoświatowa), jakby przewidując, że istotnie stanie się on narzędziem używanym na całym świecie; Berners-Lee tworzył go od podstaw z zamierzeniem włączenia *WWW* do Internetu. I tak, jak przewidział w latach 60tych Marshall McLuhan, a dziesięć lat po nim jeden z ojców fantastyki naukowej Sir Arthur C. Clarke – powstała teleinformatyczna sieć, dająca ludziom dostęp do wiedzy pochodzącej z całego świata na wyciągnięcie ręki. Optymizm jej twórców tylko raz zderzył się z rzeczywistością – zakładali oni, że rozbudowanie systemu do funkcjonalności, w której użytkownicy sami będą mogli zarządzać jego treścią, to kwestia sześciu miesięcy od czasu zaprezentowania pierwszej wersji. Minęło 10 lat zanim świat internetu wkroczył w epokę *Web 2.0* – epokę blogów, forum, filmu, muzyki i telewizji w internecie. Statyczne internetowe „witryny" i „strony" miały zostać zastąpione funkcjonalnymi aplikacjami, hipertekstowe dokumenty – interaktywnymi prezentacjami z dźwiękiem i animacją.

Czasy gdy *MIT*, czy jakakolwiek inna instytucja, wytyczały kierunek i dyktowały trendy w rozwijaniu nowych metodyk pracy, czy technologii, minęły bezpowrotnie. *Web 2.0* uczynił z internautów współautorów nowego medium; każdy mógł publikować swoje pomysły i opinie, ale nowe czasy wymagały nowego „przywództwa". To właśnie wtedy, w 2001 roku, w Snowbird, w stanie Utah, odbyła się konferencja, w której udział wzięli entuzjaści nowych technologii. Jej głównym zadaniem było wypracowanie porozumienia w sprawie metodyki pracy alternatywnej do używanego od lat 70tych „modelu kaskadowego". Opracowano *Manifest Zwinnego Wytwarzania Oprogramowania*, zwany potocznie *Manifestem Agile*, od jego angielskiej nazwy *Manifesto for Agile Software*

Development. Jego główną ideą było przedkładanie[6]:

- **Ludzi i ich wzajemne interakcje (współdziałanie)** ponad procedury i narzędzia.
- **Działające oprogramowanie** nad wyczerpującą dokumentację.
- **Współpracę z klientem** nad negocjację umów.
- **Reagowanie na zmiany** nad realizowanie planu.

Najciekawszym, moim zdaniem, aspektem *Manifestu*, jest fakt, że był pierwszym formalnym dokumentem opisującym metody, którymi przez całe lata nieoficjalnie posługiwali się najwięksi programiści, jak wspomniany wcześniej współtwórca *Smalltalka* – Dan Ingalls, czy Donald Knuth, autor wielotomowej „Sztuki Programowania" i twórca systemu składania tekstu *TEX*. W gruncie rzeczy wszystkie metody, objęte wspólną nazwą *Zwinnych*, używane były wcześniej, *Manifest* wpisał je tylko w nowe ramy, zidentyfikował je. Na przykład metodyki typu *test-first* znane były już pod koniec lat 90tych, odwracały one „standardową" i – można by pomyśleć – naturalną kolejność pisania oprogramowania, zakładając, że środowisko testowe programu nie powstaje w odpowiedzi na potrzebę sprawdzenia poprawności jego działania, lecz to kod programu powstaje tak, by być w pełni zgodnym z uprzednio narzuconymi ramami testu. W związku z rosnącą rolą testów w tym okresie powstawały również pierwsze automatyczne systemy ich wykonywania na kodzie, wraz z nimi – terminologia określająca m.in. testy jednostkowe i integracyjne. Pierwsze z nich, jak sama nazwa wskazuje, dotyczą testowania wyodrębnionego, atomowego elementu w kodzie – najczęściej pojedynczej metody konkretnej klasy obiektu. Testy integracyjne mają zakres szerszy i sprawdzają współpracę między kilkoma obiektami.

W 2003r jeden z sygnatariuszy *Manifestu*, Kent Beck, uporządkował metodyki *test-first* i zaproponował rozwiązanie o nazwie *TDD* (*Test-Driven Development*, ang. dosł. Programowanie Testowo-Sterowane). W tym samym roku Dan North zaproponował metodykę będącą de facto rozwinięciem *TDD* – Programowanie Metodą Behawioralną (*BDD – Behaviour-Driven Development*). Dan North uznał, że *TDD* nie jest dostatecznie intuicyjne[7], nie istnieją wskazówki co testować, jak szczegółowo i kiedy uznać, że test zakończył się pełnym sukcesem. *BDD* miało rozwiązać ten problem – jego głównym założeniem jest tworzenie oprogramowania w oparciu nie o suche techniczne testy, ale „scenariusze" przewidujące możliwie jak najdokładniej wszelkie zachowania użytkownika aplikacji, zwłaszcza jego pomyłki i próby uzyskania niedozwolonego dostępu. Takie postępowanie wykracza poza ramy technicznego i technologicznego zarządzania aplikacją – duży nacisk kładzie na komunikację między osobami odpowiedzialnymi za projektowanie funkcjonalności, a tymi odpowiadającymi za implementację (programistami). *BDD* w istocie jest filozofią, której celem jest kierowanie całością rozwoju programu, umożliwienie zleceniodawcom współpracy ze zleceniobiorcą przy

