INSTITUTO POLITÉCNICO DE SETÚBAL
ESCOLA SUPERIOR DE TECNOLOGIAS DE SETÚBAL

Qualidade de Software
Mestrado em Engenharia de Software

Daniel Singh - 201901071
Rafael Marçalo - 201900456

18 de novembro de 2023

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

Este relatório tem como objetivo a análise da qualidade de código de um projeto realizado por outros alunos. Para esta análise iremos recorrer a uma breve experimentação das funcionalidades e requisitos do projeto. Em seguida iremos testar várias ferramentas e livrarias de análise da qualidade/consistência do código ao qual iremos em breve apresentar não só as suas funcionalidades, mas também os resultados que obtivemos com elas.

2 Análise do Caso de Estudo

Sendo esta a primeira fase desta pequena investigação, transferimos o projeto caso de estudo do **Moodle**[1] e instalámos nas nossas máquinas para análise. Desta forma, após instalarmos e configurarmos o projeto nos nossos computadores, fizemos um breve teste das suas funcionalidades e diagnosticámos alguns problemas que possam afetar a qualidade do projeto, assim sendo, elaborámos uma lista de requisitos funcionais que o projeto deveria implementar.

2.1 Análise de Requisitos

ID	**Descrição**	**Implementado**
RF1	O sistema deverá permitir inserir novos clientes	Sim
RF2	O sistema deverá permitir pesquisar pelos clientes	Sim
RF3	O sistema deverá permitir aplicar filtros às pesquisas dos clientes	Parcialmente[2]
RF4	O sistema deverá permitir atualizar a informação dos clientes	Sim
RF5	O sistema deverá permitir eliminar clientes	Sim
RF6	O sistema deverá efetuar a validação de dados ao criar um cliente	Parcialmente[3]
RF7	O sistema deverá limpar os campos prévios dos formulários, após sua utiliazação	Não

Tabela 1: Análise de requisitos entidade cliente

ID	**Descrição**	**Implementado**
RF8	O sistema deverá permitir inserir novos utilizadores	Sim
RF9	O sistema deverá permitir pesquisar pelos utilizadores	Sim
RF10	O sistema deverá permitir aplicar filtros às pesquisas dos utilizadores	Parcialmente[2]
RF11	O sistema deverá permitir atualizar os seus dados pessoais	Não
RF12	O sistema deverá permitir não atualizar os dados pessoais de utilizadores com as mesmas roles ou com hierarquia superior	Não
RF13	O sistema deverá permitir atualizar a informação de outros utilizadores com hierarquia inferior	Sim
RF14	O sistema deverá permitir eliminar outros utilizadores	Sim
RF15	O sistema deverá efetuar a validação de dados ao criar um utilizador	Parcialmente[3]
RF16	O sistema deverá limpar os campos prévios dos formulários, após sua utiliazação	Não

Tabela 2: Análise de requisitos entidade utilizador

[1] Moodle IPS - https://moodle.ips.pt/2324/
[2] Utiliza biblioteca externa
[3] Apenas no lado do cliente

ID	Descrição	Implementado
RF17	O sistema deverá permitir criar serviços	Sim
RF18	O sistema deverá permitir pesquisar pelos serviços	Sim
RF19	O sistema deverá permitir realizar pesquisas filtrados aos serviços	Sim
RF20	O sistema deverá permitir aplicar filtros às pesquisas dos serviços	Sim
RF21	O sistema deverá permitir atualizar a informação dos serviços	Sim
RF22	O sistema deverá de validar os dados ao criar um serviço	Sim
RF23	O sistema deverá permitir reabrir um serviço concluído	Sim
RF24	O sistema deverá de mostrar quantos qual o index do serviço relativo à sua prioridade	Sim
RF25	O sistema deverá limpar os campos prévios dos formulários, após sua utiliazação	Não

Tabela 3: Análise de requisitos entidade serviço

ID	Descrição	Implementado
RF26	O sistema deverá permitir enviar mensagens para outros utilizadores em tempo real	Sim
RF27	O sistema deverá permitir escolher para qual utilizador	Sim
RF28	O sistema deverá permitir visualizar o histórico de mensagens	Não
RF29	O sistema deverá permitir mensagens de grupo	Não

