Ivan Jorge Camacho Valadares

Geração de código não destrutivo em aplicações web

Orientador: Professor Doutor Pedro Hugo Queirós Alves

Universidade Lusófona - Centro Universitário de Lisboa

Escola de Comunicação, Arquitetura, Artes e Tecnologias da Informação

Lisboa

2023

Ivan Jorge Camacho Valadares

Geração de código não destrutivo em aplicações

Dissertação defendida em provas públicas para obtenção do Grau de Mestre no Curso de Mestrado em Engenharia Informática e Sistemas de Informação, conferido pela Universidade Lusófona – Centro Universitário de Lisboa, no dia 17 de março de 2023, perante o júri, nomeado pelo Despacho de Nomeação n.º: 179/2023, de 14 de Março, com a seguinte composição:

Presidente:

Prof. Doutor Paulo Jorge Tavares Guedes (ULHT).

Arguente:

Prof. Doutor João Ricardo Viegas da Costa Seco (UNL).

Orientador:

Pedro Hugo Queirós Alves (ULHT).

Universidade Lusófona - Centro Universitário de Lisboa

Escola de Comunicação, Arquitetura, Artes e Tecnologias da Informação

Lisboa

2023

Geração de código não destrutivo em aplicações web

"To make our way, we must have firm resolve, persistence, tenacity.We must gear ourselves to work hard all the way. We can never let up." (Ralph Bunche)

Resumo

De forma a acelerar o processo de desenvolvimento e o aumento da qualidade do código, ferramentas de geração de código são amplamente usadas por empresas de desenvolvimento de software.

Foram testadas algumas das ferramentas de geração de código mais utilizadas para o desenvolvimento de aplicações Web (Spring MVC, MVC.net, PHP, Python), tendo sido identificado um problema estrutural: quando é necessário alterar uma entidade que leva a uma segunda geração de código, esta torna-se destrutiva. Acontece que todo o código é reescrito e as alterações feitas a este são apagadas. No presente momento, nenhuma destas ferramentas disponibiliza uma solução para resolver esta questão.

Esta dissertação apresenta três abordagens onde, após novas gerações de código, o novo código gerado possa conviver com o código já desenvolvido pelo programador.

É apresentada a implementação dos algoritmos das abordagens onde primeiramente é demonstrada uma fase de comparação textual, ou de comparação através de Abstract Syntax Tree (AST). Após a fase de comparação é demonstrada uma fase de junção inteligente em que, com base na comparação, é gerado código final que inclui as alterações efetuadas, tanto pelo utilizador como pelo código gerado.

São apresentados os resultados onde estes algoritmos são submetidos a um conjunto de testes, envolvendo diferentes cenários de geração, alteração e regeração de código, de modo a aferir a sua eficácia.

Por último é apresentado um caso real onde foi desenvolvido um gerador de código na empresa Agap2IT que implementa os algoritmos das abordagens estudadas.

Palavras-chave: Geração de código; Programação Web; Ferramentas de Desenvolvimento

Abstract

In order to speed up the development process and increase code quality, code generation tools are widely used by software development companies.

Some of the most used code generation tools for web application development (Spring MVC, MVC.net, PHP, Python), were tested and a structural problem was identified. When an entity needs to be changed, which leads to a second code generation, it becomes destructive. It turns out that all code is rewritten, and changes made to it are deleted. Currently, none of these tools provides a solution to address this issue.

This thesis presents three different approaches. These approaches consist in the fact that new code generations allow the newly generated code to coexist with code already developed by the programmer.

The implementation of the algorithms of the approaches is presented where in the first phase a textual comparison or a comparison using AST is demonstrated. After the comparison phase an intelligent join phase is shown, where based on the comparison, the final code is generated including changes made by both the user and the generated code.

Results are presented where these algorithms are subjected to a set of tests, involving different scenarios of code generation, change, and regeneration, in order to assess their effectiveness.

Lastly, a real case is presented where a code generator was developed within the company Agap2IT, that implements the algorithms of the studied approaches.

