Minták a szoftverfejlesztésben

Jeszenszky Péter
jeszenszky.peter@inf.unideb.hu

Utolsó módosítás: 2018. december 13.

Készült Kollár Lajos korábbi anyagának felhasználásával

Minták

- A fogalom Christopher Alexander (1936–) építész nevéhez fűződik.
- Szinte azonnal átvették az elgondolást a szoftvertervezők.
- Széles körben ismertté és elterjedtté a „négyek bandájaként” ismert szerzők könyve révén váltak a szoftveriparban.

Mi a minta? (1)

- *„Minden minta olyan problémát ír le, ami újra és újra felbukkan a környezetünkben, s aztán leírja hozzá a megoldás magját, oly módon, hogy a megoldás milliószor felhasználható legyen, anélkül, hogy valaha is kétszer ugyanúgy csinálnánk.”*
  - Christopher Alexander. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press. 1977.
    - A fordítás forrása: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Programtervezési minták: Újrahasznosítható elemek objektumközpontú programokhoz*. Kiskapu, 2004.

Mi a minta? (2)

- Christopher Alexander:
  - *„Minden minta egy három részből álló szabály, mely egy bizonyos környezet, egy probléma és egy megoldás közötti kapcsolatot fejez ki.”*
    - *The Timeless Way of Building*. Oxford University Press. 1979.

Mi a minta? (3)

- Martin Fowler:
  - *„A minta egy olyan ötlet, mely egy gyakorlati környezetben már hasznosnak bizonyult, és várhatóan más környezetekben is hasznos lesz.”*
    - *Analysis Patterns: Reusable Object Models*. Addison-Wesley, 1996.

Mi a minta? (4)

- Scott W. Ambler:
  - *„A minta egy gyakori probléma vagy kérdés általános megoldásának leírása, melyből meghatározható egy konkrét probléma részletes megoldása.”*
    - http://www.ambysoft.com/books/processPatterns.html

A minta részei

- **Környezet** (***Context***):
  - Mely helyzetekben fordul elő a probléma.
- **Probléma** (***Problem***):
  - Az adott környezetben ismétlődően felmerülő probléma.
  - **Erő** (***Force***): Olyan szempontot jelent, melyet a megoldás során figyelembe kell venni.
    - Például: a megoldással szemben támasztott követelmények, megszorítások, a megoldás kívánatos jellemzői.
    - Az erők különböző nézőpontokból elemzik a problémát. Kiegészíthetik egymás vagy ellentmondhatnak egymásnak.
      - Példa ellentmondó erőkre: a rendszer kiterjeszthetősége és a kód méretének minimalizálása.
- **Megoldás** (***Solution***):
  - Hogyan oldjuk meg az ismétlődő problémát? Hogyan ellensúlyozzuk ki az erőket?
  - Egy megoldási sémát ad, nem egy részletes tervet.
  - Mentális építőkocka.

Mintakatalógusok és mintanyelvek (1)

- **Mintakatalógus** (***Pattern Catalog***)**/Mintagyűjtemény** (***Pattern Collection***)
  - Minták egy tetszőleges csoportja.
  - A gyűjtemény tartalmát tekintve lehet heterogén vagy fókuszálhat egy adott területre, problémára vagy absztrakciós szintre.
  - Szervezése történhet strukturálatlanul vagy strukturáltan.
  - A minták leírása többé-kevésbé egymástól függetlenül történik.
  - Példa: a GoF-féle katalógus.

Mintakatalógusok és mintanyelvek (2)

- **Mintanyelv** (***Pattern Language***): Egymással összefüggő olyan minták egy gyűjteménye, melyek együtt meghatároznak egy szisztematikus folyamatot szoftverfejlesztési problémák megoldására.
  - Példa: felhasználói felület tervezési minták.

Mintakatalógusok és mintanyelvek (3)

- A minták leírása mindig kötött formában történik (mintasablon).
  - A katalógusok és nyelvek eltérő formákat használnak a gyakorlatban.

Minták osztályozása

- **Architekturális minták/stílusok** (***Architectural Patterns/Styles***)
- **Tervezési minták** (***Design Patterns***)
- **Programozási idiómák/Implementációs minták** (***Programming Idioms/Implementation Patterns***)
- **Folyamatminták (*Process Patterns*)**
- **Elemzési minták (*Analysis Patterns*)**
- **Tesztelési minták** (***Test Patterns***)
- **Felhasználói felület tervezési minták** (***User Interface Design Patterns***)
- **Antiminták/Ellenminták** (***Antipatterns***)

Architekturális minták (1)

- Az architekturális minták szoftverrendszerek alapvető szerkezeti felépítésére adnak sémákat. Ehhez előre definiált alrendszereket biztosítanak, meghatározzák ezek felelősségi köreit, valamint szabályokat és irányelveket tartalmaznak a köztük lévő kapcsolatok szervezésére vonatkozólag. [POSA1]
  - Példák: mikrokernel, modell-nézet-vezérlő, …

Architekturális minták (2)

- Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.
- Frank Buschmann et al. *Pattern-Oriented Software Architecture Vol. 1–5*. Wiley, 1996, 2000, 2004, 2007.

Példa architekturális mintára: MVC (1)

- **Név**: Modell-nézet vezérlő (*Model-View-Control*)
- **Környezet**: Rugalmas ember-gép felülettel rendelkező interaktív alkalmazások.
- **Probléma**: Különösen gyakori az igény a felhasználói felületek változtatására.
  - Erők:
    - Ugyanaz az információt különböző módon jelenik meg különböző helyeken (például oszlop- vagy kördiagramon).
    - Az alkalmazás megjelenítésének és viselkedésének azonnal tükröznie kell az adatokon végzett műveleteket.
    - A felhasználói felület könnyen változtatható kell hogy legyen, akár futásidőben is.
    - Különböző *look and feel* szabványok támogatása vagy a felhasználói felület portolása nem érintheti az alkalmazás magjának kódját.
- **Megoldás**: Az interaktív alkalmazás három részre osztása:
  - A modell komponens az adatokat és a funkcionalitást csomagolja be, független a kimenet ábrázolásmódjától vagy az input viselkedésétől.
  - A nézet komponensek jelenítik meg az információkat a felhasználónak.
  - A vezérlő fogadja a bemenetet, melyet szolgáltatáskérésekké alakít a modell vagy a nézet felé.