6 http://pl.wikipedia.org/wiki/Manifest_Agile (stan z dnia 12.07.2011)

7 http://dannorth.net/introducing-bdd/ (stan z dnia 12.07.2011)

całkowitym odrzuceniu sztywnych ram dokumentacji – jest więc kompleksowym rozwiązaniem, wpisującym się w wizję wytwarzania oprogramowania *Zwinnego*. Postuluje, wobec tego, o utworzenie języka pośredniego, między językiem biznesowym (dokumentacji, którą całkowicie odrzucono) i technicznym (który dla filozofii nie jest istotny).

W tym miejscu należy przywołać ponownie osobę Alana Kaya, naukowca odpowiedzialnego nie tylko za podstawy Programowania Obiektowego i pionierskie prace nad graficznymi interfejsami użytkownika. Kay znany jest również w kulturze popularnej jako autor powiedzenia: „Jedyny sposób by przewidzieć przyszłość, to ją wymyślić", chociaż słowa te wypowiedziane zostały w odniesieniu do całej dziedziny nauki jaką jest informatyka, znakomicie wpisują się w ideę metodyki *BDD* – najlepszy sposób na przewidzenie zachowań użytkownika, odpowiedzi programu, ale również wymagań jakie w przyszłości użytkownik postawi aplikacji, to stworzenie scenariuszy dla każdej możliwości.

Pierwszym frameworkiem Dana Northa był stworzony dla języka Java *JBehave*, prace rozpoczął już w 2003r[8]. Mimo sukcesu (*Jbehave* rozwijany jest do dzisiaj) North musiał uznać, że język Java nie jest najwygodniejszym do implementacji – dowodem na to jest fakt, że wkrótce porzucił projekt, ale nie filozofię *BDD*. W 2005r zainspirowany projektem *RSpec*, platformą *BDD* w języku Ruby autorstwa Dave'a Astelsa, pracował już nad *RBehave*[9]. Chcąc uczynić z *RSpec* kompleksowe rozwiązanie połączył siły z Astelsem, Davidem Chelimskim i Aslakiem Hellesøyem – zgodnie z jego planami *RBehave* przestał istnieć jako osobny projekt i został wcielony do *Rspec*, powoli stającym się standardem w zakresie środowisk testowych platformy *Ruby on Rails* – mimo iż ten popularny framework domyślnie wyposażony jest we własne narzędzia testujące.

Norweg Hellesøy podjął później prace nad maksymalnym uproszczeniem procesu pisania testów i stworzeniem narzędzi, które umożliwiłyby tworzenie bardzo szczegółowych scenariuszy zachowania programu osobom, które nie mają wykształcenia programistycznego. Tak ziścił się pomysł Northa o zaprojektowaniu języka pośredniego między biznesowym a implementacyjnym; powstał *Cucumber* (ang. ogórek), platforma, której głównym przeznaczeniem było generowanie odpowiedniej struktury testów, w odpowiedzi na scenariusze pisane w języku o nazwie *Gherkin* (ang. korniszon), który łudząco przypomina potoczny język ludzki, nie tylko angielski – reagować ma on na słowa kluczowe 40 języków, wspieranych przez *Cucumber*.

Metodyka BDD, choć zaproponowana w 2003r jest nadal bardzo młoda, dopiero od 2010r można mówić o dojrzewaniu tej idei, edytory tekstowe zaczynają wspierać rozwiązania RSpec i Cucumber, powstają też narzędzia dedykowane[10]. Technologia stabilizuje się, przestaje tak dynamicznie zmieniać, co owocuje powstaniem

8 http://edgibbs.com/2007/12/02/jbehave-and-rspec-history/

9 http://dannorth.net/2007/06/17/introducing-rbehave/

10 http://gherkineditor.codeplex.com/

wyczerpujących dokumentacji i publikacji[11] poświęconych zarówno metodyce jak i narzędziom. Nie ma już żadnych technologicznych barier, które uniemożliwiałyby tworzenie testów i scenariuszy osobom nie biorącym udziału w pisaniu kodu.

Nie można jednak mówić o kolejnej nadchodzącej globalnej rewolucji, przede wszystkim dlatego, że BDD reprezentuje tylko jeden z wielu sposobów wdrożenia idei Programowania Zwinnego, a samo w sobie jest rozwinięciem pomysłu TDD. Inaczej niż w połowie lat 60tych XX wieku nie istnieje obecnie jeden ośrodek decyzyjny nadający kierunek całej nauce programowania. Z drugiej jednak strony trudno nazywać nowe techniki odłamami „lokalnymi" w dobie internetu, zwłaszcza w dziedzinie, która z tym medium jest tak silnie i naturalnie związana. Przyjęcie danej metodyki jest więc dyktowane tylko osobistymi upodobaniami zarządzających projektem.

11 http://pragprog.com/book/achbd/the-rspec-book