EVOLUCIJA SOFTVERA

- Razvoj softvera ne prestaje kada se sistem isporuči, već se nastavlja kroz životni vek sistema.
- Promene nekog aspekta poslovanja generišu nove zahteve za postojeći softver.
- Evolucija softvera je bitna jer su organizacije uložile velike količine novca i u potpunosti zavise od tih sistema.
- Velike kompanije troše više na održavanje postojećih sistema nego na razvoj novih.
- Prema nekim anketama, smatra se da 85-90% troškova su upravo troškovi evolucije.
- Softversko inženjerstvo kao spiralni proces koji se sastoji od:

  Zahteva, dizajna, implementacije i testiranja.
- Ovakav model implicira da je jedna organizacija odgovorna i za razvoj i za održavanje. Za softver koji se pravi po porudžbini (custom), obično jedna firma napravi softver, a onda drugi tim (koji je obično tim firme koja je poručila softver) održava.

- Alternativni pogled na životni ciklus evolucije softvera podrazumeva četiri faze:
  - Inicijalni razvoj
  - Evolucija
  - Servisiranje
  - Phaseout
- Nakon što se razvije, softver prelazi u fazu evolucije gde su dozvoljene sve velike promene arhitekture i dodavanje novih funkcionalnosti po cenu degradiranja strukture.
- U jednom trenutku, shvata se da više nije isplativo menjati softver.
- Prelazi se u fazu servisiranja gde su dozvoljene samo manje značajne promene. Počinje se sa planovima zamene softvera.
- Na kraju, dolazi se do faze Phaseout u kojoj se softver samo koristi, bez bilo kakvih modifikacija.

1. EVOLUCIONI PROCES

- Modifikacije sistema započinju sa predlozima.
- Predlozi mogu biti neki postojeći neimplementirani uslovi, zahtevi za novim uslovima (funkcionalnostima), izveštaj o bagovima i neke nove ideje za pospešavanje softvera.
- Proces identifikacije potrebnih promena i proces evolucije sistema je cikličan i nastavlja se kroz životni vek sistema.
- Evolucioni proces se može podeliti na:
  - Analiza promena (Cena modifikovanja i uticaj promena na sistem)
  - Planiranje izdanja (debagovanje, adaptacija, nove funkcionalnosti)
  - Implementacija
  - Izdavanje modifikovanog sistema klijentu
- Postupak se ponavlja ciklično uvođenjem novih predloga.

- Moguće je postojanje koraka razumevanja programa.
- Ovaj korak je potreban ukoliko nije isti tim zadužen i za implementaciju i za održavanje sistema.
- Potrebno je razumeti strukturu programa, način na koji omogućava funkcionalnost i kako bi predložene promene uticale na program.
- Koristi se kako bi se izbegli novi problemi pri izmeni sistema.

- Ponekad, potrebno je promeniti neki deo sistema u što kraćem roku. Razlozi su:
  - Ako je nastala ozbiljna greška koja mora da se ispravi ,ne bi li se nastavio normalan rad sistema.
  - Ako promene radnog okruženja sistema imaju neočekivane efekte koji onemogućuju normalan rad.
  - Ako postoje neočekivane promene u poslovanju koje koristi sistem, kao što su pojava nove konkurencije ili pojava nove legislacije koja utiče na sistem.

- U tom slučaju, potreba za brzom modifikacijom sistema onemogućava korišćenje formalnog procesa analize promena.
- Problem koji se javlja usled ovakvog pristupa je nekonzistentnost između uslova, dizajna i samog koda.
- Zbog ovoga, javlja se potreba za dodatnim ispravkama softvera, čiji je prioritet veći od dokumentovanja promena uslova i dizajna.
- Zbog toga što se koristi brzo rešenje koje radi a ne ono koje održava kvalitet strukture, ovaj proces doprinosi starenju sistema, tako da je buduće promene sve teže i teže implementirati.
- Agilne metode su izuzetno korisne pri tranziciji sa procesa razvijanja na proces evolucije jer koriste inkrementalni pristup i automatsko regresiono testiranje pri promenama sistema.
- Problematične situacije nastaju ukoliko tim koji razvija i tim koji se bavi održavanjem ne koriste isti pristup.
  (agilne metodologije i planski zasnovan pristup)

2. DINAMIKA EVOLUCIJE PROGRAMA

- Dinamika evolucije programa je studija promena sistema čiji je cilj bolje razumevanje karakteristika evolucije softvera.
- Lehman i Belady su sproveli studiju nakon koje su došli do zakona “Lehman’s laws”.
  - Održavanje sistema je neizbežan proces.
  - Kada se sistem promeni, struktura degradira.
  - Veliki sistemi imaju svoju dinamiku koja se uspostavlja rano tokom procesa razvoja. Ovo ograničava broj mogućih izmena sistema.
  - Promena resursa i članova razvojnog tima nema značajan efekat na dugoročnu evoluciju sistema.
  - Dodavanje nove funkcionalnosti proizvodi nove greške u sistemu (zato je preporučljivo ne preterivati sa novim funkcionalnostima u jednom izdanju programa jer iza toga sledi dodatan posao ispravljanja grešaka).