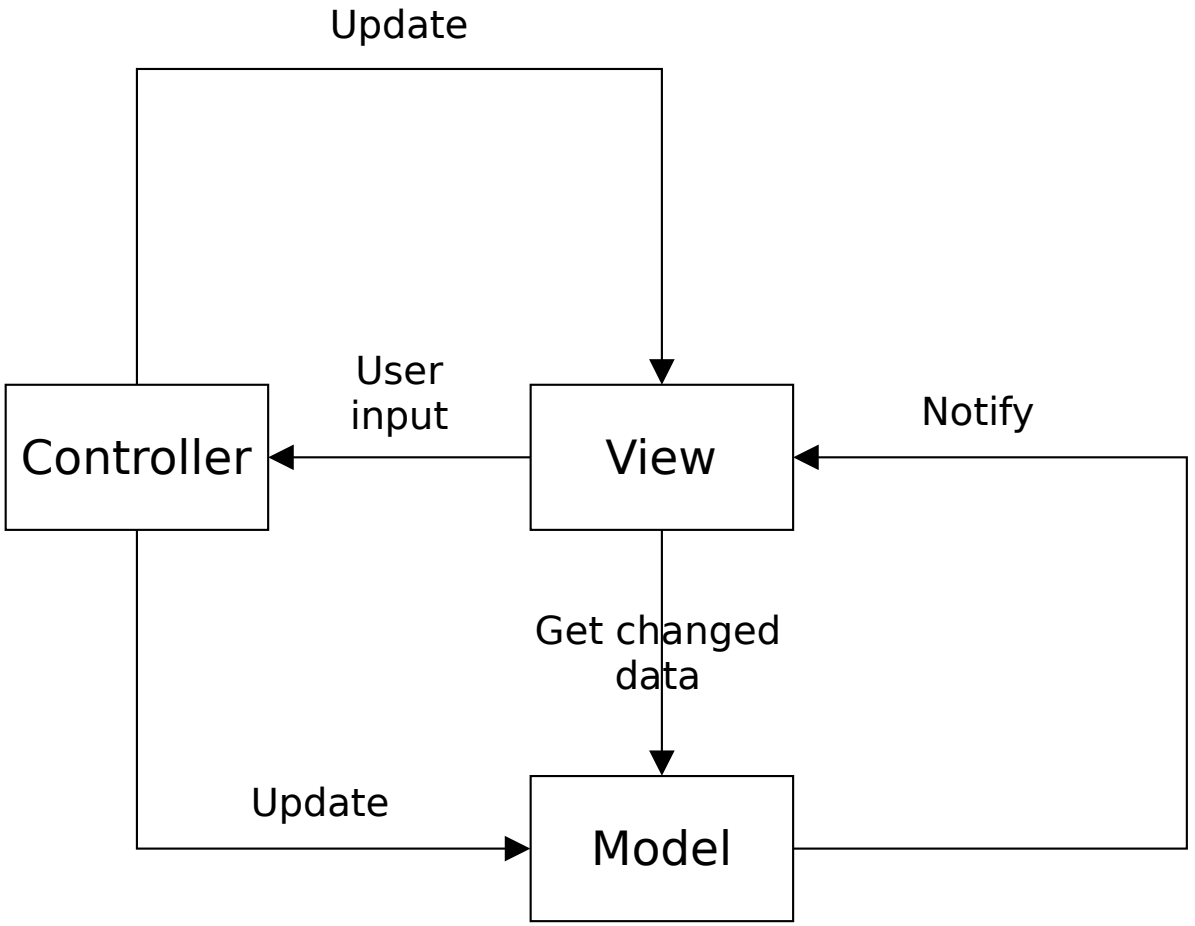
Példa architekturális mintára: MVC (2)

- A modell elválasztása a nézet komponenstől több nézetet is lehetővé tesz ugyanahhoz a modellhez.
  - Ugyanazok az adatok többféle módon is megjeleníthetők.
- A nézet elválasztása a vezérlő komponenstől kevésbé fontos.
  - Lehetővé tesz ugyanahhoz a nézethez akár több vezérlőt is.
    - A klasszikus példa ugyanahhoz a nézethez szerkeszthető és nem szerkeszthető viselkedés támogatása két vezérlővel.
  - A gyakorlatban sokszor csak egy vezérlő van nézetenként.

Példa architekturális mintára: MVC (3)

- A modell a szakterületeit valamilyen információját ábrázoló objektum, mely adatokat csomagol be.
  - Rendelkezik alkalmazás-specifikus feldolgozást végző eljárásokkal, melyeket a vezérlők hívnak meg a felhasználó nevében.
  - Függvényeket biztosít az adatokhoz való hozzáféréshez, melyeket a nézetek használnak a megjelenítendő adatok eléréséhez.
  - Regisztrálja a függő objektumokat (nézeteket és vezérlőket), melyeket értesít az adatokban történő változásokról.

Példa architekturális mintára: MVC (4)



Architekturális stílusok (1)

- Az architekturális stílusok szoftverrendszerek egy családját határozzák meg a szerkezeti felépítésük szempontjából. Az architekturális stílusok komponenseket és a közöttük lévő kapcsolatokat fejeznek ki, az alkalmazásukra vonatkozó megszorításokkal valamint a létrehozásukra szolgáló kompozíciós és tervezési szabályokkal együtt. [POSA1]
  - Példák: kliens-szerver, rétegzett, reprezentációs állapot átvitel (REST – *Representational State Transfer*), …

Architekturális stílusok (2)

- Az architekturális stílusok nagyon hasonlóak az architekturális mintákhoz, ugyanakkor több fontos tekintetben eltérnek a mintáktól.
  - A minták egy jól meghatározott ismétlődő tervezési problémát fejeznek ki, melyre egy megoldást adnak, mindezt annak a környezetnek a nézőpontjából, melyben a probléma felmerül.
  - Az architekturális stílusok egy aktuális tervezési helyezettől független nézőpontból fejeznek ki tervezési módszereket.

Tervezési minták (1)

- A tervezési minták középszintű minták, kisebb léptékűek az architekturális mintáknál.
- Alkalmazásuknak nincs hatása egy szoftverrendszer alapvető felépítésére, de nagyban meghatározhatják egy alrendszer felépítését.
- Függetlenek egy adott programozási nyelvtől vagy programozási paradigmától.

Tervezési minták (2)

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
  - Négyek bandája (GoF – *Gang of Four*)
- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Programtervezési minták: Újrahasznosítható elemek objektumközpontú programokhoz*. Kiskapu, 2004.

Tervezési minták (3)

- GoF:
  - A tervezési minták egymással együttműködő objektumok és osztályok leírásai, amelyek testre-szabott formában valamilyen általános tervezési problémát oldanak meg egy bizonyos összefüggésben.

Tervezési minták (4)

- Nyelvspecifikus ajánlott irodalom:
  - **C#**:
    - Steven John Metsker. *Design Patterns in C#*. Addison-Wesley, 2004.
    - *.NET Design Patterns in C#* http://www.dofactory.com/net/design-patterns
  - **C++**:
    - Andrei Alexandrescu. Modern C++ Design: Generic Programming and Design Patterns Applied. Addison-Wesley, 2001.
    - Alexander Shvets. *Design Patterns Explained Simply*. 2014. https://sourcemaking.com/design-patterns-ebook
  - **Groovy**:
    - *Design patterns in Groovy* http://groovy-lang.org/design-patterns.html
  - **Java**:
    - Alexander Shvets. *Design Patterns Explained Simply*. 2014. https://sourcemaking.com/design-patterns-ebook
    - Ilkka Seppälä. *Design patterns implemented in Java*. http://java-design-patterns.com/ https://github.com/iluwatar/java-design-patterns
  - **Python**:
    - Bruce Eckel et al. *Python 3 Patterns, Recipes and Idioms*. http://python-3-patterns-idioms-test.readthedocs.io/
    - Sakis Kasampalis. *A collection of design patterns/idioms in Python*. https://github.com/faif/python-patterns

Tervezési minták (5)

- A minták osztályozása céljuk szerint (GoF):
  - **Létrehozási minták** (***Creational Patterns***)
  - **Szerkezeti minták** (***Structural Patterns***)
  - **Viselkedési minták** (***Behavioral patterns***)

Tervezési minták (6)

- A GoF könyv 23 tervezési mintát ír le, időközben számos további tervezési minta született.
  - Például: absztrakt dokumentum (a*bstract document*), monád (m*onad*), tároló (r*epository*), többke (m*ultition*), iker (t*win*), …
- Új kategória: **Konkurencia minták** (***Concurrency Patterns***)
  - Például: aktív objektum (a*ctive object*), őrzött felfüggesztés (*guarded suspension*), szálkészlet (t*hread pool*), …

