Minták a szoftverfejlesztésben

Jeszenszky Péter
Debreceni Egyetem, Informatikai Kar
jeszenszky.peter@inf.unideb.hu

Utolsó módosítás: 2023. április 21.

Készült Kollár Lajos korábbi anyagának felhasználásával

Minták

- A fogalom Christopher Alexander (1936–2022) 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.
    - Az idézet 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 Professional, 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.”*
    - Scott W. Ambler. *Process Patterns: Building Large-Scale Systems Using Object Technology*. Cambridge University Press, 1998. https://scottambler.com/process-patterns/

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 (a*rchitectural patterns/styles*)
- Tervezési minták (*design patterns*)
- Programozási idiómák/implementációs minták (*programming idioms/implementation patterns*)
- Tesztelési minták (*test patterns*)
- Felhasználói felület tervezési minták (*user interface design patterns*)
- Antiminták/ellenminták (*antipatterns*)
- …

Architekturális minták (1)

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

Architekturális minták (2)

- 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, …

Példa architekturális mintára: MVC (1)

- Az MVC-t 1978-ban Trygve Reenskaug, egy norvég számítógéptudós (Oslói Egyetem) dolgozta ki.
  - Történeti háttér: https://folk.universitetetioslo.no/trygver/themes/mvc/mvc-index.html

Példa architekturális mintára: MVC (2)

- **Név**: Modell-nézet vezérlő (*Model-View-Controller*)
- **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.

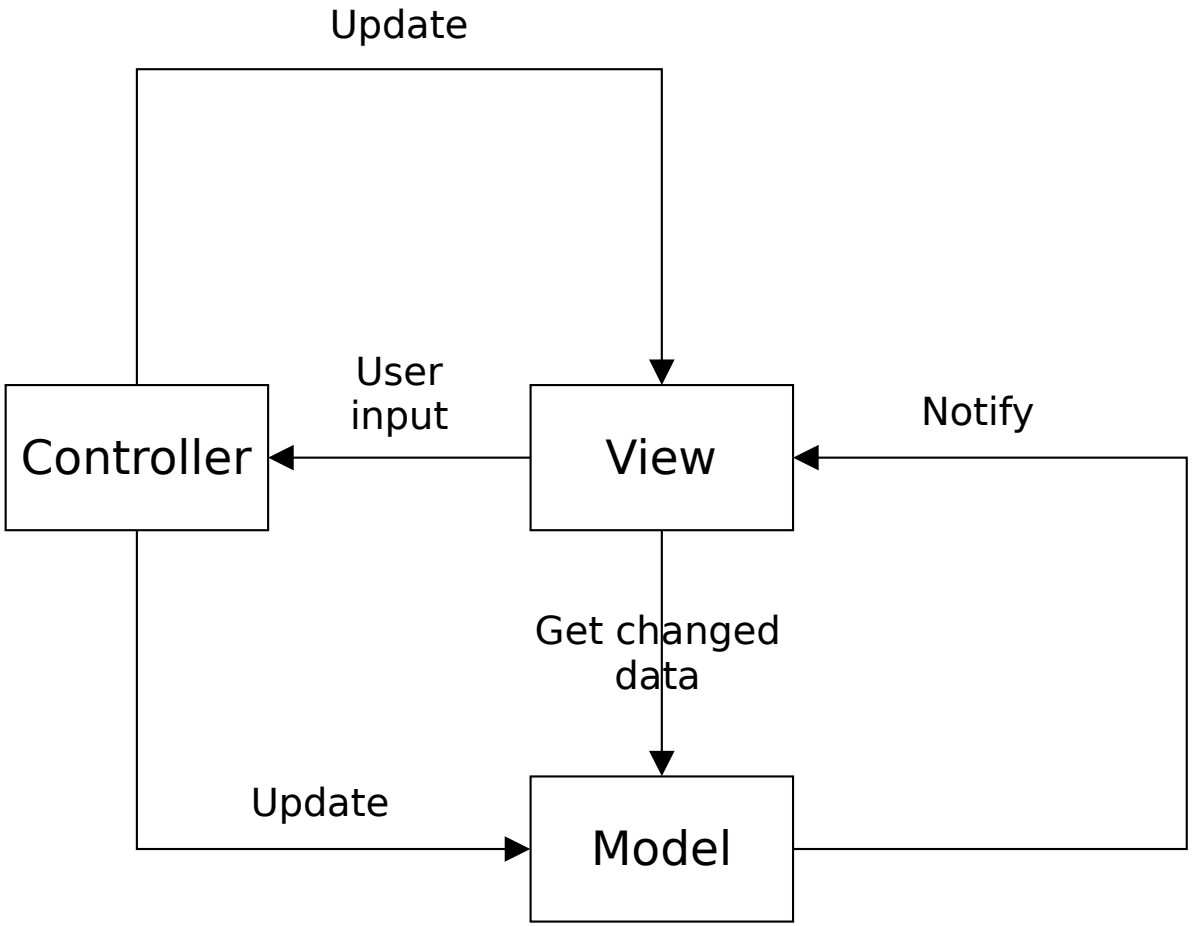
Példa architekturális mintára: MVC (3)