Keywords: Code Generation; Web Programming; Development Tools

Índice

Siglas

API Application Programming Interface 30, 31, 33, 35, 36, 37, 43
AST Abstract Syntax Tree ix, xi, 51, 52, 53, 67, 94

CRUD Create Read Update Delete 16, 23, 24, 35, 36, 41, 70, 71, 96, 97, 98
CSRF Cross Site Request Forgery 30

DAL Data Access Layer 38, 97

HTML Hypertext Markup Language 22, 23, 31

MVC Model-View-Controller 11, 12, 29, 30, 36, 45, 46, 97
MVT Model-View-Template 34

OO Orientada a Objectos 53
ORM Object–Relational mapping 38, 45, 97

REST Representational state transfer 35, 36, 42

SQL Structured Query Language 38

UML Unified Modeling Languag 97

XML Extensible Markup Language 69, 97

Capítulo 1

Introdução

Hoje, das pequenas às grandes empresas, muitas são as que preferem aplicações web para os seus serviços internos e externos. Estas aplicações garantem vantagens como:

- Acessibilidade através de múltiplos dispositivos

  As aplicações web podem ser utilizadas na maioria dos navegadores e funcionam de forma uniforme em todos os dispositivos, independentemente do tipo (computador, telemóvel, tablete, etc) que esteja a ser utilizado para aceder. Isto permite ao utilizador escolher quando e onde utilizar a aplicação, promovendo o trabalho flexível dentro das empresas e aumentando desta forma a produtividade global dos funcionários [7].

- Interoperabilidade de sistemas

  As aplicações Web são programas armazenados em servidores remotos que são entregues pela Internet através do browsers, ao contrário das aplicações desktop, que são executadas localmente no sistema operacional do dispositivo. As aplicações Web têm uma maior capacidade de integração com outros sistemas face às aplicações desktop. As aplicações desktop são isoladas em comparação com as aplicações Web, que são significativamente mais interoperáveis. Isto acontece porque devido às aplicações web se encontrarem implementadas na Internet, estas podem ser ligadas entre si mais facilmente do que dois sistemas completamente separados.

- Maior facilidade na distribuição

  Devido às aplicações web serem instaladas em servidores web, sempre que existe uma atualização, esta só necessita de ser aplicada no servidor sem necessidade de atualizar cada máquina. Isto significa que a manutenção pode ser executada através

de um ponto central e as novas atualizações são efetuadas com maior facilidade. O tempo necessário para efetuar alterações é reduzido e o sistema é consistente [81].

- Maior potencial, flexibilidade e escalabilidade

  Tal como as atualizações são simples de realizar, também é simples aumentar a capacidade da aplicação para que acompanhe o crescimento do negócio. Ao aumentar a capacidade de uma aplicação poderá ser necessário aumentar as características do computador que a contêm, como por exemplo, incluir mais espaço em disco ou mais memória. No caso de aplicações Desktop é necessário fazer estas melhorias para os vários computadores que contêm a aplicação, sendo este processo mais demoroso e dispendioso em relação às aplicações web onde é apenas necessário fazer melhorias no servidor que disponibiliza a aplicação. Além disso, quando ocorrem problemas, os servidores podem ser completamente substituídos sem qualquer efeito em todo o sistema operativo. Isto, por conseguinte, diminui qualquer tempo de inatividade.

- Proteção dos dados

  As aplicações web oferecem uma forma segura de aceder aos dados centralizados [1]. Os servidores só são acedidos diretamente pela pessoa ou equipa que os gere. Em geral, a implementação de medidas de segurança é mais simples, uma vez que é feita de forma centralizada e não existe a necessidade de manter a segurança de cada dispositivo em que a aplicação está a ser utilizada.

- Facilidade de Backup

  Os dados das aplicações web são normalmente centralizados, permitindo uma maior facilidade de Backup. Através da utilização de computação em nuvem, os servidores podem ser totalmente redundantes e replicados para evitar tempos de inatividade em caso de desastre.

Porém, a quantidade de código e a complexidade inerente a esse tipo de aplicações são elementos que podem dificultar a construção de uma aplicação web. Por isso, é importante levantar a questão de como melhorar o respetivo processo de desenvolvimento. Uma possível solução para combater estes problemas é o uso de ferramentas de geração de código [26] [51] [44], pois estas permitem:

- Garantir a coerência.

