Wprowadzenie do systemów informacyjnych

Analiza systemów informacyjnych

informatyka stosowana

Czynniki determinujące skuteczność zastosowania systemu informacyjnego

Efekt GIGO

Mimo wielu lat rozwoju ryzyko prowadzenia projektów informatycznych nadal jest duże.

W USA rozwój oprogramowania pochłania rocznie 250 mld dolarów rocznie, w postaci około 175 tys. projektów informatycznych.

Badania *Standish Group* dla rynku Amerykańskiego dowodzą, że **31%** z tych projektów jest przerywanych na jednym z etapów pośrednich, zaś **52%** projektów przekracza planowany budżet. Typowy system informatyczny jest droższy niż planowano o średnio **89%**.

Niezbędna jest właściwa

metodologia

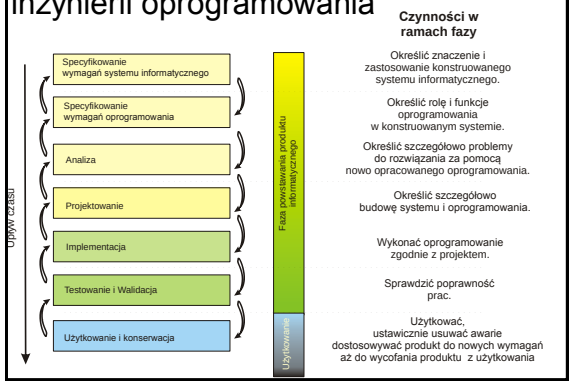
projektowania i wdrażania systemów informatycznych.

Stosowną metodologię dostarcza głównie inżynieria oprogramowania

Inżynieria oprogramowania jest praktycznym zastosowaniem wiedzy naukowej do projektowania i tworzenia systemów informacyjnych i informatycznych oraz **dokumentacji** wymaganej do ich opracowania, uruchomienia oraz pielęgnacji.

Fazy projektowania zgodnie z metodologią

inżynierii oprogramowania



Najnowsze prace odwołujące się do inżynierii oprogramowania przewidują występowanie aż **12** faz procesu projektowego.

1.Inicjalizacja systemu i wstępne planowanie:

Znalezienie potencjalnego obszaru zastosowania systemu informatycznego, który wspomoże lub zastąpi dotychczasowe metody przetwarzania informacji.

2.Analiza wymagań i specyfikacja wymagań:

Identyfikacja problemów, które nowy system informacyjny ma rozwiązać. Zarysowanie właściwości operacyjnych systemu, wydajności i zasobów sprzętowych niezbędnych do użytkowania i konserwowania systemu.

3. Specyfikacja funkcjonalna i prototypowanie:

Identyfikacja i formalizacja obiektów obliczeń, ich atrybutów i zależności. Specyfikacja transformacji, którym obiekty będą podlegać.

4. Dekompozycja problemu:

Podział systemu na logiczne podsystemy na podstawie wymagań i specyfikacji. Analiza logicznych podsystemów pod kątem użycia już istniejących komponentów informatycznych. Selekcja rozwiązań: wykonać samodzielnie, kupić, wykorzystać już istniejące.

5. Projekt architekturalny i specyfikacja konfiguracji:

Określenie zależności pomiędzy podsystemami, komponentami i modułami w sposób uwzględniający szczegóły ich budowy i wymagania wobec nich.

6. Szczegółowe projektowanie i specyfikacja komponentów:

Opracowanie i kodyfikowanie metod przetwarzania informacji w komponentach.

7. Implementacja komponentów i usuwanie błędów:

Wytwarzanie kodu źródłowego komponentów na podstawie uprzednich specyfikacji. Testowanie podstawowych funkcji komponentów.

8. Asemblacja systemu i testowanie:

Weryfikacja komponentów pod kątem kompletności i zgodności ze specyfikacją. Asemblacja produktu z komponentów, weryfikacja wydajności podsystemów systemu jako całości.

