**Software engineering**

**Softvérové inžinierstvo** sa zaoberá teóriami, metódami a nástrojmi pre profesionálny softvérový vývoj.

Softvérové inžinierstvo sa zaoberá cenovo - efektívnemu vývoju softwaru.

**Čo je software ?**

Počítačové programy a súvisiaca dokumentácia, ako požiadavky, dizajn modelov a

používateľských príručiek. Softwarové produkty môžu byť vytvorené pre konkrétneho

zákazníka alebo môžu byť vyvinuté pre trh všeobecne.

**Softwarové produkty môžu byť**

• **Generické** - vyvinutý byť predané rôznym zákazníkom, napr. PC softvér, ako je Excel alebo Word.

• **na zákazku (custom)** - vyvinutý pre jediného zákazníka podľa jeho špecifikácii.

**What is a software process?**

Súbor činností, ktorých cieľom je rozvoj alebo vývoj softvéru.

Všeobecné činnosti vo všetkých softvérových procesov sú:

· **Špecifikácia** - čo by mal systém robiť a jeho vývojove obmedzenia

· **Vývoj** - výroba softvérového systému

· **Validácia** - overenie, že softvér je to, čo zákazník chce

· **Evolúcia** - zmena programu v závislosti na meniacich sa požiadavkách.

**What is software engineering?**

Softvérové inžinierstvo je inžinierska disciplína, ktorá sa zaoberá všetkými aspektmi

softvérovej produkcie. Softvéroví inžinieri by mala prijať systematický a organizovaný

prístup k práci a využiť vhodné nástroje a postupy v závislosti od problému ktorý treba

vyriešiť, vývojove obmedzenia a dostupne zdroje.

**Dependable systems**

Spoľahlivosť odráža stupeň dôvery používateľa v systém.

**Socio-technical critical systems**

**Hardware failure** – hardware zlyha z dovodu zleho navrhu alebo vyrobnej chyby, skoncila

zivotonost

**Software failure** – software zlyha kvoli chybe v jeho specifikacii, navrhu alebo implementacii

**Operational failure** ( operacna chyba ) – ludky faktor(chyba), stava sa najcastejsie

**Vlastnosti spoľahlivosti:**

**· Dostupnosť**

**· Spoľahlivosť**

**· Bezpečnosť**

**· Zabezpečenie**

**· Životaschpnosť**

**Software Processes**

**Softvérové procesy**

Štruktúrovaná množina **aktivít** potrebná na vývoj softvérových systémov

**o Špecifikácia**

**o Dizajn**

**o Overenie**

**o Evolúcia**

Model softvérového procesu je abstraktná reprezentácia procesu .

Predstavuje popis procesu z nejakých špecifických perspektív.

**Software process models**

· **Vodopádový**

· **Inkrementálny/iteratívny**

**· Na exitujúcich komponentoch**

**Aktivity designu**

· Design architektúry

· Design databázy

· Design interfacu

· Design komponentov

**Testovacie fázy:**

· Testovanie komponentov

· Testovanie systému

· Zákaznícke testovanie

**Requirements Engineering**

**Typy požiadaviek :**

**Používateľské požiadavky**

- Sú písane v prirodzenom jazyku + diagramy služieb ktoré poskytuje

systém a ich operačné obmedzenia.

**Systémové požiadavky**

- Štruktúrovaný dokument popisujúci detailný opis systémových funkcií,

služieb a operačných obmedzení. Definuje čo má byť implementované,

takže môže byť časť kontraktu medzi klientom a dodávateľom.

**Funkcionálne požiadavky**

- Služby, ktoré by mal systém poskytovať, ako má systém reagovať na

jednotlivé vstupy a ako sa má systém správať v jednotlivých situáciách.

- Popisujú funkcionalitu alebo systémové služby.

- Závisia na type softwaru, predpokladajú používateľov a typ systému

kde je softvér používaný.

- Funkcionálne používateľské požiadavky môžu byť všeobecné vyjadrenia

čo by mal systém robiť ale funkcionálne systémové požiadavky by mali

popisovať systémové služby detailne.

**Nefunkcionálne požiadavky**

- Obmedzenie služieb alebo funkcií ponuknutých systémom ako časové

obmedzenia, obmedzenia vývojového procesu, štandardy, atď.

**Doménové požiadavky**

- Požiadavky ktoré prídu z domény systémových aplikácií a prejavia sa

v charakteristikách domény.

**Požiadavky : kompletnosť a konzistencia**

V princípe by požiadavky mali byť kompletné a konzistentné.

