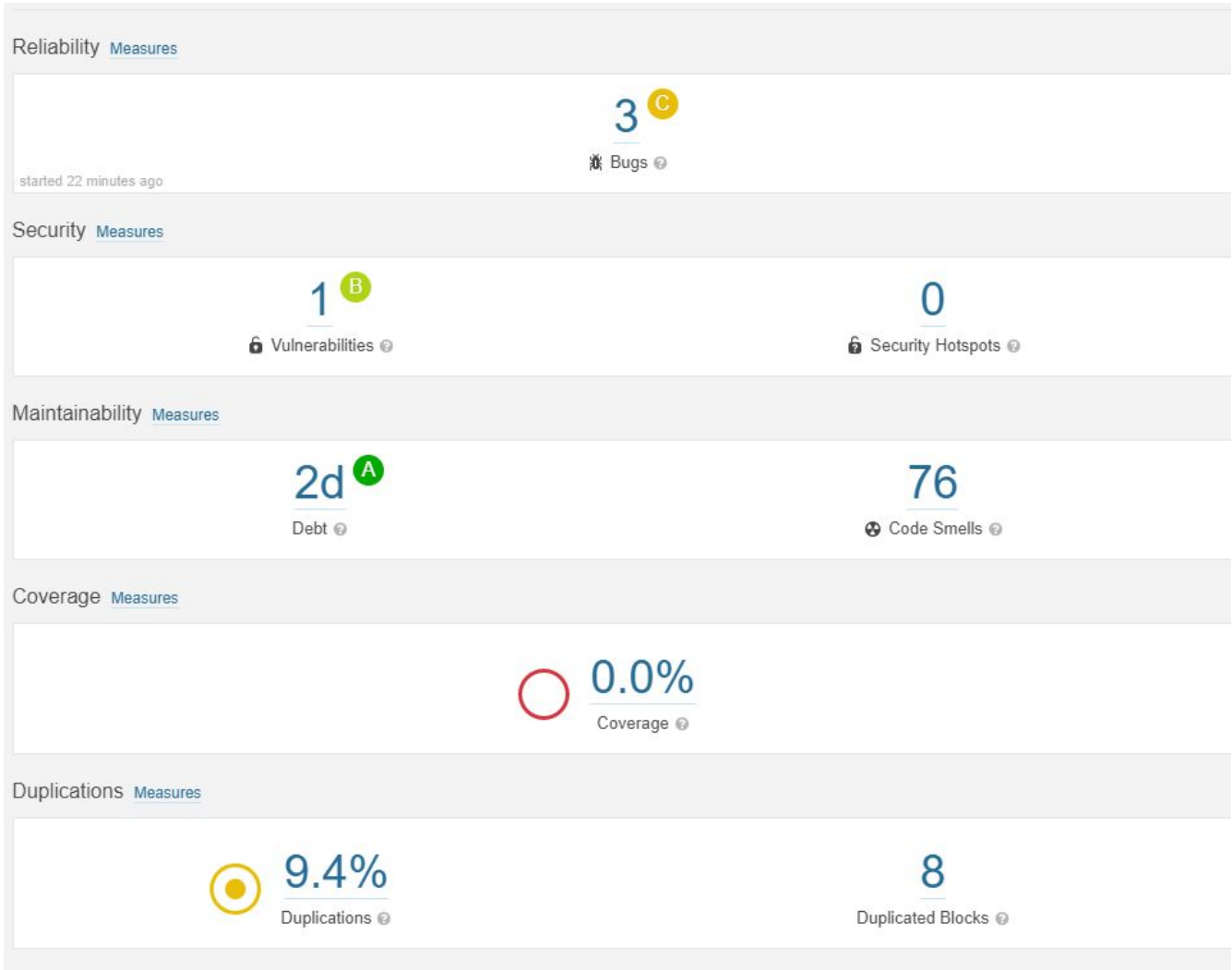


Código analisado com a API Sonarqube em 04/11/2019

Análise geral do código



BUGS

```
String cleanString = preco.replaceAll("[R$,.]", "");
new BigDecimal(cleanString)
    .setScale(2, BigDecimal.ROUND_FLOOR)
    .divide(new BigDecimal(100), BigDecimal.ROUND_FLOOR);
```

Quando a chamada para uma função não tem efeitos, qual é o sentido de fazer a chamada se os resultados são ignorados? Nesse caso, a chamada da função é inútil e deve ser descartada ou o código-fonte não se comporta conforme o esperado.