Tervezési minták (7)

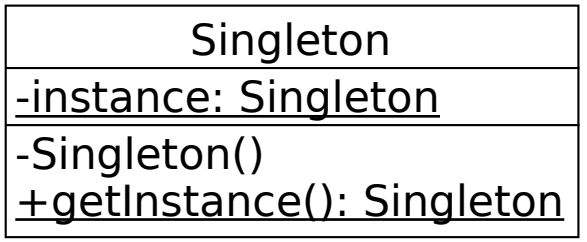
- Mintasablon (GoF):
  - **Név és besorolás** (***Name and Classification***):
  - **Cél** (***Intent***):
  - **Más néven** (***Also Known As***):
  - **Indíték** (***Motivation***):
  - **Alkalmazhatóság** (***Applicability***):
  - **Szerkezet** (***Structure***):
  - **Résztvevők** (***Participants***):
  - **Együttműködés** (***Collaborations***):
  - **Következmények** (***Consequences***):
  - **Megvalósítás** (***Implementation***):
  - **Példakód** (***Sample Code***):
  - **Ismert felhasználások** (***Known Uses***):
  - **Kapcsolódó minták** (***Related Patterns***):

Példa tervezési mintára: egyke (1)

- **Cél**: Egy osztályból csak egy példányt engedélyezni, és ehhez globális hozzáférési pontot ad megadni.
- **Indíték**: Egyes osztályok esetében fontos, hogy pontosan egy példány legyen belőlük.
- **Alkalmazhatóság**: Az egyke mintát a következő esetekben használjuk:
  - Pontosan egy példányra van szükség valamelyik osztályból, és annak elérhetőnek kell lennie az ügyfelek számára a jól ismert elérési pontokból.
  - Ennek az egyetlen példánynak alosztályokkal bővíthetőnek kell lennie, és az ügyfeleknek képeseknek kell lenniük saját kódjuk módosítása nélkül használni a bővített példányt.

Példa tervezési mintára: egyke (2)

- **Szerkezet**:



Példa tervezési mintára: egyke (3)

- Java megvalósítás (1): mohó inicializálás, szálbiztos

```
public class Singleton {

    private static Singleton instance = new Singleton();

    private Singleton() {
    } // privát láthatóságú konstruktor!

    public static Singleton getInstance() {
        return instance;
    }

}
```

Példa tervezési mintára: egyke (4)

- Java megvalósítás (2): lusta inicializálás, szálbiztos

```
public class Singleton {

    private static Singleton instance;

    private Singleton() {
    } // privát láthatóságú konstruktor!

    public static synchronized Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }

}
```

Példa tervezési mintára: egyke (5)

- Java megvalósítás (3): lusta inicializálás, szálbiztos (explicit szinkronizálás nélkül!) (*initialization-on-demand holder*)

```
public class Singleton {

    private static class Holder {
        private static Singleton instance = new Singleton();
    }

    private Singleton() {
    } // privát láthatóságú konstruktor!

    public static Singleton getInstance() {
        return Holder.instance;
    }

}
```

Példa tervezési mintára: egyke (6)

- Java megvalósítás (4): enum (J2SE 5.0–)

```
public enum Singleton {

    INSTANCE;

}
```

Példa tervezési mintára: egyke (7)

- Java megvalósítás (5): lusta inicializálás, szálbiztos (*double-checked locking*)

```
public class Singleton {

    private static volatile Singleton instance;

    private Singleton() {
    }

    public static Singleton getInstance() {
        if (instance == null) {
            synchronized (Singleton.class) {
                if (instance == null) {
                    instance = new Singleton();
                }
            }
        }
        return instance;
    }

}
```

Példa tervezési mintára: egyke (8)

- Java megvalósítás (5): lusta inicializálás, szálbiztos (*double-checked locking*) (folytatás)
  - Ez a megoldás a J2SE 5.0 előtt nem volt szálbiztos!
  - Lásd:
    - *The „Double-Checked Locking is Broken” Declaration*. http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html
    - Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea. *Java Concurrency in Practice*. Addison-Wesley, 2006. http://jcip.net/
      - A *The Java Memory Model* című 16. fejezet 16.2 alfejezete tárgyalja a szálbiztos inicializálást.

Programozási idiómák/ Implementációs minták (1)

- Egy idióma egy programozási nyelvre jellemző alacsony szintű minta. Az idiómák jelentik a legalacsonyabb szintű mintákat.
- Egy idióma leírja, hogy hogyan valósítsuk meg komponensek és kapcsolataik bizonyos vonatkozásait az adott nyelv eszközrendszerével.
- A legtöbb idióma nyelvspecifikus, létező programozási tapasztalatot hordoznak.

Programozási idiómák/ Implementációs minták (2)

- Kent Beck. *Implementation Patterns*. Addison-Wesley, 2008.
- Kent Beck. *Implementációs minták – 77 szoftverminta*. Panem, 2008.
- Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

Programozási idiómák/ Implementációs minták (3)

- Nyelvspecifikus ajánlott irodalom:
  - **C#**:
    - Bill Wagner. *Effective C#: 50 Specific Ways to Improve Your C#*. Third Edition. Addison-Wesley Professional, 2016.
    - Bill Wagner. *More Effective C#: 50 Specific Ways to Improve Your C#*. Addison-Wesley Professional, 2008.
  - **C++**:
    - Scott Meyers. *Effective C++: 55 Specific Ways to Improve Your Programs and Designs*. Third Edition. Addison-Wesley Professional, 2008.
    - Scott Meyers. *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14*. O'Reilly Media, 2014.
  - **Java**:
    - Joshua Bloch. *Effective Java*. Third Edition. Addison-Wesley Professional, 2017. http://www.informit.com/store/effective-java-9780134685991
  - **Python**:
    - Brett Slatkin. *Effective Python: 59 Specific Ways to Write Better Python*. Addison-Wesley Professional, 2015. http://www.effectivepython.com/
    - Luciano Ramalho. *Fluent Python*. O'Reilly Media, 2015. https://github.com/fluentpython

Programozási idiómák/ Implementációs minták (4)

- Példa:

```
public final class Pet implements Cloneable {

    private String name;
    private int age;

    public Pet(String name, int age) {
        if (age < 0)
            throw new IllegalArgumentException();
        this.name = name;
        this.age = age;
    }

    @Override
    public boolean equals(Object o) {
        if (o == this)
            return true;
        if (! (o instanceof Pet))
            return false;
        Pet that = (Pet) o;
        return (name == that.name || (name != null && name.equals(that.name)))
             && age == that.age;
    }

    // Hiba: nincs hashCode() és clone() metódus!
}
```

Programozási idiómák/ Implementációs minták (5)

- Példa (folytatás):

```
Set<Pet> s = new HashSet<Pet>();
Pet p = new Pet("Tardar Sauce", 4);
s.add(p);

System.out.println(s.contains(p));                              // true
System.out.println(s.contains(new Pet("Tardar Sauce", 4)));    // false

System.out.println(s.contains(p.clone()));  // fordítási hiba
```

Programozási idiómák/ Implementációs minták (6)

- Példa (folytatás):

```
public final class Pet implements Cloneable {

    // ...

    @Override
    public int hashCode() {
        int result = Integer.hashCode(age);
        result = 31 * result + (name == null ? 0 : name.hashCode());
        return result;
    }

    @Override
    public Pet clone() {
        try {
            return (Pet) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new AssertionError();
        }
    }

}
```

Folyamatminták (1)

- Scott W. Ambler. *Process Patterns: Building Large-Scale Systems Using Object Technology*. Cambridge University Press, 1998.
- Scott W. Ambler. *More Process Patterns: Delivering Large-Scale Systems Using Object Technology*. Cambridge University, 1999.

