

LAB Tema 2- Mantenimiento del Software: 2.2 Refactorización Software

Contenido

LAB Tema 2- Mantenimiento del Software: 2.2 Refactorización Software	1
Refactorizaciones llevadas acabo por Iñaxio Inda Araguás	1
"Write short units of code" (capítulo 2)	1
"Write short units of code" (capítulo 2)	5
"Duplicate code" (capítulo 4).	5
"Keep unit interfaces small " (capítulo 5)	6

Refactorizaciones llevadas acabo por Iñaxio Inda Araguás

Miembro: Iñaxio Inda Araguás

"Write short units of code" (capítulo 2)

Guideline:

- Limit the length of code units to 15 lines of code.
- Do this by not writing units that are longer than 15 lines of code in the first place, or by splitting long units into multiple smaller units until each unit has at most 15 lines of code.
- This improves maintainability because small units easy to understand, easy to test, and easy to reuse.

1ª refactorización

Codigo inicial

```
public boolean gertaeraEzabatu(Event ev) {
    Event event = db.find(Event.class, ev);
    boolean resultB = true;
    List<Question> listQ = event.getQuestions();

    for(Question q : listQ) {
        if(q.getResult() == null) {
            resultB = false;
        }
    }
    if(!resultB) {
        return false;
    } else if(new Date().compareTo(event.getEventDate())<0) {
        TypedQuery<Quote> qquery = db.createQuery("SELECT q FROM
Quote q WHERE q.getQuestion().getEvent().getEventNumber() =?1", Quote.class);
```

```

        qqquery.setParameter(1, event.getEventNumber());
        List<Quote> listQUO = qqquery.getResultList();
        for(int j=0; j<listQUO.size(); j++) {
            Quote quo = db.find(Quote.class, listQUO.get(j));
            for(int i=0; i<quo.getApustuak().size(); i++) {
                ApustuAnitza apustuAnitza =
quo.getApustuak().get(i).getApustuAnitza();
                ApustuAnitza ap1 =
db.find(ApustuAnitza.class, apustuAnitza.getApustuAnitzaNumber());
                db.getTransaction().begin();
                ap1.removeApustua(quo.getApustuak().get(i));
                db.getTransaction().commit();

                if(ap1.getApustuak().isEmpty() &&
!ap1.getEgoera().equals("galduta")) {
                    this.apustuaEzabatu(ap1.getUser(),
ap1);

                }else if(!ap1.getApustuak().isEmpty() &&
ap1.irabazitaMarkatu()){
                    this.ApustuaIrabazi(ap1);
                }
                db.getTransaction().begin();
                Sport spo
=quo.getQuestion().getEvent().getSport();
                spo.setApustuKantitatea(spo.getApustuKantitatea()-1);
                db.getTransaction().commit();
            }
        }

        db.getTransaction().begin();
        db.remove(event);
        db.getTransaction().commit();
        return true;
    }
}

```

Codigo final

```

public boolean gertaeraEzabatu(Event ev) {
    Event event = db.find(Event.class, ev);
    if (!validarResultados(event)) {
        return false;
    }

    handleApustuak(event);

    db.getTransaction().begin();
    db.remove(event);
    db.getTransaction().commit();

    return true;
}

private boolean validarResultados(Event event) {
    List<Question> listQ = event.getQuestions();
}

```

```

        for (Question q : listQ) {
            if (q.getResult() == null) {
                return false;
            }
        }

        return true;
    }

    private void handleApustuak(Event event) {
        TypedQuery<Quote> qquery = db.createQuery("SELECT q FROM Quote q
WHERE q.getQuestion().getEvent().getEventNumber() =?1", Quote.class);
        qquery.setParameter(1, event.getEventNumber());
        List<Quote> listQUO = qquery.getResultList();

        for (Quote quo : listQUO) {
            handleApuestaIndividual(quo);
        }
    }

    private void handleApuestaIndividual(Quote quo) {
        for (Apustua apustuak : quo.getApustuak()) {
            ApustuAnitza apustuAnitza = apustuak.getApustuAnitza();
            ApustuAnitza ap1 = db.find(ApustuAnitza.class,
apustuAnitza.getApustuAnitzaNumber());

            db.getTransaction().begin();
            ap1.removeApustua(apustuak);
            db.getTransaction().commit();

            Collection<Event> collection = (Collection<Event>) apustuak;
            if (collection.isEmpty() || ap1.irabazitaMarkatu()) {
                handleApustua(ap1);
            }

            updateSportCount(quo);
        }
    }

    private void handleApustua(ApustuAnitza apustuAnitza) {
        db.getTransaction().begin();
        apustuAnitza.setEgoera("galduta");
        db.getTransaction().commit();

        apustuaEzabatu(apustuAnitza.getUser(), apustuAnitza);
    }

    private void updateSportCount(Quote quo) {
        db.getTransaction().begin();
        Sport sport = quo.getQuestion().getEvent().getSport();
        sport.setApustuKantitatea(sport.getApustuKantitatea() - 1);
        db.getTransaction().commit();
    }
}