Esse mesmo erro foi notificado duas vezes

```
public class DBHelper extends SQLiteOpenHelper {
    public static final int VERSAO_BANCO = 27;
    public static final String NOME_BANCO = "SORESENHA_BD";
    public static final SimpleDateFormat dateTimeFormat = new SimpleDateFormat("dd/MM/yyyy kk:mm");
```

Nem todas as classes na biblioteca Java padrão foram gravadas para serem seguras ao encadeamento. É muito provável que usá-los de uma maneira multi-encadeada cause problemas ou exceções nos dados em tempo de execução.

Os mesmos Code Smells da análise anterior se mantiveram nessa portanto só irei apresentar os novos.

Code Smells presentes no código

Erros sublinhados de vermelho

Throwable.printStackTrace (...) imprime um Throwable e seu rastreamento de pilha em algum fluxo. Por padrão, esse fluxo System.Erro, que inadvertidamente pode expor informações confidenciais.

```
try {
    evento.setDate(DBHelper.dateTimeFormat.parse(cursor.getString(
)} catch (ParseException e) {
    evento.setDate(new Date());
    e.printStackTrace();
```

O requisito para uma cláusula padrão é a programação defensiva. A cláusula deve tomar as medidas apropriadas ou conter um comentário adequado sobre o motivo pelo qual nenhuma ação foi tomada.

Mesmo erro notificado três vezes

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
```

Strings literais duplicadas tornam o processo de refatoração propenso a erros, pois você deve atualizar todas as ocorrências.

```
eventoDAO.update(evento);
} else {
    throw new SoresenhaAppException( 1 "Usuario tentou alterar banco sem ser parceiro");
```

```
return eventoDAO.cadastrar(evento);
} else {
    throw new SoresenhaAppException( 2 "Usuario tentou alterar banco sem ser parceiro");
}
```

```
} else {
    throw new SoresenhaAppException( 3 "Usuario tentou alterar banco sem ser parceiro");
}
```

A atualização correta de um campo estático a partir de um método não estático é difícil de corrigir e pode facilmente levar a erros se houver várias instâncias de classe e / ou vários threads em execução. Idealmente, os campos estáticos são atualizados apenas a partir de métodos estáticos sincronizados.

Esta regra levanta um problema sempre que um campo estático é atualizado a partir de um método não estático.

```
super.onCreate();
context = this;
```

```
@Override
public void beforeTextChanged(CharSequence s, int start, int count, int after) {
```

```
@Override
public void onTextChanged(CharSequence s, int start, int before, int count) {
```

Defina uma constante em vez de duplicar esse "TEXT", 4 vezes

```
String QUERY_COLUNAFESTA = "CREATE TABLE " + TABELA_FESTA + "("
    + COLUNA_IDFESTA + " INTEGER PRIMARY KEY, "
    + COLUNA_NOMEFESTA + 1 " TEXT, "
```

```
    + COLUNA_PRECOFESTA + 2 " TEXT, "
    + COLUNA_DATAFESTA + 3 " TEXT, "
    + COLUNA_CRIADORFESTA + " INTEGER, "
    + COLUNA_DESCRICAOFESTA + " TEXT)";
```

```
dh.execSQL(QUERY_COLUNAFESTA);
```

```
private void criarTabelaUsuario(SQLiteDatabase db) {
    String QUERY_COLUNAUSUARIO = "CREATE TABLE " + DBHelper.TABELA_USUARIO + "("
        + DBHelper.COLUNA_ID + " INTEGER PRIMARY KEY, " + DBHelper.COLUNA_SERPARCEIRO
        + " INTEGER DEFAULT 0, " + DBHelper.COLUNA_NOME + 4 " TEXT, " + DBHelper.COLUNA_EMAIL
        + " TEXT," + DBHelper.COLUNA_SENHA
        + " TEXT)";
```

Blocos de código duplicados 2 classes iguais que poderia facilmente ser uma

