Arquiteturas de Software
Refactoring ESS Online Trading Platform

Rogério Gomes Lopes Moreira - A74634

Perfil de Engenharia de Sistemas de Software, Universidade do Minho

Resumo Este relatório serve como descrição dos procedimentos realizados para o trabalho prático número 3 de Arquiteturas de Software. Este projeto tem como objetivo um estudo aprofundado sobre code smells e técnicas de refactoring e a sua aplicabilidade.

1 Introdução

Neste projeto pretende-se fazer um estudo aprofundado sobre code smells, as respetivas técnicas de refactoring e a sua aplicabilidade. Como ponto de partida servem os trabalhos práticos da disciplina onde se pretendia desenvolver um Trader de CFDs. É assim esperado uma melhoria da estrutura interna após a aplicação das técnicas acima referidas. Este relatório serve como descrição dos procedimentos realizados durante o projeto, sendo que este está dividido em três partes distintas: uma primeira parte onde se descrevem os procedimentos referentes ao trabalho prático número 1, uma segunda parte onde se descrevem os procedimentos referentes ao trabalho prático número 2 e por fim, selecionou-se um anti-padrão não presente no código original e aplicou-se este anti-padrão.

As ferramentas usadas foram:

1. SourceMonitor (métricas estáticas)
2. Eclipse (IDE)
3. TheadMXBEan Library
4. nanoTime Library

Métricas dinâmicas

```
long startTimeNano = System.nanoTime( );
ThreadMXBean threadMXBean = ManagementFactory.
    getThreadMXBean();
long time = threadMXBean.getCurrentThreadCpuTime();
long taskTimeNano  = System.nanoTime( ) - startTimeNano;
```

2 Trabalho Prático 1

2.1 Code Smell detetados e respetivo Refactoring

Long Method

Listing 1.1. Original

```
public void
    portfolioUpdater(){
    /*42 linhas*/
    }, 5000,200000);
   }
}
```

Listing 1.2. Refactored

```
   public void
       atualizaUsers(){
      allUsers.entrySet().
          forEach((u) -> {
          it.forEach((c) ->
              {
          atualizaCFD(c);
         });
    });}

 public void
    portfolioUpdater(){
    atualizaPortfolio();
    }, 5000,200000);
   }
}
```

O refactoring aplicado foi Extract Method, dividiu-se o método original em três métodos distintos. Tornando o método original mais pequeno e portanto, de mais fácil leitura e compreensão.

Large Class

Verificou-se que as classes Trader continha muitos métodos e linhas de código. Portanto, identificou-se o code smell Large Class. O refactor usado para resolver o problema foi o **Extract Interface**, ou seja, criou-se uma interface a partir da classe Trader, pertimindo assim uma melhor compreensão da classe.

Switch Statements

Listing 1.3. Original

```
updater.start();
do{
menumain.executa();
switch(menumain.getOpcao()
    ){
case 1:
signIn();
break;
case 2:
signUp();
break;
}
}while (menumain.getOpcao
    ()!=0);
```

Listing 1.4. Refactored

```
public enum MenuInicial {
SIGNIN(1, TraderApp::
    signIn),
SIGNUP(2, TraderApp::
    signUp),
UNKNOWN(3, () ->
    doSomething());
private int value;
 private Runnable
     execution;
private MenuInicial(int
    val, Runnable toRun) {
     int value = val;
     execution = toRun;
}
public void execute() {
     execution.run();
}
static void execute(int
    code) {
     for (MenuInicial item
         : values()) {
          if (item.value ==
              code) {
               item.run();
                   break;
          }
     }
     throw new IAException(
         "unknown");
     }
};
```

Verificou-se que havia um code smell relacionado com **Switch Statements**. Para isto cria-se uma classe Enum para cada menu e o utilizador invoca a opção a que corresponde uma função.