**Kompletné**

- Mali by obsahovať popis všetkých potrebných vlastností

**Konzistentné**

- Nemali by obsahovať žiadny konflikt alebo protirečenie v popise

systémových vlastností.

V praxi, je nemožné písať kompletné a konzistentné požiadavky.

**Nefunkcionále pož. Klasifikácia**

**Produktové požiadavky**

- Požiadavky ktoré, špecifikujú, že dodaný produkt sa musí určitým

spôsobom správať napr. spúštacia schopnosť, spoľahlivosť, atď.

**Organizačné požiadavky**

- Požiadavky ktoré sú výsledkom organizačných pravidiel a procedúr

ako napr. process standards used, implementation requirements,

etc.

**Externé požiadavky**

- Požiadavky ktoré vyplývajú z vonkajších faktorov pre systém a jeho

vývojový proces ako napr. legislatívne požiadavky, a pod.

Príklady nefunkcionálnych požiadaviek

**Produktové požiadavky**

o 8.1 používateľské rozhranie pre LIBSYS by malo byť

implementované ako jednoduché HTML bez framov alebo Java

appletov.

**Organizačné požiadavky**

o 9.3.2 Vývojový proces systému a doručiteľné dokumenty by sa mali

prispôsobiť procesom a doručiteľnosti definovaných

v XYZ=SP-STAN-95

**Externé požiadavky**

o 7.6.5 Systém by nemal prezradiť žiadne osobné informácie

o zákazníkoch okrem ich mena a ref. čísla pre operátorov systému

**Requirements management planning**

**System Modeling**

**Pohľady**

· Externý

· Interakčný

· Štrukturálny

· Behaviorálny – správanie a ako odpovede na udalosti

**Contex models (kontextove modely)**

Kontextové modely sú použité pre ilustráciu prevádzkových podmienok

systému, ktoré ukazujú, čo leží mimo hraníc systému.

Sociálne a organizačné otázky môžu mať vplyv na rozhodnutie o tom, kam

umiestniť hranice systému.

**Architektonické modely** ukazujú systému a jeho vzťahu s ostatnými systémami.

**Process models**

Procesné modely ukazujú celkový proces a procesy, ktoré sú podporované

systémom.

Dátový tok modelov môže byť použitý na zobrazenie procesov a toku informácií

z jedného procesu na iný.

Tieto typy diagramu sú niekedy označované ako **workflow diagramy**

**Activity diagram – workflow a dataflow**

**Interaction models**

*Interakcia môže byť:*

· Používateľská

· Systém so systémom

· Medzi komponentami

*Využíva Use-case, sekvenčný a komunikačný diagram*

**Behavioural models**

Behaviorálne modely sa používajú na opis celkové správania systému.

Dva typy správanie modelu sú:

o Spracovanie dát, modely, ktoré ukazujú, ako sú dáta spracované, pretože

sa pohybuje v systéme;

o Stavové modely, ktoré ukazujú systémy reakcie na udalosti.

Tieto modely ukazujú rôzne pohľady takže oba **musia opísať správanie systému.**

**model-driven engineering**

+ vysoka abstrakcia, generovanie kodu, prisposobivost pre rozne platformy

- abstraktne modely nemusia byt spravne pre implementaciu

**Architectural Design**

Architektonický návrh - ako by mal byť systém organizovaný

- Navrhuje celkovu strukturu system

Výstupon návrhu je **architektonický model –** ako je systém organizovaný ako set komunikačných komponentov

**Architektúra**

· *V malom* – zameraná na individuálne program, ako je program rozložený na komponenty

· *Vo veľkom* - zameraná komplexné systémy obsahujúce iné systémy, program a komponenty, ako je program rozložený na komponenty

**Výhody explicitnej architektúry:**

· komunikácia so zúčastnenými stranami

· systémová analýza

· široká možnosť znovupoužitia

**Box a line diagramy**: veľmi abstraktné, užitočné pre komunikáciu

a plánovanie projektu

Jazyky pre opis boli vyvynuté ale veľmi sa nepoužívajú, skor UML.

**Znovupoužitie architektúry(reuse):** Systémy v rámci rovnakej domény

majú často podobné architektúry, ktoré odrážajú doménové koncepty,

linky aplikácie produktov sú postavené okolo jadra architektúry s

variantmi, ktoré spĺňajú konkrétne požiadavky zákazníkov.