  Quando se tem muitos programadores diferentes a trabalhar durante muito tempo, manter a coerência do código é fundamental. Não se pretende abrir dois ficheiros com objetivos semelhantes e ter estilos de programação completamente distintos. Quando os ficheiros têm um aspeto demasiado diferente, é necessário que os programadores percam tempo a ler cada um dos ficheiros para compreender o seu conteúdo. Este tempo é exponencial ao tamanho da aplicação. A utilização de geradores de código ajuda a equipa a manter a consistência e homogeneidade do

código. O mesmo problema deve ser resolvido da mesma forma em todos os sítios da aplicação facilitando assim a sua manutenção e escalabilidade [5].

- Servir como uma ferramenta de aprendizagem para engenheiros de software júniores.

  A consistência que os geradores de código proporcionam é eficaz para os engenheiros de software júniores. Quando se trazem novos elementos para uma equipa estes não possuem o mesmo conhecimento dos programadores séniores. Na ótica de que os geradores de código produzem arquiteturas corretamente desenhadas e implementam corretamente padrões de desenho e boas praticas, o código produzido acaba por ser utilizado como uma ferramenta de ensino, ajudando-os a aprender conceitos e soluções que os programadores séniores tiveram de aprender de uma forma mais complexa. Este código ao ser gerado e não criado por engenheiros de software júniores proporciona um aumento na qualidade do mesmo [27].

- Garantir um aumento na velocidade e produtividade.

  Tempo é dinheiro. É moroso escrever uma aplicação para um problema complexo e pode demorar ainda mais tempo a adquirir os conhecimentos necessários para fazê-lo. Em vez de escrever o código uma vez e utilizá-lo apenas nesse único local, é possível escrever um gerador que escreve o código automaticamente e futuramente poupar tempo cada vez que é preciso resolver esse problema no futuro. Desta forma ao longo do tempo escreve-se menos código o que se reflete numa poupança de tempo e de custos, tornando possível lançar soluções mais rapidamente. Escrever menos código significa também uma redução de bugs que se traduz numa menor necessidade de manutenção.

A geração automática de código vai ajudar a aumentar a eficácia da produção complexa de software, reduzindo o custo e o tempo associados ao esforço da escrita de código, aumentando a produtividade dos programadores [70] e melhorando a qualidade do código por meio da redução de erros [2]. De um modo geral, a ferramenta irá criar código base e repetido deixando assim tempo para o programador se concentrar em criar funcionalidades específicas e exigentes [30] e também focar-se em problemas de alto de nível de design. Porém podem existir desvantagens como:

- A geração de código que não é utilizado, fazendo com que seja necessário percorrer todo o projeto para apagá-lo.

- O aumento da complexidade do código.

- A pouca flexibilidade, devido ao código gerado ser restrito a certos padrões de desenho, arquiteturas ou frameworks, fazendo com que o programador escreva código "compatível" com o gerador. Este código pode acabar por ser mais complexo e menos eficiente.

Um gerador de código apresenta tanto vantagens como desvantagens, tornando necessário que antes da sua criação se avalie a aplicação a desenvolver, de forma a entender se esta vai realmente beneficiar da geração de código.

Apesar de existirem diversas ferramentas de geração de código, todas elas sofrem do mesmo problema estrutural: quando é necessário alterar uma entidade que leva a uma segunda geração de código, ou existe uma atualização na ferramenta, esta torna-se destrutiva. Acontece que todo o código é reescrito e as alterações feitas a este são apagadas. Pretende-se então, chegar a uma solução onde após novas gerações de código, o novo código gerado possa conviver com o código já desenvolvido pelo programador. Para esta solução é necessário que haja previamente uma avaliação do código de maneira a entender-se se falamos do mesmo código ou se é diferente. No caso de ser diferente, pretende-se que exista uma junção entre o código existente com o código que está a ser gerado, de maneira a que o resultado final continue com as funcionalidades para que está a ser gerado e com as funcionalidades alteradas. Para atingir este objetivo propõe-se o estudo e desenvolvimento de abordagens que serão divididas em duas fases. Uma fase de comparação que consiste em comparar as instruções do método original com o método alterado pelo utilizador e o método novamente gerado (2º geração) e uma segunda fase que consiste em juntar de uma maneira inteligente as instruções do método alterado com o método novamente gerado (2º geração). Para esta fase serão demonstradas três abordagens diferentes: A abordagem de marcação de início e fim de código que consiste em criar marcadores que indiquem o começo e o final do código gerado, a abordagem do diferencial de instruções, que propõe um algoritmo que faz uma comparação textual entre 3 métodos (1º geração, 2ª geração e alterado) removendo ou adicionando as instruções necessárias de forma a refletir tanto as alterações do utilizador, como as do gerador. E por fim, a abordagem de Diff And Patch baseada numa biblioteca de Diff-Match-Patch.