- 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 (4)

- 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 (5)



Példa architekturális mintára: MVC (6)

- Változatok:
  - Hierarchikus modell-nézet-vezérlő (HMVC)
  - Model-view-presenter (MVP)
  - Model-view-viewmodel (MVVM)

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 helyzettől független nézőpontból fejeznek ki tervezési módszereket.

Tervezési minták (1)

- Felhasznált irodalom:
  - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 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 (2)

- 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 (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 Professional, 2004.
    - Vaskaran Sarcar. *Design Patterns in C#: A Hands-on Guide with Real-world Examples*. 2nd ed. Apress, 2020. https://github.com/Apress/design-patterns-csharp-2e
    - *.NET Design Patterns in C#* https://www.dofactory.com/net/design-patterns
  - **C++**:
    - Andrei Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison-Wesley Professional, 2001.
    - Fedor G. Pikus. *Hands-On Design Patterns with C++*. Packt Publishing, 2019. https://github.com/PacktPublishing/Hands-On-Design-Patterns-with-CPP
  - **Groovy**:
    - *Design patterns in Groovy* https://groovy-lang.org/design-patterns.html

Tervezési minták (5)

- Nyelvspecifikus ajánlott irodalom (folytatás):
  - **Java**:
    - Vaskaran Sarcar. *Java Design Patterns*. 2nd ed. Apress, 2019. https://github.com/Apress/java-design-patterns-2e
    - Ilkka Seppälä. *Design patterns implemented in Java*. https://java-design-patterns.com/ https://github.com/iluwatar/java-design-patterns
  - **JavaScript**:
    - Addy Osmani. *Learning JavaScript Design Patterns*. O'Reilly Media, 2012. https://github.com/addyosmani/essential-js-design-patterns
  - **Python**:
    - Bruce Eckel et al. *Python 3 Patterns, Recipes and Idioms*. https://python-3-patterns-idioms-test.readthedocs.io/en/latest/
    - Sakis Kasampalis. *A collection of design patterns/idioms in Python*. https://github.com/faif/python-patterns
    - Brandon Rhodes. *Python Design Patterns*. https://python-patterns.guide/ https://github.com/brandon-rhodes/python-patterns

Tervezési minták (7)

- Nyelvspecifikus ajánlott irodalom (folytatás):
  - **Scala**:
    - Ivan Nikolov. *Scala Design Patterns*. 2nd ed. Packt Publishing, 2018. https://github.com/nikolovivan/scala-design-patterns
  - **C#**, **C++**, **Go**, **Java**, **PHP**, **Python**, **Ruby**, **Swift**, **TypeScript**:
    - Alexander Shvets. *Dive Into Design Patterns*. https://sourcemaking.com/design-patterns-ebook
    - Refactoring.Guru. *Design Patterns*. https://refactoring.guru/design-patterns https://github.com/RefactoringGuru

Tervezési minták (8)

- A minták osztályozása céljuk szerint (GoF):
  - **Létrehozási minták** (***creational patterns***): az objektumok létrehozásával foglalkoznak.
  - **Szerkezeti minták** (***structural patterns***): azzal foglalkoznak, hogy hogyan alkotnak osztályok és objektumok nagyobb szerkezeteket.
  - **Viselkedési minták** (***behavioral patterns***): az osztályok vagy objektumok egymásra hatását valamint a felelősségek elosztását írják le.