3. ODRŽAVANJE SOFTVERA

- Održavanje softvera je opšti proces menjanja sistema nakon što je isporučen.
- Postoje tri različita tipa održavanja softvera:
  - Ispravka grešaka
    - Greške u kodu se relativno jeftino ispravljaju, a greške u dizajnu skuplje.
  - Adaptacija okruženja
    - Ovaj tip održavanja je potreban kada se neki aspekt okruženja sistema promeni, kao što su hardver ili operativni sistem.
  - Dodavanje funkcionalnosti
    - Kada se uslovi koje treba da zadovoljava sistem promene usled organizacionih ili poslovnih prilika, potrebno je proširiti mogućnosti softvera.

- Efektivno je ulagati u dobar dizajn i implementaciju sistema kako bi se smanjili troškovi u budućnosti.
- Ako je dizajn sistema kvalitetan, lakše ga je razumeti i promeniti a samim tim i troškovi evolucije su manji.
- Tehnike kao što su precizna specifikacija, korišćenje objektno orijentisane paradigme i upravljanje konfiguracijom doprinose smanjenju troškova.
- Što je sistem ili komponenta sistema kompleksnija to ju je skuplje održavati.
- Primećeno je da je održavanje najviše fokusirano na mali broj kompleksnih komponenti.
- Iz tog razloga potrebno je zameniti kompleksne komponente sistema sa jednostavnijim alternativama.

- Nad sistemima koji su zastareli i koje je teško razumeti kako rade tzv. (legacy sistemi) ili kod sistema kod kojih je struktura degradirana, primenjuje se tehnika reinženjeringa.
- Tehnike reinženjeringa mogu da uključuju:
  - Redokumentovanje sistema
  - Refaktorisanje arhitekture sistema
  - Prevođenje programa u neki moderni programski jezik
  - Modifikovanje i ažuriranje strukture i vrednosti podataka sistema
- Funkcionalnost sistema se ne menja.
- Koraci reinženjeringa:
  - Prevod izvornog koda
  - Obrnuti inženjering (analiza programa, pomaže pri dokumentovanju)
  - Unapređenje strukture programa
  - Modularizacija programa (grupisanje delova i uklanjanje redundantnosti)
  - Reinženjering podataka (redefinisanje šema, uklanjanje grešaka i duplikata)
- Nije primenljiv i ne mogu se automatizovati koraci u svim situacijama. Nekad je praktičnije i isplativije iznova razvijati sistem.

- Refaktorisanje je proces unapređenja programa sa ciljem usporenja degradacije kroz promene.
- Unapređuje se struktura i čitljivost a smanjuje kompleksnost.
- Može se shvatiti kao preventivno održavanje, koje smanjuje količinu problema izazvanih budućim promenama.
- Refaktorisanje nije isto što i reinženjering.
- Refaktorisanje je kontinualni proces koji traje kroz ceo razvojni i evolutivni proces.
- Regresiono testiranje smanjuje rizik od uvođenja novih grešaka korišćenjem refaktorisanja.
- Česte situacije gde se koristi refaktorisanje:
  - Dupli kod
  - Dugacke metode
  - Switch naredbe
  - “Clumping” podataka (više istih podataka na različitim mestima)
  - Špekulativna uopštavanja

4. RUKOVOĐENJE LEGACY SISTEMA

- Mnogi legacy sistemi čine krucijalne delove poslovnog sistema. Javlja se potreba za adaptacijom ovih sistema.
- Pravi se realistična procena svih legacy sistema neke organizacije, na osnovu poslovne vrednosti i kvaliteta sistema.
- Strategije rukovođenja:
  - Izbaciti sistem iz upotrebe
  - Ostaviti sistem nepromenjenim i nastaviti sa održavanjem
  - Uraditi reinženjering sistema kako bi se pospešila održivost
  - Zameniti ceo ili delove sistema sa novim sistemom

- Procena poslovne vrednosti:
  - Iskorišćenost sistema
    - Retko korišćeni sistemi su kandidati za izbacivanje iz upotrebe, doduše treba voditi računa, postoje krucijalni sistemi koji se retko koriste (npr. upisivanje godine studenta).
  - Podržani poslovni procesi
    - Sistem koji koristi neefikasne poslovne procese takođe ima nisku poslovnu vrednost.
  - Pouzdanost sistema
    - Ako sistem nije pouzdan tj. ako problemi direktno utiču na korisnika onda sistem ima nisku poslovnu vrednost.
  - Izlazi sistema
    - Ako je rezultat rada nije bitan po poslovanje tj. može se lako generisati na neki drugi način, onda sistem ima nisku poslovnu vrednost.

- Procena kvaliteta sistema iz tehnološke perspektive (uglavnom se zasniva na pouzdanosti sistema, težini održavanja sistema i dokumentaciji sistema).
  - Broj zahteva za promenom sistema
    - Što je ovaj broj veći, to je struktura sistema lošija pa i kvalitet sistema.
  - Broj korisničkih interfejsa
    - Ovo je bitan faktor kod sistema baziranih na formama gde svaka forma moze da se shvati kao odvojeni korisnički interfejs. Što više interfejsa, to je veća verovatnoća da postoje nekonzistentnosti i redundantnosti kod tih interfejsa.
  - Količina podataka koja se koristi od strane sistema
    - Veća količina podataka povlači veću verovatnoću za nekonzistentnošću među podacima, što smanjuje kvalitet sistema.
- Objektivna procena bi trebalo da se koristi pri odlučivanju strategije rukovođenja legacy sistema, mada se u većini slučajeva odluke zasnivaju na organizacionim ili političkim prilikama.