9. Przegląd dokumentacji i dostarczenie systemu:

Opracowywanie i systematyzowanie dokumentacji powstałej w trakcie prowadzenia projektu pod kątem raportów dla odbiorcy.

10. Opracowanie procedur instalacyjnych i instalacja:

Opracowanie dokumentacji instalacyjnej opisującej sposób instalacji systemu.

11. Szkolenie dla użytkowników:

Zapoznanie użytkowników z systemem. Zapoznanie ich z możliwościami i ograniczeniami systemu.

12. Użytkowanie i konserwacja oprogramowania:

Użytkowanie, usuwanie błędów dostrzeżonych w trakcie użytkowania, rozbudowywanie systemu o nowe właściwości, poprawa wydajności systemu.

Jednak nawet najlepsza metodologia nie zabezpiecza przed błędami, bo istnieje **luka poznawcza** w projektach informatycznych

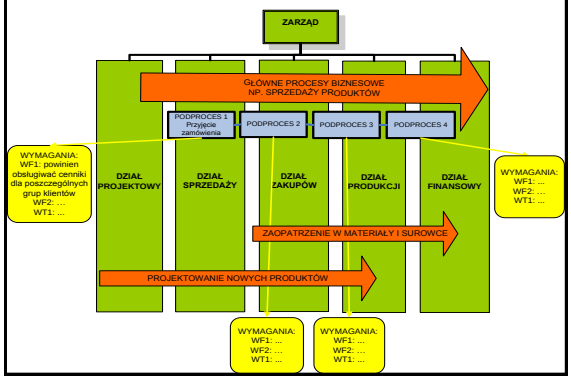
Zagadnienia zaliczające się do luki poznawczej czasem nie są w trakcie analizy dostrzegane i gdy nie zostaną wyczerpująco opracowane, to można je określać mianem ryzykownych punktów projektu.

Omówimy teraz metodykę prowadzenia analizy, która powinna to ryzyko zminimalizować.

Budowa systemu informacyjnego musi rozpocząć się od **analizy**.

Analiza ma na celu ustalenie, jaki jest aktualnie używany system informacyjny firmy, dla której tworzony jest projekt, oraz ustalenie, jaki ma być system docelowy, ten który należy zaprojektować.

Przykładowy wynik prawidłowej analizy



Kroki wchodzące w skład procesu analizy wymagań i potrzeb:

Opisanie wymagań,

Analizowanie wymagań,

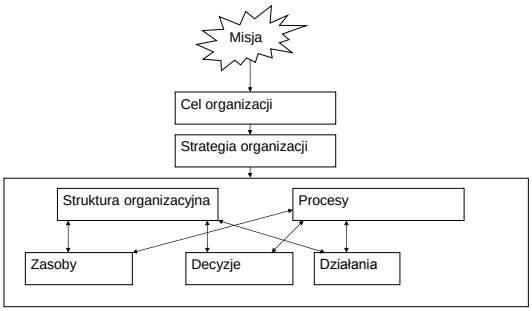
Negocjowanie wymagań,

Specyfikowanie wymagań,

Modelowanie działania systemu informacyjnego,

Zatwierdzenie wymagań.

Model podejścia analitycznego w organizacji



Każde spotkanie analityczne przeprowadzane jest w formie wywiadu prowadzonego przez projektanta i opartego o ankietę.

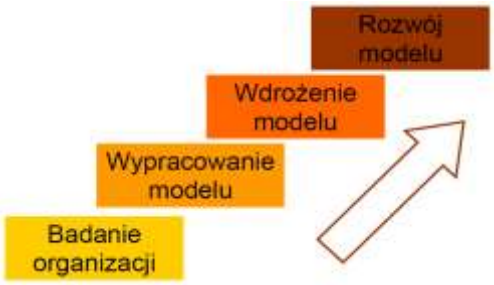
- Należy rejestrować miejsce i czas przeprowadzanego spotkania, wraz z ogólnymi ustaleniami zawartymi w trakcie analizy.
- Istnieje także potrzeba rejestrowania najważniejszych punktów analizy a także
- umożliwienia trwałego przechowywania wypełnionej w trakcie wywiadu ankiety, zawierającej wszystkie niezbędne dane potrzebne do przygotowania raportu z analizy.