Folyamatminták (2)

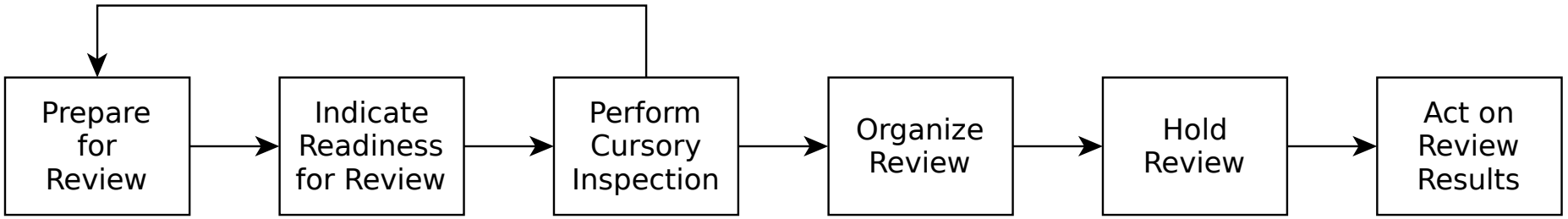
- **Folyamat**: Tevékenységsorozat, melynek során egy vagy több bemenetből egy vagy több kimenet kerül előállításra.
- **Folyamatminta**: Objektumorientált szoftverfejlesztéshez használható általános módszerek, műveletek és/vagy feladatok (tevékenységek) gyűjteménye.
  - Szervezett módon alkalmazva őket kialakítható egy szervezet szoftverfolyamata
    - Mivel a folyamatminták nem határozzák meg, hogy pontosan hogyan kell elvégezni egy feladatot, újrafelhasználható építőelemekként lehet őket használni a szervezet igényeire szabott szoftverfolyamat kialakításához
  - Három fajtája: feladat, folyamat és fázis folyamatminta

Folyamatminták (3)

- **Feladat folyamatminta** (***Task Process Pattern***): Egy adott feladat elvégzésének részletes lépéseit ábrázolja.
  - Példa: **Műszaki felülvizsgálat**
- **Szakasz folyamatminta** (***Stage Process Pattern***): Magasabb szintű folyamatminta, mely gyakran számos feladat folyamatmintából áll. Azon lépéseket ábrázolja, melyeket egy projektszakaszban gyakran iteratívan kell elvégezni.
  - Példa: **Programozás**
- **Fázis folyamatminta** (***Phase Process Pattern***): Egy projektfázis szakasz folyamatmintái közötti kölcsönhatásokat ábrázolja. A fázis folyamatminták végrehajtása soros módon történik.
  - Példa: **Létrehozás**

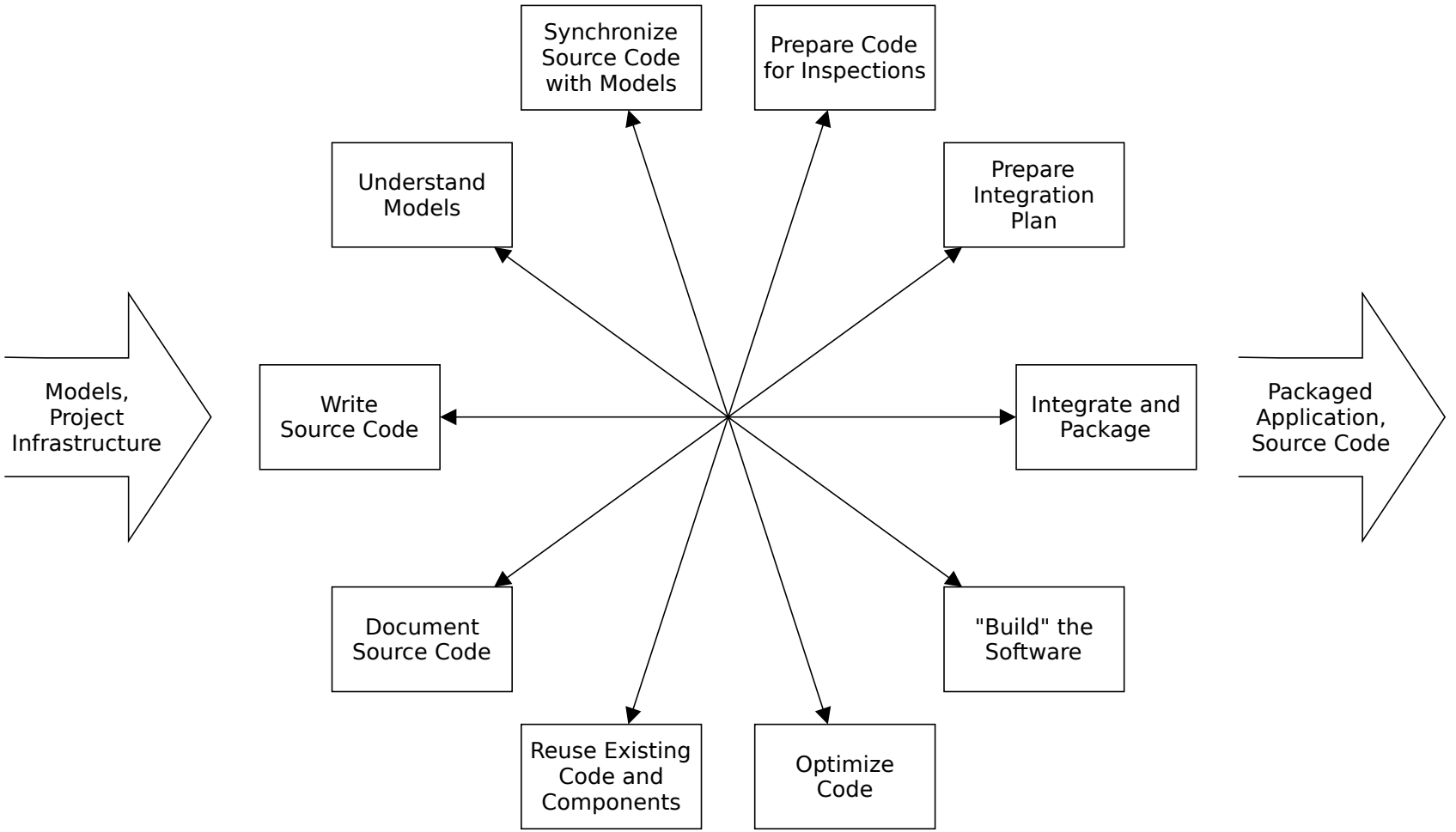
Folyamatminták (4)

- Példa feladat folyamatmintára: **Műszaki felülvizsgálat** (*Technical Review*)



Folyamatminták (5)

- Példa szakasz folyamatmintára: **Programozás** (***Program***)



Elemzési minták (1)

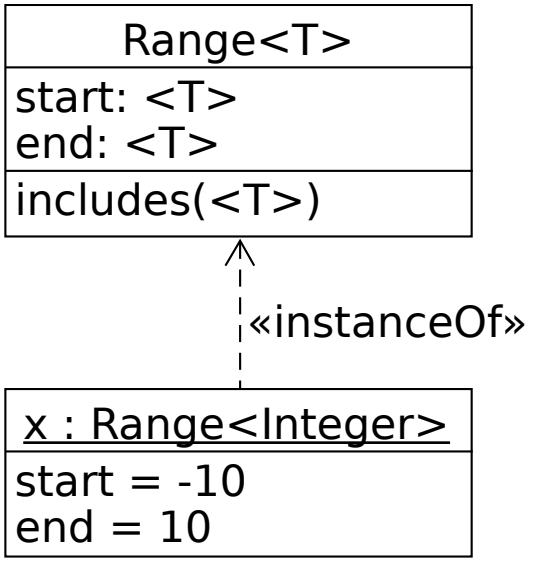
- Az elemzési minták az üzleti modellezésben gyakori konstrukciókat reprezentáló fogalmak csoportjai, melyek egy vagy több szakterületre is vonatkozhatnak.