1.1 Contribuições

As contribuições desta dissertação podem ser vistas como uma tentativa de resolver o problema estrutural dos geradores de código. Podemos resumir assim as contribuições deste estudo da seguinte forma:

1. Um processo inovador para geração de código não-destrutivo;
2. Uma implementação desse processo;
3. Os resultados da utilização dessa implementação num cenário real;

1.2 Estrutura da Dissertação

O resto do presente documento está estruturado para expandir os conceitos introduzidos por este capítulo. O conteúdo dos capítulos seguintes é, por conseguinte, o seguinte:

1. No capítulo 2 é demonstrado o funcionamento das ferramentas de geração de código descrevendo o processo de geração de uma aplicação web e mostrando exemplos que incluem uma geração destrutiva de código.
2. O capítulo 3 descreve o estado atual dos geradores de código existentes no mercado, as suas vantagens e os seus problemas.
3. O capítulo 4 é focado na geração de código não-destrutivo, os tipos de comparação de métodos possíveis e as várias abordagens.
4. No capítulo 5 é explicada a implementação das ferramentas para testar as abordagens descritas no capítulo 3.
5. No capítulo 6 são apresentados os resultados dos testes e taxas de sucesso das abordagens descritas no capítulo 3.
6. O capítulo 7 apresenta uma análise de um caso real onde foi criado um gerador de código num ambiente empresarial.
7. Por último reflete-se sobre a utilização de geradores de código e o uso das abordagens descritas destacando as vantagens e limitações destas.

Capítulo 2

Geração de código

Este capítulo demonstra como funcionam as ferramentas de geração de código descrevendo o processo de geração de uma aplicação web incluindo os seus requisitos e a sua arquitetura. Mostra exemplos práticos de gerações e por fim apresenta um caso de geração destrutiva de código e o seu problema.

2.1 Ferramentas de geração de código

De um modo geral, as ferramentas de geração de código criam um projeto que contempla uma arquitetura base. É possível ter um gerador que, seguindo um conjunto de regras, crie vários tipos de projetos com diversas arquiteturas como por exemplo, o JHipster. Para o caso descrito, optou-se por geradores de aplicações web com arquiteturas em três níveis devido não só às vantagens das aplicações web anteriormente referidas, como também pela observação de várias aplicações desenvolvidas onde a maioria se enquadrava neste tipo.

2.2 Requisitos de uma aplicação gerada

Como em todos os sistemas de software, as aplicações geradas também têm os seus próprios requisitos [8]. Os principais requisitos estão listados abaixo:

- Funcionalidade desde a camada de base de dados à interface gráfica.
- Conexão a diferentes tipos de base de dados e funcionamento de forma similar. Clientes diferentes podem funcionar com infraestruturas já montadas que contêm as suas base de dados.
- Boa estruturação e documentação

- Divisão da arquitetura por camadas de modo a ser possível alterações numa camada sem impactar na outra. Deve por exemplo ser possível mudar o aspeto visual apenas alterado a camada de apresentação.
- Permissão de alterações ao código gerado de forma rápida e fácil.
- Facilidade na atualização ou desenvolvimento de novas funcionalidades.
- Debug na aplicação deve ser fácil, rápido e transparente.
- Utilização das tecnologias mais recentes.
- Funcionalidades comuns às aplicações web como logging, multilinguagem, auditoria, etc.

2.3 Arquitectura de uma aplicação gerada

A arquitetura de uma aplicação é a estrutura base do sistema [80]. Uma boa arquitetura ajuda a definir atributos tais como desempenho, qualidade, escalabilidade, portabilidade, capacidade de manutenção e usabilidade. A não existência de uma arquitetura ou a má implementação desta, pode levar a alguns problemas básicos, tais como a alteração de código em determinada área passar a criar bugs em outras áreas da aplicação, fazendo com que adicionar novas funcionalidades à aplicação possa torna-se extremamente difícil. Um gerador de código garante que a aplicação implementa uma boa arquitetura obrigando os programadores a seguirem os seus padrões e implementações.