```

Descripción del error concreto detectado y descripción de la refactorización realizada

He dividido la función original en funciones más pequeñas para mejorar la legibilidad y mantenibilidad del código. Cada función tiene como máximo 15 líneas de código

`public boolean gertaeraEzabatu(Event ev)`: Este es el método principal que toma un objeto Event como argumento y tiene la responsabilidad de eliminar un evento si ciertas condiciones se cumplen.

`private boolean validarResultados(Event event)`: Esta función toma un objeto Event como entrada y verifica si todas las preguntas dentro del evento tienen resultados no nulos. Realiza lo siguiente:

Obtiene la lista de preguntas del evento.

Itera a través de estas preguntas y verifica si alguna de ellas tiene un resultado nulo.

Si encuentra una pregunta sin resultado, retorna false, lo que indica que no se pueden eliminar eventos con preguntas sin resultados. De lo contrario, retorna true.

`private void handleApustuak(Event event)`: Esta función toma un objeto Event como entrada y se encarga de manejar las apuestas relacionadas con ese evento. Hace lo siguiente:

Realiza una consulta en la base de datos para obtener una lista de objetos Quote relacionados con el evento.

Itera a través de la lista de cotizaciones y llama a la función `handleIndividualQuote(quo)` para manejar cada una de ellas.

`private void handleIndividualQuote(Quote quo)`: se encarga de manejar las apuestas individuales. Realiza lo siguiente:

Itera a través de la lista de apuestas (`Apustuak`)

Para cada apuesta, obtiene el objeto `ApustuAnitza` asociado y lo busca en la base de datos.

Inicia una transacción en la base de datos y elimina la apuesta (`apustuak`) del objeto `ApustuAnitza`.

Luego, verifica si la apuesta está vacía y no se ha perdido o si ha ganado. Si es cierto, llama a `handleApustua(ap1)`.

Finalmente, llama a `updateSportCount(quo)` para actualizar el contador de deportes.

`private void handleApustua(ApustuAnitza apustuAnitza)`: Esta función toma un objeto `ApustuAnitza` como entrada y se encarga de actualizar a "galduta" (perdida).

Llama a `apustuaEzabatu` para eliminar la apuesta de un usuario específico.

Finaliza la transacción en la base de datos.

`private void updateSportCount(Quote quo)`: se encarga de actualizar el contador de apuestas

"Write short units of code" (capítulo 2)

Guideline:

- Limit the length of code units to 15 lines of code.
- Do this by not writing units that are longer than 15 lines of code in the first place, or by splitting long units into multiple smaller units until each unit has at most 15 lines of code.
- This improves maintainability because small units easy to understand, easy to test, and easy to reuse.

2º refactorización

La función que antes hemos refacorizado nos incumple la regla

Codigo inicial

```
public boolean gertaeraEzabatu(Event ev) {
    Event event = db.find(Event.class, ev);
    if (!validarResultados(event)) { // Punto de bifurcación 1
        return false;
    }

    handleApustuak(event); // Punto de bifurcación 2

    db.getTransaction().begin(); // Punto de bifurcación 3
    db.remove(event); // Punto de bifurcación 4
    db.getTransaction().commit(); // Punto de bifurcación 5

    return true;
}
```

De forma que para que no la incumpla es tan sencillo como modificar el 1º if

```
public boolean gertaeraEzabatu(Event ev) {
    Event event = db.find(Event.class, ev);

    if (validarResultados(event)) {
        handleApustuak(event);
        db.getTransaction().begin();
        db.remove(event);
        db.getTransaction().commit();
        return true;
    }

    return false;
}
```

De forma que pasamos de 5 bifurcaciones a 2

"Duplicate code" (capítulo 4).

