# 4. Walidacja i weryfikacja modeli.

Weryfikacja to sprawdzenie zgodności systemu z określonymi wymaganiami („Are we building the system right?”). Walidacja to ocena systemu podczas lub na końcu procesu jego rozwoju na zgodność z rzeczywistymi wymaganiami użytkownika („Are we building the right system?”). Mówiąc inaczej, walidacja określa, czy system realizuje oczekiwania klienta, natomiast weryfikacja stwierdza, czy system działa prawidłowo (spełnia wymagania funkcjonalne i niefunkcjonalne). Czynności te wzajemnie się uzupełniają i są istotnym elementem procesu wytwarzania oprogramowania. Część procesu walidacji i weryfikacji (V&V, z ang. validation and verification) stanowi to, co tradycyjnie określa się mianem testowania.

Na etapie weryfikacji sprawdzane jest, czy produkty danej fazy wytwarzania są zgodne z nałożonymi na nią założeniami. Nie jest natomiast sprawdzane, czy specyfikacja jest prawidłowa – to jest przedmiotem walidacji. Weryfikacja zatem nie pozwoli na wykrycie błędów związanych z nieprawidłową specyfikacją wymagań. Walidacja natomiast sprawdza, czy oprogramowanie wykonuje to, czego wymaga od niego użytkownik, czyli jest odpowiedzialna za znalezienie błędów w specyfikacji systemu.

Weryfikacja i walidacja powinny być przeprowadzane we wszystkich fazach wytwarzania oprogramowania. Realizowane są one na dwa uzupełniające się sposoby:

* poprzez inspekcję wymagań, projektu i kodu (analiza statyczna) – analiza (może być częściowo zautomatyzowana), której celem jest znalezienie potencjalnych nieprawidłowości,
* poprzez testowanie programu (analiza dynamiczna) – uruchomienie aplikacji, zasilenie jej danymi testowymi i sprawdzenie, jakie działanie generowane jest przez system; zaobserwowane działanie porównywane jest z oczekiwanym.

Zgodnie z sugestią Ani dodaję opis metod weryfikacji i walidacji. To rozdział z mojej pracy, jest w miarę krótki, więc pozwolę sobie w większej części go tu wstawić ☺

**Sposoby oceny jakości**

Sposoby oceny jakości produktów programowych wywodzą się z metod wykrywania defektów w oprogramowaniu. Istnieją dwa podstawowe podejścia do rozwiązania tego problemu: analiza dynamiczna oraz analiza statyczna.

**Analiza dynamiczna**

Analiza dynamiczna oznacza przede wszystkim testowanie. Jest możliwa zwykle dopiero na końcowych etapach cyklu wytwarzania. W przypadku modeli UML, ten sposób oceny nie wydaje się być dobrym podejściem ze względu na konieczność opracowania szczegółowego modelu, możliwego do uruchomienia. Nawet, jeżeli model taki zostanie opracowany, podejście to umożliwi analizę jakości dopiero po zakończeniu procesu modelowania systemu, nie w jego trakcie. Zgodnie ze znanym prawem inżynierii oprogramowania, usunięcie błędów wykrytych na koniec procesu tworzenia oprogramowania (w tym wypadku jego modelu) może być zbyt kosztowne, aby to podejście uznać za opłacalne.

**Analiza statyczna**

Analiza statyczna oznacza ocenę jakości bez uruchamiania oprogramowania. Ta metoda obejmuje przeglądy, formalne dowody poprawności czy interpretacje symboliczne. Ze względu na brak konieczności uruchamiania artefaktu, analiza statyczna jest dobrym narzędziem do oceny jakości modeli.

**Interpretacje symboliczne**

Interpretacje symboliczne oznaczają analizę i interpretację tekstu programu w celu określenia przepływu danych oraz sterowania wykonywaną najczęściej przez narzędzia (np. kompilatory). Mogą one obejmować od kontroli składni po bardziej zaawansowane mechanizmy, np. analizę charakterystyk pamięciowo-czasowych. Interpretacje symboliczne nie muszą być ograniczone do kodu. W przypadku wykorzystania notacji formalnej lub półformalnej (np. języka UML, zwłaszcza wzbogaconego o warunki wyrażone w języku OCL) możliwa jest automatyczna analiza niektórych anomalii.

**Formalne dowody poprawności**

Formalne dowody poprawności obejmują wykazywanie spełnienia określonych warunków działania programu (np. asercji) przez cały czas oraz tworzenie specyfikacji systemu w języku formalnym i jej kolejne transformacje, aż do uzyskania kodu programu. Przykładem zastosowania tej metody może być użycie algebry procesów równoległych i związanej z nią notacji CSP.