A arquitectura de três camadas, que separa as aplicações em três camadas de computação lógica e física, é a arquitectura de software predominante para as aplicações cliente-servidor tradicionais [24]. O Padrão de arquitetura em três níveis fornece meios para estruturar e decompor aplicações em três camadas, cada camada fornece um diferente nível de responsabilidade. Um nível lida com a parte da apresentação do sistema, outro trata da lógica de negócio, sendo o núcleo do sistema e a última camada representa o armazenamento de dados [28]. A estrutura de uma camada de três níveis é mostrada na (Fig.2.1)

Apesar da arquitetura de multicamada de três níveis ser a arquitetura mais amplamente adotada, existem outras como a arquitetura a duas camadas, esta é a arquitetura cliente-servidor original, constituída por um nível de apresentação e um nível de dados onde a lógica empresarial pode existir no nível de apresentação, no nível de dados ou em ambos. Na arquitetura de dois níveis, a camada de apresentação - e consequentemente o utilizador final - tem acesso direto à camada de dados, e a lógica empresarial é muitas vezes limitada. Existe também a arquitetura de múltiplos níveis que se refere a qualquer arquitetura de aplicação com mais de um nível. Mas aplicações com mais de três camadas são raras, porque as camadas adicionais oferecem poucos benefícios e podem tornar a aplicação mais lenta, difícil de gerir e mais cara de executar. Como resultado, arquitetura

de N camadas e arquitetura multicamadas são normalmente sinónimos de arquitetura de três camadas [24].As vantagens de uma arquitetura de três camadas são:

- Desenvolvimento mais rápido: Como cada camada pode ser desenvolvida simultaneamente por equipas diferentes, uma organização pode levar a aplicação ao mercado mais rapidamente e os programadores podem utilizar as linguagens e ferramentas mais recentes e melhores para cada camada.

- Escalabilidade melhorada: Qualquer camada pode ser escalonada conforme necessário independentemente das outras.

- Fiabilidade melhorada: Uma paragem numa camada é menos suscetível de ter impacto na disponibilidade ou desempenho dos outras camadas.

- Segurança melhorada: Como a camada de apresentação e a camada de dados não podem comunicar diretamente, a camada de negócio pode funcionar como uma espécie de firewall interno, impedindo injecções de SQL e outras explorações maliciosas.

- Proporciona uma facilidade de manutenção: Estando a camada de apresentação separada da camada de negócio, se existir uma mudança na lógica de negócio esta não tem impacto na camada de apresentação.