Tervezési minták (9)

- A GoF könyv 23 tervezési mintát ír le, időközben számos további tervezési minta született.
  - Például: függőség befecskendezés (dependency injection), tároló (*repository*), többke (*multiton*), iker (*twin*), folyékony interfész (fluent interface), …
- Új kategória: **Konkurencia minták** (***Concurrency Patterns***)
  - Például: aktív objektum (a*ctive object*), őrzött felfüggesztés (*guarded suspension*), szálkészlet (t*hread pool*), …

Tervezési minták (10)

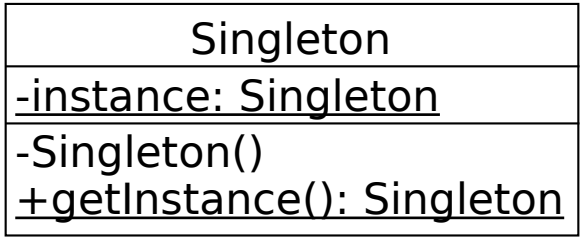
- Mintasablon (GoF):
  - **Név és besorolás** (***Name and Classification***):
  - **Cél** (***Intent***):
  - **Más néven** (***Also Known As***):
  - **Indíték** (***Motivation***):
  - **Alkalmazhatóság** (***Applicability***):
  - **Szerkezet** (***Structure***):
  - **Résztvevők** (***Participants***):
  - **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 esetben használjuk:
  - Pontosan egy példányra van szükség valamelyik osztályból, és annak elérhetőnek kell lennie az ügyfelek számára a jól ismert elérési pontokból.

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() {
  } // privát láthatóságú konstruktor!

  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*. https://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 Professional, 2006. https://jcip.net/
      - A *The Java Memory Model* című 16. fejezet 16.2 alfejezete tárgyalja a szálbiztos inicializálást.

Programozási idiómák/ Implementációs minták (1)

- Felhasznált irodalom:
  - Kent Beck. *Implementation Patterns*. Addison-Wesley Professional, 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 (2)

- 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 (3)

- Nyelvspecifikus ajánlott irodalom:
  - **C#**:
    - Bill Wagner. *Effective C#: 50 Specific Ways to Improve Your C#*. 3rd ed. Addison-Wesley Professional, 2016.
    - Bill Wagner. *More Effective C#: 50 Specific Ways to Improve Your C#*. 2nd ed. Addison-Wesley Professional, 2008.
  - **C++**:
    - Scott Meyers. *Effective C++: 55 Specific Ways to Improve Your Programs and Designs*. 3rd ed. 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*. 3rd ed. Addison-Wesley Professional, 2017. https://github.com/jbloch/effective-java-3e-source-code
  - **Python**:
    - Brett Slatkin. *Effective Python: 90 Specific Ways to Write Better Python*. 2nd ed. Addison-Wesley Professional, 2019. https://effectivepython.com/ https://github.com/bslatkin/effectivepython
    - Luciano Ramalho. *Fluent Python*. 2nd ed. O'Reilly Media, 2021. https://github.com/fluentpython/example-code-2e

Programozási idiómák/ Implementációs minták (4)

- Példa:

```
import java.util.Objects;

public final class Movie implements Cloneable {
  private String title;
  private int year;

  public Movie(String title, int year) {
    if (year < 1878) {
      throw new IllegalArgumentException();
    }
    this.title = title;
    this.year = year;
  }

  @Override
  public boolean equals(Object o) {
    if (o == this) {
      return true;
    }
    if (!(o instanceof Movie)) {
      return false;
    }
    Movie m = (Movie) o;
    return Objects.equals(title, m.title) && year == m.year;
  }

  // Error: no hashCode() and clone() methods are provided!
}
```

Programozási idiómák/ Implementációs minták (5)

- Példa (folytatás): mintaillesztés az `instanceof` operátorhoz (Java SE 16)

```
import java.util.Objects;

public final class Movie implements Cloneable {
  private String title;
  private int year;

  public Movie(String title, int year) {
    if (year < 1878) {
      throw new IllegalArgumentException();
    }
    this.title = title;
    this.year = year;
  }

  @Override
  public boolean equals(Object o) {
    if (o == this) {
      return true;
    }
    return (o instanceof Movie m) && Objects.equals(title, m.title) && year == m.year;
  }

  // Error: no hashCode() and clone() methods are provided!
}
```