**Architektúra a systémové charakteristiky**: výkon, zabezpečenie,

bezpečnosť, dostupnosť, udržovateľnosť

**Architektonické pohľady**

· logický - abstrakcie

· procesný - ako počas behu system interaguje

· vývojový – ako je system zložený(pre vývojárov)

· fyzický – HW a SW komponenty

**Architektonické vzory -** popis dobrých dizajnových postupov, ktoré boli vyskúšané a testované v rôznych prostrediach.

**Systém organizácie**: odráža základnú stratégiu, ktorá sa používa pre

štruktúru systému

3 organizačné štýly sú široko používané:

**1.štýl zdieľaného uložiska dát**

**2.štýl zdieľania služieb a serverov**

**3.štýl zvrstvený alebo štýl abstraktných strojov**

**Vrstvová architektúra**

· pre interakciu subsystémov

· system ako vrstvy poskytujúce služby

**Úložisko /Repository architecture:** Podsystémy musia vymieňať dáta. To možno vykonať

dvoma spôsobmi:

a)Zdieľané údaje sú držané v centrálnej databáze a môžu byť prístupné

všetkým podsystémom

b)Každý podsystém udržiava svoje vlastné databázy a dáta explicitne

odovzdáva ostatným podsystémom.

**-výhody**: efektívny spôsob ako zdielať veľké množstvo dát, zdieľaný

model je vydávaný ako úložisko schémy

**-nevýhody**: dátová evolúcia je nákladná a zložitá, nevyhnutný kompromis

medzi podsystémami, ťažko distribuovať efektívne

**Client-Server Model**: distribuovaný systém model, ktorý ukazuje, ako sú

dáta a ich spracovanie distribuované v celej rade komponentov

**-výhody**: jednoduchá distribúcia dát, efektívnosť pri využívaní zariadení

v sieti, jednoduchý upgrade serverov alebo pridanie nového

**-nevýhody**: redundantné riadenie v každom serveri, žiadny centrálny

register mien a služieb, neefektívna výmena dát medzi podsystémami

**Modulárne rozdelenie štýlov**- štýly rozdeľujúce podsystémy do modulov

**Podsystémy a modely**- **podsystém**= systém vo svojom vlastnom systéme,

ktorého prevádzka je nezávislá na službách poskytovaných inými podsystémami,

**modul**= je súčasťou systému, ktorý poskytuje služby iným komponentom, ale

aby normálne nebol považovaný za samostatný systém.

**Modulárne rozdelenie**: štrukturálna úroveň, kde sú podsystémy rozdelené do

modulov

Dva modulárne modely rozdelenia zahŕňajú:

a) Objektový model, kde sa systém rozdelí na interagujúce objekty

b) Potrubie alebo model dátového toku, kde sa systém rozdelí do funkčných modulov, ktoré transformujú vstupy na výstupy.

**Objektové modely**: usporiadavajú systém do súboru voľne previazaných

objektov s dobre definovanými rozhraniami

-výhody: objekty sú voľne viazané, čo umožňuje, že môžu byť upravené bez

ovplyvnenia ďalších objektov, objekty môžu odrážať entity reálneho sveta,OO

jazyky sú široko používané

**Funkciovo orientované prúdové spracovanie(Function-oriented**

**pipelining):** Funkčné transformácie spracovávajú svoje vstupy za účelom

produkcie výstupov, môže byť označené ako pipe a filter mode. Varianty tohto

prístupu sú veľmi časté. Keď sú transformácie sekvenčné, to je sekvenčný

model, ktorý sa vo veľkej miere používa v systémoch spracovania dát.

-výhody: intuitívna organizácia pre komunikáciu so zainteresovanými stranami,

jednoduché pridanie nových transformácii, jednoduché realizovať buď ako

súbežný alebo sekvenčný systém

**Control style**: týkajú sa toku riadenia medzi subsystémami. Odlišný od modelu

systému rozdelenia. **Centralizované riadenie+ Event-based riadenie**.

**Centralizované riadenie**: Kontrolný podsystém preberá zodpovednosť za

riadenie vykonávania ostatnými subsystémami. **Call-return**(zhora nadol) model +

**Manager model**(súbežnosť systémov)

**Udalosťami riadené systémy**: Hnaný externe generovanými udalosťami, kt.

načasovanie presahuje kontrolu podsystémov kt. riadia udalosti. Dv principal

event-driven modely:

**a)Broadcast models:** Účinný pri integrácii subsystémov na rôznych počítačoch

v sieti. Subsystémy zaregistrujte záujem o konkrétnu udalosť. Ak sa tieto príznaky

vyskytnú, je kontrola prevedená na podsystému, ktorý dokáže spracovať udalosť.

b) **Interrupt-driven models.** Používa sa v real-time systémoch, kde rýchla

odozva na udalosť je nevyhnutná Sú známe typy prerušenia definované pre každý

typ. Každý typ je spojený s lokáciou pamäte a prepnutia hardwaru spôsobí prevod

do jeho jednotky.