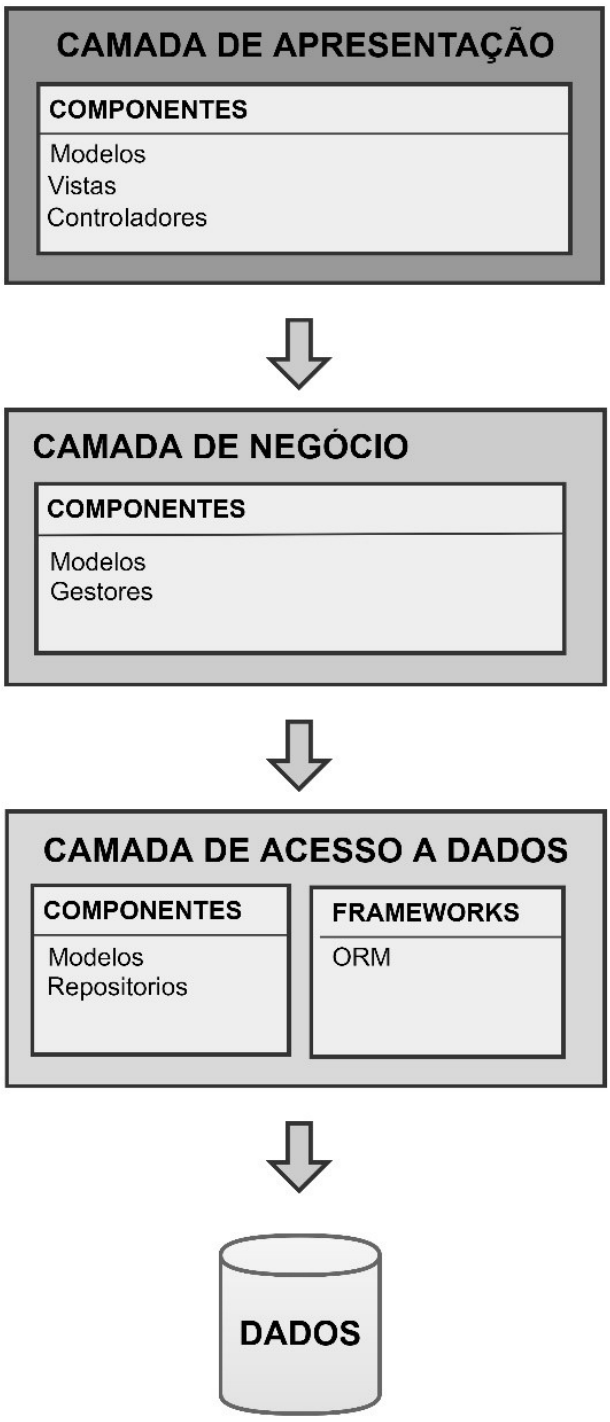


Figura 2.1: Exemplo prático de uma arquitetura de três níveis.

O elemento "Dados" representa a base de dados, onde a informação da empresa está guardada. Sistemas de base de dados mais comuns poderão ser SQL Server, Oracle, Postgres.

A camada de acesso a dados é responsável por realizar o acesso e a persistência aos dados. Contém também os modelos que representam cada uma das tabelas que se encontram na base de dados. De forma a aceder/manipular a informação das tabelas são usados repositórios para cada uma delas. Para o acesso à base de dados é usada uma framework de ORM (object relational mapping) que nos permite conectar a vários tipos de base de dados de uma forma transparente [64].

A camada de negócio é responsável por armazenar a lógica da aplicação, visto que contempla os modelos de negócio que representam a informação dos modelos de dados (1 para 1) podendo conter propriedades extras que ajudam ao nível do negócio. Já nos gestores, é onde vai ficar toda a lógica de negócio. Para cada modelo existe um gestor.

A camada de apresentação é responsável por realizar a interface com o utilizador, esta integra os modelos de visualização que representam a informação dos modelos de negócio (1 para 1) podendo conter propriedades extras que ajudem ao nível da visualização. Os controladores contêm as rotas para cada uma das páginas, e as vistas contêm o conteúdo das páginas. A camada de apresentação segue o padrão de desenho Model-View-Controller (MVC). O padrão de desenho é um dos mais importantes padrões de desenho em ciência da computação. Enquanto a maioria dos padrões aborda problemas específicos, o MVC descreve a arquitetura de um sistema de objetos [67]. Pode ser aplicado a subsistemas isolados ou a aplicações inteiras [13]. As suas vantagens são:

- Modularidade
  O modelo de desenho MVC é uma extensão dos princípios de separação de interesses e encapsulamento. A separação de Interesses declara que o software deve ser separado de acordo com os tipos de trabalho que ele executa. Por exemplo, uma aplicação que inclua uma lógica para identificar itens importantes a serem exibidos ao utilizador, e que formata esses itens de uma maneira específica para torná-los mais percetíveis. O comportamento responsável por escolher os itens a serem formatados deve ser mantido separado do comportamento responsável por formatar os itens, uma vez que esses comportamentos são questões separadas que são relacionadas apenas coincidentes entre si [57]. Diferentes partes de uma aplicação devem usar o encapsulamento para isolá-las de outras partes da aplicação. As camadas e os componentes da aplicação devem poder ajustar a sua implementação interna sem dividir seus colaboradores, desde que contratos externos não sejam violados. O uso adequado do encapsulamento ajuda a obter um acoplamento flexível e uma modularidade nos designs da aplicação, pois os objetos e os pacotes podem ser substituídos por implementações alternativas, desde que a mesma interface seja mantida [61].