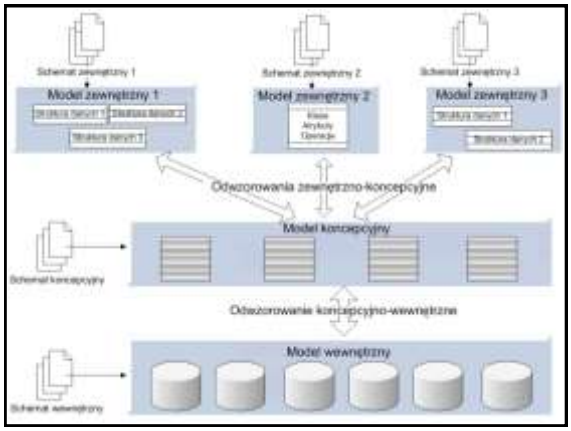
Kolejną fazą etapu, następującą po cyklu spotkań, jest opracowanie raportu z analizy, stanowiącego podsumowanie dokonanej analizy, wraz z opisem przedsiębiorstwa i ukazaniem wymogów oraz potrzeb stawianych wobec systemu.

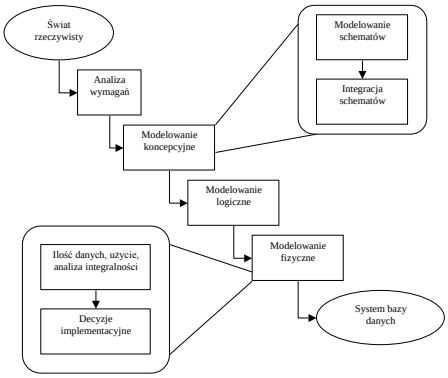
Raport standardowo sporządzany jest bazując na schemacie, dopasowywanym ściśle do danego przedsiębiorstwa, zawiera następujące pozycje:

- Wymagania ogólne;
- Finanse/Księgowość;
- Zaopatrzenie i magazyn (jeżeli istnieje dział);
- Produkcja (jeżeli istnieje dział);
- Sprzedaż;
- Zakupy;
- Infrastruktura IT;
- Inne (ściśle zależne od przedsiębiorstwa).

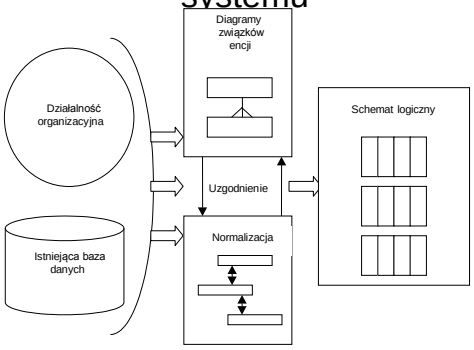
Analiza typowo wiedzie przez tworzenie serii modeli



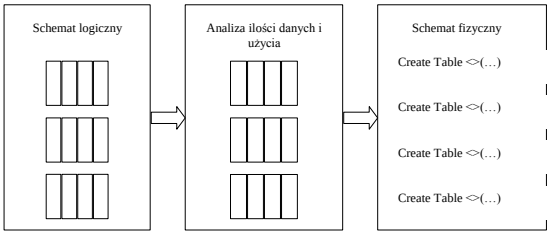




Etapy tworzenia modeli systemu



Przejście od modelu do fizycznej realizacji systemu



Badanie organizacji

- Przegląd dokumentacji projektowej
- Wywiady z przedstawicielami organizacji
  - Kierownicy projektów
  - Przedstawiciele komitetów sterujących
  - Dyrektorzy funkcyjni
  - Zarząd
  - Klienci (wewnętrzni/zewnętrzni)
- Opracowanie raportu z wynikami badań i rekomendacjami