```
private EditText editNome;
private EditText editDesc;
private EditText editPreco;
private EditText editDate;
private EditText editHora;
private EventoServices eventoServices = new EventoServices(this);
private Calendar eventoDate = Calendar.getInstance();
private SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
private SimpleDateFormat horaFormat = new SimpleDateFormat("kk:mm");

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_criar_evento);
    editNome = findViewById(R.id.nomeFestaedit);
    editPreco = findViewById(R.id.precoFestaedit);
    editDesc = findViewById(R.id.descFestaEdit);
    editDate = findViewById(R.id.dataFestaEdit);
    editHora = findViewById(R.id.horaFestaEdit);
    Button criarBtn = findViewById(R.id.criarFestabutton);
    criarListeners(criarBtn);
}

private void criarListeners(Button criarBtn) {
    criarBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (!validate()) return;
            String nome = editNome.getText().toString();
            String descricao = editDesc.getText().toString();
            String preco = editPreco.getText().toString();
            String cleanString = preco.replaceAll("[R$,.] ", "");
            BigDecimal parsed = new BigDecimal(cleanString)
                .setScale(2, BigDecimal.ROUND_FLOOR)
                .divide(new BigDecimal(100), BigDecimal.ROUND_FLOOR);
            Evento evento = new Evento(nome, descricao, parsed, eventoDate.getTime());
            try {
                eventoServices.criar(evento);
            } catch (SoresenhaAppException e) {
                Toast.makeText(v.getContext(), "Você não tem permissão para alterar eventos", Toast.LENGTH_LONG).show();
                startActivity(new Intent(v.getContext(), ListaEventoActivity.class));
                e.printStackTrace();
            }
            Intent backMenu = new Intent(CriarEventoActivity.this, ListaEventoActivity.class);
            backMenu.setFlags(backMenu.getFlags() | Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_SINGLE_TOP);
            startActivity(backMenu);
        }
    });
    listenersData();
    editPreco.addTextChangedListener(new MoneyTextMask(editPreco));
}
```

```

private void listenersData() {
    final DatePickerDialog.OnDateSetListener date = new DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker view, int year, int monthOfYear, int dayOfMonth) {
            editDate.setText(dateFormat.format(eventoDate.getTime()));
            eventoDate.set(year, monthOfYear, dayOfMonth);
            editDate.setError(null);
        }
    };

    final TimePickerDialog.OnTimeSetListener time = new TimePickerDialog.OnTimeSetListener() {
        @Override
        public void onTimeSet(TimePicker timePicker, int i, int i1) {
            eventoDate.set(
                eventoDate.get(Calendar.YEAR),
                eventoDate.get(Calendar.MONTH),
                eventoDate.get(Calendar.DAY_OF_MONTH),
                i,
                i1
            );
            String hora = horaFormat.format(eventoDate.getTime());
            editHora.setText(hora);
            editHora.setError(null);
        }
    };

    editDate.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            new DatePickerDialog(v.getContext(), date, eventoDate
                .get(Calendar.YEAR), eventoDate.get(Calendar.MONTH),
                eventoDate.get(Calendar.DAY_OF_MONTH)).show();
        }
    });

    editHora.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            new TimePickerDialog(
                v.getContext(),
                time,
                eventoDate.get(Calendar.HOUR_OF_DAY),
                eventoDate.get(Calendar.MINUTE),
                true
            ).show();
        }
    });
}

```