- Flexibilidade
  A característica mais poderosa de MVC é provavelmente a capacidade de substituir sem esforço as vistas, ou utilizar múltiplas vistas, sem alterar o modelo de dados

ou o controlador. Um exemplo disso é a aplicação iTunes. A aplicação iTunes da Apple utiliza vários objetos diferentes de visualização para exibir o conteúdo da sua biblioteca de música e vídeo. Há uma vista de tabela, um browser, uma vista de miniaturas e uma vista de capas. Todos os quatro apresentam a mesma informação do mesmo modelo de dados, apenas de formas diferentes. Ao separar o modelo de dados das vistas, podem ser desenvolvidas novas vistas sem fazer quaisquer alterações ao modelo de dados ou às outras vistas. Mesmo grande parte da lógica do controlador, tal como a seleção da canção atual, o arrastar e largar, a ação de reprodução da canção e a obtenção de informações podem ser implementadas uma vez e utilizadas alternadamente com qualquer uma das quatro vistas [9]. Programar com objetos MVC bem concebidos é como trabalhar com um conjunto Lego gigante. Qualquer objeto (modelo de dados, controlador ou vista) é intercambiável com qualquer objeto funcionalmente equivalente. A substituição de objetos, ou a anexação de múltiplos objetos, não perturba o resto do desenho. O mantra do padrão de desenho MVC é criar aplicações complexas através da interligação de objetos simples.

- Reutilização
  A reutilização está relacionada com a flexibilidade. Ao se abstrair a funcionalidade das classes, os objetos podem ser reutilizados em outras aplicações ou para outros fins. Os objetos reutilizados mais frequentemente são modelos de dados e as vistas.

- Baixo acoplamento e alta coesão
  Na programação orientada a objetos é considerada má prática misturar apresentação e código de dados. Em vez disso, as classes chamadas controladores são definidas para mediar a comunicação entre a apresentação e as classes de dados. Estas classes de Controladores desacoplam a camada de apresentação das classes da camada de dados, permitindo, consequentemente, que sejam definidas mais especificamente. Promove a alta coesão reduzindo a complexidade dos módulos, aumentando a reutilização de módulos e aumentando a capacidade de manutenção do sistema. A alta coesão cria uma separação de preocupações permitindo a especialização e focalização do programador como também o desenvolvimento paralelo se trabalhar em equipa. Model-View-Controller (MVC) é tipicamente composto de três classes como mencionado no seu nome [37] [88].

- Processo de desenvolvimento mais rápido
  Devido à sua separação é possível os programadores estarem paralelamente a trabalhar nos modelos, controladores ou vistas. É uma vantagem em comparação com outros padrões de desenho. Isto torna o trabalho mais rápido, poupa tempo, e ajuda a gerir os recursos de forma eficaz.

- Facilidade e especificidade em testes
  Devido também ao baixo acoplamento e alta coesão da arquitetura é possível facilmente construir uma coleção de testes específicos que podem ser executados

rapidamente [23]. Testes unitários podem ser criados diretamente sobre as ações dos controladores de modo a testar se a funcionalidade se encontra correta. [82].

Como um exemplo da arquitetura de três níveis é demonstrando uma aplicação que a única função é mostrar a data de nascimento e idade de um utilizador.