SonarLint y sonarcloud nos indica dónde hay duplicidad de código.

Guideline:

- Do not copy code.
- Do this by writing reusable, generic code and/or calling existing methods instead.
- This improves maintainability because when code is copied, bugs need to be fixed at multiple places, which is inefficient and error-prone.

Codigo inicial

3º Refactorizacion

```
private void handleApustua(ApustuAnitza apustuAnitza) {
    db.getTransaction().begin();
    apustuAnitza.setEgoera("galduta");
    db.getTransaction().commit();

    apustuaEzabatu(apustuAnitza.getUser(), apustuAnitza);
}
```

El código que maneja la transacción y la eliminación de un apustua es similar en múltiples lugares.

Codigo final

```
private void handleApustua(ApustuAnitza apustuAnitza) {
    db.getTransaction().begin();
    try {
        apustuAnitza.setEgoera("galduta");
        apustuaEzabatu(apustuAnitza.getUser(), apustuAnitza);
    } finally {
        db.getTransaction().commit();
    }
}
```

Al usar un bloque try-finally la transacción se realiza y detecta cualquier error. Esto simplifica el código y elimina la duplicación innecesaria de la lógica de transacción.

"Keep unit interfaces small " (capítulo 5) .

También sonarcloud nos ayuda a localizar los lugares con código repetido.

Guideline:

- Limit the number of parameters per unit to at most 4.
- Do this by extracting parameters into objects.
- This improves maintainability because keeping the number of parameters low makes units easier to understand and reuse.

En el DataAcces no hay métodos con mas de 4 parámetros, pero hay métodos que se pueden simplificar, como es el caso de storeRegistered. El cual al crear una clase supletoria que se encargue de los detalles de los Registrados,

Codigo inicial

```
public void storeRegistered(String username, String password, Integer
bankAccount) {
    db.getTransaction().begin();
    Registered ad = new Registered(username, password, bankAccount);
    db.persist(ad);
    db.getTransaction().commit();
}
```

Codigo final:

En DataAccess.java

```
public void storeRegistered(RegisteredDetails registeredDetails) {
    db.getTransaction().begin();
    Registered ad = new Registered(registeredDetails.getUsername(),
registeredDetails.getPassword(), registeredDetails.getBankAccount());
    db.persist(ad);
    db.getTransaction().commit();
}
```

RegisteredDetails.java

```
package domain;
```

```
public class RegisteredDetails {
    private String username;
    private String password;
    private Integer bankAccount;

    public RegisteredDetails(String username, String password, Integer
bankAccount) {
        this.username = username;
        this.password = password;
        this.bankAccount = bankAccount;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Integer getBankAccount() {
        return bankAccount;
    }
}
```

```

    public void setBankAccount(Integer bankAccount) {
        this.bankAccount = bankAccount;
    }
}

```

En BLFacadeImplementation

```

@WebMethod
public void storeRegistered(String username, String password, Integer
bankAccount) {
    dbManager.open(false);
    dbManager.storeRegistered(username, password, bankAccount);
    dbManager.close();
}

```

En RankingLortuDABTest

Codigo Inicial

```

public void testRankingLortuUnicoRegistro() {
    List<Registered> unicoRegistro = new ArrayList<Registered>();

    Registered user1 = mock(Registered.class);
    when(user1.getUsername()).thenReturn("user1");
    when(user1.getPassword()).thenReturn("psswr1");
    when(user1.getBankAccount()).thenReturn(10123);

    String name = user1.getUsername();
    String password = user1.getPassword();
    Integer bankAccount = user1.getBankAccount();

    unicoRegistro.add(user1);

    Mockito.doReturn(unicoRegistro).when(dbManager).rankingLortu();

    BLFacadeImplementation facade = new
BLFacadeImplementation(dbManager);
    facade.storeRegistered(name, password, bankAccount);
    List<Registered> result = facade.rankingLortu();
    assertEquals(unicoRegistro, result);
}

```

Codigo Final

```

public void testRankingLortuUnicoRegistro() {
    List<Registered> unicoRegistro = new ArrayList<Registered>();

    Registered user1 = mock(Registered.class);
    when(user1.getUsername()).thenReturn("user1");
    when(user1.getPassword()).thenReturn("psswr1");
    when(user1.getBankAccount()).thenReturn(10123);

    String name = user1.getUsername();
    String password = user1.getPassword();
    Integer bankAccount = user1.getBankAccount();
}