Programozási idiómák/ Implementációs minták (6)

- Példa (folytatás):

```
var set = new HashSet<>();
var movie = new Movie("Shrek", 2001);
set.add(movie);

System.out.println(set.contains(movie));                    // true
System.out.println(set.contains(new Movie("Shrek", 2001)));// false
System.out.println(set.contains(movie.clone()));            // compile error
```

Programozási idiómák/ Implementációs minták (7)

- Példa (folytatás):

```
public final class Movie implements Cloneable {

  // ...

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

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

}
```

Tesztelési minták (1)

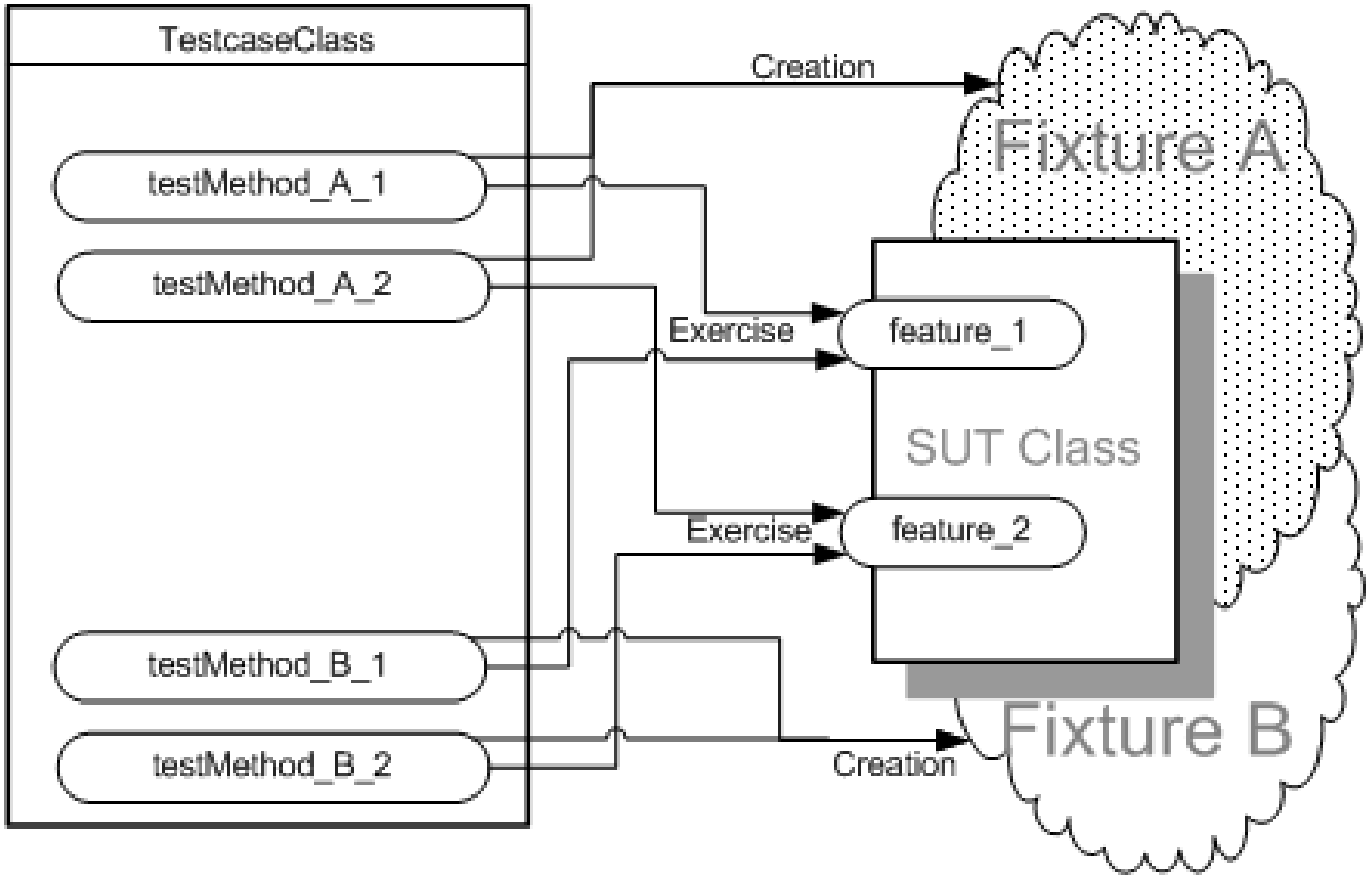
- Felhasznált irodalom:
  - Gerard Meszaros. *xUnit Test Patterns: Refactoring Test Code*. Addison-Wesley Professional, 2007. http://xunitpatterns.com/
    - A könyv egy mintanyelvet ad automatizált tesztek az xUnit családba tartózó egységteszt keretrendszerekkel történő írásához.