Temporary Field Smell

Listing 1.5. Original

```
public class Trader
    implements
    Serializable{
private boolean logged =
    false;
private User loggedUser;
private Map<String,User>
    allUsers = new TreeMap
    <>();
```

Listing 1.6. Refactored

```
ppublic class Trader
    implements
    Serializable{
private User loggedUser;
private Map<String,User>
    allUsers = new TreeMap
    <>();
```

Na classe Trader era utilizada uma variável de instância "logged"que era apenas usada em alguns casos, não era crucial uma vez que a mesma informação estava disponível noutra variável de instância "loggedUser". Posto isto, foi decidido remover a variável e utilizar apenas a variável "loggedUser". Caso esta esteja a null então é porque não há nenhum utilizado logado, caso contrário há.

Comments

Documentou-se as maiores classes TraderApp e Trader já que não apresentavam comentários a nenhuma função. Teve-se também em conta a relevância dos comentários em cada função, garantindo que realmente acrescentariam informação.

2.2 Métricas antes e depois do refactoring

Para que se possa perceber quais as implicações do refactoring no código foram medidas uma série de itens antes e depois do processo. Existem métricas estáticas (ex: número de linhas de uma classe) e métricas dinâmicas, medidas em tempo de execução (ex: tempo para o utilizador executar x tarefas). As tarefas para as métricas dinâmicas são as seguintes:

A **tarefa 1** consistia na seguinte sequência encadeada de passos:

1. Registar utilizador
2. Login do utilizador
3. Comprar CFD
4. Fechar sessão

A **tarefa 2** consistia na seguinte sequência encadeada de passos:

1. Login do utilizador
2. Vender CFD
3. Fechar sessão

A **tarefa 3** consistia na seguinte sequência encadeada de passos:

1. Registar utilizador
2. Login do utilizador
3. Adicionar ativo à watchlist
4. Imprimir watchlist
5. Fechar sessão

De seguida listam-se os resultados das medições.

Antes do Refactoring

Classe	Nº de Linhas	Expressões	% Ramos usados	Chamadas	% Comentários	Classes	Métodos/Classes	Média de Expressões/Método	Complexidade máxima	Profundidade máxima	Profundidade média	Complexidade média
Trader	188	125	16.8	132	5.3	2	11.50	4.30	9	9+	2.99	3.04
TraderApp	252	185	19.5	141	3.2	2	7.50	11.13	7	5	2.58	2.87
Menu	60	41	17.1	16	1.7	1	5.00	5.60	5	4	1.85	2.40
StockHelper	32	19	21.1	4	0.0	1	3.00	4.00	3	3	1.63	2.33
CFD	97	55	16.4	9	9.3	1	9.00	3.78	10	3	1.69	2.00
StockFetcher	85	61	1.6	32	32.9	1	1.00	50.00	2	3	2.11	2.00
User	123	71	8.5	28	0.0	1	16.00	2.75	7	3	1.63	1.38
CompanyNotFoundException	16	4	0	1	56.3	1	1.00	1.00	1	2	0.75	1.00
ComparatorUser	10	5	0	1	0.0	1	1.00	1.00	1	2	0.60	1.00
SaldoException	16	4	0	1	56.3	1	1.00	1.00	1	2	0.75	1.00
SemAutorizacaoException	16	4	0	1	56.3	1	1.00	1.00	1	2	0.75	1.00
SerializationUtil	31	22	0	10	0.0	1	2.00	5.50	1	2	1.09	1.00
Stock	125	80	0	0	0.0	1	20.00	1.90	1	2	1.44	1.00
UtilizadorExistenteException	16	4	0	1	56.3	1	1.00	1.00	1	2	0.75	1.00
Utils	25	4	0	1	36.0	1	1.00	1.00	1	2	0.75	1.00
CFDtype	16	3	0	0	60.0	1	0.00	0.00	0	1	0.33	0.00

Tabela 1. Métricas estáticas das classes

Tarefa	Elapsed Time	CPU Time
Tarefa 1	0.454875s	28.6271518s
Tarefa 2	0.334007s	65.1614337s
Tarefa 3	0.448339s	22.2790403s