**Agilne metody**

· Procesy špecifikácie, návrhu a implementácie sú súbežné. Neexistuje žiadna podrobná špecifikácia a projektová dokumentácia je minimálna.

· Systém je vyvíjaný v sérii krokov. Koncoví užívatelia zhodnotia každý prírastok a podávajú návrhy

· Časté nové verzie

· Sústredenie sa na kód

**Cieľom je redukovať náklady(vynechanim dokumentacie) a byŤ schoplný rýchlo ragovať a zmeny požiadavok, bez nadmerného prerábania.**

(pre porovnanie)Plan-driven development

· Založený na oddelenych stupnoch vyvojovych statdii

· Vystup je produkovany kazdom stupni

· Vodopadovy model alebo inkrementalny vyvoj

**Extrémne programovanie**

**Extrémne programovanie (XP)** má "extrémny" prístup pre iteratívny vývoj.

o Nové verzie môžu byť postavené niekoľkokrát za deň;

o Prírastky sú dodávané zákazníkom každé 2 týždne;

o Všetky testy musia prebehnúť úspešne

**XP a agilné princípy**

· Inkrementálny vývoj je podporovaný prostredníctvom malých, častých verzií systému.

· Zapojenie zákazníkov znamená angažmán na plný úväzok s tímom.

· Zmena podporovaná prostredníctvom pravidelných verzií systému.

· Zachovanie jednoduchosti neustálym refactoringom kódu.

**zapojenie zákazníka**

· Zapojenie zákazníka je kľúčovou súčasťou XP, kde zákazník je súčasťou vývojového tímu.

· Úloha zákazníka je:

o Ak chcete pomôcť rozvíjať príbehy, ktoré definujú požiadavky

o Ak chcete pomôcť priority funkcie, ktoré majú byť realizované v každej verzii

o Ak chcete pomôcť rozvíjať akceptačné testy, ktoré posudzujú, či systém spĺňa jeho

požiadavky.

**požiadavky scenára**

· V XP sú užívateľské požiadavky vyjadrené ako scenáre alebo užívateľské príbehy.

· Tie sú napísané na kartách a vývojový tím to používa na plnenie úloh. Tieto úlohy sú základom plánu a odhadov nákladov.

· Zákazník si vyberá príbehy pre zaradenie do budúceho vydania na základe svojich priorít a odhadov plánu.

**XP a zmena**

· Konvenčná múdrosť v softvérovom inžinierstve je navrhnúť zmeny. Stojí za to stráviť čas a úsilie na predvídanie zmien, pretože to znižuje náklady neskôr v životnom cykle.

· XP však tvrdí, že to nie je užitočné lebo zmeny nemožno spoľahlivo predpokladať.

· Skôr sa navrhuje neustále zlepšovanie kódu (refactoring), aby sa menilo ľahšie, keď má byť zmena vykonaná.

**Refactoring**

· Refactoring je proces zlepšovania kódu, kde je kód reorganizovaný a prepísaný, aby bol efektívnejší, zrozumiteľnejší, atď

· Refactoring je nutný, pretože časté zmeny znamenajú, že kód je vyvinutý postupne, a preto má tendenciu byť chaotickým.

· Refactoring by nemal meniť funkčnosť systému.

· Automatizované testovanie zjednodušuje refactoring, lebo môžete zistiť, či zmenený kód stále beží,ak testy úspešne.

**Testovanie v systéme Windows XP**

· Test-first vývoj.

· Inkrementálny vývoj testu od scenárov.

· Zapojenie užívateľov v oblasti vývoja a validácie testu.

· Automatizované testovacie sa používa na spustenie všetkých čiastkových skúšok zakaždým, keď je nová verzia postavená.

**Test-first vývoj**

· Písanie testov pred kódom upresňuje požiadavky, ktoré majú byť realizované.

· Testy sú písané ako programy skôr ako údaje tak, aby mohli byť vykonané automaticky. Test zahŕňa kontrolu, že kód bol správne vykonaný.

· Všetky predchádzajúce a nové testy sú automaticky spustené, keď je pridaná nová funkcia. Preto sa skontrolujte, či do novej funkcie nebola zavedená chyba.

**Pár programovanie (Pair programming)**

· V XP programátori pracujú vo dvojiciach, sedia spolu na vývoji kódu.

· To pomáha rozvíjať spoločné vlastníctvo kódu a šíriť vedomosti naprieč tímom.

· Slúži ako proces neformálne preskúmania, že na každý riadok kódu sa pozrela viac ako 1 osoba.

· Povzbudzuje refactoring lebo celý tím môže ťažiť zo zlepšovania kódu.

· Merania ukazujú, že produktivita vývoja s párom programovania je podobná ako u dvoch ľudí, ktorí pracujú nezávisle na sebe.

**Kľúčové praktiky**

· **User stories**

· **Refactoring**

· **Test-first**

· **Pair programming**

**SCRUM**

· Agilna metoda, sustredi sa na viac na riadenie iterativneho vyvoja ako na konkretne agilne praktiky

· Obsahuje serie sprintov

· Na konci projektu sa robi dokumentacia, manulaly, posudzuju sa ponaucenia z projektu

· Sprint – fixna dlzka 2-4 tyzdne

o zacina z Backlogu – suhrn prace ktora sa ma robit na projekte

o kazdy clen timu si vezme nieco z backlogu co bude v danom sprinete robit

**Scrum master** – riadi denne schodzky, sleduje backlog, meria progres oproti backlogu, komunikuje so zakaznikom a manazmentom mimo timu

**Výhody scrumu**

· Rozdelenie uloh na casti

· Nestabilne poziadavky nespomaluju proces

· Kazdy clen vidi vsetko co sa deje, dobra komunikacia

· Zakaznik vidi prirastky a moze dat feedback

Software Testing

Má ukázať že program robí to čo má robiť a odhaľovať chyby predtým ako sa začne používať.

Odhaľuje prítomnosť chýb.

Základná verifikácia validácia.

**Validačné testy** – testujú očakávané správanie

**Defektné/vadné testy** – majú odhaliť defekty, nemusia odrážať to ako sa systém normálne používa

**Verifikácia** – softvér by mal zobpovedať špecifikácii

**Validácia** – softvér má robiť to čo používateľ požaduje

**Softvérová inšpekia**

– ľudia skúmajú zdroj reprezentácie, cieľom je zistiť anomálie a vady

– nevyžaduje spustenie systému(može byť aj pred impl)

– može byť aplikovaná na hocijakú reprezentáciu systému(požiadavky, design, konf a test dáta)

– efektívna technika pre objavovanie chýb

Inšpekcia a testovanie

– komplementárne(doplnkové) niesú opozitá

– používajú sa počas verifikačného a validačného procesu

– inš nemôže kontrolovať nefunkcionálne požiadavky

**Štádiá testovania**

– Vývojové testovanie – počas vývoja(bugy a vady)

o *Unit testy* (defektné testy )– testuju funkcionalitu objektu a metod

o *Komponent testy* - test komponentov

o  *Systémové testy* - test celeho systemu ako spojenie komponentov do jedneho – testovanie interakcii

– Release testovanie – testeri testujú aplikáciu predtým ako ju dostanú skutočný používatelia(Requirements-based testing, Performance testing, Stress testing)

– Používateľské testovanie – používatelia testujú systém v ich prostredí

**Testovanie triedy objektu**

· Kompletný test pokrytia triedy zahŕňa:

o Testovanie všetkých operácii spojených s objektom;

o Nastavovanie a vyskúšanie všetkých atribútov objektu;

o Testovanie objektu vo všetkých jeho možných stavoch.

· Dedičnosť robí návrh triedy objektu zložitejším, pretože informácie nie sú lokalizované.

**Testovacie stratégie**

– **Partition testing**(oddielové/oblasťové) – vstupy s rovnakými vlastnosťami by mali byt spracované rovnakým sposobom

– **Guideline-based testing** - kde pomocou testovacích pokynov pre výberáme testovacie prípady.

**Testovanie rozhraní**

· Cieľom je nájsť chyby spôsobené chybami v rozhraniach alebo nesprávnymi predpokladmi o rozhraniach.

· Zvlášť dôležité pre objektovo orientovaný vývoj, keďže objekty sú definované ich rozhraniami.

**Chyby rozhraní**

· Nesprávne použitie rozhrania

o Volajúci komponent volá ďalší komponent a spraví chybu v použití svojho komponentu (napr. parametre v nesprávnom poradí).

· Nerozumenie rozhraniu

o Volajúci komponent vloží predpoklady o chovaní volaného komponentu, ktoré sú

nesprávne.

· Chyby časovania

o Volaný a volajúci komponent pracujú rôzne rýchlo a pristupované je k starým

informáciám.

**Pokyny pre testovanie rozhrania**

· Navrhneme testy tak, aby parametre volanej procedúry boli na extrémnych koncoch ich rozsahov.

· Vždy testujeme pointre parametrov s NULL pointrami.

· Navrhneme testy, ktoré spôsobia pád komponentu.

· Použijeme stress testing v systémoch predávania správ.

· V systémoch so zdieľanou pamäťou, obmeňujeme poradie, v ktorom sú komponenty

aktivované.

**White-box (clear-box, glass-box) testing**

**Štrukturálne testovanie** - cieľom je vykonávať všetky príkazy programu.

**Testovanie ciest** - cieľom je prejsť všetky programové cesty (cykly, if-výkazy, ...) aspoň raz

**Black-box methods: partitioning, guideline-based testing**

**Developmenttesting**

**Unittesting (UIT) – functions, objects**

**Componenttesting**

**Interface testing**

**Systemtesting**

· testovanie integrácie = integračné testovanie systému (SIT)

**User testing**

· Testovanie užívateľmi (UAT)

**Reuse-based software engineering**

· ***opätovné použitie systému***

· ***opätovné použitie aplikácie***

· ***opakované použitie komponentov***

· ***opakované použitie objektov a funkcií***

**Framework**

- integrovaná sada softvérových častí (napríklad tried, objektov a komponentov), ​​ktoré spolupracujú tak, aby sa dali opakovane použiť

- sú generické -> môžu byť rozšíriteľné – pridaním tried a metód

**Framework classes**

· **System infrastructure frameworks**- Podporujú rozvoj systému

· **Middleware integration frameworks**- Štandardy a triedy, ktoré podporujú komponentnú komunikáciu a výmenu informácií.

· **Enterprise application frameworks**- Podporujú rozvoj špecifických typov aplikácií, ako sú telekomunikácie alebo finančné systémy.

**Softvérové ​​produktové línie a aplikačné rodiny** sú aplikácie s generickými funkciami, ktoré môžu byť upravené a nakonfigurované pre použitie v špecifickom kontexte.

**Zloženie aplikácie**:

· **Core komponenty** – základná podpora infraštruktúry(nemodifikujú sa)

· **Konfigurovateľné komponenty**

· **Špecializované komp pre špecifickú doménu**

**Produktové línie**

· Nemusia byť objektovo orientované

· Produktové línie často kontrolujú aplikácie pre zariadenia

· Softvérové ​​produktové rady sa skladajú z rodiny aplikácií, zvyčajne tej istej organizácie.

· Špecializácia

o **platformová** – rôzne verzie softvéru vyvýjané pre rôzne platformy

o **prostredí** - rôzne verzie softvéru pre rôzne OS

o **funkcionálna** - rôzne verzie softvéru vyvýjané pre roznych zákazníkov s rôznymi požiadavkami

o **procesná** - rôzne verzie softvéru vyvýjané pre podporu rôznych biznis procesov

**Konfigurácia produktovej línie**

· **Design time configuration**

o Organizácia, ktorá vyvíja softvér upraví spoločnú radu produktov prostredníctvom rozvoja, výberu alebo prispôsobením komponentov

· **Deployment time configuration**

o Všeobecný systém je určený pre konfiguráciu zo strany zákazníka či odborníkov pracujúcich so zákazníkom.

**Konfigurovateľné aplikačné systémy**

· generické aplikačné systémy, ktoré môžu byť navrhnuté tak, aby podporovali zvláštny druh obchodu, obchodnú činnosť alebo niekedy aj kompletné obchodné podniky.

**Systémy pre špecifické oblasti**, ako sú systémy pre podporu obchodných funkcii (napr. Pre správu dokumentov) poskytujú funkcie, ktoré budú pravdepodobne potrebné radom potenciálnych používateľov.

**Enterprise ResourcePlanning(ERP)**- je všeobecný systém, ktorý podporuje bežné obchodné procesy, ako je objednávanie a fakturácia, výroba, atď

**Application system integration problems**

· málo kontroly nad funkcionalitou a výkonom

· interoperabilita - Rôzne aplikačné systémy môžu robiť rôzne predpoklady

čo znamená, že integrácia je ťažká

· žiadna kontrola nad evolúciou systému

· podpora zo strany predajcov systému