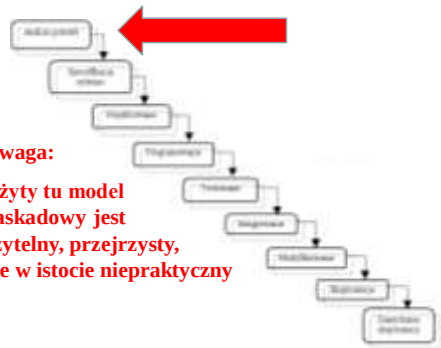
Wypracowanie modelu

- Selekcja zespołu – autorytety, agenci zmiany
- Przygotowanie merytoryczne zespołu
- Prezentacja wniosków z raportu
- Zespołowe tworzenie modelu
- Spisanie modelu
- Przyjęcie modelu przez zespół

Wdrożenie modelu

- Projekty pilotowe ze wsparciem zewnętrznym
- Optymalizacja modelu po projektach pilotowych
- Zatwierdzenie kolejnej iteracji modelu
- Konsekwentne stosowanie modelu we wszystkich nowych projektach

Proces analizy zajmuje szczególne miejsce w procesie projektowania systemu informacyjnego

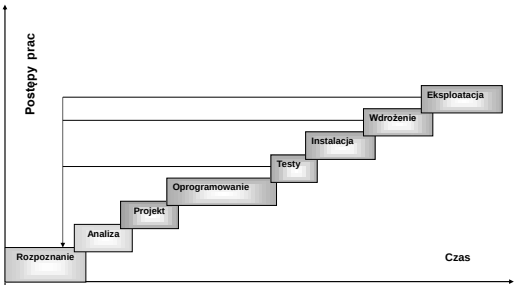


Uwaga:

Użyty tu model kaskadowy jest czytelny, przejrzysty, ale w istocie niepraktyczny

Mimo rozbicia procesu analizy i projektowania na wiele stopni „kaskady” niektóre z tych stopni mogą się okazać zbyt strome i zbyt wysokie

Wadą modelu kaskadowego jest to, że pewne czynności muszą być wykonywane **kolejno** i do kontroli dochodzi za późno



Ale to niestety z reguły niewiele pomaga.

Do zmiany sposobu myślenia o projektowaniu systemów informacyjnych zmusił informatyków

kryzys oprogramowania

Kryzys oprogramowania - symptomy (1)

Symptomy kryzysu oprogramowania:

USA:

Utrzymanie 10 mld. linii kodu istniejących programów **kosztuje 70 mld. $ rocznie**

31% nowych projektów jest anulowane przed zakończeniem; **koszt 81 mld. $**

53% projektów jest kończone z przekroczeniem zaplanowanego czasu, budżetu i z ograniczeniem planowanego zbioru funkcji systemu

Zaledwie **16%** projektów jest kończone w zaplanowanym czasie, bez przekroczenia budżetu i okrajania funkcjonalności

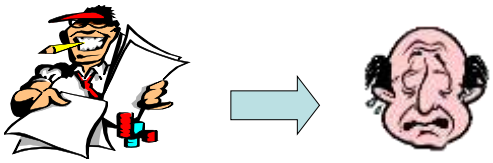
Kryzys oprogramowania - symptomy (2)

Średnia wydajność wykonawców oprogramowania spadła o 13% w ciągu dwóch lat;

Stosunek najlepszej wydajności do najgorszej od 1990 r. rozszerzył się od **4:1 do 600:1.**