Tabela 2. Métricas dinâmicas das classes

Depois do Refactoring

Classe	Nº de Linhas	Expressões	% Ramos usados	Chamadas	% Comentários	Classes	Métodos/Classes	Média de Expressões/Método	Complexidade máxima	Profundidade máxima	Profundidade média	Complexidade média
CFD	97	55	16.4	9	9.3	1	9.00	3.78	10	3	1.69	2.00
CFDtype	15	3	0.0	0	60.0	1	0.00	0.00	0	1	0.33	0.00
CompanyNotFoundException	16	4	0.0	1	56.3	1	1.00	1.00	1	2	0.75	1.00
ComparatorUser	10	5	0.0	1	0.0	1	1.00	1.00	1	2	0.60	1.00
Menu	60	41	17.1	16	1.7	1	5.00	5.60	5	4	1.85	2.40
MenuInicial	28	16	12.5	6	0.0	1	3.00	2.33	3	4	1.44	1.67
SaldoException	16	4	0.0	1	56.3	1	1.00	1.00	1	2	0.75	1.00
SemAutorizacaoException	16	4	0.0	1	56.3	1	1.00	1.00	1	2	0.75	1.00
SerializationUtil	31	22	0.0	10	0.0	1	2.00	5.50	1	2	1.09	1.00
Stock	125	80	0.0	0	0.0	1	20.00	1.90	1	2	1.44	1.00
StockFetcher	85	61	1.6	32	32.9	1	1.00	50.00	2	3	2.11	2.00
StockHelper	32	19	21.1	4	0.0	1	3.00	4.00	3	3	1.63	2.33
Trader	192	119	17.6	131	0.0	2	11.00	4.32	9	9+	3.05	3.14
TraderApp	285	185	19.5	141	14.4	2	7.50	11.13	7	5	2.58	2.87
TraderInterface	40	20	0.0	0	0.0	1	16.00	0.00	0	1	0.80	0.00
User	123	71	8.5	28	0.0	1	16.00	2.75	7	3	1.63	1.38
UtilizadorExistenteException	16	4	0.0	1	56.3	1	1.00	1.00	1	2	0.75	0.00
Utils	25	4	0.0	1	36.0	1	1.00	1.00	1	2	0.75	0.00

Tabela 3. Métricas estáticas das classes

Tarefa	Elapsed Time	CPU Time
Tarefa 1	0.335594s	16.8276316s
Tarefa 2	0.246420s	38.3032377s
Tarefa 3	0.330771s	13.0960804s

Tabela 4. Métricas dinâmicas das classes

3 Trabalho Prático 2

3.1 Code Smell detetados e respetivo Refactoring

Large Class

Verificou-se que a classe Trader continha muitos métodos e linhas de código. Portanto, identificou-se o code smell. O refactor usado para resolver o problema foi o **Extract Interface**, ou seja, criou-se uma interface a partir da classe Trader, permitindo aumentar a compreensibilidade.

Switch Statements

Listing 1.7. Original

```
updater.start();
do{
menumain.executa();
switch(menumain.getOpcao()
    ){
case 1:
signIn();
break;
case 2:
signUp();
break;
}
}while (menumain.getOpcao
    ()!=0);
```

Listing 1.8. Refactored

```
public enum MenuInicial {
SIGNIN(1, TraderApp::
    signIn),
SIGNUP(2, TraderApp::
    signUp),
UNKNOWN(3, () ->
    doSomething());
private int value;
 private Runnable
     execution;
private MenuInicial(int
    val, Runnable toRun) {
     int value = val;
     execution = toRun;
}
public void execute() {
     execution.run();
}
static void execute(int
    code) {
     for (MenuInicial item
         : values()) {
          if (item.value ==
              code) {
               item.run();
                   break;
          }
     }
     throw new IAException(
         "unknown");
     }
};
```

Verificou-se que havia um code smell relacionado com **Switch Statements**. Para isto cria-se uma classe Enum para cada menu e o utilizador invoca a opção a que corresponde uma função.