```
private boolean validate() {
    limparCampos();
    if (!validarCampos()) return false;
    return true;
}

private boolean validarCampos() {
    boolean res = true;
    if (!validarPreco()) res = false;
    if (!validarNome()) res = false;
    if (!validarDescricao()) res = false;
    if (!validarData()) res = false;
    if (!validarHora()) res = false;
    return res;
}

private boolean validarNome() {
    String nome = editNome.getText().toString();
    if (nome.isEmpty()) {
        editNome.setError("Nome não pode ser vazio");
        return false;
    }
    return true;
}

private boolean validarDescricao() {
    String desc = editDesc.getText().toString();
    if (desc.isEmpty()) {
        editDesc.setError("Descrição não pode ser vazia");
        return false;
    }
    return true;
}

private boolean validarPreco() {
    String preco = editPreco.getText().toString();
    if (preco.isEmpty()) {
        editPreco.setError("Preço não pode ser vazio");
        return false;
    }
}
```

```

        return true;
    }

    private boolean validarDescricao() {
        String desc = editDesc.getText().toString();
        if (desc.isEmpty()) {
            editDesc.setError("Descrição não pode ser vazia");
            return false;
        }
        return true;
    }

    private boolean validarPreco() {
        String preco = editPreco.getText().toString();
        if (preco.isEmpty()) {
            editPreco.setError("Preço não pode ser vazio");
            return false;
        }
        try {
            String cleanString = preco.replaceAll("[R$,.]", "");
            new BigDecimal(cleanString)
                .setScale(2, BigDecimal.ROUND_FLOOR)
                .divide(new BigDecimal(100), BigDecimal.ROUND_FLOOR);
        } catch (Exception e) {
            editPreco.setError("Preço não é conversível em número");
            return false;
        }
        return true;
    }

    private boolean validarData() {
        String data = editDate.getText().toString();
        Calendar now = Calendar.getInstance();
        now.set(Calendar.HOUR_OF_DAY, 0);
        now.set(Calendar.MINUTE, 0);
        if (data.isEmpty()) {
            editDate.setError("Data não pode ser vazia");
            return false;
        } else if (eventoDate.before(now)) {
            editDate.setError("Esta data já passou");
            return false;
        }
    }
}

```



```

        return true;
    }

    private boolean validarData() {
        String data = editDate.getText().toString();
        Calendar now = Calendar.getInstance();
        now.set(Calendar.HOUR_OF_DAY, 0);
        now.set(Calendar.MINUTE, 0);
        if (data.isEmpty()) {
            editDate.setError("Data não pode ser vazia");
            return false;
        } else if (eventoDate.before(now)) {
            editDate.setError("Esta data já passou");
            return false;
        }
        return true;
    }

    private boolean validarHora() {
        String hora = editHora.getText().toString();
        Calendar now = Calendar.getInstance();
        if (hora.isEmpty()) {
            editHora.setError("Hora não pode ser vazia");
            return false;
        } else if (eventoDate.before(now)) {
            editHora.setError("Esta hora já passou");
            return false;
        }
        return true;
    }

    private void limparCampos() {
        editPreco.setError(null);
        editDesc.setError(null);
        editNome.setError(null);
        editDate.setError(null);
        editHora.setError(null);
    }
}

```

Concatenação desnecessária e que pode impactar no desempenho

```

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    logger.log(Level.INFO, "Upgrading DB from "+oldVersion+" to " +newVersion);
}

```

Se um campo privado for declarado, mas não usado no programa, ele poderá ser considerado código morto e, portanto, deverá ser removido. Isso melhora a manutenção, pois os desenvolvedores não se perguntarão para que serve a variável.

```
public class ParceiroServices {  
    private ParceiroDAO parceiroDAO;
```

Parâmetros não utilizados são enganosos. Quaisquer que sejam os valores passados para esses parâmetros, o comportamento será o mesmo.

```
public ParceiroServices(Context context) {this.parceiroDAO = new ParceiroDAO();}
```

Os programadores não devem comentar o código, pois ele incha os programas e reduz a legibilidade.

O código não utilizado deve ser excluído e pode ser recuperado do histórico de controle de origem, se necessário.