Elemzési minták (2)

- Martin Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, 1996.
- *tagged by: analysis patterns* https://martinfowler.com/tags/analysis%20patterns.html

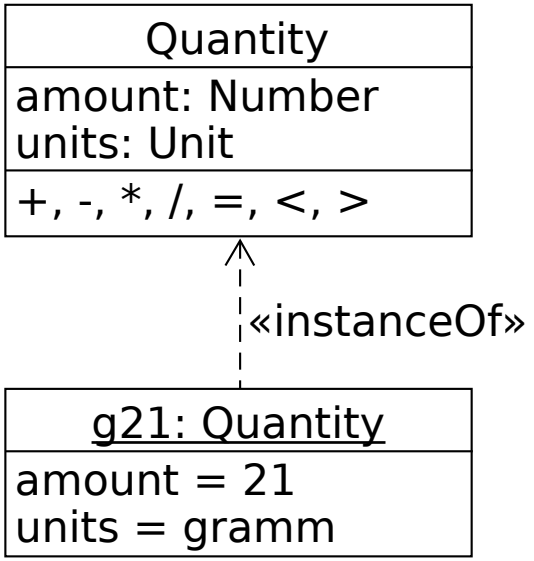
Elemzési minták (3)

- Példa elemzési mintára: tartomány https://martinfowler.com/eaaDev/Range.html
  - Egy értéktartományt ábrázol.



Elemzési minták (4)

- Példa elemzési mintára: mennyiség https://www.martinfowler.com/eaaDev/quantity.html
  - Egy mennyiségi egységgel ellátott numerikus értéket ábrázol.



Tesztelési minták (1)

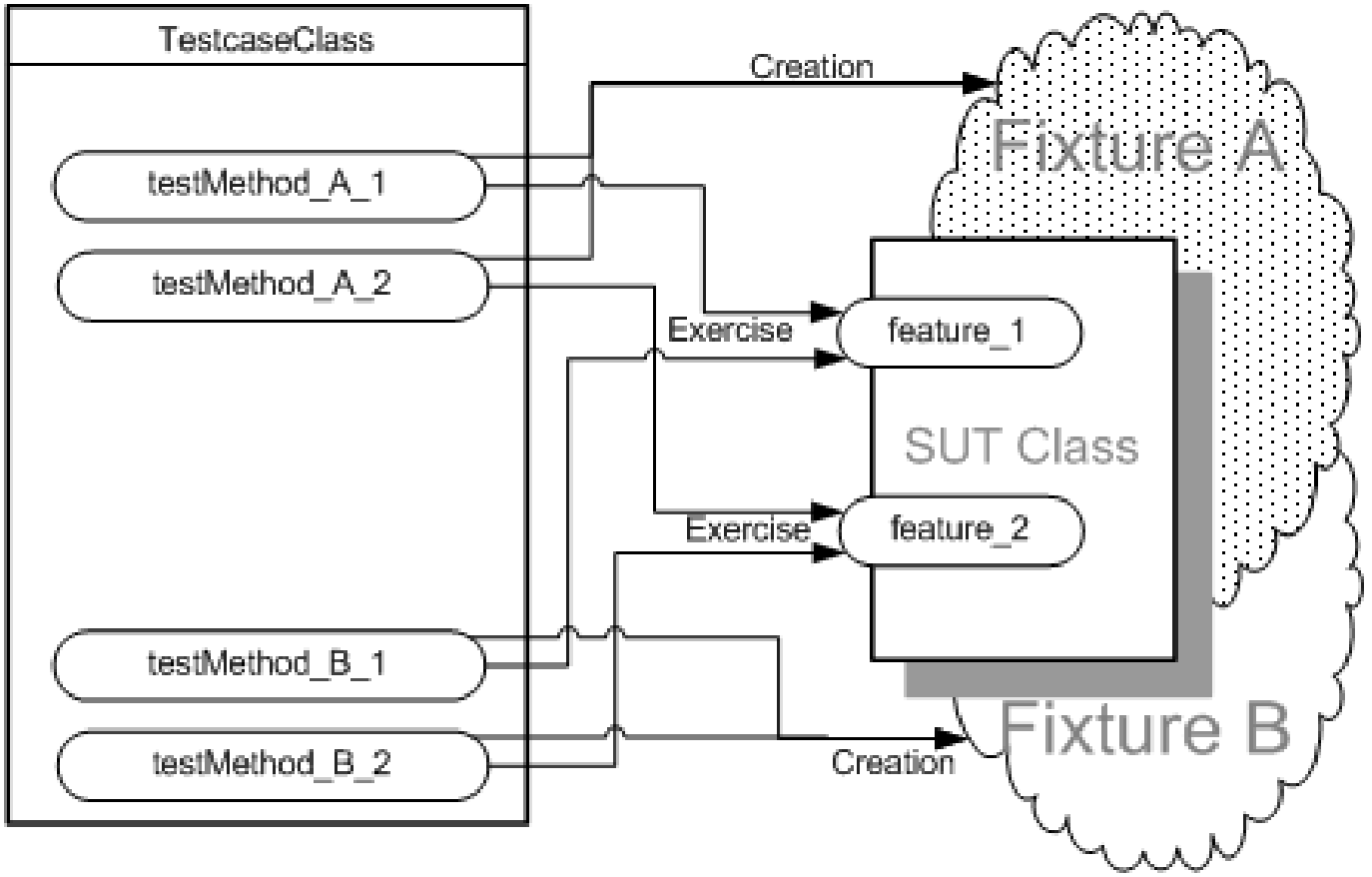
- Gerard Meszaros. *xUnit Test Patterns: Refactoring Test Code*. Addison-Wesley, 2007. http://xunitpatterns.com/

Tesztelési minták (2)

- **Osztályonkénti teszteset osztály** (***Testcase Class per Class***) (617. oldal):
  - Hogyan szervezzük tesztmetódusainkat teszteset osztályokba?
    - Egy adott osztályt tesztelő összes tesztmetódust helyezzük egy teszteset osztályba.
  - Mikor használjuk?
    - Ha nincs túl sok tesztmetódus vagy csak most kezdtünk el teszteket írni a tesztelendő rendszerhez.
    - Ahogy nő a tesztek száma és jobban megértjük a tesztadat (*fixture*) követelményeinket, a teszteset osztályunkat több osztályra vághatjuk szét (lásd: *Testcase Class per Fixture*, *Testcase Class per Feature*).