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

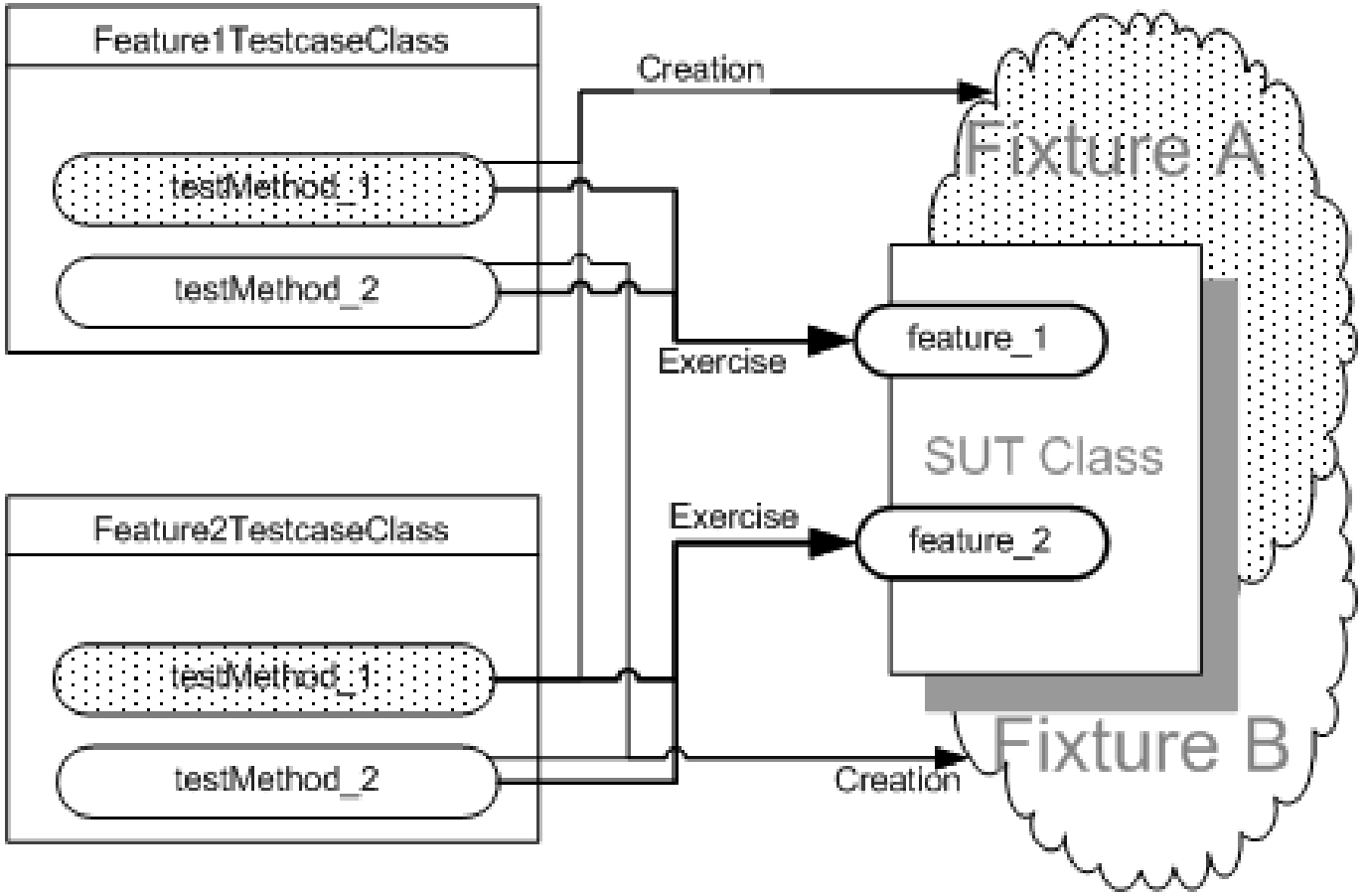


Tesztelési minták (4)