Tabela 4: Análise de requisitos entidade mensagem

ID	Descrição	Implementado
RF30	O sistema deverá permitir fazer login	Não
RF31	O sistema deverá permitir fazer logout	Não
RF32	O sistema deverá de verificar permissões no routing	Não

Tabela 5: Análise de requisitos entidade sistema

3 Ferramentas de Análise

Após um diagnóstico de requisitos do projeto, avançamos para uma análise a nível de possíveis problemas de código, para esta tarefa, efetuamos uma pesquisa de ferramentas que nos possam ajudar com esta tarefa, das quais, achamos importante destacar:

- **ESLint** é uma ferramenta de análise de código estático que verifica o código JavaScript procurando diversos problemas comuns, como erros de sintaxe, problemas de formatação, violações de estilo de código e possíveis bugs.
- **Prettier Code Formatter** é um formatador de código. Impõe um estilo consistente analisando seu código e reimprimindo-o conforme um conjunto de regras que levam em consideração diversos parâmetros como (comprimento máximo da linha, agrupando o código quando necessário, etc.). Utilizando esta ferramenta conseguimos garantir a uniformização do código desenvolvido ao longo do projeto.
- **Jest** é uma framework para testes de código compatível com bastantes projetos (Babel, TypeScript, Node, React, Angular, Vue e mais...). Suporta mocking, gera reports de code coverage.
- **Mocha** é outra framework de testes para Node capaz de correr no browser. Esta framework contém "interfaces" de testes que podem ser utilizados para vários tipos de desenvolvimento(TDD, BDD, exports, qunit, require, etc...). O Mocha também permite o fácil teste de funções assíncronas e de promises.
- **Chai**, livraria de testes especificada para testes do tipo BDD/TDD, esta livraria geralmente é integrada noutras frameworks específicas de testes(como o Mocha, por exemplo).

4 Linting

Para a análise de linting do código usamos a ferramenta ESLint, devido à sua alta configurabilidade e por ser uma das ferramentas de linting mais utilizadas, como podemos observar no esquema abaixo:

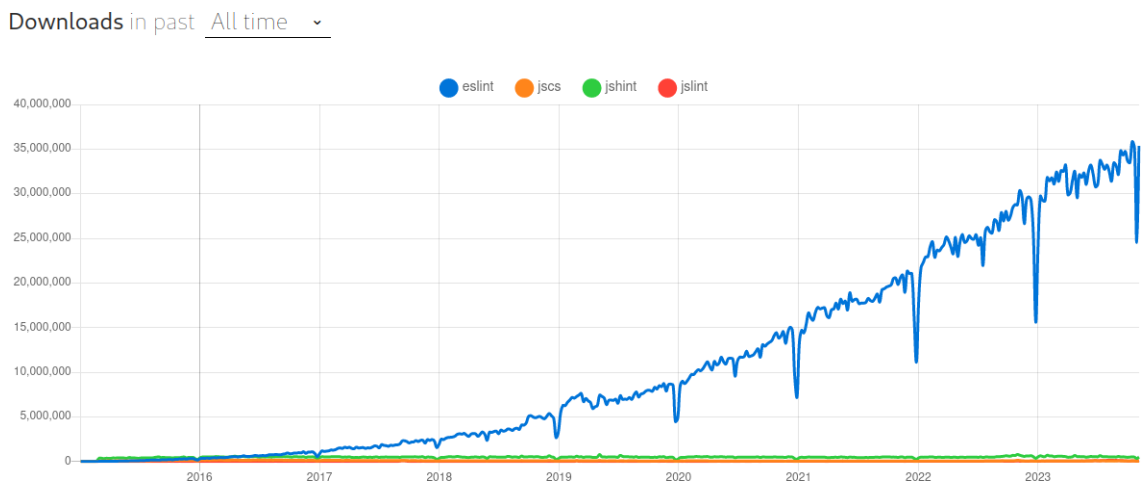

Figura 1: Estatísticas do download dos Linters ao longo do tempo[4]