```



```

        unicoRegistro.add(user1);
        RegisteredDetails rd = new RegisteredDetails(name, password,
bankAccount);
        Mockito.doReturn(unicoRegistro).when(dbManager).rankingLortu();

        BLFacadeImplementation facade = new
BLFacadeImplementation(dbManager);
        facade.storeRegistered(rd);
        List<Registered> result = facade.rankingLortu();
        assertEquals(unicoRegistro, result);
    }

```

En RegisterGUI.java la función `actionPerformed`

LAB REFACTORIZACIÓN

JOSE MARIA GIMENO

Bad smell 2: "Write simple units of code"

Número máximo de ramificaciones ≤ 4 .

 Refactor this method to reduce its Cognitive Complexity from 17 to the 15 allowed. [+10 locations]

Código inicial:

```
977 public boolean gertaeraEzabatu(Event ev) {
978     Event event = db.find(Event.class, ev);
979     boolean resultB = true;
980     List<Question> listQ = event.getQuestions();
981
982     for(Question q : listQ) {
983         if(q.getResult() == null) {
984             resultB = false;
985         }
986     }
987     if(!resultB) {
988         return false;
989     }else if(new Date().compareTo(event.getEventDate())<0) {
990         TypedQuery<Quote> qquery = db.createQuery("SELECT q FROM Quote q WHERE q.getQuestion().getEvent().getEventNumber()=?1", Quote.class);
991         qquery.setParameter(1, event.getEventNumber());
992         List<Quote> listQUO = qquery.getResultList();
993         for(int j=0; j<listQUO.size(); j++) {
994             Quote quo = db.find(Quote.class, listQUO.get(j));
995             for(int i=0; i<quo.getApustuak().size(); i++) {
996                 ApustuAnitza apustuAnitza = quo.getApustuak().get(i).getApustuAnitza();
997                 ApustuAnitza apl = db.find(ApustuAnitza.class, apustuAnitza.getApustuAnitzaNumber());
998                 db.getTransaction().begin();
999                 apl.removeApustua(quo.getApustuak().get(i));
1000                 db.getTransaction().commit();
1001                 if(apl.getApustuak().isEmpty() && !apl.getEgoera().equals("galduta")) {
1002                     this.apustuaEzabatu(apl.getUser(), apl);
1003                 }else if(!apl.getApustuak().isEmpty() && apl.irabazitaMarkatu()){
1004                     this.ApustuaIrabazi(apl);
1005                 }
1006                 db.getTransaction().begin();
1007                 Sport spo =quo.getQuestion().getEvent().getSport();
1008                 spo.setApustuKantitatea(spo.getApustuKantitatea()-1);
1009                 db.getTransaction().commit();
1010             }
1011         }
1012     }
1013     db.getTransaction().begin();
1014     db.remove(event);
1015     db.getTransaction().commit();
1016     return true;
1017 }
1018 }
```

Código refactorizado:

```
984 public boolean gertaeraEzabatu(Event ev) {
985     Event event = db.find(Event.class, ev);
986     boolean resultB = true;
987     List<Question> listQ = event.getQuestions();
988
989     resultB = questionList(resultB, listQ);
990     if(!resultB) {
991         return false;
992     }else if(new Date().compareTo(event.getEventDate())<0) {
993         TypedQuery<Quote> qquery = db.createQuery("SELECT q FROM Quote q WHERE q.getQuestionior
994         qquery.setParameter(1, event.getEventNumber());
995         List<Quote> listQUO = qquery.getResultList();
996         for(int j=0; j<listQUO.size(); j++) {
997             Quote quo = db.find(Quote.class, listQUO.get(j));
998             bucle(quo);
999         }
1000     }
1001     db.getTransaction().begin();
1002     db.remove(event);
1003     db.getTransaction().commit();
1004     return true;
1005 }
1006
1007 private void bucle(Quote quo) {
1008     for(int i=0; i<quo.getApustuak().size(); i++) {
1009         ApustuAnitza apustuAnitza = quo.getApustuak().get(i).getApustuAnitza();
1010         ApustuAnitza apl = db.find(ApustuAnitza.class, apustuAnitza.getApustuAnitzaNumber());
1011         db.getTransaction().begin();
1012         apl.removeApustua(quo.getApustuak().get(i));
1013         db.getTransaction().commit();
1014         if(apl.getApustuak().isEmpty() && !apl.getEgoera().equals("galduta")) {
1015             this.apustuaEzabatu(apl.getUser(), apl);
1016         }else if(!apl.getApustuak().isEmpty() && apl.irabazitaMarkatu()){
1017             this.ApustuaIrabazi(apl);
1018         }
1019         db.getTransaction().begin();
1020         Sport spo =quo.getQuestion().getEvent().getSport();
1021         spo.setApustuKantitatea(spo.getApustuKantitatea()-1);
1022         db.getTransaction().commit();
1023     }
1024 }
1025
1026 private boolean questionList(boolean resultB, List<Question> listQ) {
1027     for(Question q : listQ) {
1028         if(q.getResult() == null) {
1029             resultB = false;
1030         }
1031     }
1032     return resultB;
1033 }
1034 }
```

Como se puede observar, he refactorizado para obtener dos métodos más y así reducir el número de ramificaciones. De esta manera, desde el método principal, se hace u a llamada a estos dos métodos para obtener la información necesaria, pero de mejor manera que ya se ramifica menos el código principal.

Bad smell 1: "Write short units of code"

Límite de líneas de código = 15.

Para solucionar este bad smell, me remito al código anterior, ya que con la creación de los métodos vistos anteriormente, reducimos el número de líneas de código del método principal (gertaeraEzabatu) que era de 32, a 15 que tiene después de la refactorización.

Los otros métodos creados, bucle y QuestionList, tienen 4 y 14 líneas de código respectivamente.

Bad smell 3: "Duplicate code"

No copiar código.

Para solucionar este bad smell, se ha creado un String con la palabra ApustuaEgin para que sirva de parámetro de entrada en "Transaction".

Código inicial:

```
Transaction t1 = new Transaction(reg1, apA1.getBalioa(), new Date(), "ApustuaEgin");
Transaction t3 = new Transaction(reg2, apA4.getBalioa(), new Date(), "ApustuaEgin");
Transaction t4 = new Transaction(reg3, apA5.getBalioa(), new Date(), "ApustuaEgin");
Transaction t5 = new Transaction(reg4, apA3.getBalioa(), new Date(), "ApustuaEgin");
Transaction t6 = new Transaction(reg4, apA6.getBalioa(), new Date(), "ApustuaEgin");
Transaction t7 = new Transaction(reg1, apA7.getBalioa(), new Date(), "ApustuaEgin");
Transaction t8 = new Transaction(reg1, apA8.getBalioa(), new Date(), "ApustuaEgin");
Transaction t9 = new Transaction(reg2, apA9.getBalioa(), new Date(), "ApustuaEgin");
Transaction t10 = new Transaction(reg2, apA10.getBalioa(), new Date(), "ApustuaEgin");
Transaction t11 = new Transaction(reg3, apA11.getBalioa(), new Date(), "ApustuaEgin");
Transaction t12 = new Transaction(reg3, apA12.getBalioa(), new Date(), "ApustuaEgin");
```

Código refactorizado:

```
String a = "ApustuaEgin";
Transaction t1 = new Transaction(reg1, apA1.getBalioa(), new Date(), a);
Transaction t3 = new Transaction(reg2, apA4.getBalioa(), new Date(), a);
Transaction t4 = new Transaction(reg3, apA5.getBalioa(), new Date(), a);
Transaction t5 = new Transaction(reg4, apA3.getBalioa(), new Date(), a);
Transaction t6 = new Transaction(reg4, apA6.getBalioa(), new Date(), a);
Transaction t7 = new Transaction(reg1, apA7.getBalioa(), new Date(), a);
Transaction t8 = new Transaction(reg1, apA8.getBalioa(), new Date(), a);
Transaction t9 = new Transaction(reg2, apA9.getBalioa(), new Date(), a);
Transaction t10 = new Transaction(reg2, apA10.getBalioa(), new Date(), a);
Transaction t11 = new Transaction(reg3, apA11.getBalioa(), new Date(), a);
Transaction t12 = new Transaction(reg3, apA12.getBalioa(), new Date(), a);
```

Bad smell 4: "Keep unit interfaces small"

Se ha analizado todo el código de Bets22, y no hay métodos que necesiten más de 4 parámetros de entrada, por lo que no es necesario refactorizar el código para evitar este bad smell.