- **Képességenkénti teszteset osztály** (***Testcase Class per Feature***) (624. oldal):
  - Hogyan szervezzük tesztmetódusainkat teszteset osztályokba?
    - A tesztmetódusokat teszteset osztályokba csoportosítjuk az alapján, hogy a tesztelt rendszer melyik tesztelhető képességét mozgatják meg.
  - Mikor használjuk?
    - Amikor jelentős számú tesztmetódus van és nyilvánvalóbbá akarjuk tenni a tesztelt rendszer minden egyes képességének specifikációját.
    - Sajnos ez nem teszi a tesztmetódusokat egyszerűbbé vagy könnyebben érthetővé.

Tesztelési minták (5)

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



Felhasználói felület tervezési minták (1)

- Felhasznált irodalom:
  - Jenifer Tidwell. *Designing Interfaces: Patterns for Effective Interaction Design*. 2nd ed. O'Reilly Media, 2010.
  - Jenifer Tidwell, Charles Brewer, Aynne Valencia. *Designing Interfaces: Patterns for Effective Interaction Design*. 3rd ed. O'Reilly Media, 2020.
  - Christian Crumlish, Erin Malone. *Designing Social Interfaces: Principles, Patterns, and Practices for Improving the User Experience*. 2nd ed. O'Reilly Media, 2015.

Felhasználói felület tervezési minták (2)

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

Felhasználói felület tervezési minták (3)

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

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://web.archive.org/web/20190430234207/https://www.mozilla.org/hu/firefox/new/
    - https://web.archive.org/web/20190901005650/https://www.mozilla.org/hu/firefox/new/

Antiminták (1)

- Felhasznált irodalom:
  - 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 ed. Auerbach, 2011.

Antiminták (2)

- Az antiminta kifejezést Andrew Koenig alkotta meg:
  - „Egy antiminta pont olyan, mint egy minta, kivéve azt, hogy megoldás helyet valami olyat ad, ami látszólag megoldásnak néz ki, de nem az.”
    - Lásd: Andrew Koenig. *Patterns and Antipatterns*. Journal of Object-Oriented Programming 8.1 (2005), p. 46–48.
  - Idézve: Martin Fowler. *AntiPattern*. 25 August 2015. https://martinfowler.com/bliki/AntiPattern.html
- Egy problémára adott általánosan előforduló megoldások, melyek kifejezetten negatív következményekkel járnak.
- Bármely szinten megjelenhetnek.

Antiminták (3)

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

Antiminták (4)

- Egy nagyon hasonló fogalom a tervezési szag (*design smell*).
  - A tervezési szagok olyan struktúrák a tervezésben, melyek alapvető tervezési elvek megsértését jelzik és negatív hatással vannak a tervezés minőségére.
  - Irodalom:
    - Girish Suryanarayana, Ganesh Samarthyam, Tushar Sharma. *Refactoring for Software Design Smells: Managing Technical Debt*. Morgan Kaufmann, 2014.

Antiminta sablon (1)

- **Antiminta neve** (***Anti-Pattern 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.

Antiminta sablon (3)

- **Általános alak (*General Form*):** Az antiminta általános leírása, mely gyakran tartalmaz ábrát is. Nem egy példa, hanem egy általános változat.
- **Tünetek és következmények** (***Symptoms and Consequences***): Az antiminta tüneteinek és az antiminta által 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.

Antiminta sablon (4)

- **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.
- **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.
- **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.

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) https://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.
- **Alkalmazhatóság más nézőpontokra és szintekre**: Az architekturális és vezetői nézőpontok is kulcsszerepet játszanak a massza antiminta megelőzésében.

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 jelenik 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.
- **Kapcsolódó megoldások**: analízis-paralízis, lávafolyás