Uniwersytet im. Adama Mickiewicza
Wydział Nauk Społecznych

Praca magisterska

"Agile methodology" jako przykład społecznego tworzenia wiedzy

"Agile methodology" as an Example of Distributed Cognition

Michał Owsiak

Promotor:
dr Andrzej W. Nowak

Poznań 2010

Poznań, dnia 18.05.2010

OŚWIADCZENIE

Ja, niżej podpisany/a Michał Owsiak student/ka Wydziału Nauk Społecznych im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt: "Agile methodology" jako przykład społecznego tworzenia wiedzy

napisałem/am samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem/am z pomocy innych osób, a w szczególności nie zlecałem/am opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem/am tej rozprawy lub jej części od innych osób.

Oświadczam również, że egzemplarz pracy dyplomowej w formie wydruku komputerowego jest zgodny z egzemplarzem pracy dyplomowej w formie elektronicznej.

Jednocześnie przyjmuję do wiadomości, że gdyby powyższe oświadczenie okazało się nieprawdziwe, decyzja o wydaniu mi dyplomu zostanie cofnięta.

.....

☐ Wyrażam zgodę na to, aby po egzaminie dyplomowym dołączyć przedstawioną przeze mnie pracę dyplomową do baz systemu Plagiat.pl, gdzie będzie ona wykorzystywana wyłącznie w celach porównywania jej z innymi pracami powstającymi na uczelniach stosujących system antyplagiatowy Plagiat.pl.

.....

☒ Nie wyrażam zgody na to, aby po egzaminie dyplomowym dołączyć przedstawioną przeze mnie pracę dyplomową do baz systemu Plagiat.pl, gdzie będzie ona wykorzystywana wyłącznie w celach porównywania jej z innymi pracami powstającymi na uczelniach stosujących system antyplagiatowy Plagiat.pl.

.....

Abstrakt

Niniejsza praca ma na celu przybliżenie problematyki związanej z komunikacją w grupach projektowych odpowiedzialnych za projektowanie, wytwarzanie oraz wdrażanie systemów informatycznych. W pracy postawiona zostaje teza, iż komunikacja (zarówno wewnętrzna jak i zewnętrzna) w tego typu zespołach może być znacznie bardziej efektywna, gdy opiera się na poznaniu rozproszonym. W pracy niniejszej oparto się na pracach Edwina Hutchinsa oraz badaniach opartych o jego dorobek. Poddane krytyce zostały również niektóre założenia zwinnych metodyk projektowych.

słowa kluczowe: Edwin Hutchins, poznanie rozproszone, distributed cognition, zwinne zespoły projektowe, XP, Scrum, agile programming

Wstęp

Idealny architekt powinien być erudytą, matematykiem, obeznany z wiedzą historyczną, okazywać zainteresowanie filozofią, być osłuchanym z muzyką, posiadać podstawową wiedzę na temat medycyny, znać podstawy astronomi.

Vitruvius, około 25 lat przed Chrystusem

Rewolucja informatyczna wymusiła wykształcenie nowej dziedziny nauki pozwalającej na skodyfikowanie i unormowanie procesu wytwarzania oprogramowania - inżynierii oprogramowania. Celem wspomnianej gałęzi informatyki jest opracowanie jak najbardziej wydajnych metodyk prowadzenia projektów informatycznych.

Mimo tego, że istnieje szereg metod wspomagających proces tworzenia aplikacji, które to metody powinny wpływać na lepsze zarządzanie projektami informatycznymi, statystyki pozostają bezlitosne. Spośród wszystkich prowadzonych projektów informatycznych aż 50% przekracza termin lub budżet przeznaczony na ich realizację. Wśród tych projektów, około 30% problemów wynika ze złej komunikacji pomiędzy twórcami oraz odbiorcami[1]. Naturalnym więc wydaje się zadanie pytania z czego mogą wynikać wspomniane problemy w komunikacji międzyludzkiej.

Cytowany na początku rozdziału Vitruvius, pisząc o idealnym architekcie nie miał na myśli architektów aplikacji komputerowych - nie mógł przewidzieć, że 20 wieków po jego śmierci pojawi się zawód architekta systemów obliczeniowych. Niemniej jednak, trafnie zauważył, że architekt to nie tylko rzemieślnik znający się na swoim fachu. Miał świadomość, że prawdziwy architekt, tworzący ponadczasowe budowle musi znać się na czymś więcej niż tylko na swoim fachu - że musi być osobą wszechstronną. Dzisiejsza

[1] D. Leffingwell, D. Widrig, *Zarządzanie wymaganiami*, WNT, Warszawa 2003, s. 7

automatyzacja życia, wiara w łatwy dostęp do informacji, sprawiły że architekci systemów komputerowych zapomnieli o tej prawdzie sprzed czasów Chrystusa. Stawiają na technologię, procesy i maszyny zamiast sięgnąć po to, co od zawsze prowadziło do sukcesu - elastyczność, indywidualizm oraz niczym nieskrępowaną komunikację międzyludzką.

W tej pracy nie zostanie udzielona jednoznaczna odpowiedź na pytanie o źródła problemów komunikacyjnych w projektach informatycznych. Wynika to z tego, że przyczyny problemów komunikacyjnych są niezwykle różnorodne i złożone. W tej pracy nie zostanie również podana jednoznaczna receptura na wykształcenie idealnego architekta, wybitnej jednostki stojącej na czele zespołu ludzkiego. Zostaną natomiast poddane analizie różne metodyki projektowe w odniesieniu do tak zwanego poznania rozproszonego i ukazane zostanie jak można usprawnić komunikację międzyludzką oraz jak można wykorzystać osiągnięcia kognitywistyki do tej klasy problemów.

1 Komunikacja w grupach

1.1 Poznanie rozproszone

Każda grupa społeczna bierze udział w przekazywaniu informacji pomiędzy jej członkami. Tak jak pojedyncze jednostki mogą kumulować wiedzę opartą o swoje doświadczenia tak samo cała społeczność może być beneficjentem wiedzy każdego z członków danej społeczności, dzięki nadbudowywaniu i poszerzaniu bazy wiedzy. W takim przypadku możemy mówić o poznaniu rozproszonym. W modelu tym informacja przechowywana jest w większej niż minimalna wymagana ilość zasobów koniecznych do przechowania danej porcji informacji[2]. Wiedza ta, często nie jest bezpośrednio dostępna (żaden z członków społeczności nie dysponuje całą wiedzą), jest redundantna (obszary wiedzy, posiadanej przez członków grupy, pokrywają się), może być przechowywana na różne sposoby (muzyka, pismo, przekazy ustne). Ważnym kryterium jakości tej wiedzy jest to, że cała grupa jest jej beneficjentem. Dzięki niej członkowie grupy mogą pozyskiwać nową wiedzę w oparciu o doświadczenia całej grupy. Christopher P. Nemeth ujmuje to następującymi słowy:

> (...) *poznanie rozproszone to współdzielenie świadomości dotyczącej celów działania, planów oraz detali tych działań, których pojedyncza jednostka nie byłaby w stanie objąć*[3].

Spoglądając na sposoby przekazywania wiedzy możemy zauważyć trzy typowe sposoby gromadzenia i przekazywania wiedzy: proces poznawczy może być rozdystrybuowany pomiędzy członkami grupy, może być zależny od wewnętrznej i zewnętrznej struktury oraz może być zależny od czasu - w takim znaczeniu, że wynik wcześniejszych działań może wpływać na działania po nich

---

[2] Robert A. Wilson, *The MIT Encyclopedia of the Cognitive Sciences*, s. 236

[3] C. P. Nemeth, *Using Cognitive Artifacts to Understand Distributed Cognition*, s. 727

następujące[4]. Zewnętrzna i zewnętrzna struktura nazywana jest również aktorami oraz siatką zależności. Według Latoura, aktorzy poza-ludzcy (*ang. nonhuman actors*), to materialne i technologiczne czynniki, które umożliwiają stabilizację struktur i instytucjonalizację praktyk społecznych[5].

Znajomość tych zależności nie daje jednak jasnej odpowiedzi na pytanie o to, jak procesy poznawcze jednostki można przenieść na całą grupę. Inaczej mówiąc, w jaki sposób grupa może być beneficjentem dotychczasowej wiedzy i umiejętności danej jednostki. W ramach nauk społecznych powstało wiele teorii traktujących o tym zagadnieniu. Durkheim oraz jego uczniowie (głównie Halbwachs) twierdzili, że pamięć nie może być rozpatrywana jako własność wyizolowanej jednostki. Roberts proponował, aby wszelkiego rodzaju organizacje traktować jako pewnego rodzaju strukturę architektoniczną na poziomie grupy społecznej. Aby ocenić możliwości poznania kognitywnego grupy, starał się odnaleźć lokalizację informacji, rodzaj informacji oraz możliwości jej transferu. Schwartz proponował model rozproszony kultury, w którym nacisk położony został na rozkład wierzeń w danej grupie społecznej. Romney, Weller, and Batchelder zajmowali się ilościowym podejściem do problematyki wzorców kulturowych - zlokalizowanie i opisanie takiego wzorca pociągało za sobą poszukiwanie odpowiedzi na pytanie: dlaczego dany wzorzec powstał? Dawkins zaproponował aby przyjąć „istnienie" bytów niezależnych, w znaczeniu autonomicznych od naszego umysłu - *memów*. Byty te mogą „przenosić się" pomiędzy umysłami, a tym samym „infekować" je określoną ideą.

> *Już kiedyś wskazałem, że istnieją całkowicie niegenetyczne replikatory, które spotkać można jedynie w środowiskach porozumiewających się ze sobą umysłów. Nazwałem je „memami" (...) Fenotypowe efekty memu mogą mieć formę słów, muzyki, obrazów, kroju ubrań, gestów rąk i grymasów twarzy, specjalnych umiejętności (...) Są one zewnętrznymi*

---

[4] E. Hutchins, *Distributed Cognition,* s. 1

[5] Ł. Afeltowicz, *Czy technika pozbawia nas pracy?*, s. 108

i widocznymi przejawami memów mieszczących się w mózgu. Mogą być dostrzeżone przez narządy zmysłów innych osobników i mogą wdrukować się w ich umysły.[6]

Koncepcja ta wyewoluowała w późniejszym czasie w memetykę.

Poznanie rozproszone ma szczególne zastosowanie w dziedzinach związanych z nauką - a dokładniej z pracownikami naukowymi. Ta grupa zawodowa z założenia zbudowana jest na fundamentach poznania kognitywnego oraz wiedzy rozproszonej. Badaniem tego zjawiska zajmował się między innymi Thagard. Interesującą konkluzją jego prac jest odkrycie zależności pomiędzy sposobami komunikacji w grupach, a efektywnością poznania kognitywnego[7]. Ten fakt będzie miał znaczenie na późniejszym etapie niniejszej pracy, gdy analizowane będą sposoby komunikacji w zespołach projektowych. Innym badaczem środowiska naukowego był Latour, który analizował wpływ czynności kognitywnych wewnątrz grup społecznych oraz pomiędzy ludźmi i ideami na rezultaty pracy naukowej. Kolejnym ciekawym aspektem komunikacji w grupach jest kwestia racjonalności określonych idei na poziomie indywidualnym i na poziomie agregowanym. Te kwestie poruszane były między innymi w ramach teorii gier. Klasycznym przykładem może tutaj być dylemat więźnia - to co racjonalne w jednej sytuacji, nie musi być racjonalne w innej. Działanie więźnia zależy nie tylko od jego indywidualnych preferencji, ale również od relacji w grupie i spodziewanego zysku, wynikającego z podjęcia określonych decyzji - prace Rappaporta[8].

Inną, bardzo ważną koncepcją jest propozycja spojrzenia na umysł ludzki nie, jak na zamknięty wewnątrz jednostki konstrukt pozwalający na racjonalizację działań, ale jak na konstrukt oparty zarówno o wewnętrzne struktury, takie jak mózg czy neurony, ale również o to, co zewnętrzne. Koncepcja ta nosi miano umysłu rozszerzonego.

---

[6] R. Dawkins, *Fenotyp rozszerzony*, s. 146

[7] E. Hutchins, *Distributed Cognition,* s. 2

[8] E. Hutchins, *Distributed Cognition,* s. 3

1.1.1 Umysł rozszerzony

Koncepcja umysłu rozszerzonego została zaproponowana przez Andy'iego Clarka i Davida Chalmersa. W swojej pracy, *The Extended Mind*[9], autorzy postulują, odejście od klasycznego umiejscawiania umysłu wewnątrz jednostki - granica umysłu nie przebiega na *skórze i czaszce* (*ang. skin and skull*). Odchodzą też od klasycznego eksternalizmu, w którym to co zewnętrze pozostaje zawsze zewnętrzne. W ich koncepcji, eksternalizmu aktywnego, świat zewnętrzny oddziałuje na procesy kognitywne jednostki.

W ujęciu tym, jednostka jest zanurzona w świecie i procesach, które ją otaczają. Procesy oraz inne czynniki zewnętrzne mają wpływ na generowanie nowej wiedzy.

> *Wszystkie części składowe systemu odgrywają ważną rolę przyczynowo-skutkową i wspólnie realizują pewne zachowania tożsame z poznaniem. Usunięcie poszczególnych elementów wpływa negatywnie na proces poznawczy całości. Według nas, tego typu działanie, wyrażone jako kooperacja pojedynczych, zewnętrznych w stosunku do podmiotu elementów, może być postrzegana jako proces kognitywny tożsamy z poznaniem jakie zachodzi wewnątrz umysłu*[10]*.*

Idea Clarka i Chalmersa znacznie różni się od postulowanego przez Putnama braku wpływu czynników zewnętrznych na proces poznawczy. Eksternalizm aktywny odrzuca historię jako jedyne źródło poznania i wskazuje na równie ważne, jeżeli nie kluczowe, znaczenie czynników zewnętrznych w procesie poznawczym. Dodatkowo, autorzy wskazują na brak istotności w rozróżnieniu

---

[9] A. Clark, D. J. Chalmers, *The Extended Mind - tłumaczenie tekstu ukazało się nakładem: A. Clark, D. Chalmers, Umysł rozszerzony, w: M. Miłkowski, R. Poczobut (red.), Analityczna metafizyka umysłu. Najnowsze kontrowersje, Wydawnictwo IFiS PAN, Warszawa 2008, s. 342-357.*

[10] A. Clark, D. J. Chalmers, *The Extended Mind*, s. 4

tego co zewnętrzne i wewnętrzne w procesie akwizycji wiedzy. Informacja o świecie, pozwalająca na generowanie wiedzy, może być albo częścią składową naszego umysłu, na przykład w postaci wspomnień, ale może też być zapisaną kartką papieru do której mamy dostęp zawsze wtedy, gdy określona porcja informacji jest nam potrzebna. W rozumowaniu tym, autorzy posługują się eksperymentem myślowym, w którym dwie osoby, realizujące to samo zadanie, posługują się różnymi źródłami wiedzy. Pierwsza z nich, opiera się tylko na swoich przekonaniach wewnętrznych i własnej pamięci. Druga osoba, posługuje się zarówno własnymi przekonaniami jak i zewnętrznym źródłem informacji - notatnikiem. Autorzy dochodzą do wniosku, że notatnik pełni taką samą rolę jak pamięć jednostki. Informacja znajdująca się w notatniku pozwala na wykształcenie się przekonania, które przed chwilą jeszcze nie było naszym udziałem - dokładnie tak samo, jak informacja „przywołana" z naszej pamięci. Różnica polega tylko na tym, że w tym pierwszym przypadku, źródło informacji leży „poza skórą"[11].