Tesztelési minták (3)

- A kép forrása: *Testcase Class per Class* http://xunitpatterns.com/Testcase%20Class%20per%20Class.html



Felhasználói felület tervezési minták (1)

- Újrafelhasználható megoldások gyakran előforduló felhasználói felület tervezési problémákra.
  - Asztali, webes és mobil felhasználói felületek.

Felhasználói felület tervezési minták (2)

- Jenifer Tidwell. *Designing Interfaces: Patterns for Effective Interaction Design*. Second Edition. O'Reilly Media, 2010. http://designinginterfaces.com/
- Christian Crumlish, Erin Malone. *Designing Social Interfaces: Principles, Patterns, and Practices for Improving the User Experience*. Second Edition. O'Reilly Media, 2015. http://www.designingsocialinterfaces.com/

Felhasználói felület tervezési minták (3)

- További tervezési minta gyűjtemények:
  - Anders Toxboe. *UI-Patterns.com*. http://ui-patterns.com/
  - *Oracle Alta UI Patterns* http://www.oracle.com/webfolder/ux/middleware/alta/patterns.html
  - *USPTO UI Design Library* https://uspto.github.io/designpatterns/
  - *VMware UI Patterns Library* http://ui-patterns.vmware.com/
  - ...

Felhasználói felület tervezési minták (4)

- Mintasablon (Tidwell):
  - **Mit**:
  - **Mikor használd**:
  - **Miért**:
  - **Hogyan**:
  - **Példák**:

Felhasználói felület tervezési minták (5)

- **Mély háttér** (Tidwell, 499. oldal):
  - **Mit**: Helyezz egy képet vagy színátmenetet az oldal hátterébe, mely vizuálisan visszaugrik az előtér elemek mögött.
  - **Mikor használd**: Az oldalnak hangsúlyos vizuális elemei vannak (mint például szövegblokkok, vezérlő elemek csoportjai vagy ablakok), és nem nagyon sűrű az elrendezés. Azt szeretnéd, hogy az oldal jellegzetes és szemrevaló legyen, vizuális márkaépítési stratégiára is gondolhatsz. Valami érdekesebb oldalhátteret szeretnél egy sima fehérnél vagy szürkénél.
  - **Miért**: A lágy fókuszú, színátmenetes és más mélységjelzőkkel rendelkező hátterek hátrahúzódnak az előterükben lévő élesebben definiált tartalom mögött. A tartalom ilyen módon „lebegni” látszik a háttér előtt. Ez a pszeudo-3D kinézet egy erős figura-alap hatást eredményez.
  - **Hogyan**: Használj olyan hátteret, mely rendelkezik ezen jellemzők közül eggyel vagy többel: lágy fókusz, színátmenetek, mélységjelzők, nincsenek erős fókuszpontok.
  - **Példák**: https://www.mozilla.org/hu/firefox/new/

Antiminták (1)

- Egy problémára adott általánosan előforduló megoldások, melyek kifejezetten negatív következményekkel járnak.
- Bármely szinten megjelenhetnek.

Antiminták (2)

- William H. Brown, Raphael C. Malveau, Hays W. McCormick III, Thomas J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons, 1998.
- https://sourcemaking.com/antipatterns
- Phillip A. Laplante, Colin J. Neill, Joanna F. DeFranco. *AntiPatterns: Managing Software Organizations and People*. 2nd Edition. Auerbach, 2011.

Antiminták (3)

- Nézőpont szerint az alábbi három kategória:
  - Szoftverfejlesztési antiminták
  - Szoftver architekturális antiminták
  - Szoftverprojekt vezetési antiminták

Szoftverfejlesztési antiminták (1)

- **A massza** (***The Blob***): A procedurális stílusú tervezés egy olyan objektumhoz vezet, melyé a felelősség oroszlánrésze, miközben a legtöbb más objektum csak adatokat tárol vagy egyszerű folyamatokat hajt végre.
- **Folyamatos korszerűtlenség** (***Continuous Obsolescence***): A technológia olyan gyorsan változik, hogy a fejlesztőknek nehézséget jelent lépést tartani a szoftver naprakész verzióival és megtalálni az együttműködni képes termékkiadás kombinációkat.
- **Lávafolyás** (***Lava Flow***): Halott kód és feledésbe merült tervezési információk fagynak be egy állandóan változó tervbe.

Szoftverfejlesztési antiminták (2)

- **Funkcionális dekompozíció** (***Functional Decomposition***): Tapasztalt, nem objektumorientált fejlesztők objektumorientált nyelven terveznek és valósítanak meg egy alkalmazást. Az így előálló kód hasonlít egy strukturált nyelven (Pascal, FORTRAN) történő megvalósításhoz. Hihetetlenül bonyolult lehet, mert az ügyes procedurális programozók ötletes módokat agyalnak ki a jól bevált módszereik egy objektumorientált architektúrában történő replikálására.
- **Kopogó szellem** (***Poltergeist***): Osztályok nagyon korlátozott szerepekkel és tényleges életciklusokkal. Gyakran indítanak el folyamatokat más objektumoknak.
- **Vasmacska** (***Boat Anchor***): Olyan programrész vagy hardverelem, mely nem szolgál semmilyen hasznos célt az aktuális projektben. Gyakran olyan költséges szerzemény, mely még inkább ironikussá teszi a megvásárlását.

Szoftverfejlesztési antiminták (3)

- **Arany kalapács** (***Golden Hammer***): Egy jól ismert technológia vagy fogalom rögeszmés alkalmazása sok problémára.
- **Zsákutca** (***Dead End***): Egy újrafelhasználható komponens módosítása, ha a módosított komponenst többé nem a szállító tartja karban és nem is támogatja.
- **Spagetti kód** (***Spaghetti Code***): Az *ad hoc* programszerkezet megnehezíti a kód bővítését és optimalizálását.

Szoftverfejlesztési antiminták (4)

- **Bemenet gányolás** (***Input Kludge***): *Ad hoc* algoritmusok alkalmazása a program bemenetének kezelésére.
- **Séta aknamezőn** (***Walking through a Minefield***): Napjaink szoftvertechnológiáinak használata hasonló egy csúcstechnológiai aknamezőn való sétához, mivel számos hiba van a kiadott szoftvertermékekben.
- **Vágólap programozás** (***Cut−and−Paste Programming***): A programsorok másolása révén történő újrafelhasználás jelentős karbantartási problémákat okoz.
- **„Gomba” menedzsment** (***Mushroom Management***): A rendszerfejlesztők elszigetelten tartása a rendszer végelhasználóitól, akikhez így másodkézből jutnak el a követelmények közvetítőkön (mérnökökön, menedzsereken, követelményelemzőkön) keresztül.

Szoftver architekturális antiminták (1)

- **Automatikusan generált kályhacső** (***Autogenerated Stovepipe***): Az antiminta egy létező szoftverrendszer osztott infrastruktúrára történő migrálásakor fordul elő. Akkor jelentkezik, amikor az interfészeket újratervezés nélkül veszik át az osztott infrastruktúrához.
- **Kályhacső vállalat** (***Stovepipe Enterprise***): Egy vállalaton belül egymástól függetlenül tervezett rendszerek gátolják az interoperabilitást és az újrafelhasználást, felhajtják a költségeket.
- **Kályhacső rendszer** (***Stovepipe System***): Az alrendszerek integrálása *ad hoc* módon történik egy rendszeren belül, több integrációs stratégia és mechanizmus felhasználásával. A kályhacső vállalat megfelelője egyetlen rendszer esetére.