A nossa experiência a utilizar o **ESLint** tem sido bastante benéfica, ajudou-nos a identificar vários problemas no código analisado. Devido a configurabilidade da ferramenta, esta permitiu-nos identificar e relatar diversos problemas desde erros de sintaxe a problemas mais complexos relacionados com a qualidade e a manutenibilidade do código **JavaScript**. Adicionalmente podemos também referir que a habilidade de definir regras personalizadas ajudaram-nos a adaptar a ferramenta aos requisitos do projeto.

Com a ajudar deste custom formatter foi nos possível organizar os resultados obtidos do output do **ESLint**.

[4]Imagem obtida através de https://npmtrends.com/eslint-vs-jscs-vs-jshint-vs-jslint

4.1 Resultados

Através desta ferramenta, foi nos possível efetuar várias auditorias ao código utilizando várias configurações, permitindo-nos assim destacar as estatísticas

4.1.1 Análise simples

Para esta análise, configuramos o **ESLint** para apenas reportar erros de sintaxe e alguns problemas de código em geral(variáveis inutilizadas, valores *undefined*, *espaços* misturados com *tabs*, etc...).

Problema Anotado	**Total**
Variáveis Inutilizadas	40
Valores *undefined*	189
Espaços Misturados com *Tabs*	2

Tabela 6: Análise de Lint Simples

4.1.2 Análise simples (Client-side)

Nesta análise decidimos fazer distinção entre problemas de código de cliente obtendo assim os seguintes resultados:

Problema Anotado	**Total**
Variáveis Inutilizadas	23
Valores *undefined*	189
Espaços Misturados com *Tabs*	2

Tabela 7: Análise de Lint Simples (Client-side)

4.1.3 Análise simples (Server-side)

Nesta análise decidimos fazer distinção entre problemas de código de servidor obtendo assim os seguintes resultados:

Problema Anotado	**Total**
Variáveis Inutilizadas	17

Tabela 8: Análise de Lint Simples (Server-side)

4.1.4 Análise detalhada

Para a análise detalhada, configuramos o **ESLint** para reportar os problemas da análise simples, assim como os problemas de estilo de código. Para esta análise, escolhemos os estilo de código *Standard* onde pudemos observar os seguintes resultados:

Problema Anotado	**Total**
Utilização de *aspas*	772
Utilização de *ponto e vírgula*	1302
Espaço a terminar linha	171
Linhas em branco	26
Indentação incorreta	2062
Espaço inserido várias vezes de seguida	11
Espaço inserido antes de comentário	50
Espaço em falta antes do *parênteses* da *função*	121
Espaço em falta antes de blocos de código	67
Operador ternário desnecessário	1
Trocar variável para *const*	1
Usar *var*	11
Espaço antes de uma palavra-chave	2
Estilo das *chavetas*	93
Espaço entre as =>	85
Objeto abreviado	6
Caractere terminal de ficheiro	5
Espaço dentro da definição de objeto	5
Variáveis Inutilizadas	11
Espaços padding blocos de código	7
Hardcoded callback	24
Valores *undefined*	189
Propriedades sem *aspas*	6
Tabs vazios	2
Nova linha após *parênteses* de objeto	9
Nova linha após propriedade de objeto	9
Falta de *espaços* entre operadores lógicos e aritméticos	22
Vírgula em propriedades finais de objetos	2
Espaços entre parênteses	2
Espaços múltiplos	2

Tabela 9: Análise de Lint Detalhada

4.1.5 Análise dos resultados obtidos

Observando a análise de código detalhada podemos constatar que muitos dos problemas apontados pela ferramenta, poderiam ser evitados através da utilização de uma ferramenta de formatção de código *(ex: **Prettier Code Formatter**)*.

O **Prettier** é uma ferramenta de formatação de código que analisar o código e de seguida, o imprime de novo num estilo consistente. Ao contrário dos linters tradicionais que se focam em identificar e corrigir erros de codificação específicos, o principal objetivo do **Prettier** é impor um estilo de código consistente e opinativo em todo o código-fonte.

Um dos principais benefícios do **Prettier** é a sua simplicidade e configuração *barebones*. Ele pretende minimizar a necessidade dos programadores tomarem decisões sobre o estilo de código, fornecendo um conjunto padrão de regras sendo estas facilmente configuradas.

Os programadores podem integrar o **Prettier** no seu *workflow* usando-o diretamente a partir da linha de comandos ou integrando-o diretamente em editores de código e IDEs.

5 Testes Unitários

Os testes unitários são uma prática fundamental no desenvolvimento de software. Estes, envolvem o teste de unidades/componentes individuais do código fonte por forma a garantir a boa funcionalidade das suas operações. Estas unidades/componentes podem variar entre funções, métodos ou objetos. O objetivo principal destes testes é certificar cada uma destas unidades/componentes por forma a verificar se os retornos estão corretos, consoante a variedade de inputs possíveis. Ao se isolar e testar pequenas porções do código, os programadores mais facilmente conseguem identificar e corrigir bugs no ciclo de desenvolvimento.

5.1 Metodologias e Práticas

O Desenvolvimento Orientado a Testes (TDD) e o Desenvolvimento Orientado ao Comportamento (BDD) são duas metodologias essenciais no campo do desenvolvimento e teste de software. Tanto o TDD quanto o BDD desempenham um papel crucial na garantia da qualidade e confiabilidade de aplicações de software.

5.1.1 Test-Driven Development

O TDD é uma metodologia de desenvolvimento de software na qual testes unitários são usados para orientar o desenvolvimento da aplicação. Com o TDD, os programadores escrevem os testes antes de escrever o código. Com isto os programadores conseguem ter a certeza de que o código que escreveram faz o trabalho pretendido e os testers podem executar os seus testes tendo a certeza de que o código funcionará corretamente.

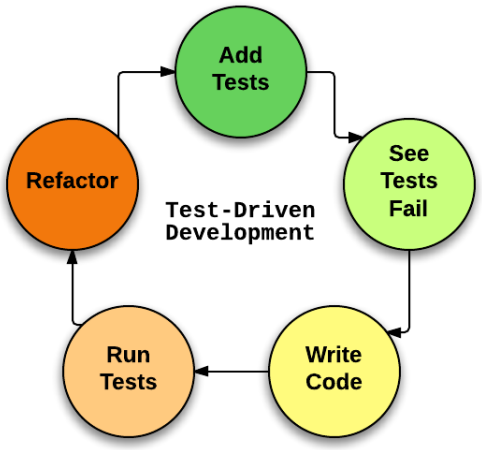

Figura 2: Fluxograma Desenvolvimento Orientado a Testes(TDD)

Processo Test Driven Development

- Escrever um teste que define uma nova função ou melhoria;
- Escrever a quantidade mínima de código para passar no teste;
- Refatorar o código para remover duplicações e melhorar seu design.

Benefícios Test Driven Development

- Tempo de trabalho reduzido: O TDD não permite que novo código seja escrito, a menos que o código existente seja testado com sucesso e sem falhas. Até que as falhas sejam completamente resolvidas e removidas, o processo de escrita do código é interrompido;
- Rápido feedback: Como os testes focam-se em secções especificas do código, os programadores podem receber feedback mais rapidamente permitindo-lhes implementar alterações de maneira mais eficiente;
- Maior produtividade de desenvolvimento: Com o TDD, o foco está na produção de código funcional, e não no design de casos de teste;
- Código mais flexível e de fácil manutenção: Como cada parte do código é testada antes de passar para a próxima parte do processo de desenvolvimento de software, o código mantém a funcionalidade e é adaptável no futuro.

5.1.2 Behavior-Driven Development

O Desenvolvimento Orientado ao Comportamento (BDD) é uma extensão do TDD que reforça testar o comportamento da aplicação do ponto de vista do utilizador final. O BDD envolve a criação de especificações executáveis que definem o comportamento da aplicação. Este, baseia-se em testar comportamentos em vez de detalhes de implementação, facilitando a colaboração entre programadores, testers e utilizadores. Os cenários de BDD geralmente são descritos num formato mais legível e centrado ao utilizador, permitindo que utilizadores não programadores possam compreender os testes.

Processo Behavior-Driven Development As etapas para o Desenvolvimento Orientado ao Comportamento são bastante simples:

- O comportamento é descrito normalmente utilizando uma user story, que permite que a equipa discuta exemplos concretos da nova funcionalidade e as expectativas do seu comportamento;
- A ação é então escrita transformando os exemplos em documentação de forma que possa ser automatizada;
- O teste é executado para auxiliar e orientar os programadores no desenvolvimento do código;
- O código é então criado para realizar esta funcionalidade tornando assim o código funcional.

Benefícios Behavior-Driven Development Existem vários benefícios em usar BDD para desenvolvimento de software, incluindo:

- Incorporação da experiência do utilizador(UX): o BDD foca-se na experiência do utilizador, como tal, permite que a equipa desenvolva uma perspetiva mais ampla e observe lacunas no seu design;
- Custo-benefício: Como o BDD estabelece prioridades para utilizadores, programadores e investidores, este, permite que os recursos sejam utilizados de forma otimizada no desenvolvimento de software;
- Testes simples entre browsers: o BDD concentra-se no comportamento de utilizador, o que significa que oferece uma estrutura ideal para testes entre browsers.

5.1.3 TDD e BDD

Em resumo, o TDD é uma prática de desenvolvimento que se foca em testar o código de forma isolada, enquanto que o BDD é uma metodologia de equipa que se foca em testar o comportamento da aplicação do ponto de vista do utilizador. Embora partilhem semelhanças, eles têm algumas diferenças chave na finalidade, no tipo de colaboração, na linguagem e nos casos de teste.

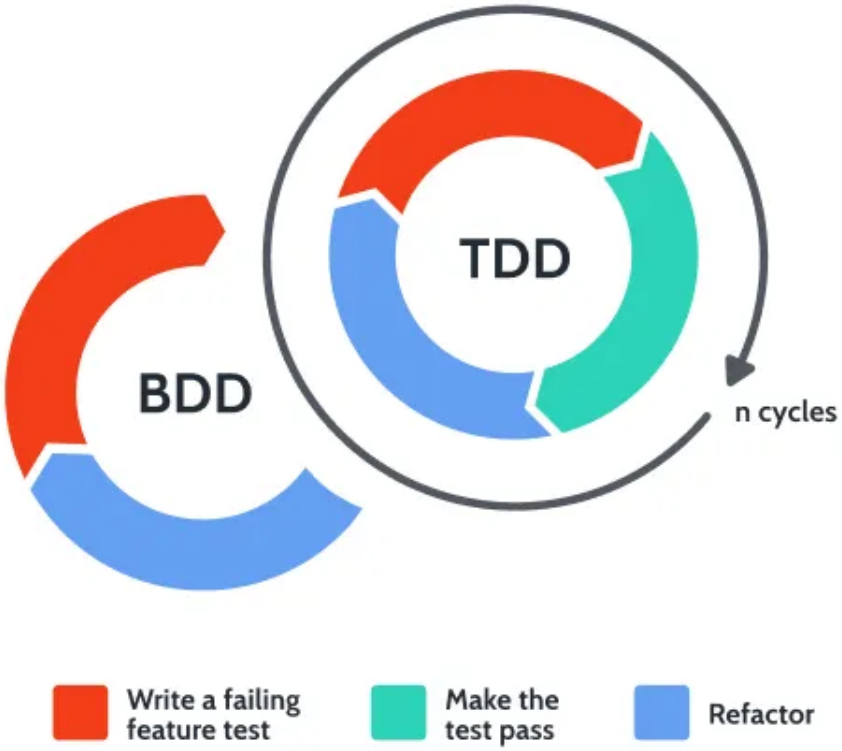

Figura 3: Fluxograma TDD com BDD