- **Uzależnienie** organizacji **od systemów komputerowych** i przyjętych technologii przetwarzania informacji, które nie są stabilne w długim horyzoncie czasowym.
- **Problemy współdziałania niezależnie zbudowanego oprogramowania**, szczególnie istotne przy dzisiejszych tendencjach integracyjnych.
- **Problemy przystosowania już istniejących** i działających **systemów** do nowych wymagań, tendencji i platform sprzętowo-programowych.
- **Frustracje informatyków** wynikające ze zbyt szybkiego postępu w zakresie narzędzi i metod wytwarzania oraz uciążliwości i długotrwałości procesów produkcji i pielęgnacji oprogramowania. Znaczące zmiany w przemyśle informatycznym następują co 5-7 miesięcy w porównaniu do 5-7 lat w innych dziedzinach.

Wszystko to spowodowało, że butni i pewni siebie informatycy musieli „spuścić z tonu”

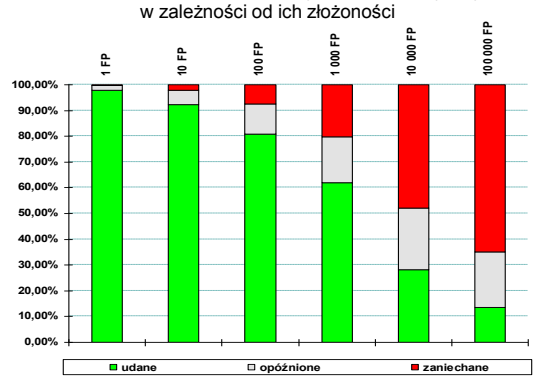


Kryzys oprogramowania - przyczyny

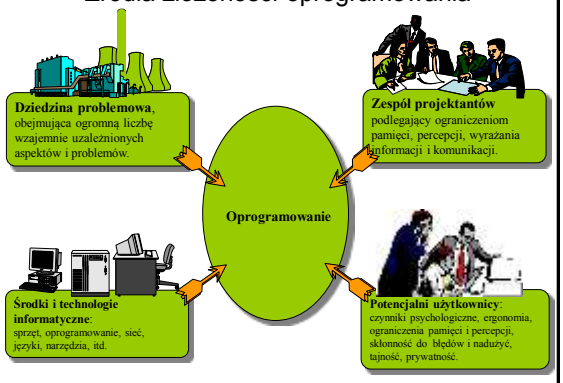
- Konflikt pomiędzy odpowiedzialnością, jaka spoczywa na współczesnych SI, a ich zawodnością wynikającą ze złożoności i ciągle niedojrzałych metod tworzenia i weryfikacji oprogramowania.
- Długi i kosztowny cykl tworzenia oprogramowania, wysokie prawdopodobieństwo niepowodzenia projektu programistycznego.
- Niska kultura ponownego użycia wytworzonych komponentów projektów i oprogramowania (reuse); niski stopień powtarzalności poszczególnych przedsięwzięć.
- Długi i kosztowny cykl życia SI, wymagający stałych (często globalnych) zmian.
- Eklektyczne, niesystematyczne narzędzia i języki programowania.

Podstawowym powodem kryzysu oprogramowania jest złożoność produktów informatyki i procesów ich wytwarzania.

Powodzenia i niepowodzenia projektów informatycznych w zależności od ich złożoności



Źródła złożoności oprogramowania



Walka ze złożonością oprogramowania (1)

Złożoność powoduje, że głównym problemem w procesie konstrukcji produktów informatycznych stał się człowiek (analityk, projektant, programista, ...) z jego różnymi uwarunkowaniami fizycznymi, psychologicznymi i mentalnymi.

Wniosek: Technologie komputerowe powinny być bardziej zorientowane na ludzi, niż na maszyny.

Co robić?