Szoftver architekturális antiminták (2)

- **Védd a hátsód** (***Cover Your Assets***): A dokumentum-vezérelt szoftverfolyamatok gyakran nem túl hasznos követelményeket és specifikációkat eredményeznek, mivel a szerzők kitérnek a fontos döntések hozása elől.
- **Gyártótól függés** (***Vendor Lock-In***): Adott gyártók architektúráitól való nagymértékű függés.
- **Hamis jegy** (***Wolf Ticket***): Olyan termék, mely úgy állít magáról nyíltságot és szabványnak való megfelelést, hogy annak nincs kényszerítő ereje.

Szoftver architekturális antiminták (3)

- **Hallgatólagos architektúra (*Architecture by Implication*)**: Nem készül architekturális terv, mert a fejlesztők elbizakodottak a hasonló rendszerekkel kapcsolatban szerzett korábbi tapasztalataik miatt.
- **Bizottsági tervezés** (***Design by Committee***): Egy bizottság által készített terv túlságosan bonyolult architektúrájú és hiányzik belőle a koherencia.
- **Svájci bicska** (***Swiss Army Knife***): A svájci bicska egy rendkívül bonyolult osztály interfész. A tervező megpróbál az osztály összes lehetséges felhasználásáról gondoskodni. Más programozók számára nehéz a bonyolult interfész megértése, nem világos a használata még egyszerű esetre sem.
- **A spanyolviasz feltalálása** (***Reinvent the Wheel***): A szoftverprojektek közötti technológia átvitel átható hiánya jelentős mértékű újra-feltaláláshoz vezet.

Antiminta sablon (1)

- **Antiminta neve** (***AntiPattern Name***):
  - Az antiminta egyedi (pejoratív) neve.
- **Más néven** (***Also Known As***):
  - Az antiminta más ismert elnevezései.
- **Leggyakoribb előfordulási szint** (***Most Frequent Scale***):
  - A szoftvertervezési modell mely szintjén fordul elő jellemzően az antiminta. A következő kulcsszavak kerülhetnek ide: idióma, mikroarchitektúra, keretrendszer, alkalmazás, rendszer, vállalat, globális/ipar.
- **Újragyártott megoldás neve** (***Refactored Solution Name***):
  - Az újragyártott megoldási sémát azonosítja.
- **Újragyártott megoldás típusa** (***Refactored Solution Type***):
  - Azt jelzi, hogy milyen fajta tevékenységet jelent az antiminta megoldása. A következő kulcsszavak kerülhetnek ide: szoftver, technológia, folyamat, szerep.

Antiminta sablon (2)

- **Kiváltó okok** (***Root Causes***):
  - Az antiminta kiváltó okait megadó kulcsszavak, melyek a következők lehetnek: sietség, fásultság, szűklátókörűség, lustaság, fösvénység, tudatlanság, büszkeség, elhanyagolt felelősség.
- **Kiegyensúlyozatlan erők** (***Unbalanced Forces***):
  - Azokat a tényezőket megadó kulcsszavak, melyeket figyelmen kívül hagytak, rosszul használtak vagy túl sokszor használtak a mintában. A választási lehetőségek közé tartoznak a következők: a funkcionalitás kezelése, a teljesítmény kezelése, a bonyolultság kezelése, a változás kezelése, az IT erőforrások kezelése, a technológia-transzfer kezelése.
- **Anekdotaszerű példa** (***Anecdotal Evidence***):
  - Opcionális rész, mely az antimintához kötődő ismert mondásokat tartalmaz.
- **Háttér** (***Background***):
  - Opcionális rész, mely további példákat tartalmazhat a probléma előfordulási helyeiről vagy további hasznos vagy érdekes általános háttérinformációkat.
- **Általános alak** (***General Form***):
  - Az antiminta általános leírása, mely gyakran tartalmaz ábrát is. Nem egy példa, hanem egy általános változat.

Antiminta sablon (3)

- **Tünetek és következmények** (***Symptoms and Consequences***):
  - Az antiminta tüneteinek és az antiminta általa okozott következmények felsorolása.
- **Tipikus okok** (***Typical Causes***):
  - Az antiminta (konkrét) okainak felsorolása.
- **Ismert kivételek** (***Known Exceptions***):
  - Azokat a kivételes eseteket írja le ez a rész, melyeknél az antiminta nem káros.
- **Újragyártott megoldás** (***Refactored Solution***):
  - Az általános alaknál megfogalmazott antimintára adott megoldást ismerteti ez a rész lépésekre bontva.
- **Változatok** (***Variations***):
  - Opcionális rész, mely az antiminta általános alakjának esetleges változatait sorolja fel és fejti ki.

Antiminta sablon (4)

- **Alkalmazhatóság más nézőpontokra és szintekre (*Applicability to Other Viewpoints and Scales*):**
  - Hogyan befolyásolja a minta a többi nézőpontot: vezetési, architekturális, fejlesztési. Ugyancsak itt kerül leírásra, hogy milyen mértékben releváns a minta más szintek szempontjából.
- **Példa** (***Example***):
  - Ez a rész szemlélteti a megoldás a problémára való alkalmazását. Rendszerint megjelenik itt a probléma sematikus ábrája, a probléma leírása, a megoldás sematikus ábrája, a megoldás leírása.
- **Kapcsolódó megoldások** (***Related Solutions***):
  - Az adott antimintához szorosan kapcsolódó tervezési minták és antiminták kerülnek itt felsorolásra, valamint kifejtésre kerülnek a különbségek.

Szoftverfejlesztési antiminta: a massza (1)

- **Antiminta neve**: a massza (*The Blob*)
- **Más néven**: *Winnebago*, az Isten osztály (*The God Class*)
- **Leggyakoribb előfordulási szint**: alkalmazás
- **Újragyártott megoldás neve**: a felelősségek újraosztása
- **Újragyártott megoldás típusa**: szoftver
- **Kiváltó okok**: lustaság, sietség
- **Kiegyensúlyozatlan erők**: a funkcionalitás, a teljesítmény és a bonyolultság kezelése
- **Anekdotaszerű példa**: „Ez az osztály az architektúránk szíve.”

Szoftverfejlesztési antiminta: a massza (2)

- **Háttér**: A massza (*The Blob*) (1958) http://www.imdb.com/title/tt0051418/
- **Általános alak**:
  - A massza olyan tervezésnél fordul elő, ahol a feldolgozást egyetlen osztály sajátítja ki magának, a többi osztály pedig elsősorban adatokat zár egységbe.
  - Olyan osztálydiagram jellemzi, mely egyetlen bonyolult vezérlő osztályból és azt körülvevő egyszerű adat osztályokból áll.
  - A massza általában procedurális tervezésű, habár reprezentálható objektumokkal és implementálható objektumorientált nyelven.
  - Gyakran iteratív fejlesztés eredménye, ahol egy megvalósíthatósági példakódot (*proof−of−concept code*) fejlesztenek idővel egy prototípussá, végül pedig egy éles rendszerré.

Szoftverfejlesztési antiminta: a massza (3)

- **Tünetek és következmények**:
  - Egyetlen osztály nagyszámú attribútummal, művelettel vagy mindkettővel. Általában a massza jelenlétét jelzi egy 60-nál több attribútummal és művelettel rendelkező osztály.
  - Egymáshoz nem kapcsolódó attribútumok és műveletek bezárása egyetlen osztályba.
  - …
  - Egy ilyen osztály túl bonyolult újrafelhasználáshoz vagy teszteléshez.
  - Költséges lehet egy ilyen osztály a memóriába való betöltése. Még egyszerű műveletekhez is sok erőforrást használ.
- **Tipikus okok**:
  - Az objektumorientált architektúra hiánya.
  - (Bármilyen) architektúra hiánya.
  - Az architektúra kikényszerítésének hiánya.
  - Túl korlátozott beavatkozás.
  - Kódolt katasztrófa (rossz követelmény specifikáció).