Temporary Field Smell

Listing 1.9. Original

```
public class Trader
    implements
    Serializable{
private boolean logged =
    false;
private User loggedUser;
private Map<String,User>
    allUsers = new TreeMap
    <>();
```

Listing 1.10. Refactored

```
ppublic class Trader
    implements
    Serializable{
private User loggedUser;
private Map<String,User>
    allUsers = new TreeMap
    <>();
```

Na classe Trader era utilizada uma variável de instância "logged"que era apenas usada em alguns casos, não era crucial uma vez que a mesma informação estava disponível noutra variável de instância "loggedUser". Posto isto, foi decidido remover a variável e utilizar apenas a variável "loggedUser". Caso esta esteja a null então é porque não há nenhum utilizado logado, caso contrário há.

Comments

Documentou-se as maiores classes TraderApp e Trader já que não apresentavam comentários a nenhuma função. Teve-se também em conta a relevância dos comentários em cada função, garantindo que realmente acrescentariam informação.

Data Class

Verificou-se que as classes **CFD** e **Asset** eram Data Class, ou seja, apenas guardavam dados. As classes devem conter dados mas também métodos para operar nesses dados. Contudo, verificou-se que segundo o esquema atual do projeto não seria possível ter as classes de outra forma.

Dead Code

Listing 1.11. Original

```
public class User
    implements Comparable<
    User>, Serializable;
public class CFD
    implements
    Serializable;
public class Trader
    implements
    Serializable;
```

Listing 1.12. Refactored

```
public class User
    implements Comparable<
    User>;
public class CFD;
public class Trader;
```

Verificou-se que as classes User, CFD e Trader continuavam a implementar o Serializable apesar de o recurso estar "morto", ou seja, não era utilizado. Procedeu-se então à sua remoção.

Data Clumps

Listing 1.13. Original

```
MongoClientURI uri = new
    MongoClientURI("
    mongodb://java:fLN?]Nu
    ~;8-@ds231315.mlab.com
    :31315/traderapp");
MongoClient client = new
    MongoClient(uri);
DB db = client.getDB( "
    traderapp" );
DBCollection coll = db.
    getCollection("users")
    ;
```

Listing 1.14. Refactored

```
DBCollection coll = new
    MongoConnection("users
    ").getColl();
```

Nas classes NotificationsDAO e UserDAO verificou-se que o mesmo bloco de código, que permitia a ligação inicial à base de dados, era utilizado em todos os métodos. Posto isto, procedeu-se ao refactoring **Extract Class** e criou-se uma nova classe MongoConnection e em cada DAO uma variável de instância de MongoConnection que permite fazer a ligação à base de dados. A classe MongoConnection é a seguinte:

Listing 1.15. Classe criada

```
public class MongoConnection {
private final MongoClientURI uri = new MongoClientURI("
    mongodb://");
private final MongoClient client = new MongoClient(uri);
private final DB db = client.getDB( "traderapp" );
private final DBCollection coll;
public MongoConnection(String name) {coll = db.
    getCollection(name);}
public DBCollection getColl() {return coll;}
}
```

3.2 Métricas antes e depois do refactoring

Para que se possa perceber quais as implicações do Refactoring no código foram medidas uma série de itens antes e depois do processo. Existem métricas estáticas (ex: número de linhas de uma classe) e métricas dinâmicas, medidas em tempo de execução (ex: tempo para o utilizador executar x tarefas). As tarefas para as métricas dinâmicas são as seguintes:

A **tarefa 1** consistia na seguinte sequência encadeada de passos:

1. Registar utilizador
2. Login do utilizador
3. Comprar CFD
4. Fechar sessão

A **tarefa 2** consistia na seguinte sequência encadeada de passos:

1. Login do utilizador
2. Vender CFD
3. Fechar sessão

A **tarefa 3** consistia na seguinte sequência encadeada de passos:

1. Registar utilizador
2. Login do utilizador
3. Adicionar ativo à wathlist
4. Imprimir watchlist
5. Fechar sessão

De seguida listam-se os resultados das medições.