Zakładając z powyższego, że świat zewnętrzny odgrywa rolę w naszych procesach myślowych, możemy przyjąć że nie jest jednak tak, jak sugeruje Putnam, że za nasze stany mentalne odpowiada tylko nasza historia i doświadczenie. To, co czyni z informacji przekonanie, to sposób jej funkcjonowania, a nie źródło pochodzenia. Nie jest tak naprawdę ważne czy źródło informacji umiejscowione jest wewnątrz, czy też na zewnątrz naszego mózgu.

Przyjmując takie założenia możemy pytać dalej. Jaki wpływ na nasze poznanie mają inne jednostki? Czy w przypadku współpracy wielu niezależnych podmiotów nie mamy do czynienia z czymś w rodzaju „umysłu uwspólnionego".

---

[11] A. Clark, D. J. Chalmers, *The Extended Mind,* s. 12

1.1.2 Umysł rozproszony

Źródła naszych informacji mogą rozciągać się w różnych kierunkach, w różnych wymiarach. Mogą to być informacje płynące z naszych notatek, książek, wykładów. Mogą to być nasze przekonania wpojone nam w procesie wychowania. Ciekawym przykładem tutaj może być dostęp do tak zwanych FAQ (*ang. Frequently Asked Questions*) w świecie informatyki. Takie zbiory FAQ pozwalają na błyskawiczne rozwiązanie problemów związanych z konfiguracją systemów operacyjnych, pisaniem określonych fragmentów oprogramowania czy wyszukiwania gotowych algorytmów. Zbiory te, nie posiadają zwykle gotowych rozwiązań dla każdego możliwego problemu, ale prezentują rozwiązania problemów zbliżonych do tego, z którym właśnie się spotkaliśmy. Analiza tych rozwiązań może znacznie nas przybliżyć do wynalezienia rozwiązania dobrego w naszej sytuacji. Wiele osób będzie pewnie argumentowało, że traktowanie zbiorów Internetu jako przedłużenia naszego umysłu to nadużycie, niemniej jednak - w tym przypadku - Internet to nic innego jak notatnik. Jeżeli wiemy gdzie szukać, jeżeli potrafimy dobrze szukać, to w pracy nad rozwiązaniem naszego zadania intelektualnego dysponujemy notatnikami wszystkich tych, którzy spotkali się kiedyś z problemem podobnym do naszego.

Jeszcze ciekawszym przypadkiem jest współdzielenie umysłu z innymi podmiotami. Jeżeli przyjmiemy że inne jednostki mogą wpływać na nasze decyzje poprzez swoje zachowanie, poprzez informacje jakich nam udzielają, to możemy mówić o ich wpływie na nasze przekonania oraz wiedzę jaką posiadamy. Jeżeli w trakcie negocjacji biznesowych, jedna ze stron skutecznie ukryje niewygodne dla siebie fakty, to uzyska dużo lepszy wynik niż wtedy, gdy druga strona dysponuje pełną informacją potrzebną w trakcie negocjowania kontraktu. W takim przypadku, możemy mówić o tym, że oddziaływanie środowiska zewnętrznego osłabiło nasze kompetencje umysłowe. Z drugiej strony, możemy mieć do czynienia z taką sytuacją, w której inne osoby pozwalają nam na odciążenie naszej pamięci i wspierają nas niezbędną do

podjęcia decyzji informacją. Sekretarka, rozporządzająca naszym kalendarzem, znacznie ułatwia nam podejmowanie decyzji oraz odciąża nasz umysł od konieczności przechowywania informacji. Pozwala nam tym samym na operowanie na wyższym poziomie abstrakcji. Już nie musimy dbać o szczegóły, możemy skupić się na tym co dla nas najważniejsze w trakcie spotkań - przeglądnięcie materiałów przed prezentacją, prowadzenie rozmów w kuluarach, wyciąganie wniosków w oparciu o poszczególne porcje informacji oraz temu podobne.

Umysł rozproszony znajduje szczególny wymiar w interdyscyplinarnych grupach projektowych. Osoby zajmujące się różnymi profesjami mogą w trakcie pracy nad określonym zagadnienie skupić się na tych aspektach zagadnienia, które są im najbliższe, jednocześnie mając oparcie u innych członków zespołu tam, gdzie ich własna wiedza nie sięga. I tutaj pojawia się pytanie. Czy jeżeli w trakcie takiego grupowego procesu, jedna z osób poda rozwiązanie problemu, łącząc wszystkie pozostałe dziedziny oraz podając sensowną ścieżkę wnioskowania, to czy powinniśmy tutaj mówić o jednostkowym sukcesie czy raczej kolaboracji wielu umysłów i artefaktów. Efekt synergii nie pozostaje tutaj bez znaczenia. Współpraca wszystkich członków zespołu pozwala na spojrzenie na problem z innej strony. Można by to podsumować cytatem, *„wszyscy wiedzą, że czegoś nie da się zrobić, i przychodzi taki jeden, który nie wie, że się nie da, i on to właśnie robi”*. Właśnie w takich grupach - interdyscyplinarnych - pojawia się możliwość generowania nowej wiedzy w sposób całkowicie niedostępny grupom hermetycznym, skupionym na jednym wizerunku świata i pozbawionym zewnętrznego źródła informacji.

Praca w zespołach projektowych może przyjmować różne formy. W niniejszej pracy zaprezentowane zostaną różne modele pracy takich grup. Wykazane zostanie, że praca w grupach opartych o koncepcję umysłu rozproszonego przynosi znacznie lepsze rezultaty niż praca w grupach skupionych na procesie samym w sobie zamiast na wymianie informacji.

1.2 Sposoby dystrybucji wiedzy

Prowadząc rozważania nad procesami poznawczymi (poznaniem rozproszonym) należy postawić zasadnicze pytanie: czy procesy te prowadzą do działania poznawczego, czyli czy generują wiedzę[12]? W rozważaniach niektórych kognitywistów[13] można wyróżnić trzy podstawowe modele pozwalające na dystrybucję wiedzy: umysł we wspólnocie, wspólnota umysłu oraz rozproszenie czynności poznawczych w czasie.

Koncepcja umysłu we wspólnocie zakłada możliwość rozprowadzenia wiedzy między członkami grupy społecznej. W modelu tym zakłada się, że wiedza zostaje przekazywana dzięki interakcji kilku czynników: pamięci, zdefiniowanej jako zdolność do zapamiętywania wrażeń zmysłowych, organizacji jako pewnej platformy komunikacji, interakcji oraz napięcia komunikacyjnego wynikającego np. z presji czasu w trakcie podejmowania decyzji.

Koncepcja wspólnoty umysłu zakłada możliwość rozproszenia procesów poznawczych w sensie koordynacji między strukturami wewnętrznymi i zewnętrznymi. Według Minsky'ego jedyną możliwością analizy danej grupy, pod względem możliwości dystrybucji i wytwarzania nowej wiedzy, jest założenie iż grupa ta składa się z niezależnych podmiotów (agentów) będących w różnych połączeniach i konfiguracjach. Taki gotowy konstrukt należy poddać testom, poprzez stawianie kolejnych zadań, co pozwala na ocenę „jakości" badanej struktury i możliwości jej adaptacji - jako całości - do nowych problemów. To podejście będzie rozważane w późniejszej części niniejszej pracy, gdy pod uwagę będą brane różne sposoby komunikacji w grupach projektowych.

Koncepcja rozproszenia czynności poznawczych w czasie zakłada, że to następstwo wydarzeń ma kluczowe znaczenie w dystrybucji wiedzy. W koncepcji tej zdarzenia wcześniejsze mogą, ale nie muszą, wpływać na

---

[12] J. Podgórski, *Charakterystyka pojęcia kolektywu „rozproszonego"*, s. 153

[13] Hollan J., *Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research*, s. 174–196.

późniejsze, a dodatkowo powstała ścieżka zdarzeń może prowadzić do generacji nowej wiedzy. Najbardziej istotnym jest tutaj odpowiedź na pytanie *Jak zaplanować działania w skali czasowej pomiędzy członkami lub elementami tworzącymi formę jakiejkolwiek autonomicznej struktury, a zarazem inteligentnej?*[14] Ten aspekt poznania będzie miał szczególne znaczenie w procesach dystrybucji wiedzy wewnątrz zespołów programistycznych wtedy, gdy kluczowym czynnikiem jest czas - np. błyskawiczna reakcja na dane zdarzenie może prowadzić do wygenerowania nowej porcji wiedzy, natomiast opieszałość - wręcz przeciwnie - do zaciemnienia obrazu całości.

Wspomniane koncepcje poznania rozproszonego próbują ukazać dynamikę poznawczą grupy podmiotów generujących wiedzę niezależnie od tego, czy będą to byty sztuczne - artefakty, czy też myślące jednostki. Poznanie rozproszone odnosi się zarówno do procesów poznawczych w sensie globalnym (grypy, struktury makro), do działań szczegółowych (pojedynczych podmiotów), jak również do wpływu czasu (następstwa).

Poznanie rozproszone jest odkrywczą koncepcją pozwalającą na ujęcie procesów poznawczych z kilku perspektyw i spojrzenie na procesy technologiczne z innego punktu widzenia niż tylko jako na proces technologiczne czy ekonomiczne. Te kwestie są niezwykle istotne w teorii zarządzania zespołami ludzkimi - bardzo często kładzie się w tej teorii nacisk na to aby jednostka była traktowana jako zasób, co wydaje się być bardzo niefortunnym podejściem.

1.3 Artefakty jako narzędzie dystrybucji wiedzy

Poznanie rozproszone wymaga nie tylko platformy komunikacyjnej (społeczność), ale również metod przekazywania informacji, np. muzyka, obraz, przekaz słowny, tekst pisany. Te zewnętrzne formy przekazu informacji to artefakty. Takimi artefaktami mogą być również konstrukty będące sumą bądź wynikiem operowania na innych artefaktach. Przed rokiem 1830 chemicy znali

---

[14] J. Podgórski, *Charakterystyka pojęcia kolektywu „rozproszonego”*, s. 154

pismo i potrafili się nim posługiwać. Niemniej jednak, to dopiero wprowadzenie przez Dumasa odpowiedniej notacji pozwoliło na prowadzenie teoretycznych rozważań dotyczących reakcji chemicznych. Zapis:

$$CH_4 + 2\ O_2 \longrightarrow CO_2 + 2\ H_2O$$

jest, z jednej strony, tylko napisem, z drugiej strony jest zapisem reakcji chemicznej. Według Latoura[15], między innymi tego typu artefakty pozwalają na znaczne upowszechnienie wiedzy oraz nagły jej przyrost w grupie. Wspólna platforma wymiany informacji pozwala na łatwiejsze generowanie nowej wiedzy w oparciu o posiadaną informację. Ciekawym przykładem artefaktów służących temu samemu celowi, ale wyrażonych na kilka różnych sposobów, są reprezentacje symboliczne pochodnej funkcji. Istnieją różne notacje opisujące pochodną[16]:

Zapis pochodnej	Autor
$\frac{d^n y}{dx^n}, \quad \frac{d^n f}{dx^n}(x), \quad \frac{d^n}{dx^n} f(x)$	Leibniz
$(f')' = f''$	Lagrange
$\dot{y}, \ddot{y}$	Newton

Warto tutaj zauważyć, że zapis Leibniza:

$$\frac{d^n y}{dx^n}, \quad \frac{d^n f}{dx^n}(x), \quad \frac{d^n}{dx^n} f(x)$$

jest znacznie bardziej uniwersalny, ponieważ pozwala na ujęcie rzędu pochodnej jako argumentu funkcji. Zapis Newtona:

$$\dot{y}, \ddot{y}$$

---

[15] R. N. Giere, *Distributed Cognition: Where the Cognitive and the Social Merge*, s. 4

[16] http://en.wikipedia.org/wiki/Derivative - *stan na 16.05.2010*

już tego nie dopuszcza. Wydawałoby się, że sposób zapisu nie powinien mieć aż tak dużego znaczenia, niemniej jednak (na podstawie powyższego) wyraźnie widać, że zapis Newtona jest uboższy przez co nie znalazł zastosowania w matematyce (mimo tego, że jest używany w świecie fizyki, gdzie zwykle wystarcza posługiwanie się pochodną maksymalnie trzeciego rzędu).

Takie ujęcie problemu ma bardzo ciekawe konsekwencje. Można by powiedzieć, że reprezentacja problemu ma kluczowe znaczenie dla wiedzy, a dokładniej dla możliwości jej generowania. Jak pisał Hilbert:

> *Przyjmując ten punkt widzenia za przedmioty teorii liczb uważam, dokładnie odwrotnie niż Frege i Dedekind, same znaki... Tkwi w tym wyraźne stanowisko filozoficzne, które jak sądzę jest pożądane dla uzasadnienia czystej matematyki, jak zresztą w ogóle dla całego naukowego myślenia, rozumienia i komunikowania: na początku - jak się tu mówi* - jest znak (za Ernstem Cassirerem[17]).

1.4 Zaburzenia dystrybucji wiedzy

Dystrybucja wiedzy w grupie może być zaburzona na różne sposoby. Pierwszy, i chyba podstawowy problem, to sytuacja w której grupa nie spełnia założeń koniecznych dla zaistnienia poznania rozproszonego. Jednym z tych założeń jest realizacja wspólnych praktyk kulturowych, czyli znacznie upraszczając, po prostu współpraca pomiędzy jednostkami (niekoniecznie bezpośrednia). Ważne jest tutaj to, aby zwrócić uwagę na fakt, że taka współpraca nie musi opierać się na bezpośrednim kontakcie, a raczej na pewnych praktykach. Środowisko naukowe to zbiór rozproszonych po świecie jednostek. Niemniej jednak, dzięki określonym praktykom (publikacje prac naukowych, konferencje, sympozja) środowisko to może wymieniać się informacjami i na ich podstawie generować nową wiedzę. Jeżeli tych wszystkich elementów by zabrakło, dystrybucja wiedzy byłaby znacznie utrudniona.

---

[17] Ernst Cassirer, *Symbol i język*, s. 49

Innym przykładem problemu dystrybucji wiedzy może być żargon specyficzny dla danej grupy zawodowej. Jeżeli w trakcie prac nad systemem komputerowym grupa programistów w obecności klienta wypowie zdanie: „*Problemy z deploymentem web-service'u na nowym nodzie wynikają z nieprawidłowej wersji bibliotek*", to ciężko tutaj mówić o jakiejkolwiek szansie na powstanie nowej wiedzy w wyniku współpracy klienta i programisty. Ten przekaz będzie całkowicie niezrozumiały dla laika komputerowego i nie będzie wnosił niczego nowego do bazy wiedzy wszystkich osób zajmujących się danym zagadnieniem. Oczywiście dociekliwy klient mógłby zwrócić się z prośbą o wyjaśnienie wszystkich terminów, niemniej jednak nie byłby w stanie zrozumieć wszystkich implikacji jakie wynikają z definicji i kontekstu. Jest to klasyczny problem komunikacji wewnątrz grup hermetycznych. O tym problemie szerzej będzie traktował rozdział poświęcony metodyce Rational Unified Process. Również metodyka Scrum próbuje rozwiązać ten problem - ale na innej płaszczyźnie.

1.5 Podsumowanie

Rozproszona dystrybucja wiedzy w grupie pozwala na generowanie nowej wiedzy przez wszystkich jej członków. Pozwala również na takie operowanie informacją, które nie wymaga istnienia centralnego elementu sterującego. Rozproszenie wiedzy pozwala na wyciąganie wniosków na podstawie różnych faktów, potencjalnie ze sobą nie związanych. Ze względu na to, że w projektach informatycznych bardzo często mamy do czynienia z zamkniętymi zespołami pracującymi nad rozwiązaniem określonego problemu (coraz częściej są to zespoły interdyscyplinarne), koncepcja Hutchinsa może mieć tutaj bardzo duże zastosowanie. Dzięki antropologicznemu i filozoficznemu podejściu do problemu możemy osiągnąć lepsze rezultaty na polu zarządzania informacją i wiedzą generowaną przez współpracujących ze sobą ludzi.

W dalszej części pracy zostaną omówione dwa podstawowe paradygmaty tworzenia oprogramowania: klasyczny oraz zwinny. Obydwa

dotyczą tej samej problematyki - wytwarzania oprogramowania - niemniej jednak traktują o niej na dwa różne sposoby. Metodyki klasyczne kierują się ku formalizacji procesu, zaś metodyki zwinne ku interakcji międzyludzkiej.

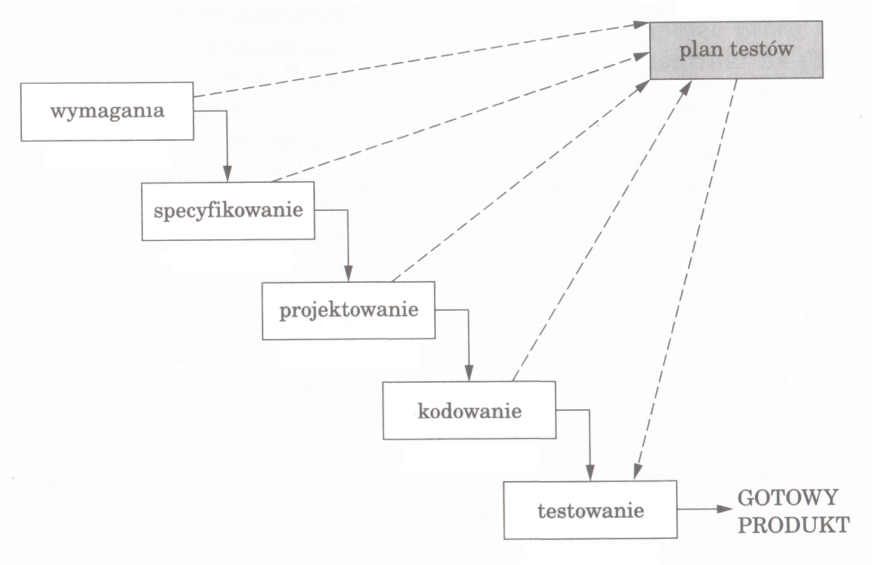
2 Komunikacja w klasycznych metodykach projektowych

Inżynieria oprogramowania to dosyć nietypowa nauka w obszarze nauk ścisłych. Z jednej strony posługuje się fizycznymi urządzeniami (komputery, myszki, monitory, dyski twarde), z drugiej strony jest w zasadzie ograniczona jedynie możliwościami danego języka programowania, w którym wytwarzany jest dany system. Ze względu na to, że stworzenie kolejnego języka programowania nie jest czymś szczególnie trudnym, można zaryzykować stwierdzenie, że dziedzina ta jest jedynie ograniczona wyobraźnią twórców narzędzi programistycznych. Z jednej strony jest to bardzo wygodna sytuacja, gdyż twórcy oprogramowania mogą swoim odbiorcom zaoferować niemalże niczym nieograniczone produkty. Z drugiej strony, odbiorcy są przyzwyczajeni do tego, że „wszystko jest możliwe”. Ta sytuacja może rodzić konflikty, szczególnie wtedy gdy jednak okazuje się że realizacja pewnych wymagań odbiorcy produktu jest albo niemożliwa, albo na tyle skomplikowana, że przekracza znacznie koszty projektu. Dodatkowo może dojść do sytuacji, gdy pewne oczywiste dla klienta aspekty dotyczące projektu, są całkowicie niezrozumiałe dla zespołu tworzącego oprogramowanie. Takim sytuacjom mają przeciwdziałać metodyki projektowe. Każda z metodyk, jakie znajdziemy na rynku inżynierii oprogramowania, ma na celu zapewnienie skutecznego i szybkiego procesu wytwarzania oprogramowania. Każda z nich została stworzona z myślą o poprawie wydajności zespołów programistycznych, polepszeniu relacji klient-twórca oraz poprawie jakości dostarczanych produktów. W niniejszym rozdziale postaram się przybliżyć klasyczne metodyki projektowe. Różnią się one od metodyk zwinnych, przede wszystkim tym, że stawiają nacisk na proces - odstawiając niejako uczestników przedsięwzięcia na boczny tor. Oczywiście biorą one pod uwagę istnienie żywych ludzi z krwi i kości, niemniej jednak traktowani są oni raczej jako zasób, niż żywe istoty.

2.1 Klasyczne metodyki projektowe - model kaskadowy

Jedną z pierwszych, najlepiej rozpoznanych metodyk wytwarzania oprogramowania, jest model kaskadowy. Pierwsze wzmianki o tym podejściu do procesu wytwarzania oprogramowania pojawiły się w roku 1970[18]. W modelu tym zakłada się, że realizacja przedsięwzięcia programistycznego powinna przebiegać wedle pewnych, z góry ustalonych reguł.

Model ten zakłada, że istnieją pewne fazy (następujące po sobie) oraz osoby, które za poszczególne fazy są odpowiedzialne. W modelu klasycznym wyróżnić można następujące fazy: faza wymagań, faza specyfikowania, faza projektowania, faza kodowania, faza testowania, faza dostarczenia produktu oraz faza pielęgnacji produktu (Rys. 1).



Rys. 1 Model kaskadowy (*źródło: D. Hamlet, Podstawy techniczne inż. oprogramowania, s. 18*)

Każda z nich cechuje się innymi założeniami i celami. Fazy te można scharakteryzować w następujący sposób[19]:

---

[18] Royce, Winston, *Managing the Development of Large Software Systems*, Proceedings of IEEE WESCON 26, 1970 r.

[19] D. Hamlet, *Podstawy techniczne inżynierii oprogramowania, s.16*

- Faza wymagań - w tej fazie zbierane są wszelkie wymagania i oczekiwania dotyczące oprogramowania. Klient definiuje jaki produkt go interesuje, jak chce go wdrożyć i co „otrzymać”
- Faza specyfikowania - w tej fazie architekci opracowują zręby systemu, decydują co musi być wytworzone, jakimi metodami, w jaki sposób zostanie wdrożone u klienta, kto będzie odpowiadał za proces implementacji, kto za wdrożenie oraz w jaki sposób będzie przebiegała pielęgnacja produktu po wdrożeniu
- Faza projektowania - w tej fazie następuje szczegółowe projektowanie systemu. Osoby odpowiedzialne za daną technologię decydują jak będzie zaprojektowany system w szczegółach, jak będą łączyły się ze sobą poszczególne części systemu oraz w jaki sposób zostaną zaprogramowane. W wyidealizowanym modelu kaskadowym, projekt systemu jest na tyle szczegółowy i dopracowany, że osoby programujące system muszą jedynie „przepisać” idee projektantów na dany język programowania (co oczywiście w świecie rzeczywistym zwykle nie ma miejsca)
- Faza kodowania - w tej fazie programiści implementują system, a następnie wdrażają go u klienta
- Faza pielęgnacji - w tej fazie osoby odpowiedzialne za utrzymanie systemu (projektanci, programiści) dbają o to aby realizował on powierzone zadania.

W powyższym modelu należy zwrócić uwagę na jedną bardzo ważną kwestię. Do każdej fazy przypisani są ludzie o określonych kompetencjach. Fazy następują po sobie, a powrót z jednej fazy do drugiej nie jest możliwy. Model ten powoduje pewne komplikacje. Do najważniejszych należą:

- jeżeli w danej fazie popełniony został błąd (np. w fazie projektowania) i zostanie on wykryty dopiero w fazie kodowania, to nie ma możliwości ponownego zaprojektowania systemu,

- fazy są od siebie dosyć ostro oddzielone co skutkuje tym, że komunikacja między poszczególnymi grupami pracowników jest utrudniona. Architekci nie komunikują się bezpośrednio z projektantami, czy programistami,
- ze względu na to, że każda grupa ma inne cele, mogą pojawiać się spięcia pomiędzy przedstawicielami poszczególnych grup.

Mimo wszystkich towarzyszących mu komplikacji, model ten jest nadal bardzo często stosowany w ramach procesu wytwarzania oprogramowania. Najczęściej jest on wykorzystywany w projektach podatnych na kwestie „polityczne", które dotyczą działań pomiędzy partnerami, które nie mają na celu merytorycznego rozstrzygania sporów, a umocnienie pozycji jednego z partnerów poprzez niekoniecznie etyczne posunięcia. Takim działaniem może być np. bardzo ścisłe egzekwowanie terminów umów mimo znajomości realiów w jakich wytwarzany jest projekt, czy też pomijanie w dyskusji czynników zewnętrznych, które mają wpływ na projekt, a nie zostały wcześniej uwzględnione.

> *Wprawdzie wszystkie znane rodzaje ryzyka muszą być wskazane, nawet politycznie niepoprawne (zatajanie ich nie oznacza pozbycia się problemów - powoduje tylko, że trudniej nimi kierować)*[20]

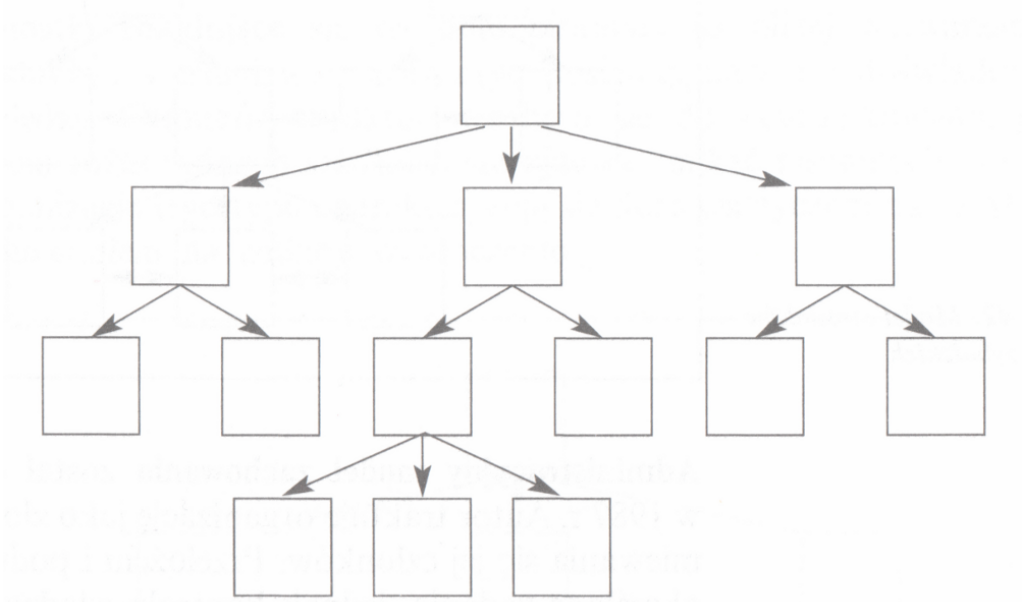
niemniej jednak część z nich (czynników ryzyka) bywa pomijana ze względu na znikome prawdopodobieństwo ich wystąpienia - te właśnie czynniki są często przedmiotem gry politycznej. Model kaskadowy jest również idealny w sytuacji, gdy mamy do czynienia z klasyczną, hierarchiczną strukturą zarządzania - gdzie określone stanowisko pracy jednoznacznie definiuje określony zakres obowiązków. Dzięki temu przepływ informacji pomiędzy pracownikami w naturalny sposób dostosowuje się do struktury organizacji.

---

[20] J. Cadle, *Zarządzanie procesem tworzenia systemów informacyjnych*, s. 273

2.1.1 Komunikacja w modelu kaskadowym

Struktury hierarchiczne jako model komunikacji wykorzystują bardzo często model Webera[21]. W modelu tym komunikacja przebiega wzdłuż ścieżek zależności pomiędzy pracownikami. Dodatkowo, struktury takie cementują zależności pomiędzy pracownikami oraz pewne modele zachowań. Jak to obcesowo wyjaśnia Jack Welch z General Electric - hierarchia jest organizacją zwróconą twarzą w stronę dyrekcji, a tyłkiem do klienta[22].



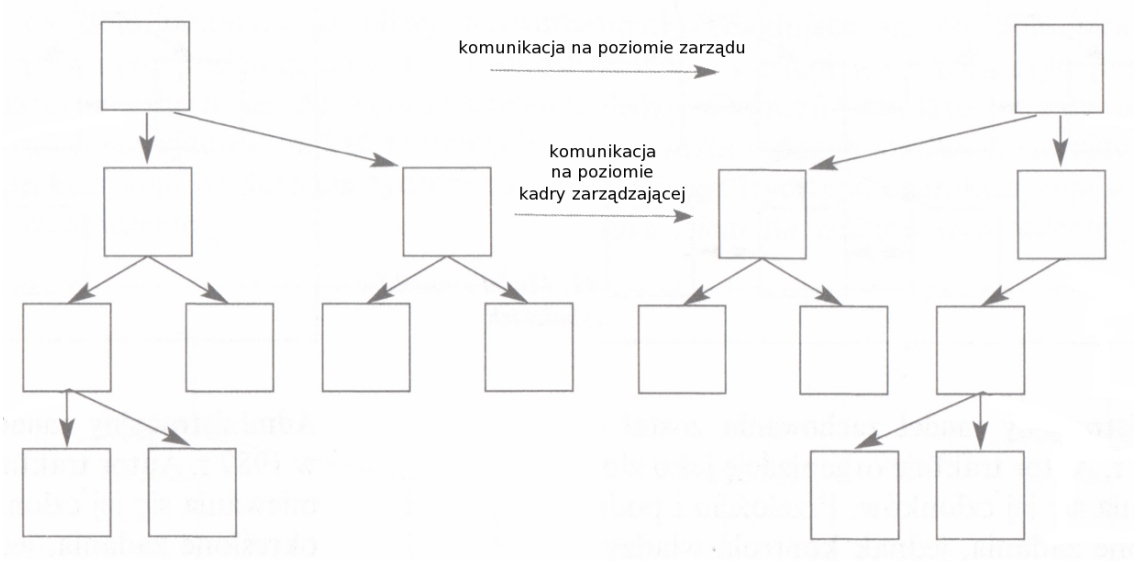
Rys. 2 Model biurokracji Webera (*źródło: B. Dobek-Ostrowska, Podstawy komunikowania społecznego, s. 117*)

Warto się jednak zastanowić, w jaki sposób przebiega komunikacja pomiędzy zespołami współpracującymi nad danym projektem, które to zespoły należą do różnych organizacji. Komunikacja taka przypomina model Webera z tą różnicą, że komunikacja zachodzi do określonego poziomu organizacji. Co za tym idzie, osoby bezpośrednio odpowiedzialne za produkt nie prowadzą rozmów dotyczących jego funkcjonalności.

---

[21] B. Dobek-Ostrowska, *Podstawy komunikowania społecznego*, s. 116

[22] K. A. Nordström, J. Ridderstråle, *Funky biznes*, WIG-Press, Warszawa 2001, s.135



Rys. 3 Model Webera w komunikacji między organizacjami (*źródło własne*)

W tym modelu komunikacji należy zwrócić uwagę na fakt ograniczonej komunikacji pomiędzy zespołami. Zachodzi ona tylko do wyznaczonego przez obie organizacje poziomu. Ma to oczywiście uzasadnienie jeżeli weźmiemy pod uwagę teorię zarządzania. Nieograniczona komunikacja pomiędzy zespołami mogłaby prowadzić do niekontrolowanej wymiany informacji pomiędzy pracownikami różnych organizacji. Warto też wspomnieć o tym, że priorytety oraz interesy osób na różnych poziomach w organizacji nie muszą być do końca zgodne. Interesy sponsora (zarządu) wcale nie muszą być takie same jak interesy użytkowników (najniższy poziom w hierarchii).

> *Sponsor, to osoba odpowiadająca przed zarządem za inwestycję (...) będzie więc: definiować biznesowe zamierzenia przedsięwzięcia, uzasadniać przedsięwzięcie przed zarządem, otrzymywać zgodę na niezbędne inwestycje, inicjować przedsięwzięcie, nadzorować je, informować zarząd o postępach, w razie konieczności będzie przerywać przedsięwzięcie. Sponsor jest więc „właścicielem" produktu, ale nie musi być jego użytkownikiem*[23].

[23] J. Cadle, *Zarządzanie procesem tworzenia systemów informacyjnych*, s. 45

Z kolei „zwykły użytkownik” ma inne priorytety. *Użytkownik, to osoba która będzie korzystała z udogodnień systemu w codziennej pracy, a więc osoba, na którą najbardziej bezpośrednio będzie miało wpływ dane przedsięwzięcie. Użytkownik będzie:*

> *(...) definiował szczegółowe wymagania systemu, upewniał się że system obsługuje funkcje biznesowe, uczestniczył w testach systemu. Te wszystkie kwestie wymagają od użytkownika tego, aby dokładnie wiedział zarówno jakie cechy będą w systemie zaimplementowane, ale również to jak mają one działać*[24].

Takie rozróżnienia pomiędzy sponsorem a użytkownikiem, prowadzą do ciekawego problemu. Mianowicie, grupy najbardziej zainteresowane wymianą informacji są tej możliwości pozbawione. Osoby najbardziej odpowiedzialne za wytwarzanie produktu (programiści) nie mogą komunikować się z tymi, którzy z tego produktu będą później korzystać (z użytkownikami). Taka sytuacja musi się w oczywisty sposób odbić na końcowym rezultacie współpracy. Bardzo często dochodzi do tego, że informacja w trakcie przemieszczania się z najniższego poziomu hierarchii w jednej organizacji do najniższego poziomu hierarchii w drugiej organizacji, zostaje zniekształcona. Jest to bardzo niekorzystne, ponieważ w efekcie prawdziwe oczekiwania użytkowników mogą być źle sprecyzowane.

2.1.2 Zarządzanie zmianą w modelu kaskadowym

Model kaskadowy posiada zasadniczą wadę - jest nieodporny na zmiany. Zakłada się w nim, że komunikacja pomiędzy zleceniodawcą a twórcą systemu jest, zawsze pełna i kompletna już na etapie specyfikowania wymagań. Jest to bardzo odważne założenie i często nie mające nic wspólnego z rzeczywistością. D. Leffingwell określa ten problem mianem *problemu „kamienia”*

---

[24] J. Cadle, *Zarządzanie procesem tworzenia systemów informacyjnych*, s. 46

Zagadnienia te, są zagadnieniami typu «przynieś mi kamień». Kiedy przynosimy klientowi kamień, klient przygląda mu się przez chwilę i mówi: «Tak, ale to, czego tak naprawdę chciałem, miało być małym niebieskim kamieniem». Po otrzymaniu małego niebieskiego kamienia klient wysuwa następne żądanie - chce, by mały niebieski kamień był na dodatek okrągły[25].

Prowadzi to do sytuacji, w której produkt dostarczony klientowi w żadnej mierze nie spełnia jego oczekiwań. Dodatkowo, w modelu tym zakłada się, że przejścia pomiędzy fazami są płynne, a komunikacja pełna. Słowem przyjmuje się, że zawsze mamy do czynienia z pełną wiedzą na temat produktu oraz efektów prac poszczególnych grup ludzi. Trzeba z pełną świadomością przyznać, że taki obraz jest niezwykle naiwny. Niemniej jednak, model ten był często stosowany.

Krytykowanie modelu kaskadowego w obszarze zarządzania zmianą nie zawsze jest jednak uzasadnione. W przypadku projektów o ścisłym harmonogramie oraz projektów wymagających procedur przetargowych, ten model ma sens. W sytuacjach, gdy projekt obarczony jest procedurami wynikającymi z ustaw regulujących życie publiczne ciężko wręcz sobie wyobrazić sytuację, w której podmioty mogą dokonywać swobodnej redefinicji wymagań (np. po zakończeniu procedur przetargowych). W takiej sytuacji, niestety, podmioty mają związane ręce i muszą podążać raz wybraną ścieżką. W takich przypadkach model ten posiada jeszcze jedną ważną zaletę - pełną dokumentację. Dzięki temu w każdym momencie trwania projektu można zweryfikować jego zgodność z pierwotnymi założeniami. Niemniej jednak, co zostało już wcześniej powiedziane, model ten utrudnia dystrybucję wiedzy pomiędzy wszystkimi zainteresowanymi stronami.

---

[25] D. Leffingwell, *Zarządzanie wymaganiami*, WNT, Warszawa 2003, s. XIX

2.1.3 Współdzielenie wiedzy w modelu kaskadowym

Zasadniczym problemem modelu kaskadowego jest brak możliwości ścisłej współpracy pomiędzy zleceniodawcą, a stroną dostarczającą produkt. Zleceniodawca pytany jest o zdanie tylko raz - na samym początku projektu. Potem wszystko toczy się własnym torem. W takiej sytuacji mówienie o kognitywnym współdzieleniu wiedzy jest wręcz niemożliwe[26]. Jednostki biorące udział w przedsięwzięciu nie mają możliwości swobodnej wymiany informacji, współdzielenia się wiedzą, czy wymiany informacji inaczej, jak za pomocą ustalonych z góry reguł. Taka sytuacja jest dalece niekomfortowa. Wydaje się, że proponowane przez kognitywistów metody współdzielenia wiedzy nie mogą w ogóle w tego typu metodykach zaistnieć. Sztuczne ograniczenie liczby punktów styku na linii użytkownik-projektant prowadzi do sytuacji, w której oba zespoły pracują niejako oddzielnie. Dodatkowym problemem jest brak możliwości redefiniowania raz ustalonych kwestii.

2.2 Klasyczne metodyki projektowe - model spiralny

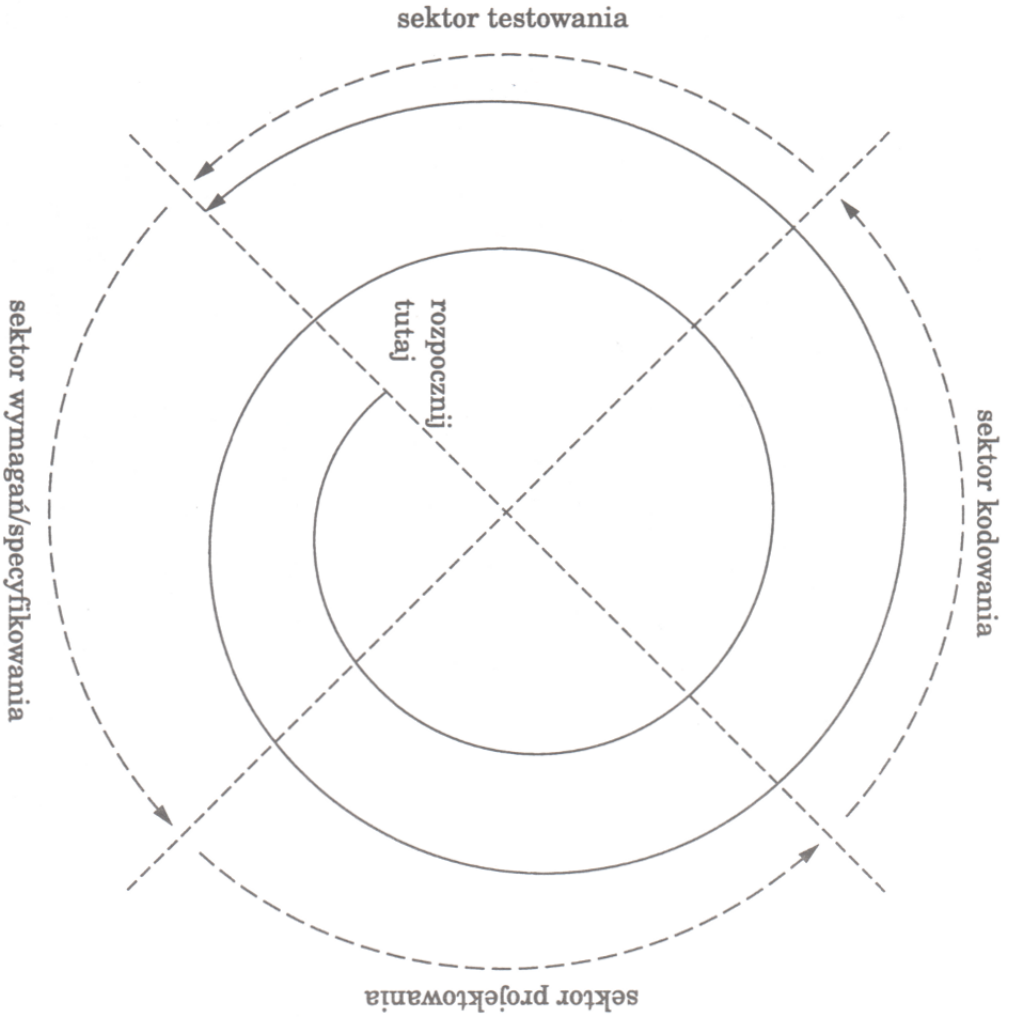
Próbą sprostania niewydolnościom modelu kaskadowego było wprowadzenie do inżynierii oprogramowanie modelu spiralnego. Pierwsze wzmianki o takim sposobie prowadzenia projektów pojawiły się w artykule Barry'ego Boehma w 1986 roku[27], zatem prawie dwadzieścia lat od momentu pojawienia się pierwszych wzmianek w literaturze dotyczących modelu kaskadowego. Model spiralny różni się od modelu kaskadowego wprowadzeniem dodatkowych możliwości analizy i „poprawiania" wymagań stawianych przez użytkowników. W modelu spiralnym, fazy projektowe następują po sobie dopóty, dopóki ostateczna wersja produktu nie spełnia wymagań klienta. Poprawiona zostaje znacznie komunikacja klient-twórca.

---

[26] E. Hutchins, *Distributed Cognition,* s. 2

[27] G. Boehm, *A Spiral Model of Software Development and Enhancement*, CM SIGSOFT Software Engineering Notes Volume 11 , Issue 4

Model ten zakłada cykliczne występowanie faz: zbierania wymagań, projektowania, kodowania, testowania (rys. 4).



Rys. 4 Model spiralny (*źródło: D.Hamlet, Podstawy techniczne inż. oprogramowania, s. 99*)

Każda z faz następuje po sobie (podobnie jak w modelu kaskadowym). Zasadniczą różnicą jest tutaj natomiast możliwość weryfikacji ścieżki, którą podąża całe przedsięwzięcie. Każdy powrót do fazy specyfikacji wymagań pozwala na ponowny kontakt z klientem i weryfikację założeń dotyczących produktu. Mówiąc obrazowo, językiem Leffingwella, każdy powrót do fazy zbierania wymagań to powrót do klienta z kolejnym kamieniem - trochę innym od poprzedniego. Jeżeli chodzi o same fazy tworzenia produktu programowego, to są one w zasadzie zgodne z tymi w modelu kaskadowym. Następują po

sobie, są niezależne - w sensie komunikacji pomiędzy ich uczestnikami, wymagają jasnego określenia kompetencji osób biorących udział w danej fazie. Zasadniczym minusem tego modelu jest fakt, że każda iteracja może trać bardzo długo (czas jest tutaj liczony raczej w miesiącach i latach, niż w dniach). Dodatkowo, nadal mamy do czynienia z brakiem pełnej komunikacji pomiędzy wszystkimi uczestnikami przedsięwzięcia.

2.2.1 Komunikacja w modelu spiralnym

W modelu spiralnym, komunikacja przypomina model Webera. Mamy do czynienia z hierarchicznym przekazywaniem informacji pomiędzy członkami zespołu. Pojawia się jednak zasadnicza różnica - komunikacja nie jest już jednostronna. W modelu kaskadowym, komunikaty przebiegały tylko w kierunku od osób odpowiedzialnych za bardziej ogólny opis systemu do osób odpowiedzialnych za bardziej szczegółowy opis. W modelu spiralnym, komunikacja przebiega również w drugą stronę. Osoby odpowiedzialne za szczegóły systemu przekazują wiedzę dotyczącą detali implementacji oraz gotowy produkt architektom systemu. Ci z kolei, bazując na tym co otrzymali, mogą nanosić odpowiednie poprawki, dzięki czemu mogą ulepszać produkt.

2.2.2 Zarządzanie zmianą w modelu spiralnym

Model spiralny dopuszcza wprowadzanie zmian w procesie tworzenia aplikacji. Dzięki temu istnieją możliwości korygowania decyzji błędnych, bądź zbędnych - zarówno po stronie zleceniodawcy jak i zleceniobiorcy. Kolejne powroty do fazy wymagań sprawiają, że klienci i producenci mogą na nowo definiować założenia dotyczące końcowego produktu. W przypadku podjęcia decyzji o zmianach proces przebiega tak, jak każdej z poprzednich iteracji. Do pracy przystępują architekci, którzy na nowo opisują zachowanie systemu. Następnie programiści dokonują niezbędnych zmian w produkcie - tak, aby odzwierciedlały poprawki architektoniczne, aby wreszcie testerzy mogli dokonać

weryfikacji założeń architektonicznych poprzez testowanie gotowego produktu. Tak przygotowana aplikacja zostaje wdrożona i oddana klientowi, który ponownie może zgłaszać swoje uwagi i przystąpić do ponownej dyskusji nad projektem.

Już w tym momencie warto zasygnalizować kwestię definicji iteracji. W metodykach klasycznych iteracja zakłada stworzenie gotowego produktu - niezależnie od tego czy klient jest w pełni usatysfakcjonowany czy nie. Kolejne iteracje wiążą się zwykle z ponownym przystąpieniem do negocjacji. W tym sensie nie możemy mówić o ciągłości jednego procesu. Jest to raczej zbiór niezależnych, ale powiązanych ze sobą oddzielnych procesów.

2.2.3 Współdzielenie wiedzy w modelu spiralnym

W modelu spiralnym, podobnie jak w modelu kaskadowym, brakuje możliwości ścisłej współpracy pomiędzy zleceniodawcą, a stroną dostarczającą produkt. Wprawdzie zleceniodawca ma możliwość weryfikacji planów i wprowadzania zmian do produktu, niemniej jednak takie zmiany mogą być wprowadzane tylko w sytuacji ponownego powrotu do fazy specyfikacji wymagań. Jest to oczywiście bardziej komfortowa sytuacja niż w modelu kaskadowym - dopuszczamy możliwość wprowadzenia zmian, niemniej jednak nadal nie posiadamy swobody wymiany informacji i nadal podlegamy ograniczeniom organizacyjnym. Tak jak w modelu kaskadowym, wydaje się, że kognitywistyczne metody współdzielenia wiedzy nie mogą w ogóle w tej metodyce zaistnieć. A już na pewno nie z taką siłą jak powinny. I tu mamy do czynienia ze sztucznym ograniczeniem liczby punktów styku na linii użytkownik-projektant, co z kolei prowadzi do sytuacji, w której oba zespoły pracują niejako oddzielnie.

2.3 Rational Unified Process jako metoda opisu procesów w klasycznych metodykach projektowych

Twórcy klasycznych metodyk projektowych doskonale zdawali sobie sprawę z problemów wynikających ze złej komunikacji pomiędzy członkami zespołów projektowych. Do problemu tego dosyć zabawnie, ale jakże prawdziwie podchodzi Tom DeMarco:

> *Nie mamy zielonego pojęcia o rozwiązywaniu konfliktów - Tompkins mówił do swego menedżerskiego zespołu marzeń. - I nie mówię tylko o nas, ale o całej branży (...) - Konflikty są wszechobecne w naszej branży - ciągnął Tompkins. - Nie da się stworzyć i wdrożyć nawet najmniejszego systemu, by nie natknąć się na jakiś poważny konflikt. Konflikty są wszechobecne w naszej branży, ale nie wiemy o nich praktycznie nic.*[28]

Można powiedzieć, że niemal każda dziedzina działalności człowieka ociera się o konflikty. Niemniej jednak specyfika komunikacji w projektach informatycznych jest szczególna. Jak już zostało wcześniej wspomniane, mamy w tych projektach do czynienia ze swego rodzaju zamianą wymagań przyszłych użytkowników systemu na programy wyrażone w określonych językach programowania. Musi więc dojść do pełnego porozumienia pomiędzy twórcami systemu, a tymi, którzy systemu będą używać. Mało tego, „tłumaczenie" idei może przebiegać w wielu krokach i na każdym etapie może mieć inną formułę - czym innym jest opis architektury systemu, a czym innym jest pisanie aplikacji komputerowej.

Jedną z metod pozwalających na sprawny opis projektu jest wykorzystanie metodyki Rational Unified Process[29] (RUP). Metodyka ta pozwala na dokładne określenie zależności pomiędzy uczestnikami procesu, precyzyjne modelowanie systemu, formalizowanie wymiany informacji pomiędzy poszczególnymi etapami projektu (wykorzystywany jest do tego język

---

[28] T. DeMarco, *Zdążyć przed terminem*, Wydawnictwo Studio Emka, Warszawa 2002, s. 204

[29] http://www-01.ibm.com/software/awdtools/rup/ - *stan na 16.05.2010*

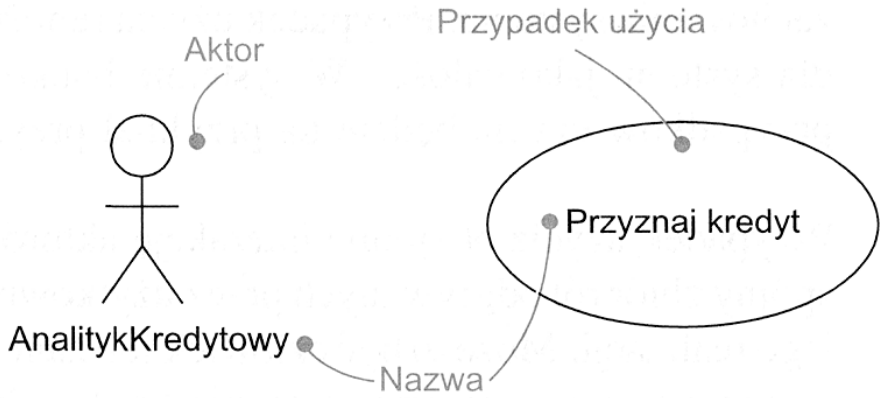
UML[30] - Unified Modeling Language). Metodyka ta stara się usystematyzować proces w jak największym stopniu, dzięki czemu ma on być łatwo weryfikowalny na każdym jego etapie. Sama idea RUP jest słuszna, niemniej jednak w realizacji może być trudna.

Metodyka RUP stawia sobie za jeden z głównych celów usprawnienie komunikacji pomiędzy członkami zespołu pracującego nad określonym zagadnieniem. Aby ten proces przebiegał sprawnie i aby redukować wszelkie nieporozumienia do minimum, zakłada się w RUP konieczność istnienia uniwersalnej platformy komunikacji. Tą platformą jest język UML. Jest to sztuczny twór, który przy pomocy piktogramów, opisów słownych i języków formalnych ma na celu ujednolicenie opisu zagadnień związanych z wytwarzaniem oprogramowania. Warto zwrócić uwagę, że jest to ukłon w kierunku koncepcji Latoura, który postuluje, że to właśnie takie artefakty pozwalają na znaczne polepszenie komunikacji pomiędzy członkami zespołów, oraz umożliwiają wymianę i generowanie nowej wiedzy[31]. Główną ideą tego języka jest możliwość dekompozycji systemu od najbardziej ogólnych stwierdzeń użytkowników (np. „użytkownik powinien mieć możliwość wygenerowania raportu w formacie PDF") do bardzo szczegółowych modeli systemu komputerowego (na poziomie pojedynczych wywołań metod obiektów - jeden z najniższych poziomów opisu działania aplikacji komputerowej). Taka „uniwersalność" języka musi nieść ze sobą pewne koszty. Jednym z nich jest dosyć wysoki stopień komplikacji samego języka (dozwolone piktogramy, sposób zapisu diagramów, sposób opisu problemów). Poniższe przykłady ilustrują „pojemność" UMLa. Z jednej strony powinniśmy mieć możliwość opisu najbardziej ogólnych ram systemu (rys. 5)

---

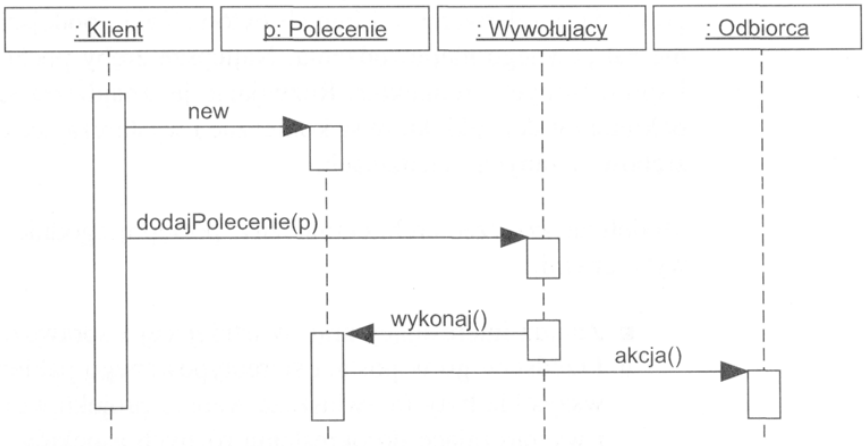
[30] http://www.uml.org/ - *stan na 16.05.2010*

[31] R. N. Giere, B. Moffatt, *Distributed Cognition: Where the Cognitive and the Social Merge*, s. 5



Rys. 5 Przykładowy Use-case - przypadek użycia (źródło: G. Boch, *UML przewodnik użytkownika, s. 226*)

z drugiej strony chcemy mieć możliwość szczegółowego opisu interakcji na poziomie aplikacji (rys. 6).



Rys. 6 Przykładowy model interakcji (źródło: G. Boch, *UML przewodnik użytkownika, s. 403*)

Takie ujęcie problemu ma jeden zasadniczy cel - całość systemu powinna być opisywalna przy pomocy pojedynczego narzędzia (języka). Pozwala to na redukcje narzędzi niezbędnych do wytworzenia systemu oraz na ujednolicenie platformy komunikacji. Ma to jednak poważną wadę - stopień skomplikowania języka. W podstawowym ujęciu UML składa się z czternastu diagramów - każdy opisujący inne aspekty systemu. Można sobie wyobrazić, że opanowanie rodziny tak różnorodnych sposobów opisu sytemu jest trudne - co zresztą traktowane jest jako jeden z podstawowych argumentów na rzecz porzucenia tego typu metodyk projektowych.

Innym aspektem RUP jest formalizacja ścieżek komunikacyjnych oraz zależności pomiędzy członkami zespołu. Rational Unified Process wprowadza ogromną liczbę przeróżnych stanowisk, z którymi wiążą się różne zakresy obowiązków (jest to około dwudziestu różnych ról wewnątrz grupy projektowej - patrz Dodatek B). Przy takiej różnorodności stanowisk można sobie jedynie wyobrazić jak bardzo musi być sformalizowana komunikacja pomiędzy pracownikami - nie można tutaj mówić o indywidualizmie czy łatwym współdzieleniu wiedzy. Według Hutchinsa, to wzorce zachowań sprawiają, że pojawia się wiedza „uwspólniona" - *Praktyki kulturowe łączą ludzi dzięki czemu przyczyniają się do tego iż pojawiają się pewne wzorce zachowań. Niektóre z tych zachowań są realizowane jedynie przez jednostki inne - rozprzestrzeniają się na całą grupę oraz artefakty. Wzorce zachowań, które cyklicznie występują w ramach praktyk kulturowych mogą prowadzić do pojawiania się pewnych zależności funkcjonalnych - zanik zachowań zbędnych oraz rozprzestrzenianie się zachowań pożądanych*[32]*.* Pojawia się tutaj ciekawe pytanie - jeżeli w metodyce Rational Unified Process kładzie się bardzo duży nacisk na ostre definiowanie zakresu obowiązków członków zespołu, to siłą rzeczy określa się pewne ramy komunikacyjne pomiędzy osobami należącymi do danej grupy, tym samym narzuca się pewne ramy „kulturowe" wynikające z żargonu, którym się posługują, wykonywanych czynności, czy realizowanych zadań. W takiej sytuacji ciężko jest mówić o wymianie i współdzieleniu wiedzy pomiędzy wszystkimi uczestnikami procesu, ponieważ nie ma przestrzeni niezbędnej do tego, aby wiedza mogła się upowszechniać lub zanikać poprzez odpowiednie wahania entropii. Oczywiście, dzięki formalizacji komunikacji i ujednoliceniu języka możemy mówić o pewnym przenoszeniu informacji do artefaktów - w tym przypadku dokumentacji - niemniej jednak ryzykownym byłoby stwierdzenie, że wiedza ta jest w miarę równy sposób współdzielona przez wszystkich członków zespołu projektowego.

---

[32] E. Hutchins, *Distributed Cognition*, s. 5 (*tekst źródłowy*: *patrz Dodatek C pkt 1*)

2.4 Podsumowanie

Klasyczne metodyki projektowe starają się odpowiedzieć na pytanie jak usprawnić proces i w jaki sposób dobrze go udokumentować. Niestety, nie przykładają dużej wagi do relacji międzyludzkich i do komunikacji nieformalnej. Zakładają też, że do współdzielenia wiedzy konieczna jest jakaś uniwersalna platforma - metajęzyk pozwalający na opisanie artefaktów, komunikacji oraz procesów rządzących przedsięwzięciem. Te praktyki, konieczne ze względu na formalne wymogi przedsięwzięcia, mogą być niejednokrotnie zastąpione bardziej elastycznymi i mniej oficjalnymi formami współpracy. Wydaje się, że w niektórych sytuacjach taki stopień formalizacji przedsięwzięcia ma uzasadnienie. Niemniej jednak może prowadzić do niepotrzebnego rozrostu aparatu pojęciowego i żargonu jakim posługuje się grupa. W takiej sytuacji koszt „wejścia” nowych osób do grupy jest bardzo wysoki - muszą przyswoić sobie język formalny oraz zrozumieć przy jego pomocy intencje twórców systemu. W metodykach zwinnych nacisk zostaje położony na inne aspekty współpracy - na interakcje, współdzielenie wiedzy i jej rozproszenie w zespole.

3 Komunikacja w zwinnych zespołach projektowych

Zwinne metodyki projektowe są odpowiedzią na „ociężałe" metodyki klasyczne.

> *Jest to koncepcja zorientowana na czynnik ludzki, niejako na przekór obiegowym wyobrażeniom menadżerów o ludziach jako wzajemnie uzależnionych „zębatkach" w skomplikowanej machinie programistycznej. Zrywają one generalnie z pojmowaniem procesu tworzenia oprogramowania według proceduralnych przepisów (na wzór książki kucharskiej), na rzecz prostych i klarownych reguł i praktyk współdziałania*[33].

W metodykach tych ogromny nacisk kładzie się na wymianę informacji, współdzielenie wiedzy i współtworzenie produktu przez wszystkich członków zespołu. Metodyki zwinne, mimo tego, że mają podobne założenia (zgodne są z manifestem Agile Software Development - patrz Dodatek A), to jednak różnią się w sposobie przekazywania wiedzy i sposobie zarządzania zespołem. Powstały one w oparciu o przekonanie, że nie wszystko co techniczne musi być w sposób techniczny zarządzane - są one rezultatem zanegowania „pewników" rządzących światem inżynierii oprogramowania. *Najlepszym możliwym przekonaniem jest to, że wszelkie powszechne przekonania są błędne*[34], mawiał Ken Olsen (założyciel Digital Equipment Corp.) - co jest to bardzo bliskie kartezjańskiemu przeświadczeniu o konieczności redefiniowania pojęć i sceptycznemu podchodzeniu do wszelki prawd ostatecznych

---

[33] D. Astels, G. Miller, M. Novak, *eXtreme programming*, HELION 2002, s. 11

[34] K. A. Nordström, J. Ridderstråle, *Funky biznes*, s.100

To prawda, iż nie widzimy, aby burzono wszystkie domy w mieście jedynie po to, aby je przebudować w inny sposób i upiększyć dzięki temu ulice; widzimy jednak, iż wielu burzy swoje domostwa, aby je odbudować na nowo: niekiedy nawet zmuszeni są do tego, kiedy domom grozi zawalenie, a fundamenty nie są dosyć mocne (...) odnośnie zaś wszystkich przekonań, jakie we mnie dotąd wpojono, nie mogę uczynić nic lepszego, jak zabrać się do usunięcia ich z siebie na dobre, aby później wstawić w to miejsce albo inne lepsze, albo nawet te same, jeśli tylko dostosuję je do miary rozumu"[35].

Na tym gruncie negacji, tego co powszechnie było uznane za najlepsze w świecie inżynierii oprogramowania, narodziły się metody „*zwinne*". Negacji poddane zostały najważniejsze elementy klasycznych metodyk projektowych: konieczność istnienia precyzyjnie zdefiniowanego procesu, konieczność istnienia dokumentacji, konieczność realizacji produktu zgodnie z pierwotnymi założeniami, konieczność dokładnego spisywania kontraktów. Zamiast skupiać się na technicznych aspektach procesu, zaczęto wysuwać na pierwszy plan ludzi (jednostki), interakcję międzyludzką oraz współdzielenie się informacjami. Zaczęto realizować idee komunikacji opartej o kognitywne współtworzenie wiedzy. Najbardziej znanymi i powszechnie uznawanymi metodykami, w świecie inżynierii oprogramowania są metodyki *eXtreme Programming*[36] oraz *Scrum*[37]. Co ciekawe, mimo toczącej się w branży IT debaty na temat wyższości jednej ze wspomnianych metod nad drugą, są one w pewnym sensie komplementarne. Wprawdzie każda z nich opiera się na manifeście Agile Software Development, niemniej jednak, *eXtreme Programming* skupia się na technicznych aspektach programowania (implementowania aplikacji), natomiast *Scrum* na procesach międzyludzkich. Nie daje jednoznacznej odpowiedzi w jaki sposób programować, a raczej stara się odpowiedzieć na pytanie jak się

---

[35] Descartes, *Rozprawa o metodzie*, str. 19

[36] http://www.extremeprogramming.org/ - *stan na 16.05.2010*

[37] http://www.scrumalliance.org/ - *stan na 16.05.2010*

komunikować. W tym sensie, obie metody mogą być stosowane równolegle, chociaż są to raczej rozważania teoretyczne[38].

W niniejszym rozdziale obie metodyki zostaną omówione w kontekście komunikacji wewnętrznej oraz realizacji założeń kognitywistów w kwestiach społecznego tworzenia wiedzy. Zostaną one również poddane krytyce, jeżeli chodzi o problemy związane z dużymi przedsięwzięciami informatycznymi.

3.1 Zwinne metodyki projektowe - eXtreme Programming

Zwinne metodyki projektowe, w swoich założeniach chcą uciec od formalizacji procesów, co jest związane z odmiennym podejściem do „zarządzania" ludźmi.

> *Formalizacja procesu ma swoje zalety i wady. Istnieje tendencja do opracowywania zasad przystosowanych do tzw. „przeciętnego człowieka" w danej firmie - cóż jednak począć, gdy wyobraźnia na temat „przeciętności" w konkretnych warunkach rozmija się z rzeczywistością*[39]?

Pytanie, czy jest możliwe stworzenie takiego modelu postępowania w grupie, który jednak pozwoli na uzyskanie lepszych efektów współpracy między członkami zespołu. Kent Beck, analizując problemy występujące w klasycznych metodykach projektowych, postanowił zhumanizować proces wytwarzania oprogramowania. Jego proces wytwarzania produktów programowych kładzie nacisk na pewne elementy procesu, które wg Becka, są kluczowe, aby osiągnąć sukces. Model ten jest:

- Modularny - podział całości na mniejsze części
- Iteracyjny - daną rzecz tworzymy poprzez aplikowanie kolejnych iteracji

---

[38] Softhouse, *Scrum in five minutes*, http://www.softhouse.se/Uploades/Scrum_eng_webb.pdf, s.11

[39] D. Astels, *eXtreme programming*, s. 15

- Przyrostowy - całość tworzymy z małych, pozwalających na pełne ich poznanie części
- Ograniczony czasowo - każda część projektu posiada nieprzekraczalne ramy czasowe
- Oszczędny - minimalizacja działań (czym prostsze działania, tym łatwiejsze zarządzanie ryzykiem)
- Zdolny do adaptacji - nowe potrzeby, to nowe działania (należy być gotowym na zmiany)
- Zbieżny - sens mają tylko te działania, które prowadzą do celu
- Zorientowany na ludzi - dobre rzeczy tworzone są przez zdolnych ludzi (człowiek nie może być redukowany do „koła zębatego")
- Zorientowany na współdziałanie - dobre działanie jest tam, gdzie dobra komunikacja
- Komplementarny - działania cząstkowe oddziałują na siebie, ważne aby się uzupełniały, a nie były sobie przeciwstawiane[40]

Powyższe zasady stanowią trzon metodyki eXtreme Programming. Bardzo ważnym elementem, na który warto zwrócić uwagę, jest umieszczenie czynnika ludzkiego jako elementu ważnego z punktu widzenia procesu. Pojawia się jednak pytanie, czy to co zostało zdefiniowane powyżej, nie jest niczym innym, jak tylko kolejnym procesem?

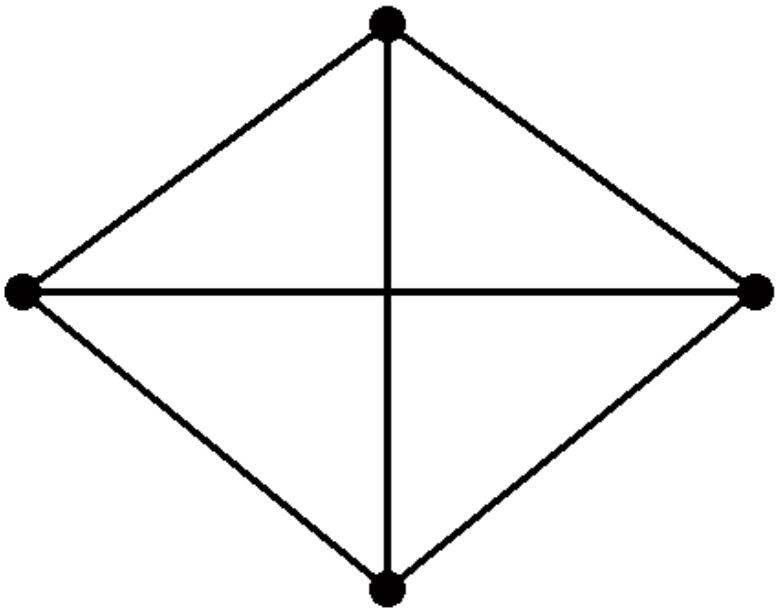
W pewnym sensie można powiedzieć, że Kent Beck chcąc uciec od definiowanych odgórnie procesów, sam wpadł w pułapkę bycia nieomylnym. Z założenia odrzuca on wszelkie metodyki oparte o eXtreme Programming, jeżeli rezygnują one z któregokolwiek opisanego przez niego postulatu. Niemniej jednak należy przyznać, że metodyka ta jest bardziej miękka niż techniczne metodyki klasyczne.

Czynnik ludzki odgrywa kluczową rolę w eXtreme Programming. Ludzie, jako podstawa procesu, są odpowiedzialni za tworzenie wiedzy i współdzielenie jej pomiędzy wszystkimi członkami zespołu. Dzięki takiemu podejściu do

---

[40] D. Astels, *eXtreme programming*, s. 13

problemu przestajemy mieć do czynienia z ograniczeniami w komunikacji. Komunikacja nie przypomina już klasycznego modelu Webera, ale jest swobodną wymianą informacji pomiędzy członkami grupy odpowiadającej za produkt (rys. 7).

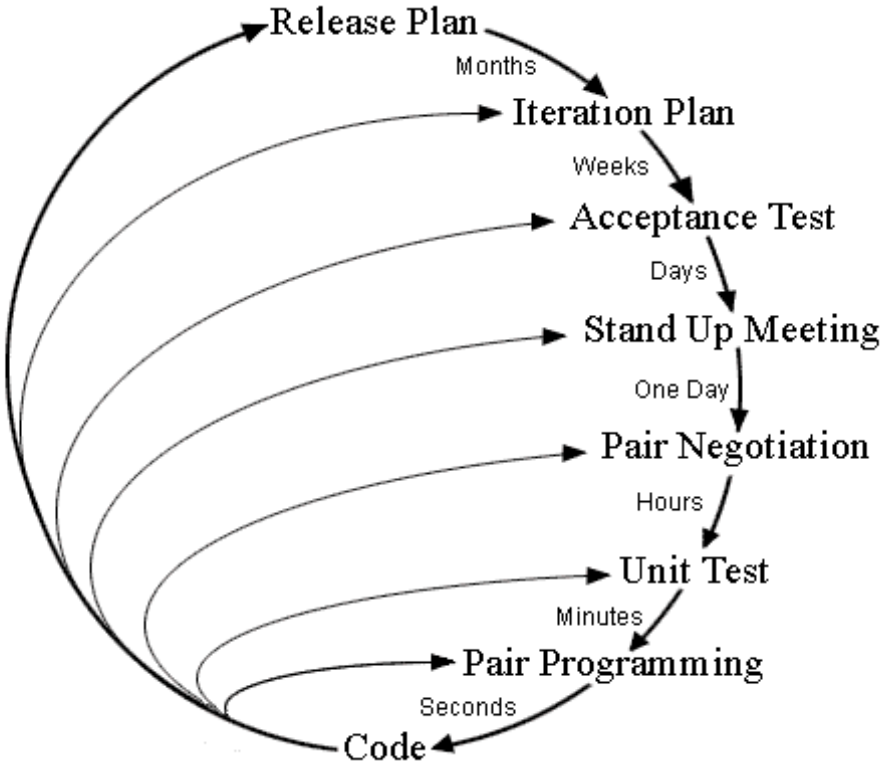


Rys. 7 Komunikacja w zespołach korzystających z metodyki eXtreme programming (*źródło własne*)

Dzięki takiemu modelowi komunikacji informacje przepływają w grupie swobodnie, nie są filtrowane, a biorące udział w przedsięwzięciu osoby mają możliwość wyrażania swoich opinii nie tylko wewnątrz zespołu programistów, architektów. Hierarchia oraz zbiór zależności pomiędzy jednostkami został znacznie ograniczony w porównaniu do metodyki RUP (patrz Dodatek B).

Kolejnym ważnym założeniem metodyki eXtreme Programming jest przyjęcie jako pewnika, że zmiana jest kwestią czasu. Chodzi tu o wszelkie, mogące się pojawić w trakcie trwania projektu, zmiany. Nie mamy już do czynienia ze sztucznym bytem w postaci „wodospadu" kolejnych etapów przedsięwzięcia. Pojawia się w jego miejsce zbiór niekończących się iteracji. Oczywiście w praktyce, tak czy inaczej, narzucone zostają pewne ramy czasowe, pewien zakres funkcjonalny aplikacji. Niemniej jednak istnienie zmian

jest jednym z kluczowych założeń metodyki. Iteracji podlega tutaj każdy z elementów procesu: programowanie, dobieranie się w pary członków zespołu, testowanie aplikacji, spotkania robocze, testy akceptacyjne, modyfikacje planów. Słowem, zakłada się, że wszystkie te praktyki są powielane i realizowane aż do końca trwania projektu (rys. 8).



Rys. 8 Przebieg projektu zgodnie z metodyką eXtreme Programming (*źródło: http:// www.extremeprogramming.org/introduction.html - stan na 16.05.2010*)

Przyjęcie, że zmiana oraz iteracja „rządzą" projektem jest o tyle komfortowe, że tak jak w codziennym życiu, tak i w trakcie trwania jakiegokolwiek przedsięwzięcia, zmiana jest tylko kwestią czasu, natomiast realizacja naszych celów to wykonywanie małych kroczków, następujących po sobie, a nie pojedynczych skoków.

Takie ujęcie problemu wprowadza swego rodzaju komfort psychiczny wśród członków zespołu. Zmiana nie jest już postrzegana jako zagrożenie, ale

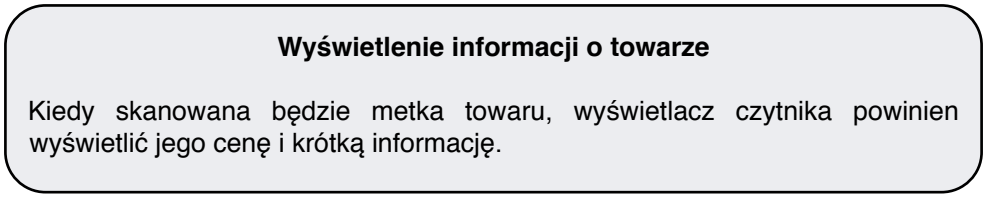
sytuacja generująca nowe możliwości. To, z kolei, prowadzi do znacznego polepszenia komfortu komunikacji pomiędzy członkami zespołu.

3.1.1 Komunikacja w modelu eXtreme Programming

eXtreme Programming zakłada niezwykłe uproszczenie hierarchii w ramach zespołów realizujących dany projekt. Możemy mówić wręcz o całkowitym spłaszczeniu struktur, dzięki czemu przepływ informacji nie jest krępowany zależnościami pomiędzy pracownikami. Jedyny, wyróżnionymi w tej strukturze osobami, są *Klient* (Customer) oraz *Kierownik* (Manager). Niemniej jednak, bardzo ważne w tej metodyce jest to, że wyróżnienie tych dwóch ról nie prowadzi do pojawienia się bariery komunikacyjnej. Wynika to z tego, że *Klient* bierze aktywny udział w tworzeniu produktu. Rezultaty kolejnych iteracji są prezentowane *Klientowi* natychmiast po ich realizacji i oczekują na testy akceptacyjne. Jeżeli okaże się, że testy te kończą się pozytywnie, zespół przechodzi do kolejnej iteracji. Jeżeli okaże się, że wypadają one niepomyślnie, zespół musi ponownie dokonać koniecznych zmian w danym fragmencie oprogramowania. Inną sytuacją jest ta, w której *Klient* stwierdza, że rezultat który widzi, nie jest dokładnie tym, czego oczekiwał. W takiej sytuacji następuje renegocjacja założeń wstępnych i powtórna implementacja. Wydawać by się mogło, że taka praktyka jest zabójcza dla każdego projektu - *Klient* może przecież w nieskończoność zmieniać zdanie co do oczekiwanej formy produktu, który zamówił. Takiej sytuacji może przeciwdziałać jedynie poprawne ustalenie reguł współpracy zanim rozpocznie się sam proces wytwarzania produktu.

Jednym z podstawowych elementów komunikacji w projektach eXtreme Programming są tak zwane historie użytkownika. Rezygnuje się tutaj z twardego definiowania wymagań na rzecz snucia opowieści. Jest to zbliżenie się do praktyk grupowych wymiany informacji. Już nie formalizm czy diagramy są ważne, a opowieść z perspektywy przyszłego użytkownika. Jest to bardzo ważny aspekt tej metodyki, ponieważ kładzie ona nacisk nie na to *jak* tworzyć aplikację komputerową, ale *co* ma zostać stworzone. Przykładowa historia

użytkownika może wyglądać tak, jak zostało to przedstawione na poniższym rysunku.



Dzięki takiemu podejściu do problemu pozwalamy wszystkim członkom grupy na to, aby posiadali wspólną platformę komunikacji. Opowiedzieć swoją historię potrafi praktycznie każdy. Dzięki temu osoba stojąca przed problemem zdefiniowania wymagań, nie musi się obawiać braku znajomości odpowiedniego żargonu - po prostu opowiada swoją historię.

Zaprezentowana powyżej forma komunikacji ma jedną zasadniczą przewagę nad metodykami klasycznymi: klient staje się częścią grupy tworzącej produkt. Staje się pełnoprawnym członkiem zespołu, ze wszystkimi tego konsekwencjami - jeżeli chodzi o komunikację, wymianę informacji, udział w generowaniu nowej wiedzy. Wydaje się, że taka formuła komunikacji jest niezbędna, niezależnie od tego jakiego modelu poznania rozproszonego byśmy nie przyjęli. Abstrahując od tego, czy będziemy posługiwa się modelem umysłu we wspólnocie (Mind in Society), wspólnoty umysłów (The Society of Mind)[41], czy rozproszenia czynności w czasie[42]. Branie czynnego udziału w praktykach kulturowych, a takimi można nazwać jakikolwiek proces wytwórczy, jest elementem koniecznym korzystania z wiedzy rozproszonej pomiędzy członkami zespołu. Tak więc metodyka eXtreme Programming w jak największym stopniu (na ile to możliwe) pozwala na współdzielenie wiedzy pomiędzy wszystkimi partnerami biznesowymi.

---

[41] E. Hutchins, *Distributed Cognition*, s. 2

[42] J. Podgórski, *Charakterystyka pojęcia kolektywu „rozproszonego”*, s. 154

3.1.2 Zarządzanie zmianą w modelu eXtreme Programming

Zmiana jest nieodzowna wszelkim procesom wytwórczym. Ważne jest, aby grupa potrafiła zmianę przyjąć w taki sposób, aby koszty adaptacji były jak najniższe. Klasyczne modele projektowe zakładały, że zmiany w ogóle nie występują (model kaskadowy) lub występują stosunkowo rzadko (model spiralny). Metodyki zwinne przyjmują z goła inne założenie. Zmiana jest nieodzowna procesowi wytwarzania oprogramowania.

Zmiana w metodyce eXtreme Programming wiąże się z przejściem do kolejnej iteracji. Jest to czas kolejnego kontaktu z klientem oraz czas wymiany informacji pomiędzy grupą projektową, a osobą zamawiającą produkt. Ważne jest, że iteracje w projektach zwinnych są znacznie krótsze, niż w klasycznych metodykach. W przypadku metodyk zwinnych jest to czas od kilkunastu dni do miesiąca. W metodykach klasycznych, iteracje mogą być liczone nawet w latach. Należy zwrócić uwagę na jeszcze jeden fakt: przed rozpoczęciem każdej z iteracji w procesie ustalania jej przebiegu bierze również udział klient[43]. Zwykle proces ten wygląda następująco: grupa projektowa prezentuje klientowi nową iteracje (gotowy, działający produkt), klient ocenia, w jakim stopniu to co otrzymał, pokrywa się z jego oczekiwaniami. Jeżeli okaże się, że w produkcie należy wprowadzić zmiany, to razem z grupą projektową przygotowuje nową listę zadań do realizacji. W kolejnym kroku, grupa projektowa (wykorzystując metodę burzy mózgów) ocenia swoje możliwości w kontekście nowych zadań przeznaczonych do realizacji w trakcie następnej iteracji. Po tej fazie następuje implementacja. Taki model komunikacji sprawia, że wiedza jest non-stop przetwarzana przez wszystkich członków zespołu oraz prezentowana klientowi w bardzo krótkich odstępach czasu. Pozwala to na współdzielenie wizji tego, nad czym wszyscy członkowie przedsięwzięcia pracują (zarówno klient jak i twórcy systemu).

---

[43] W. C. Wake, *Extreme Programming Explored*, s. 119

3.1.3 Współdzielenie wiedzy w modelu eXtreme Programming

Podstawową jednostką bezpośredniego dzielenia się wiedzą oraz jej tworzenia w eXtreme Programming są krótkie spotkania (stand-up meetings[44]). Podczas spotkań tego typu przekazywane są najważniejsze informacje pomiędzy członkami zespołu. Każda osoba zwięźle opowiada co udało jej się osiągnąć dnia poprzedniego, mówi otwarcie o problemach, które napotkała oraz propozycjach możliwych rozwiązań. Dzięki temu wszyscy członkowie zespołu wiedzą dokładnie „na czym stoją". Taki sposób współpracy ma zasadniczą zaletę. Mobilizuje do dokładnej i rzetelnej pracy. Wynika to z tego, że eXtreme programming postuluje pracę w parach. W takiej sytuacji, osoby rzetelne mogą nie być zainteresowane współpracą z osobami nieprzykładającymi się do pracy, co może ostatecznie skutkować separacją „nielubianych" pracowników.

Programowanie w parach to drugi element skutecznego współdzielenia się informacją i generowania nowej wiedzy. W metodyce eXtreme Programming zakłada się, że osoby pracujące nad zadanym problemem rozwiązują go razem[45]. Przed przystąpieniem do pracy członkowie zespołu dobierają się w pary, w taki sposób że każdego dnia mogą współpracować z kim innym, a następnie rozpoczynają pracę. Dzięki temu każdy członek zespołu zna większość zagadnień dotyczących projektu, nad którym pracuje cały zespół. Programowanie w parach wprowadza dodatkowy element „ochronny" - nasze działania są na bieżąco analizowane przez drugą osobę (lub odwrotnie). Dzięki temu istnieje mniejsze ryzyko popełnienia błędów oraz większe szanse na znalezienie optymalnego rozwiązania w danej sytuacji.

eXtreme Programming kładzie bardzo duży nacisk na współdzielenie się wiedzą oraz współprace między członkami zespołu. Jest jednak krytykowane za bardzo rygorystyczne podejście do praktyki związanej z tą metodyką. Zakłada się, że nie są możliwe żadne odstępstwa od reguł rządzących eXtreme Programming. Prowadzić to może do napięć między pracownikami np.

---

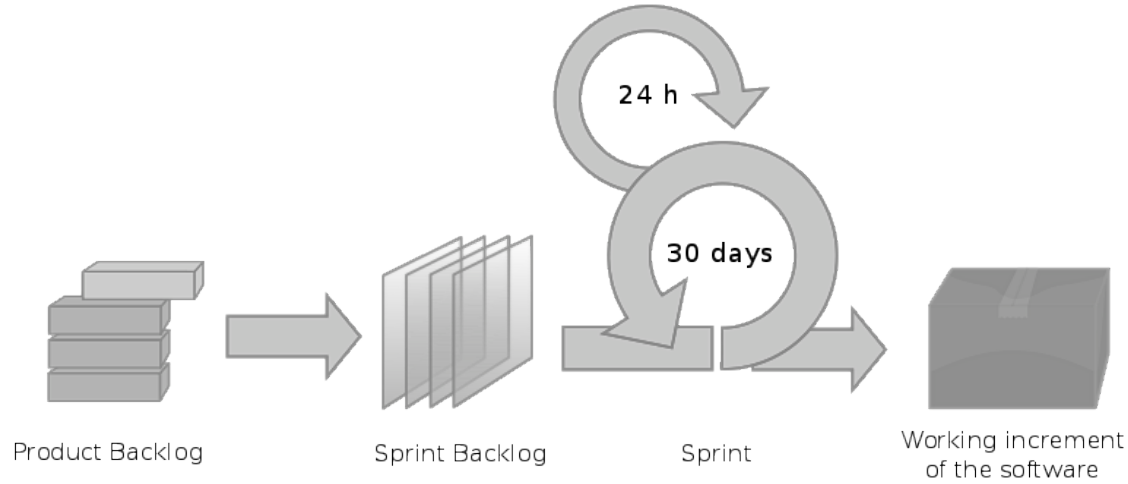
[44] K. Beck, *Planning Extreme Programming*, s. 87

[45] W. C. Wake, *Extreme Programming Explored*, s. 65

w sytuacji różnych preferencji co do czasu pracy. Problemy tego typu stara się rozwiązać inna metodyka zwinna - Scrum.

3.2 Zwinne metodyki projektowe - Scrum

Scrum to kolejna zwinna metodyka projektowa. Praca zespołu projektowego posługującego się nią zorganizowana jest wokół tak zwanych „sprintów" - iteracji, które trwają zwykle od dwóch do czterech tygodni. Podczas każdego ze sprintów zespół wybiera zadania, z listy priorytetów ułożonej przez klienta (listy historii użytkownika). Zadania te są implementowane i dostarczane klientowi w postaci potencjalnie gotowego do wdrożenia produktu[46] (*źródło: patrz Dodatek C, pkt 2*). Przebieg procesu wytwarzania oprogramowania wykorzystującego metodykę Scrum przedstawiony został na rys. 9. Zadania z listy historii użytkownika (*ang. Product Backlog* lub *user stories*) są przenoszone do listy zadań realizowanych w ramach sprintu (Sprint Backlog). Podczas każdego sprintu tworzony jest produkt, który mógłby być potencjalnie dostarczony i używany jako gotowe rozwiązanie (oczywiście z ograniczoną funkcjonalnością).



Rys. 9 Przebieg projektu zgodnie z metodyką Scrum (*źródło: http://en.wikipedia.org/wiki/Scrum_(development) - stan na 16.05.2010*)

[46] http://www.scrumalliance.org/learn_about_scrum - *stan na 16.05.2010*

Scrum jest modelem podobnym do tego jaki prezentowany był przez metodykę eXtreme Programming, niemniej jednak zasadniczą różnicą jest fakt, że to klient „zarządza" projektem. W metodyce Scrum klient bierze aktywny udział poprzez cykliczne „sortowanie" listy priorytetów. W ten sposób klient ma bezpośredni wpływ na kształt produktu, natomiast zespół ma pewność, że realizowane funkcjonalności są faktycznie najbardziej pożądanymi.

Kolejnym novum w metodyce Scrum jest zmiana roli managera projektu. Zwykle w klasycznych metodykach projektowych manager jest swego rodzaju buforem pomiędzy zespołem, a klientem, i nie bierze bezpośredniego udziału w procesie tworzenia produktu. W metodyce Scrum manager jest zarówno członkiem zespołu (tworzy produkt) jak i osobą odpowiedzialną za organizację pracy zespołu. Manager jest odpowiedzialny za prowadzenie spotkań, dbanie o artefakty, terminologię oraz płynny przebieg procesu[47].

Inną znaczącą zmianą w stosunku do metodyki eXtreme Programming jest rezygnacja z programowania w parach. W modelu Scrum kładzie się raczej nacisk na komunikację między wszystkimi członkami grupy i na sprawny przebieg codziennych spotkań (ang. stand-up meeting). Bolączką w dużych projektach informatycznych są spotkania projektowe. Bardzo często sprowadzają się one do niekończących się dysput (bardzo często ideologicznych), zamiast skupiać się na meritum problemu.

> *Spotkanie robocze powinno mieć program, powinno być krótkie i na temat, powinno wymagać obecności jedynie zainteresowanych osób, powinno być prowadzone zgodnie z agendą (szczególnie powinno się kłaść nacisk na stronę związaną z czasem trwania poszczególnych punktów programu), uczestnicy powinni być pewni, że omówione zostaną tylko kwestie wymienione w agendzie*[48].

---

[47] K. Schwabe, *Agile Project Management with Scrum*, s. 31

[48] T. DeMarco, *Zdążyć przed terminem*, s. 252

W metodyce Scrum, postulat ten realizowany jest właśnie przez spotkania „na stojąco” (*ang. stand-up meeting*). Podczas spotkania zainteresowane osoby (jest to zwykle cały zespół - bo de facto każdy członek zespołu odpowiedzialny jest za każdą z części systemu) omawiają w ciągu kilkunastu minut co zostało zrobione, czego nie mogli zrobić, co powoduje blokadę ich pracy. Po takim spotkaniu wszyscy członkowie zespołu przechodzą do pracy nad własnymi zadaniami.

Dzięki wprowadzeniu swobodnej wymiany informacji, model ten promuje współdzielenie wiedzy ponad dokumentowanie jej. Ma to oczywiście ujemne strony - zostaną one omówione w części poświęconej współdzieleniu wiedzy.

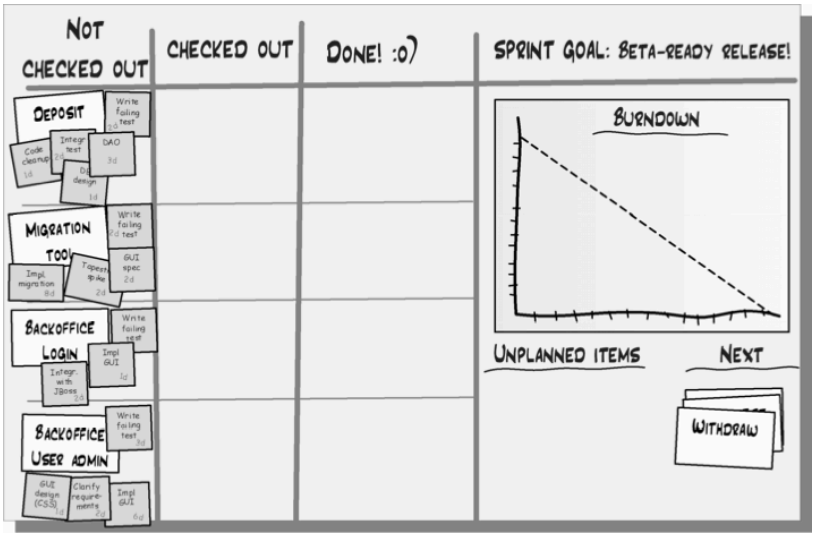
3.2.1 Komunikacja w modelu Scrum

Komunikacja w metodyce Scrum opiera się na trzech podstawowych elementach:

- swobodnym przepływie informacji pomiędzy klientem a zespołem,
- swobodnym przepływie informacji pomiędzy członkami zespołu,
- pełnej informacji o aktualnym stanie prac.

Te trzy elementy sprawiają, że komunikacja (podobnie jak w modelu eXtreme Programming) jest swobodna, a pracownicy mogą się komunikować w modelu „każdy z każdym”. Zachęca się wręcz członków takich zespołów do tego, by pracowali w jednym pomieszczeniu[49] (jest to o tyle możliwe, że zespoły te liczą kilka osób), aby mieli możliwość wymiany informacji gdy tylko zajdzie taka potrzeba (na przykład poprzez spontaniczne dyskusje) oraz na bieżąco sprawdzali swoje postępy prac w odniesieniu do założonego planu. Charakterystyczne dla tej metodyki jest umieszczenie w widocznym miejscu planu projektu (rys. 10).

[49] H. Kriberg, *Scrum and XP from the Trendres*, s. 57



Rys. 10 Przykładowy wygląd tablicy z zadaniami do realizacji w ramach projektu Scrum (*źródło:* K. Schwabe, *Agile Project Management with Scrum*, s. 46)

Oczywiście sama metodyka nie narzuca ani formy, ani miejsca umieszczenia takiej tablicy - niemniej jednak zalecane jest aby była ona dobrze widoczna dla wszystkich osób pracujących w zespole, oraz aby każdy miał do niej swobodny dostęp.

Krytykowana przez przeciwników eXtreme Programming zasada programowania w parach nie jest tutaj stosowana. Programowanie w parach posiada zalety (większa mobilizacja do pracy, mniejsze prawdopodobieństwo popełnienia błędów podczas pisania programu, lepsze zrozumienie problemu), ale posiada też i wady. Jedną z podstawowych wad jest kwestia posiadanego doświadczenia. Każda z pracujących osób w projekcie posiada inne doświadczenie, jest biegła w jednej dziedzinie natomiast słabsza w innej. Z tego powodu, praca w parach może prowadzić do niepotrzebnych napięć pomiędzy współpracującymi osobami - osoba bardziej kompetentna będzie zirytowana sposobem pracy osoby mniej kompetentnej, natomiast ta druga będzie odczuwała presję z zewnątrz. Nie jest powiedziane, że tak musi być zawsze, niemniej jednak w metodyce Scrum eliminuje się to ryzyko poprzez rezygnację z programowania w parach na rzecz wymiany informacji wtedy, gdy taka konieczność istnieje. Sprzyja temu umiejscowienie wszystkich członków

zespołu w jednym pomieszczeniu w taki sposób, że mogą ze sobą swobodnie rozmawiać.

3.2.2 Zarządzanie zmianą w modelu Scrum

Metodyka Scrum, tak jak inne metodyki zwinne, zakłada że zmiany są nieodzownym elementem procesu tworzenia aplikacji. Zmiana „rządzi" projektem. Nie jest tutaj ważne, czy zmiana pochodzi z wewnątrz zespołu, czy też z zewnątrz. Ważne jest natomiast to, że osoby tworzące system zakładają, że to co stworzą będzie poddane krytyce i być może będzie musiało być kompletnie zmienione. Jest to zasadnicza zmiana w stosunku do podejścia klasycznego. W modelach klasycznych zmiana jest czymś niepożądanym, jest traktowana jako zło konieczne, a ścieżka tworzenia produktu jest wytyczona przez kontrakt, zakres wymagań oraz architekturę systemu.

Metodyka Scrum angażuje klienta w pracę nad projektem w sposób podobny do metodyki eXtreme Programming. Nie ma się zresztą czemu dziwić, ponieważ obie metodyki starają się realizować postulaty Manifestu Agile Development. Jednym z jego punktów brzmi: *Współpraca z klientem zamiast negocjowania kontraktów*[50] *(ang.: Customer collaboration over contract negotiation)*. Po otrzymaniu kolejnej iteracji, klient omawia z zespołem elementy, które pozostały do zaimplementowania i decyduje, które z nich mają dla niego kluczowe znaczenie, które z elementów mogą być oddalone w czasie - jeżeli chodzi o ich realizację, a z których można w ogóle zrezygnować. Mówiąc kolokwialnie, zespół zbiera się przed tablicą z zadaniami do realizacji i ustala przebieg prac wchodzących w skład kolejnej iteracji. Klient, dzięki łatwemu dostępowi do tablicy, może na bieżąco analizować przebieg prac.

3.2.3 Współdzielenie wiedzy w modelu Scrum

Współdzielenie wiedzy w modelu Scrum to w dużej mierze oparta o ciągłą komunikację wymiana informacji. W modelu tym wszystkie osoby

---

[50] http://agilemanifesto.org/ - *stan na 16.05.2010*

zainteresowane projektem mogą czerpać wiedzę z doświadczeń innych. Służą temu codzienne, krótkie, ale treściwe, spotkania, podczas których wszyscy prezentują swoje postępy oraz napotkane problemy. Wymianie informacji służy sposób ulokowania pracowników - jedno, przestronne pomieszczenie pozwalające zarazem uczestniczyć w dyskusjach, jak i skupić się na pracy. W modelach idealnych pomieszczenie takie składa się z dwóch, odseparowanych części. Z sali dyskusyjnej oraz sali, w której można w skupieniu pracować. Oczywiście osobnym pytaniem jest to, czy firma bądź zespół projektowy może sobie na takie warunki pracy pozwolić.

Inną kwestią, wartą podkreślenia, jest kwestia dystrybucji wiedzy pomiędzy członkami zespołu. W projektach typu Scrum de facto nie mamy do czynienia z dokumentacją projektową sensu stricte. Wiedza dotycząca projektu sprowadza się do opowieści użytkownika, zbioru notatek na tablicy projektowej oraz tego co zostało do tej pory stworzone. Z jednej strony jest to wystarczające, ponieważ wiedza i tak jest współdzielona przez wszystkie zainteresowane strony. Jednakże patrząc na tę kwestię z „politycznego" i strategicznego punktu widzenia sytuacja nie jest aż tak komfortowa jakby mogło się na pierwszy rzut oka wydawać. Mamy tutaj do czynienia z dwoma podstawowymi zagrożeniami: całkowity brak możliwości transferu wiedzy poza grupę projektową oraz katastrofalny wpływ na zespół sytuacji, w której członkowie zespołu opuszczają grupę.

Pierwszy z problemów jest bezpośrednim wynikiem rezygnacji z pełnej dokumentacji projektowej. W klasycznej metodyce projektowej klient, hipotetycznie, może przenieść projekt do innego dostawcy. Będzie to kosztowne, niemniej jednak wykonalne. W przypadku zespołu pracującego w myśl metodyki Scrum, jest to praktycznie niemożliwe. Siłą, ale jednocześnie słabością zespołów Scrumowych jest to, że wiedza jest w pełni rozproszona pomiędzy członków zespołu. W przypadku pozytywnego przebiegu projektu mamy do czynienia z efektem synergii, co skutkuje pojawieniem się wartości dodanej w postaci wzrostu wiedzy w całej grupie. Jednakże w przypadku

problemów, to co było siłą zaczyna być słabością. Wiedzy nie można, ot tak, wyciągnąć z głów poszczególnych jednostek.

Drugi problem, czyli potencjalna rezygnacja członków zespołu ze współpracy, również niesie negatywne skutki. Ze względu na to, że czas spędzony nad tworzeniem aplikacji oraz osobista wiedza składają się na doświadczenie poszczególnych osób, w przypadku ich rezygnacji z dalszego udziału w przedsięwzięciu cały zespół traci potencjalne źródło informacji i wiedzy. Wprawdzie idea współtworzenia programu przez wszystkich ma na celu minimalizację tego typu problemów, niemniej jednak ma to wpływ na resztę zespołu - szczególnie na nowych jego członków. W klasycznych metodach projektowych nową osobę w zespole można odesłać zawsze do dokumentacji technicznej, w metodyce Scrum te możliwości są znacznie ograniczone.

3.3 Podsumowanie

Ze względu na to, że metodyki zwinne promują swobodną wymianę informacji, angażują członków zespołu w aktywną pracę oraz pozwalają na niczym nieskrępowaną komunikację pomiędzy klientem a zespołem, tworzą środowisko sprzyjające wymianie informacji i generowaniu wiedzy. Podstawowymi artefaktami w metodykach zwinnych są historie użytkownika - czyli opowieści o tym, co powinno być realizowane przez system. Jest to najprostszy i najbardziej pierwotny sposób opisu rzeczywistości - opis słowny. Jest to również najwygodniejsza forma prowadzenia dyskusji - mogę opowiedzieć moją historię i wiem że ktoś inny będzie mógł się do niej odnieść, coś dopowiedzieć, o coś dopytać. Diagramy, projekty, modele formalne są wygodne z punktu widzenia kontraktu, natomiast opowieść jest wygodna z punktu widzenia precyzowania naszych oczekiwań i odczuć związanych z daną funkcjonalnością systemu. Tutaj leży główna zaleta metodyk zwinnych.

Innym aspektem integrującym grupę projektową jest jej rozmiar. W metodykach agile kładzie się nacisk na redukowanie zespołu do niezbędnego minimum. Takie podejście pozwala na powstanie związków

emocjonalnych pomiędzy współpracującymi osobami, integrację zespołu oraz swobodne współdzielenie informacji i wiedzy. Również klient uzyskuje pewność w kwestiach dotyczących informacji o projekcie, ponieważ na bieżąco uczestniczy w pracach zespołu.

Kolejnym z elementów wpływających na relacje pomiędzy członkami grupy roboczej jest liczba ról jakie mogą pełnić poszczególne osoby. Została ona w tej metodyce ograniczona do zaledwie trzech, niezbędnych: kierownik przedsięwzięcia (ang. Scrum Master), klient (ang. Product Owner) oraz zespół (ang. Scrum Team) - patrz Dodatek B. Taka minimalizacja ról prowadzi do tego, że ścieżki formalne zostają zredukowane do minimum, a odpowiedzialność za powodzenie, bądź nie, przedsięwzięcia zostaje złożona na barki jednej osoby - kierownika przedsięwzięcia. To osoba kierownika przedsięwzięcia skupia wszystkie zainteresowane strony i dba o to, aby cały proces twórczy przebiegał sprawnie.

Dla kwestii poznania rozproszonego kluczowe jest to, że metodyka Scrum promuje wymianę informacji oraz wspólny, prosty żargon projektowy - komunikację słowną. Dzięki temu wszyscy posiadają ogólną wiedzę o wszystkim, natomiast poszczególne jednostki posiadają wiedzę szczegółową dotyczącą poszczególnych aspektów projektu. Pozwala to na budowanie nowej wiedzy w oparciu o informacje własne oraz te rozproszone wśród członków zespołu.

Zakończenie

Zwinne metodyki projektowe wydają się być, na dzień dzisiejszy, jedynymi metodykami dążącymi do realizacji idei komunikacji oraz poznania rozproszonego. W swoich założeniach oparte są o podobne podstawy z jakimi mamy do czynienia w poznaniu rozproszonym. Bazują raczej na wymianie informacji, niż jej kumulowaniu. Nie przenoszą odpowiedzialności na pojedyncze jednostki.

Latourowska koncepcja artefaktów znajduje tutaj swoje odwzorowanie w stosowaniu prostych, przejrzystych metod opisu świata. Nie są to opasłe tomy dokumentacji wymagające zaawansowanych szkoleń aby zrozumieć wiedzę w nich zawartą. Skomplikowane diagramy, języki modelowania (takie jak UML), opisujące w najdrobniejszych detalach metodyki (jak np. Rational Unified Process) zostają zastąpione „gwarem" rozmów projektowych. Jest to koncepcja zbliżona do tego, co proponuje E. S. Raymond[51] w swojej *Katedrze i bazarze*. Sformalizowane stosunki i procesy przegrywają ze spontanicznym, kreatywnym podejściem do rozwiązania zadanego problemu.

Mała, zżyta grupa, jako podstawowa jednostka projektowa, zastępuje w metodykach zwinnych skomplikowane struktury hierarchiczne dużych przedsiębiorstw. Sformalizowane ścieżki komunikacyjne zostają spłaszczone, a wymiana informacji nie podlega filtrowaniu przez kolejne szczeble struktury. Takie środowisko pracy sprzyja wymianie informacji i separacji od głównych struktur przedsiębiorstwa. Dzięki temu, zespół może skupić się na pracy, a nie „szukaniu" się wzajemnie w obrębie firmy. Zamiana „open space'u", albo „kubików", na przestrzeń zarezerwowaną tylko i wyłącznie dla danego zespołu powoduje zwiększenie identyfikacji z grupą oraz przedsięwzięciem.

Wreszcie aspekt będący często przyczyną problemów w projektach - polityka - zostaje zredukowany do minimum poprzez aktywne zaangażowanie klienta w projekt. Częste spotkania robocze, bezpośrednia współpraca z klientem, sprawiają że wiedza o produkcie nie jest współdzielona tylko przez

---

[51] E. S. Raymond, *The Cathedral and the Bazaar*

członków zespołu projektowego, ale również poprzez wszystkie zainteresowane strony. Dodatkowo, ma to pozytywny wpływ na generowanie nowych idei przez klienta. Otrzymując pełną informację o tym, co chcemy zrobić, jak chcemy to zrobić oraz gdzie już jesteśmy, klient może generować nowe pomysły dotyczące produktu, odkrywać jego nowe możliwości i bezpośrednio, iteracja za iteracją, sprawdzać czy rozwój produktu idzie w dobrym kierunku.

Metodyki zwinne bywają często poddawane krytyce ze strony osób preferujących „uporządkowane" i „usystematyzowane" metodyki klasyczne. Warto jednak nadmienić, że metodyka Scrum stosowana jest nie tylko przez małe, „rodzinne" firmy produkujące oprogramowanie. Z metodyki tej korzystają tacy giganci jak Microsoft czy IBM. Wydaje się więc, że powoli, aczkolwiek systematycznie koncepcja poznania rozproszonego realizowana przez metodyki zwinne zaczyna zdobywać przyczółki tam, gdzie miejsce zarezerwowane było tylko dla dobrze opisanych i w pełni udokumentowanych procesów.

Dodatek A

Manifest Agile Software Development

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions *over* processes and tools
Working software *over* comprehensive documentation
Customer collaboration *over* contract negotiation
Responding to change *over* following a plan

That is, while there is value in the items on
the *right*, we value the items on the *left* more[52].

[52] Manifesto for Agile Software Development, http://agilemanifesto.org/ - *stan na 16.05.2010*

Dodatek B

Zróżnicowanie liczby ról w metodykach projektowych[53]

metodyka RUP (Rational Unified Process)

Project Manager, Business-Process Analyst, Business Designer, Business Reviewer, System Analyst, Use Case Specifier, Architect, User Interface Designer, Requirements Reviewer, Designer, Database Designer, Architecture Reviewer, Design Reviewer, System Integrator Implementor, Code Reviewer, Test Designer Integration Tester, Performance Tester

metodyka eXtreme Programming

Programmer, Customer, Tester, Coach, Tracker, Manager

metodyka Scrum

Scrum Master, Product Owner, Scrum Team

[53] D. Rawsthorne, *Comparing/Combining RUP, XP, and Scrum – mixing the Process Cocktail*, www.netobjectives.com

Dodatek C

Teksty źródłowe

1. “Cultural practices assemble agencies into working assemblages and put the assemblages to work. Some of these assemblages may be entirely contained in an individual, and some may span several individuals and material artifacts. The patterns of activity that are repeatedly created in cultural practices may lead to the consolidation of functional assemblages, the atrophy of agencies that are rarely used, and the hypertrophy of agencies that are frequently employed. The result can be individual learning or organizational learning, or both”[54].

2. Scrum is an agile software development framework. Work is structured in cycles of work called sprints, iterations of work that are typically two to four weeks in duration. During each sprint, teams pull from a prioritized list of customer requirements, called user stories, so that the features that are developed first are of the highest value to the customer. At the end of each sprint, a potentially shippable product is delivered[55].

3. In these cases, the human organism is linked with an external entity in a two-way interaction, creating a coupled system that can be seen as a cognitive system in its own right. All the components in the system play an active causal role, and they jointly govern behavior in the same way that cognition usually does. If we remove the external component the system’s behavioral competence will drop, just as it would if we removed part of its brain. Our thesis is that this sort of coupled process counts equally well as a cognitive process, whether or not it is wholly in the head[56].

---

[54] E. Hutchins, *Distributed Cognition*, s. 5

[55] http://www.scrumalliance.org/learn_about_scrum - *stan na 16.05.2010*

[56] A. Clark, D. J. Chalmers, *The Extended Mind*, s. 4

Literatura

1. Ł. Afeltowicz, *Czy technika pozbawia nas pracy? Dekonstrukcja ekonomicznej koncepcji bezrobocia technologicznego*, Studia Socjologiczne 2007
2. D. Astels, *eXtreme programming*, HELION, Gliwice 2000
3. K. Beck, M. Fowler, *Planning Extreme Programming*, Addison Wesley 2000
4. G. Boch, *UML przewodnik użytkownika*, WNT, Warszawa 2001
5. J. Cadle, *Zarządzanie procesem tworzenia systemow informatycznych*, WNT, Warszawa 2004
6. Ernst Cassirer, *Symbol i język*, Wydawnictwo Naukowe Wyższej Szkoły Pedagogiki i Administracji, Poznań 2004
7. R. B. Cialdini, *Wywieranie wpływu na ludzi,* GWP, Gdańsk 2003
8. A. Clark, D. J. Chalmers, *The Extended Mind*, Analysis 58:10-23, 1998. Reprinted in (P. Grim, ed) The Philosopher's Annual, vol XXI, 1998
9. A. Cooper, *Wariaci rządzą domem wariatów*, WNT, Warszawa 2001
10. R. Dawkins, *Fenotyp rozszerzony*, Prószyński i S-ka SA, Warszawa 2003
11. T. DeMarco, *Luz*, WNT, Warszawa 2005
12. T. DeMarco, *Zdążyć przed terminem*, Studio EMKA, Warszawa 2002
13. Descartes, *Rozprawa o metodzie*, Wydawnictwo ANTYK, Kęty 2002
14. *B. Dobek-Ostrowska, Podstawy komunikowania społecznego*, ASTRUM, Warszawa 2004
15. D. Hamlet, *Podstawy techniczne inżynierii oprogramowania,* WNT, Warszawa 2003
16. R. N. Giere, B. Moffatt, *Distributed Cognition: Where the Cognitive and the Social Merge*, Social Studies of Science 33/2 (April 2003) 1–10
17. A. Hunt, *Pragmatyczny programista*, WNT, Warszawa 2002
18. Hollan J., Hutchins E., Kirsh D., *Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research,* ACM Transactions on Computer-Human Interaction, Vol. 7, No. 2, s. 174–196.

19. E. Hutchins, *Distributed Cognition,* University of California - San Diego 2000
20. H. Kriberg, *Scrum and XP from the Trendres*, InfoQ Enterprise Software Development Series 2004
21. D. Leffingwell, *Zarządzanie wymaganiami*, WNT, Warszawa 2003
22. C. P. Nemeth, *Using Cognitive Artifacts to Understand Distributed Cognition*, IEEE Transactions On Systems, Man, and Cybernetics - Vol. 34, No. 6, Nov. 2004
23. K. A. Nordström, J. Ridderstråle, *Funky biznes,* WIG-Press, Warszawa 2001
24. J. Podgórski, *Charakterystyka pojęcia kolektywu „rozproszonego”*, Poznańskie Forum Kognitywistyczne, Teksty konferencyjne t. 2, 2007
25. E. S. Raymond, *The Cathedral and the Bazaar*, Copyright © 2000 Eric S. Raymond, (http://catb.org/~esr/writings/homesteading/cathedral-bazaar/)
26. Y. Rogers, *A Brief Introduction to Distributed Cognition,* School of Cognitive and Computing Sciences, University of Sussex, 1997
27. K. Schwabe, *Agile Project Management with Scrum,* Microsoft Press 2004
28. Softhouse, *Scrum in five minutes*, http://www.softhouse.se/Uploades/Scrum_eng_webb.pdf
29. R. Sun, *Cognition and Multi-Agent Interaction*, Cambridge University Press 2006
30. Leigh L. Thompson, *Creativity and Innovation in Organizational Teams*, Lawrence Erlbaum Associates, Inc., Publishers 2006
31. W. C. Wake, *Extreme Programming Explored*, Copyright 2000, William C. Wake
32. A. Walenstein, *Cognitive Support in Software Engineering Tools: A Distributed Cognition Framework*, Simon Fraser University, May 2002
33. Robert A. Wilson, *The MIT Encyclopedia of the Cognitive Sciences*, The MIT Press, 1999
34. Ed Yourdon, *Death March*, Pearson Education Inc. 2004