Należy wykorzystywać:

- **Mechanizmy abstrakcji -** pozwalają operować jednostkami bez wnikania w ich wewnętrzną strukturę (poprzez pominięcie mniej istotnych elementów, np. poprzez oddzielanie specyfikacji od implementacji), co znacząco ułatwia proces rozumienia; wyodrębnianie cech wspólnych i niezmiennych (inwariantnych) dla pewnego zbioru bytów.

Walka ze złożonością oprogramowania (2)

- **Mechanizmy kompozycji i dekompozycji,** czyli podział na części o dobrze wyrażonej semantyce i dobrze wyspecyfikowanej wzajemnej interakcji (**strukturalizację oprogramowania**):
  - można komponować większe jednostki oprogramowania z mniejszych; można dekomponować złożone struktury na fragmenty a następnie rozpatrywać te fragmenty niezależnie od siebie i niezależnie od całości.
- **Ponowne użycie** - pozwala na wykorzystanie wcześniej wytworzonych schematów, metod, wzorców, komponentów projektu, komponentów oprogramowania, itd.
- **Zasada sprzyjania naturalnym ludzkim własnościom** -pozwala na dopasowanie modeli pojęciowych i realizacyjnych systemów do mechanizmów percepcji i rozumienia świata przez ludzi.

Efekty:

Nie tylko wzrost efektywności procesu wytwarzania produktu programistycznego ale też i wzrost jakości oprogramowania, czyli np. poprawności, niezawodności, czytelności, testowalności, skalowalności, łatwej pielęgnacji, współdziałania, przenaszalności, itp.

Strukturalizacja oprogramowania

Korzyści jakie przynosi strukturalizacja oprogramowania:

- Jeśli wystarczy jedynie rozpoznać interface do komponentu, a nie jego szczegółową implementację, musi to zaowocować większą wydajnością pracy.
- Jeśli można bezpiecznie zignorować niektóre aspekty systemu (objęte przez wykorzystywany komponent) to większą uwagę można przyłożyć do swojej pracy, przez co mniej błędów wprowadza się do systemu.
- Dzięki strukturalizacji oprogramowania łatwiej znajduje się błędy (zarówno w trakcie budowania, jak i konserwacji systemu); nie wszystkie moduły muszą być testowane przy usuwaniu konkretnego błędu.
- Dobrze przetestowny, udokumentowany komponent może być wielokrotnie wykrzystywany (ponowne użycie).
- Modularna budowa ułatwia podział pracy.

"Nawet ubogi interface do źle skonstruowanego komponentu może uczynić system (jako całość) łatwiejszym do zrozumienia, a przez to do modyfikacji."

Ze strukturalizacją oprogramowania związane są dwa, opisane dalej, pojęcia: kohezja i skojarzenia.

Kohezja i skojarzenia

Kohezja (cohesion) oznacza zwartość, spoistość. Terminu tego używa się np. w odniesieniu do komponentu oprogramowania (klasy, modułu, itp.) na oznaczenie **wzajemnego zintegrowania** jego **elementów składowych**. Duża kohezja oznacza silną interakcję wewnątrz i relatywnie słabszą interakcję z zewnętrzem. Komponenty powinna cechować duża kohezja, co oznacza, że komponent stanowi dobrą, intuicyjną abstrakcję "czegoś", czyli posiada precyzyjnie określoną semantykę, jest dobrze wyizolowany z kontekstu (maksymalnie od niego niezależny) oraz posiada dobrze zdefiniowany interface.

Skojarzenie (coupling) określa stopień powiązania między komponentami, np. dla klas: jak często obiekty jednej klasy występują razem z obiektami innych klas, jak często obiekty jednej klasy wysyłają komunikaty do obiektów innej klasy, itp. Możliwe są skojarzenia silne, słabe czy w ogóle brak skojarzenia. Duża ilość silnych skojarzeń miedzy elementami składowymi (high coupling) jest tym, czego powinno się unikać.

Analiza stopnia kohezji i wzajemnych skojarzeń stanowi podstawę do konstruowania architektury systemu, czyli wyróżniania elementów składowych systemu, określania ich wzajemnych interakcji oraz sposobów przesyłania między nimi danych.