```
//public Parceiro getParceiro(String empresa){return parceiroDAO.get(empresa);}
```

```
//  
// public boolean CheckIfisParceiro(Usuario usuario){return parceiroDAO.check(usuario);}
```

Uma vez obsoleto, classes e interfaces e os seus membros devem ser evitados, ao invés de usado, herdada ou estendida. A descontinuação é um aviso de que a classe ou interface foi substituída e, eventualmente, será removida. O período de suspensão permite que você faça uma transição suave para longe do envelhecimento, logo-a-ser-aposentado tecnologia.

```
public void onClick(View v) {  
    if (!validate()) return;  
    String nome = editNome.getText().toString();  
    String descricao = editDesc.getText().toString();  
    String preco = editPreco.getText().toString();  
    String cleanString = preco.replaceAll("[R$,.]", "");  
    BigDecimal parsed = new BigDecimal(cleanString)  
        .setScale(2, BigDecimal.ROUND_FLOOR)
```

```
    .divide(new BigDecimal(100), BigDecimal.ROUND_FLOOR);
```

Imports não utilizados

```
import com.ufrpe.bsi.soresenha.R;  
import com.ufrpe.bsi.soresenha.infra.app.MecanismoPersistencia;
```

```
import com.ufrpe.bsi.soresenha.infra.gui.MenuActivity;  
import com.ufrpe.bsi.soresenha.infra.negocio.SessaoUsuario;  
import com.ufrpe.bsi.soresenha.infra.persistencia.SessaoUser;
```

```
import com.ufrpe.bsi.soresenha.infra.negocio.SessaoUsuario;
```

Classe vazia

```
package com.ufrpe.bsi.soresenha.parceiro.persistencia;  
  
public class ParceiroDAO {
```

Complexidade Ciclomática






Complexidade	Avaliação
1-10	Método simples. Baixo risco.
11-20	Método razoavelmente complexo. Moderado risco.
21-50	Método muito complexo. Elevado risco.
51-N	Método de altíssimo risco e bastante instável.

Eventos

Cyclomatic Complexity **125**

dominio	18
gui	89
negocio	9
persistencia	9

Gui de eventos

 ConsultarEventoActivity.java	2
 CriarEventoActivity.java	30
 EditarEventoActivity.java	32
 ListaEventoActivity.java	8
 RecyclingAdapterFesta.java	17

domínio de eventos

Cyclomatic Complexity 18

 Evento.java	18
---	----

negócio de eventos

Cyclomatic Complexity 9

 EventoServices.java	9
---	---

persistência de eventos

Cyclomatic Complexity 9

 EventoDAO.java	9
--	---

Infra

Cyclomatic Complexity 44

app	5
gui	11
helper	6
negocio	8
persistencia	14

Infra app

Cyclomatic Complexity 5

MecanismoPersistencia.java	3
SoResenhaApp.java	2

Infra gui

Cyclomatic Complexity 11

ConfigurationActivity.java	5
MenuActivity.java	4
SplashScreenActivity.java	2

Infra negócio

Cyclomatic Complexity 8

SessaoUsuario.java	6
SoresenhaAppException.java	2

Infra Helper

Cyclomatic Complexity 6

 MoneyTextMask.java

6

Infra persistência

Cyclomatic Complexity 14

 DBHelper.java

7

 SessaoUser.java

7

Usuário

Cyclomatic Complexity 96

 dominio

13

 gui

57

 negocio

15

 persistencia

11

Usuário domínio




Cyclomatic Complexity 13

 Usuario.java

13

Usuário gui

Cyclomatic Complexity 57

 EditUserActivity.java	22
 LoginActivity.java	5
 RegisterActivity.java	30

Usuário negócio

Cyclomatic Complexity 15

 UsuarioServices.java	15
--	----

Usuário persistência

Cyclomatic Complexity 11

 UsuarioDAO.java	11
---	----

Parceiro

Cyclomatic Complexity 5

 dominio	4
 negocio	1
 persistencia	0

Parceiro domínio

Cyclomatic Complexity 4

 Parceiro.java

4

Parceiro Negócio

Cyclomatic Complexity 1

 ParceiroServices.java

1

Parceiro persistência

Cyclomatic Complexity 0

 ParceiroDAO.java

0