Antes do Refactoring

Depois do Refactoring

Como podemos observar pelos tempos, todos eles baixaram depois do refactoring efetuado. Em grande parte devido ao refactoring do Data Clump uma vez que permitiu estabelecer apenas duas ligações à base de dados e não várias, conforme os pedidos aos métodos. Diminuindo por isso o tempo de inicialização.

Classe	Nº de Linhas	Expressões	% Ramos usados	Chamadas	% Comentários	Classes	Métodos/Classes	Média de Expressões/Método	Complexidade máxima	Profundidade máxima	Profundidade média	Complexidade média
Asset	57	32	0.0	0	0.0	1	10.00	1.60	1	2	1.44	1.00
CFD	102	65	13.8	10	0.0	1	11.00	3.82	10	3	1.71	1.82
CFDtype	8	3	0.0	0	0.0	1	0.00	0.00	0	1	0.33	0.00
CompanyNotFoundException	7	4	0.0	1	1.7	1	1.00	1.00	1	2	0.75	1.00
ComparatorUser	10	5	0.0	1	0.0	1	1.00	1.00	1	2	0.60	1.00
Menu	60	41	17.1	16	0.0	1	5.00	5.60	5	4	1.85	2.40
notificationsDAO	83	60	1.7	59	0.0	1	4.00	11.50	2	3	1.72	1.25
SaldoException	7	4	0.0	1	0.0	1	1.00	1.00	1	2	0.75	1.00
SemAutorizacaoException	7	4	0.0	1	0.0	1	1.00	1.00	1	2	0.75	1.00
Trader	217	152	13.8	140	0.0	3	9.00	4.52	10	9+	3.32	3.00
TraderApp	284	222	18.0	181	0.0	3	6.00	10.94	7	5	2.49	2.88
User	123	71	8.5	28	0.0	1	16.00	2.75	7	3	1.63	1.38
userDAO	247	173	5.2	184	0.0	1	12.00	12.67	6	5	2.23	1.82
UtilizadorExistenteException	8	4	0.0	1	0.0	1	1.00	1.00	1	2	0.75	1.00
Utils	17	4	0.0	1	0.0	1	1.00	1.00	1	2	0.75	1.00

Tabela 5. Métricas estáticas das classes

Tarefa	Elapsed Time	CPU Time
Tarefa 1	1.423028s	38.1617271s
Tarefa 2	1.1273s	22.0295465s
Tarefa 3	1.392363s	32.893317s

Tabela 6. Métricas dinâmicas das classes

4 Anti-Padrão

Um dos objetivos iniciais do projeto era a implementação de um anti-padrão, à escolha. O anti-padrão selecionado foi o **Input Kludge**, que consiste na má gestão dos inputs dos utilizados na interface, ou seja, quando por exemplo o input permite texto livre do utilizador que pode conter texto inválido e prejudicial ao programa.

Para a implementação do anti-padrão foram removidas todas as verificações de texto do input do utilizador. Por exemplo, foram removidas as verificações se a empresa inserida existe, se o email é válido e se o texto é numérico (nos casos em que tem que ser).

Classe	Nº de Linhas	Expressões	% Ramos usados	Chamadas	% Comentários	Classes	Métodos/Classes	Média de Expressões/Método	Complexidade máxima	Profundidade máxima	Profundidade média	Complexidade média
Asset	57	32	0.0	0	0.0	1	10.00	1.60	1	2	1.44	1.00
CFD	102	65	13.8	10	0.0	1	11.00	3.82	10	3	1.71	1.82
CFDtype	8	3	0.0	0	0.0	1	0.00	0.00	0	1	0.33	0.00
CompanyNotFoundException	7	4	0.0	1	1.7	1	1.00	1.00	1	2	0.75	1.00
ComparatorUser	10	5	0.0	1	0.0	1	1.00	1.00	1	2	0.60	1.00
Menu	60	41	17.1	16	0.0	1	5.00	5.60	5	4	1.85	2.40
MenuInicial	30	17	11.8	6	0.0	1	3.00	2.33	3	4	1.35	1.67
MongoConnection	22	14	0.0	4	0.0	1	2.00	1.00	1	2	0.71	1.00
notificationsDAO	66	42	2.4	45	20.9	1	4.00	7.50	2	3	1.71	1.25
SaldoException	7	4	0.0	1	13.9	1	1.00	1.00	1	2	0.75	1.00
SemAutorizacaoException	7	4	0.0	1	17.6	1	1.00	1.00	1	2	0.75	1.00
Trader	273	146	14.4	139	0.0	3	8.67	4.54	10	9+	3.38	3.08
TraderApp	339	230	17.4	187	0.0	3	6.00	11.28	7	5	2.45	2.88
TraderInterface	51	21	0.0	9	0.0	1	17.00	0.00	0	1	0.81	0.00
User	123	71	8.5	28	0.0	1	16.00	2.75	7	3	1.63	1.38
userDAO	203	135	6.7	150	0.0	1	12.00	9.67	6	5	2.23	1.82
UtilizadorExistenteException	8	4	0.0	1	0.0	1	1.00	1.00	1	2	0.75	1.00
Utils	17	4	0.0	1	0.0	1	1.00	1.00	1	2	0.75	1.00

Tabela 7. Métricas estáticas das classes

Tarefa	Elapsed Time	CPU Time
Tarefa 1	0.481199s	24.7417202s
Tarefa 2	0.381198s	14.28260503s
Tarefa 3	0.470829s	21.3260066s

Tabela 8. Métricas dinâmicas das classes

4.1 Métricas

Antes da implementação do Anti-Padrão

Classe	Nº de Linhas	Expressões	% Ramos usados	Chamadas	% Comentários	Classes	Métodos/Classes	Média de Expressões/Método	Complexidade máxima	Profundidade máxima	Profundidade média	Complexidade média
Asset	57	32	0.0	0	0.0	1	10.00	1.60	1	2	1.44	1.00
CFD	102	65	13.8	10	0.0	1	11.00	3.82	10	3	1.71	1.82
CFDtype	8	3	0.0	0	0.0	1	0.00	0.00	0	1	0.33	0.00
CompanyNotFoundException	7	4	0.0	1	1.7	1	1.00	1.00	1	2	0.75	1.00
ComparatorUser	10	5	0.0	1	0.0	1	1.00	1.00	1	2	0.60	1.00
Menu	60	41	17.1	16	0.0	1	5.00	5.60	5	4	1.85	2.40
notificationsDAO	83	60	1.7	59	0.0	1	4.00	11.50	2	3	1.72	1.25
SaldoException	7	4	0.0	1	0.0	1	1.00	1.00	1	2	0.75	1.00
SemAutorizacaoException	7	4	0.0	1	0.0	1	1.00	1.00	1	2	0.75	1.00
Trader	217	152	13.8	140	0.0	3	9.00	4.52	10	9+	3.32	3.00
TraderApp	284	222	18.0	181	0.0	3	6.00	10.94	7	5	2.49	2.88
User	123	71	8.5	28	0.0	1	16.00	2.75	7	3	1.63	1.38
userDAO	247	173	5.2	184	0.0	1	12.00	12.67	6	5	2.23	1.82
UtilizadorExistenteException	8	4	0.0	1	0.0	1	1.00	1.00	1	2	0.75	1.00
Utils	17	4	0.0	1	0.0	1	1.00	1.00	1	2	0.75	1.00

Tabela 9. Métricas estáticas das classes

Tarefa	Elapsed Time	CPU Time
Tarefa 1	1.423028s	38.1617271s
Tarefa 2	1.1273s	22.0295465s
Tarefa 3	1.392363s	32.893317s

Tabela 10. Métricas dinâmicas das classes

Depois da implementação do Anti-Padrão

Classe	Nº de Linhas	Expressões	% Ramos usados	Chamadas	% Comentários	Classes	Métodos/Classes	Média de Expressões/Método	Complexidade máxima	Profundidade máxima	Profundidade média	Complexidade média
Asset	57	32	0.0	0	0.0	1	10.00	1.60	1	2	1.44	1.00
CFD	102	65	13.8	10	0.0	1	11.00	3.82	10	3	1.71	1.82
CFDtype	8	3	0.0	0	0.0	1	0.00	0.00	0	1	0.33	0.00
CompanyNotFoundException	7	4	0.0	1	1.7	1	1.00	1.00	1	2	0.75	1.00
ComparatorUser	10	5	0.0	1	0.0	1	1.00	1.00	1	2	0.60	1.00
Menu	60	41	17.1	16	0.0	1	5.00	5.60	5	4	1.85	2.40
MenuInicial	30	17	11.8	6	0.0	1	3.00	2.33	3	4	1.35	1.67
MongoConnection	22	14	0.0	4	0.0	1	2.00	1.00	1	2	0.71	1.00
notificationsDAO	66	42	2.4	45	20.9	1	4.00	7.50	2	3	1.71	1.25
SaldoException	7	4	0.0	1	13.9	1	1.00	1.00	1	2	0.75	1.00
SemAutorizacaoException	7	4	0.0	1	17.6	1	1.00	1.00	1	2	0.75	1.00
Trader	203	144	11.1	130	0.0	3	9.00	4.22	10	9+	3.35	3.35
TraderApp	330	229	17.2	186	0.0	3	6.00	11.28	7	5	2.45	2.88
TraderInterface	51	21	0.0	0	0.0	1	17.00	0.00	0	1	0.81	0.00
User	123	71	8.5	28	0.0	1	16.00	2.75	7	3	1.63	1.38
userDAO	203	135	6.7	150	0.0	1	12.00	9.67	6	5	2.23	1.82
UtilizadorExistenteException	8	4	0.0	1	0.0	1	1.00	1.00	1	2	0.75	1.00
Utils	17	4	0.0	1	0.0	1	1.00	1.00	1	2	0.75	1.00

Tabela 11. Métricas estáticas das classes

Tarefa	Elapsed Time	CPU Time
Tarefa 1	0.481010s	24.7417182s
Tarefa 2	0.381150s	14.28260403s
Tarefa 3	0.470809s	21.3260026s

Tabela 12. Métricas dinâmicas das classes

5 Conclusões

Tendo em conta o projeto desenvolvido e tudo o que foi apreendido durante o processo podemos tirar as seguintes conclusões:

- Os processos de refactoring podem ser muitas vezes complexos quando o projeto já está numa fase muito adiantada;
- É importante ir corrigindo os diversos code smells à medida que o projeto está a ser desenvolvido;
- A revisão do código a cada iteração do projeto é de elevada importância;
- Nem todos os code smells têm um refactoring óbvio e nem sempre é possível corrigir o code smell e obter os mesmos resultados;
- Em geral, o tempo de execução melhorou com o refactoring dos code smells, tendo em conta também a sua natureza;
- É necessário ter em atenção à possível implementação de anti-padrões durante o desenvolvimento de um projeto, tendo também em consideração que estes podem ter várias naturezas e estarem relacionados com várias etapas distintas do desenvolvimento;
- Tanto os vários anti-padrões como os code smells são, por vezes, bastante difíceis de detetar, só sendo possível melhorar a sua deteção e correção com a experiência.

Por tudo isto, o desenvolvimento deste projeto revela uma importância bastante elevada uma vez que foi assim possível aprimorar os nossos conhecimentos na área.