Zadania inżynierii oprogramowania

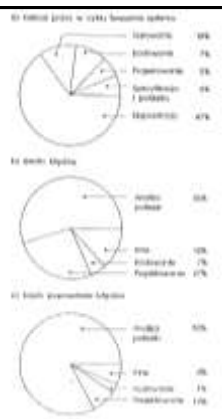
Zadania stojące przed inżynierią oprogramowania w walce z narastającą złożonością oprogramowania:

- Redukcja złożoności oprogramowania;
- Propagowanie wykorzystywania technik i narzędzi ułatwiających pracę nad złożonymi systemami;
- Upowszechnianie metod wspomagających analizę nieznanych problemów oraz ułatwiających wykorzystanie wcześniejszych doświadczeń;
- Usystematyzowanie procesu wytwarzania oprogramowania, tak aby ułatwić jego planowanie i monitorowanie;
- Wytworzenie wśród producentów i nabywców przekonania, że budowa dużego systemu wysokiej jakości jest zadaniem wymagającym profesjonalnego podejścia.

Dlaczego czasem sie nie udaje?

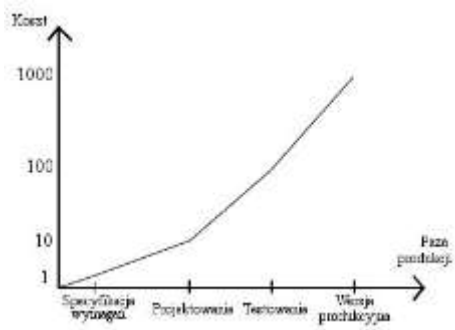
- Brak wystarczającego zaangażowania kierownictwa firmy w prace nad projektem.
- Niewystarczające zasoby ludzkie przydzielone do prac nad projektem
- Opór użytkowników przed zmianą
- Zły wybór systemu - niedopasowanie oprogramowania do procesów biznesowych w przedsiębiorstwie
- Nierealistyczne oczekiwania dotyczące zwrotu z inwestycji
- Niewystarczające przeszkolenie użytkowników
- **Złe zarządzanie pracami projektowymi i niewłaściwy plan projektu**
- **Kłopoty z komunikacją**
- **Złe oszacowanie kosztów wdrożenia i przesadne próby oszczędności**
- **Problemy techniczne i organizacyjne**

Czynnik z licencją na zabijanie (projektów) czyli James **Błąd** w akcji …

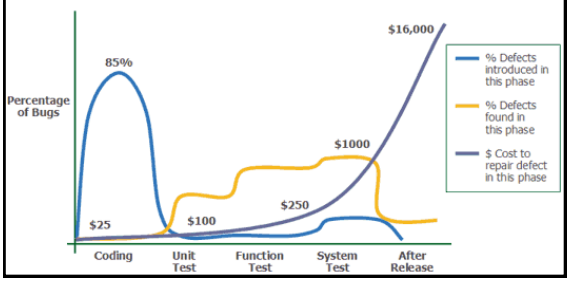


Krytycznym elementem każdego projektu są błędy, które się w nim pojawiają

Koszt naprawy błędu w zależności od fazy produkcji, w jakiej błąd będzie wykryty



Większość błędów powstaje na etapie analizy wstępnej, natomiast wykrywane są one późno, co powoduje koszty



Źródłem błędów pojawiających się na etapie analizy jest najczęściej nieskuteczne porozumienie się twórców systemu (przeprowadzających analizę) oraz użytkowników, którzy posiadają całą niezbędną wiedzę.

Problem ten będzie przedmiotem szczegółowszej analizy w kolejnym wykładzie

Wprowadzenie do systemów informacyjnych

Analiza systemów informacyjnych

© UEK w Krakowie Ryszard Tadeusiewicz