Zastosowanie zarówno interpretacji symbolicznych, jak i formalnych dowodów poprawności do oceny modeli UML nie jest ani oczywiste, ani trywialne. Problem ich użycia wynika z nieformalnej semantyki języka. Ponadto, wiele aspektów jakości (np. kompletność) wynika bezpośrednio ze specyfikacji wymagań systemowych (zwykle wyrażonych nieformalnie w postaci języka naturalnego) lub z ogólnej wiedzy dziedzinowej (jeszcze mniej formalnej). Gdyby podstawę konstrukcji modelu stanowiła formalna specyfikacja, użycie opisanych w poprzednich dwóch punktach metod byłoby nie tylko możliwe, ale wręcz automatyzowalne. Niestety, podejście takie stosowane jedynie w przypadku krytycznych systemów, np. związanych z bezpieczeństwem zdrowia i życia ludzi.

**Przeglądy**

Przegląd jest to proces kontroli określonego artefaktu z udziałem osoby lub grupy osób nie będących jego autorami, mający na celu wykrycie defektów oraz możliwości ulepszeń. Ze względu na opisane powyżej wady interpretacji symbolicznej oraz dowodów formalnych podejście to wydaje się być najbardziej odpowiednim do oceny jakości modeli UML.

Idea przeglądów jest na tyle popularna i chętnie stosowana, że doczekała się wielu rozwiązań szczegółowych. Wśród nich można wyróżnić m.in.:

* przegląd zespołowy (ang. team review),
* przejście (ang. walkthrough),
* inspekcję (ang. inspection),
* programowanie w parach (ang. pair programming).

Przegląd zespołowy jest to zaplanowany i ustrukturyzowany proces, nie narzucający zbytniej formalizacji. Uczestnicy procesu pełnią następujące role:

* Lider – osoba odpowiedzialna za powodzenie przeglądu i nadzór nad nim. Zajmuje się kwestiami organizacyjnymi, przewodniczy spotkaniu, sporządza końcowy raport oraz przekazuje wyniki przeglądu autorom artefaktu poddawanego przeglądowi.
* Sekretarz – osoba odpowiedzialna za zapisywanie uwag zgłaszanych w trakcie przeglądu w czytelnej i zrozumiałej dla odbiorców (głównie autorów) formie.
* Prezenter – podczas spotkania inicjuje dyskusję i wskazuje aktualnie rozważany fragment przeglądanego artefaktu.
* Kontroler – osoba odpowiedzialna za wykrywanie defektów w przeglądanym dokumencie. Zapoznaje się on z kontrolowanym dokumentem oraz dokumentami źródłowymi i na tej podstawie wskazuje defekty, zgłaszane następnie na spotkaniu.
* Autor – osoba odpowiedzialna za wytworzenie kontrolowanego artefaktu. Wynik jego pracy jest poddawany przeglądowi, otrzymuje on też wynik procesu przeglądu. Może uczestniczyć w spotkaniach, aby służyć kontrolerom wyjaśnieniami.

Przejście jest techniką, której zadaniem jest nie tylko wykrywanie defektów, ale również zapoznanie szerszego grona z kontrolowanym dokumentem. W tej formie przeglądu dużo większe znaczenie ma rola prezentera (zwykle jednego z autorów), który referuje słuchaczom treść dokumentu w ustalonym przez siebie porządku oraz odpowiada na wątpliwości i pytania zgłaszane przez słuchaczy. Przejście jest mniej formalną techniką, niż przegląd zespołowy, gdyż nie następuje w niej np. raportowanie wyników. Wadą tego podejścia jest duży wpływ prezentującego na przebieg spotkania. Kontrolerzy nie są dogłębnie zapoznani z dokumentem, ponadto mogą nie być w stanie obiektywnie ocenić artefaktu, robiąc to raczej przez pryzmat prezentacji i osobowości prezentera.

Inspekcja jest najbardziej sformalizowanym sposobem przeglądu. Proces przeglądu – jego etapy, zakres odpowiedzialności uczestników inspekcji są dokładnie zdefiniowane. Sformalizowane mogą być nawet np. tempo przeglądu dokumentu czy struktura zgłaszania defektu. Ogranicza to uczestników inspekcji i czyni ich pracę bardziej mechaniczną, ma jednak na celu zwiększenie skuteczności procesu. W kontekście inżynierii oprogramowania najczęściej wspomina się inspekcję Fagana.

Programowanie w parach to technika przeglądu, w której uczestniczą dwie osoby – autor oraz kontroler. Technika ta powstała z myślą o wykrywaniu defektów w kodzie źródłowym, ale możliwe jest jej zastosowanie również do innych artefaktów (np. modeli). W przeciwieństwie do technik opisanych powyżej, podczas programowania w parach kontrola ma miejsce na etapie wytwarzania dokumentu. Technika ta została spopularyzowana przez Kenta Becka wraz z jedną z lekkich metodyk wytwarzania oprogramowania – programowaniem ekstremalnym (ang. extreme programming).

We wszystkich technikach przeglądu kontrolerzy mogą być pozostawieni bez żadnych wskazówek (podejście ad hoc), najczęściej jednak są oni wspomagani jakąś techniką. Najpopularniejszą jest użycie list kontrolnych (ang. checklists). Listy kontrolne zawierają zagadnienia, na które kontroler musi udzielić odpowiedzi. Zwykle zagadnienia te mają postać poleceń bądź pytań o znane rodzaje defektów.