Szoftverfejlesztési antiminta: a massza (4)

- **Ismert kivételek**: A massza antiminta elfogadható kompatibilitási okokból megtartott korábbi rendszer becsomagolásakor.
- **Újragyártott megoldás**: A megoldás kódújraszervezéssel jár
  - Összetartozó attribútumok és műveletek csoportjainak azonosítása.
  - Természetes helyet kell keresni ezen funkcionalitás-csoportok számára és oda kell áthelyezni őket.
  - A redundáns asszociációk eltávolítása.

Szoftverfejlesztési antiminta: a massza (5)

- **Változatok**:
  - Viselkedési forma: osztály, mely tartalmaz egy központi folyamatot, mely interakcióban van a rendszer legtöbb más részével („központi agy osztály”).
  - Adat forma: osztály, mely tartalmaz olyan adatokat, melyeket a rendszer legtöbb más objektuma használ („globális adat osztály”).
- **Alkalmazhatóság más nézőpontokra és szintekre**: Az architekturális és a vezetési nézőpontnak is kulcsszerepe van a massza antiminta megelőzésében.
- **Példa**:

Szoftverfejlesztési antiminta: spagetti kód (1)

- **Antiminta neve**: spagetti kód (*Spaghetti Code*)
- **Leggyakoribb előfordulási szint**: alkalmazás
- **Újragyártott megoldás neve**: kódújraszervezés, kódtisztítás
- **Újragyártott megoldás típusa**: szoftver
- **Kiváltó okok**: tudatlanság, lustaság
- **Kiegyensúlyozatlan erők**: a bonyolultság, a változás kezelése
- **Anekdotaszerű példa**: „Ó! Micsoda zűrzavar!”, „Ugye tisztában vagy vele, hogy a nyelv egynél több függvényt támogat?”, „Könnyebb újraírni ezt a kódot, mint megpróbálni módosítani.”, …

Szoftverfejlesztési antiminta: spagetti kód (2)

- **Háttér**: Klasszikus, a leghíresebb antiminta, mely egyidős a programozási nyelvekkel.
- **Általános alak**:
  - Strukturálatlan, nehezen átlátható programkódként jelentik meg.
  - Objektumorientált nyelvek esetén kevés osztály jellemzi, melyeknél a metódusok megvalósítása nagyon hosszú.

Szoftverfejlesztési antiminta: spagetti kód (3)

- **Tünetek és következmények**:
  - A metódusok nagyon folyamat-orientáltak, az objektumokat gyakran folyamatoknak nevezik.
  - A végrehajtást az objektum implementáció határozza meg, nem pedig az objektum kliensei.
  - Kevés kapcsolat van az objektumok között.
  - Sok a paraméter nélküli metódus, melyek osztályszintű és globális változókat használnak.
  - Nehéz a kód újrafelhasználása. Sok esetben nem is szempont az újrafelhasználhatóság.
  - Elvesznek az objektumorientáltság előnyei, nem kerül felhasználásra az öröklődés és a polimorfizmus.
  - A további karbantartási erőfeszítések csak súlyosbítják a problémát.
  - Költségesebb a létező kódbázis karbantartása, mint egy új megoldás kifejlesztése a semmiből.

Szoftverfejlesztési antiminta: spagetti kód (4)

- **Tipikus okok**:
  - Tapasztalatlanság az objektumorientált tervezés terén.
  - Nincs mentorálás, nem megfelelő a kódátvizsgálás.
  - Nincs az implementálást megelőző tervezés.
  - A fejlesztők elszigetelten dolgoznak.
- **Ismert kivételek**: Ésszerűen elfogadható, ha az interfészek következetesek és csak az implementáció spagetti.
- **Újragyártott megoldás**: A megoldás kódújraszervezés.

Szoftverfejlesztési antiminta: spagetti kód (5)

- **Példa**:
- **Kapcsolódó megoldások**: analízis-paralízis, lávafolyás

Szoftver architekturális antiminta: gyártótól függés (1)

- **Antiminta neve**: gyártótól függés (*Vendor Lock−In*)
- **Más néven**: termékfüggő architektúra, kötözés és alávetettség
- **Leggyakoribb előfordulási szint**: rendszer
- **Újragyártott megoldás neve**: izolációs réteg
- **Újragyártott megoldás típusa**: szoftver
- **Kiváltó okok**: lustaság, fásultság, büszkeség/tudatlanság (naivság)
- **Kiegyensúlyozatlan erők**: a technológia-transzfer kezelése, a változás kezelése

Szoftver architekturális antiminta: gyártótól függés (2)

- **Anekdotaszerű példa**: Gyakran találkozunk olyan szoftverprojektekkel, melyek azt állítják, hogy architektúrájuk egy adott gyártón vagy termékvonalon alapul. Más anekdotaszerű példák a termékfrissítések és az új alkalmazástelepítések idején történnek.
  - „Amikor megpróbálom beolvasni az új adatállományokat az alkalmazás régi verziójába, lefagy a rendszer.”
  - „Architektúránk a… Megismételnéd, hogy mi is az adatbázisunk neve?”
- **Általános alak**: Egy szoftverprojekt egy terméktechnológiát alkalmaz és teljesen függővé válik a gyártó megvalósításától. Amikor frissítések történnek, a szoftver változik és együttműködési problémák jelentkeznek, folyamatos karbantartás szükséges a rendszer működésben tartásához.

Szoftver architekturális antiminta: gyártótól függés (3)

- **Tünetek és következmények**:
  - Az alkalmazói szoftver karbantartási folyamat motorja az üzleti termékek frissítése.
  - A beígért új termékfunkciók bevezetése késik, vagy soha nem történik meg.
  - A termék lényegesen eltér a hirdetett nyílt szabványoktól.
  - Ha teljesen kimarad egy termékfrissítés, gyakran újra meg kell venni a terméket és újra kell integrálni.
- **Tipikus okok**:
  - A termék eltér a közzétett nyílt szabványoktól, mivel nincs a szabványhoz tényleges megfelelőség vizsgálati folyamat.
  - A termék választása teljesen marketing információkon alapul, nem pedig részletes műszaki vizsgálaton.
  - Nem áll rendelkezésre műszaki megközelítés az alkalmazói szoftver a terméktől való olyan elkülönítésére, hogy ne függjön tőle közvetlenül.
  - A termék technológiájának bonyolultsága és általánossága messze meghaladja az alkalmazás igényeit. A terméktől való közvetlen függés miatt kudarcba fullad az alkalmazás rendszer-architektúra bonyolultságának kezelése.

Szoftver architekturális antiminta: gyártótól függés (4)

- **Ismert kivételek**: Elfogadható az antiminta akkor, ha egyetlen gyártó kódja alkotja az alkalmazáshoz szükséges kód nagy részét.
- **Újragyártott megoldás**: A megoldás egy olyan szoftverréteg létrehozásával jár, mely az alapul szolgáló infrastruktúrát vagy termékfüggő szoftverinterfészeket absztrahálja. A réteg egy olyan alkalmazói interfészt biztosít, mely teljesen szétválasztja az alkalmazói szoftvereket az alapul szolgáló interfészektől.
- **Példa**: