Objektum Orientált Tervezés

Felhasznált szakirodalom

Sike Sándor-Varga László: Szoftvertechnológia és UML

Angster Erzsébet: Az objektumorientált tervezés és programozás alapjai

Wikipédia

1. Történet

   (a) SIMULA'67 [60] (ALGOL jellegű, öröklődés, overloading, statikus és dinamikus kötés, szemétgyűjtés, korutin)

   (b) Smalltalk [72;74;76;78;80; V](Simula'67+LISP, 1970-es évek, tiszta OO, GUI)

   (c) Objective-C (Smalltalk jellegű, Macintosh), C with classes, C++ [80] (SIMULA'67, bonyolult, nincs szemétgyűjtés, index határ ellenőrzés)

   (d) Eiffel [80](tiszta OO)

   (e) Pascal [86] (Object Pascal, Turbo PAscal, Delphi)

   (f) Java (Oak)[90](egyszerűsített C++, van sz.gy., i.h.e., platformfüggetlen, nagy osztály könyvtár, nem hatékony)

   Hibrid, ha csak támogatja az OO-t (objektum azonosság, osztályozás, bezártság, de nincs öröklődés, specializálás).

2. A szoftver minősége

   (a) Helyesség: a specifikációnak megfelel

   (b) Hibatűrés: abnormális esetben is normálisan működik

   (c) Karbantarthatóság, bővíthetőség

   (d) Újrafelhasználhatóság

   (e) Kompatibilitás (belső/külső)

   (f) Felhasználóbarátság

   (g) Hordozhatóság (hardver/szoftver)

   (h) Hatékonyság (idő/hely)

   (i) Ellenőrizhetőség

   (j) Szabványosság

3. A szoftver fejlesztés lépései

   (a) Feladatmegfogalmazása; Szemléleti szint

   (b) Elemzés vagy analízis: a feladat felbontása részekre; Koncepcionális szint, csak a "MIT" az érdekes, a "HOGYAN0" nem

   (c) Tervezés: a számítógépes megvalósítás körvonalazása (adatstruktúrák, nyelvek); Technikai szint

   (d) Programozás: az ami a neve; Implementációs szint

   (e) Gépi kód

4. Az Objektum Orientált Tervezés

   '70-től: Szoftverkrízis: a hagyományos módszer már alkalmatlan a jó szoftver fejlesztésére: paradigmaváltás (szemléletmód)

   (a) Strukturált szemlélet:

      i. A teljes feladat egy absztrakt utasítás
      ii. Időbeli sorrendiség alapján történik a részekrebontás
      iii. Felülről lefelé építkezik
      iv. A szorosabban összetartozó adatelemek a folyamattól függetlenül csoportosíthatóak.
      v. Az adatcsoportok kezelése a programban szétszórva találhatóak.
      vi. Legkisebb modulja az eljárás, melynek adatai elvesznek. globális változókat kell általában használni

   (b) Objektum orientált szemlélet:

      i. Objektumok üzenetei alkotják a programot
      ii. Nincs igazi időbeliség
      iii. Lentről felfelé építkezik
      iv. Az adatokhoz kapcsoljuk az őket kezelő programrészeket
      v. Legkisebb modulja az objektum

5. A valós világ modellezése

   (a) Absztrakció: leegyszerűsítés, csak a lényeges dolgokat vesszük figyelembe

   (b) Megkülönböztetés: eltérő tulajdonság kiemelése "azonos" dolgoknál

   (c) Osztályozás: Kategorizálás

   (d) Általánosítás: Több objektum leírásából kiemeljük a közös jellemzőket "mindegyik ugyanolyan lényegileg" (Osztály alkotás)

   (e) Specializálás: Egy objektum leírásához egyedi jellemzőket adunk "olyan mint, de...." (Öröklés)

   (f) Kapcsolatok felépítése, részekrebontás: "Ez valamihez tartozik"; "Ez valamiket tartalmaz"

      i. ismeretség: függetlenül is létezhetnek, legalább az egyik ismeri/használja a másikat
      ii. egész-rész: a tartalmazó megszünése esetén megszűnik a tartalmazott is. Kompozíciónak nevezzük, ha nem vehető ki a rész.

6. Objektum orientált program jellemzői

   (a) Objektum: Információt tárol és kérésre feladatot hajt végre. Az objektum felelős a feladatainak korrekt elvégzéséért. Adatok (attribútomok) és metódusok (operációk, műveletek, azaz függvények, eljárások) összessége.

   (b) Az objektum állapota az adatok pillanatnyi értékei

   (c) Objektumorientált program: Egymással kommunikáló objektumok összessége, melyben minden objektumnak megvan a jól meghatározott feladata. (A "mit" az érdekes, nem a "hogyan".) A program a vezérlő objektum megszólításával indul (általában rögtön várakozik a felhasználóra).

   (d) Információ elrejtése: Az objektumokat CSAK üzeneteken keresztül kérhetjük meg a feladatok elvégzésére, csak ún. interfészen keresztül lehet kommunikálni vele. Az üzenet egy kívülről is elérhető metódus.

      i. Kliens, aktor, ügyfél, kérő: a feladatot végeztető "aktív" objektum pl. vezérlők. Nincs "export" felülete.
      ii. Szerver, kiszolgáló, végrehajtó: a feladatot elvégző "passzív" objektum pl. printer, konténer. Nincs "import" felülete
      iii. Ágens, ügynök: kér és végrehajt.

   (e) Láthatóság

      i. +public: Bárki bárhonnan
      ii. -private: Osztályon kívül senki sem
      iii. ♯ protected: Csak bizonyos helyekről (csak öröklésen keresztül)

   (f) Az objektum inicializálása a kezdeti adatok megadása, kezdeti tervékenységek elvégzése. Általában ez konstruktor is egyben.

   (g) Konstruktor: az objektum létrehozója, általában inicializál is.

   (h) Destruktor: az objektum megszüntetője.

   (i) Az objektum belseje sérthetetlen.

   (j) Objektum típusok

      i. Határobjektum: Interfész objektum
      ii. Konténer objektum: Objektumok kollekciója, s a kollekciót karbantartó metódusok.
      iii. Riport objektum: bekér, feldolgoz, kiír
      iv. Kontrol objektum: Vezérel, számol

   (k) this, self paraméter: magát az objektumot jelenti

   (l) Egybezárás: Az adatok és metódusok összezárása

   (m) Kód újrafelhasználhatósága: Az osztályok.

   (n) Polimorfizmus/többalakúság: Ugyanarra a kérelemre különböző objektumok különbözően reagálhatnak. Az üzenet küldőjének nem kell tudni a szerver osztályát.

   (o) Az objektumokat osztályokba soroljuk (Osztályozás, valójában az osztályból jön létre az objektum.)

   (p) Az osztály (típus) egy objektum minta, mely alapján példányokat (objektumokat) hozhatunk létre. Egy objektum csak egy osztályhoz tartozhat, s ismeri is az osztályát.

   (q) Egy osztály örökölhet tulajdonságokat (változók) és viselkedésformákat (metódusok) egy másik osztálytól, ekkor csak az eltéréseket kell megadni.

   (r) Öröklődés: Egy már meglévő osztály továbbfejlesztése. Az utódosztály az ősosztály (szuper)specializálása. Lehet több ős is.

      i. Új adatok
      ii. Új metódusok
      iii. Régi metódusok átírása

   (s) Osztályhierarchia: Az öröklődések diagramja.

   (t) Késői kötés: csak később a futás alatt derül ki a hogy melyik operációt kell végrehajtani.(Futás alatti kötés, vagy dinamikus kötés)

   (u) Virtuális metódus: címét a program csak futáskor oldja fel. Ha egy metódus virtuális, akkor öröklés után is az kell, hogy maradjon.

   (v) Absztrakt metódus: Üres virtuális metódus, csak örökítési célt szolgál

(w) Absztrakt osztály: Absztrakt metódust tartalmazó osztály, nem példányosítjuk, csak örökítési célt szolgál

(x) Osztálykönyvtár: Osztályok (komponensek) gyűjteménye

Programfejlesztési modellek

A program termékké vált (szolgáltatási funkció, minőség, előállítási költség, határidő), előállításához technológiára van szükség. Különleges termék.

1. A nagy méretű programrendszerek jellemzői

   (a) Nagy bonyolultságú rendszer (nem lehet fejben egy embernek átlátni)

   (b) Csapatmunkában készül

   (c) Hosszú élettartamú (verziók, stb)

2. A nagy méretű programrendszerek esetén felmerülő feladatok, melyekért a projektmenedzser felelős

   (a) A követelményeket előre pontosan írásban meg kell határozni

   (b) A program kidolgozásának menetét meg kell határozni (mérföldkövek, határidők, feladatok)

      i. A projekt részfeladatainak meghatározása $\left(\boxed{F_1}\ \boxed{F_2}\right)$

      ii. A részfeladatok megoldásiidejének megbecslése $\left(\boxed{\begin{matrix}F_1\\10\end{matrix}}\ \boxed{\begin{matrix}F_2\\20\end{matrix}}\right)$

      iii. A részfeladatok függőségeinek meghatározása

      iv. Mérföldkövek (ellenőrzési pontok) kijelölése $\left(\boxed{\begin{matrix}M_1\\2006.09.25\end{matrix}}\ \boxed{\begin{matrix}M_{2,4}\\2006.12.24\end{matrix}}\right)$

      Példa: Sike 20

   (c) Erőforrások meghatározása, beszerzése (szoftver, hardver, pénz, szakember)

   (d) A fejlesztés menetét is dokumentálni kell

   (e) Szervezni és irányítani kell a munkát, erőforrásokat

   (f) Igazolni kell a program jóságát

      i. Verifikáció: A specifikáció szerinti helyesség bizonyítása

      ii. Validáció: Az előírt minőségi tulajdonságok ellenőrzése (erőforrásigény, robosztusság, hatékonyság, bonyolultság, felhasználó–barátság)
      Validation: "Are we building the right product?", i.e., does the product do what the user really requires?
      Verification: "Are we building the product right?", i.e., does the product conform to the specifications?

   (g) A rendszer követésének, karbantartásának megszervezése/tervezése

3. Egyszerű programfejlesztési modell

   (a) Megoldandó feladat
   Programozás

   (b) Program az adott programozási nyelven
   Fordítás

   (c) Program gépi kódban
   Futtatás

   (d) Eredmény
   Tesztelés, javítás

4. Specifikációra épülő programfejlesztési modell

   (a) Megoldandó feladat
   Követelmények megfogalmazása

   (b) Informális leírás
   Specifikáció

   (c) **Formális leírás (absztrakt program)**
   Implementáció

   (d) Program az adott programozásinyelven
   Fordítás

(e) Program gépikódban
Futtatás

(f) Eredmény
Tesztelés, javítás

- Előnyei:
  Az absztrakt szintaxis és szemantika szempontjából pontos megoldás korai fázisban
  Az absztrakt leírás formális elemezhetősége
  A konkrét leírás (program) helyességének bizonyíthatósága az absztrakt leírás szerint
- Hátrányai:
  Magas matematikai ismeretek szükségesek a formális specifikációs módszerek alkalmazásánál
  A formális specifikáció alkalmazása nagyobb feladatoknál nehézkes és áttekinthetetlen
  A matematikai bizonyítás automatizálható, de a tételek megfogalmazása összetettebb esetekben már nehéz feladat. (Lsd. Floyd-Naur-Hoare–féle parciális helyesség bizonyítás)

5. Nagy rendszerek általános fejlesztési modellje

A formális specifikációs módszereket kisebb részfeladatoknál alkalmazzuk. A formális leírás helyébe a programrendszer tervének elkészítése lép.

(a) Megoldandó feladat
Követelmények megfogalmazása

(b) Informális leírás
Tervezés

(c) **Program rendszerterve**
Implementáció

(d) Program az adott programozásinyelven
Fordítás

(e) Program gépikódban
Futtatás

(f) Eredmény
Tesztelés, javítás

A programkészítés hagyományos fázisai

(a) Követelmények leírása

(b) Specifikáció

(c) Tervezés (vázlatos, finom)

(d) Implementáció, integráció

(e) Verifikáció, validáció

(f) Rendszerkövetés, karbantartás

Fázisok közötti kapcsolatok leírására szolgáló modellek:

(a) Vízesés modell

Probléma
Követelmények definíciója
Szoftverrendszer vázlatos terve
Szoftverrendszer finom terve
Implementáció+egységek tesztjei
Integráció+rendszer tesztje
Futtatás+karbantartás

- Előnyei:
  A gyakorlat szülte
- Hátrányai:
  A project munkájának megszervezése nagyon nehézkes (Párhuzamos munka, ellenőrzési pontok)
  Új szolgáltatás utólagos bevezetése mindenütt módosítást okoz
  A validáció az egész megismétlését követelheti meg

(b) Evolúciós modell
A megoldás iterációs folyamat eredménye, prototípusok sorozata.

- Előnye: Nem kell részletes specifikáció

- Hátrányai:
  Rövid idő alatt létrehozható rendszereknél eredményes
  A rendszer utólagos módosítása, bővítése nehézkes
  Nem tekinthető jól át a projekt
  A gyors fejlesztés a dokumentálás rovására megy

(c) Boehm-féle spirális modell (1988)

A programok iterációval készülnek, ezek spirálba olvaszthatóak, egy "spirál-menet" egy iteráció, melynek 4 szakasza van:

i. Célok, alternatívák, korlátozások
ii. Alternatívák kiértékelése, kockázatok elemzése
PROTOTÍPUS(OK)
iii. A fázis termékének megvalósítása, validáció
iv. A következő fázis megtervezése

Cél: az elemzés tárgya

Korlátozás: ami a megvalósításban határt szab a lehetőségnek

Alternatívák: a célokmegvalósításának különböző útjai

Kockázatok: az egyes alternatívák nagy valószínűséggel hibát okozó forrásai

Kockázat kezelése: stratégia felállítása a kockázat hatásának a csökkentésére

- Előnyei:
  Jó dokumentáltság
  A projekt strukturáltsága, az iterációs fázisok szabadon megválaszthatóak
  A fázisbeli szakaszok szintén iterálhatóak
  A fejlesztést mindig validáció követi
  Újabb fázisok lehetségesek minden ciklusban
- Hátrányai:
  Munkaigényes, bonyolult, költséges és nehezen oldható meg
  Nem gazdaságos a szakemberek foglalkoztatása, alig van párhuzamos munka