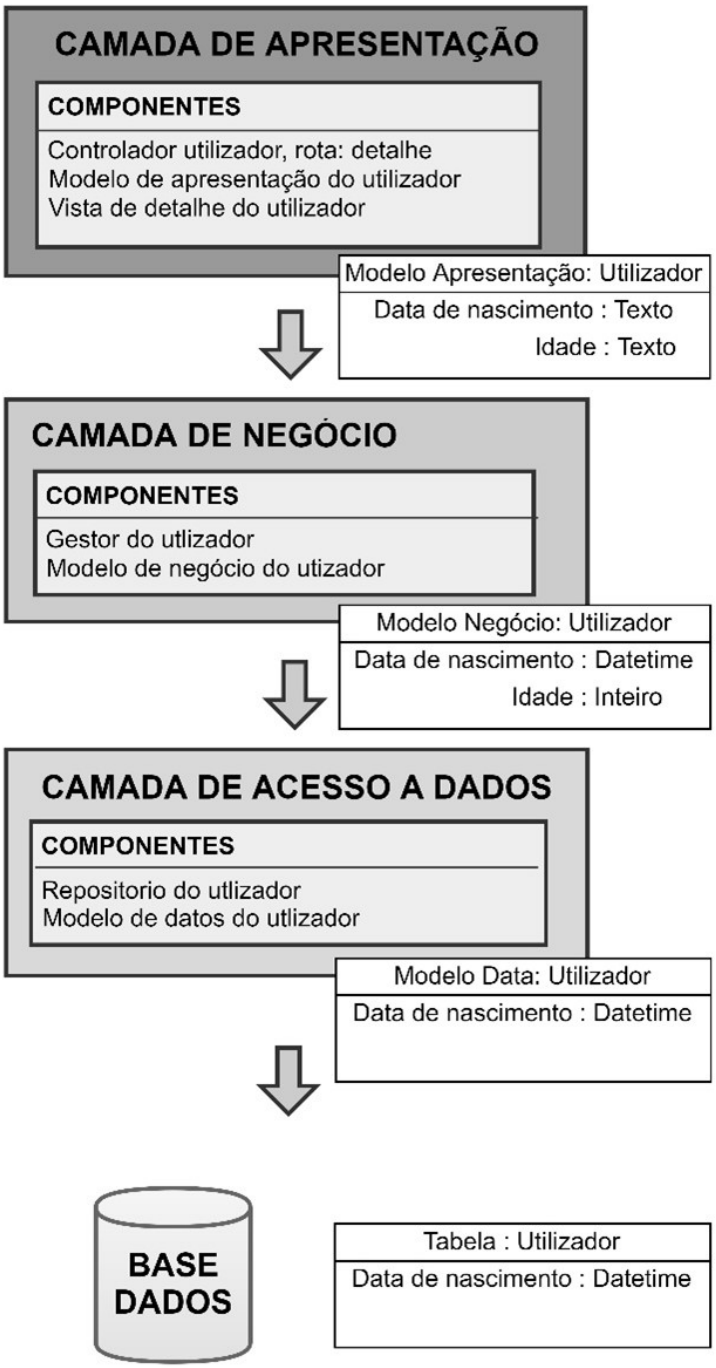


Figura 2.2: Exemplo das varias camadas e seu conteúdo em uma arquitectura de três níveis

Implementação:

1. Deve existir uma base de dados que conte com uma tabela "Utilizador" com um campo "Data nascimento". Neste campo está guardada a data de nascimento do utilizador.

2. Na camada de acesso de dados deve existir uma classe repositório do utilizador que conte com um método "GetDataNascimento", a funcionalidade deste método prende-se com o facto do ORM ir ler a data nascimento do utilizador à base de dados e mapeá-la para o modelo de dados do utilizador. O modelo de dados do utilizador contém uma propriedade "Data de nascimento".

3. A camada de negócio deve conter uma classe gestora do utilizador que implementa um método "GetDataNascimento" em que a sua funcionalidade é invocar o método "GetDataNascimento" da classe repositório do utilizador e com o facto da informação proveniente do modelo de dados do utilizador preencher a propriedade "Data de nascimento" do modelo de negócio assim como preencher a propriedade "Idade" com a diferença da data atual e da data de nascimento. Este cálculo pode ser considerado o negócio da aplicação.

4. Na camada de apresentação deve existir uma classe controladora do utilizador com uma ação "Detalhe", esta ação detalhe deve invocar o método "GetDataNascimento" da classe gestora do utilizador e com a informação proveniente do modelo de negócio do utilizador preencher as propriedades "Idade" e "Data de nascimento" do modelo de visualização, retirando a parte da informação das horas à propriedade "Data de nascimento". Podemos considerar esta alteração uma modificação para a visualização. A ação "Detalhe" por fim, deve invocar a vista "Detalhe" do utilizador onde será apresentada as propriedades do modelo de visualização do utilizador.

No exemplo (Fig.2.2) apresentado foi apenas demonstrada a ação de visualização de uma data de nascimento. Mas observando várias aplicações web verifica-se que existem ações comuns à maior parte destas. Estas operações, são as operações create, update, read e delete (CRUD). A (Fig. 2.3) demonstra que classes e métodos devem ser criados para implementar um CRUD na arquitetura de três camadas.

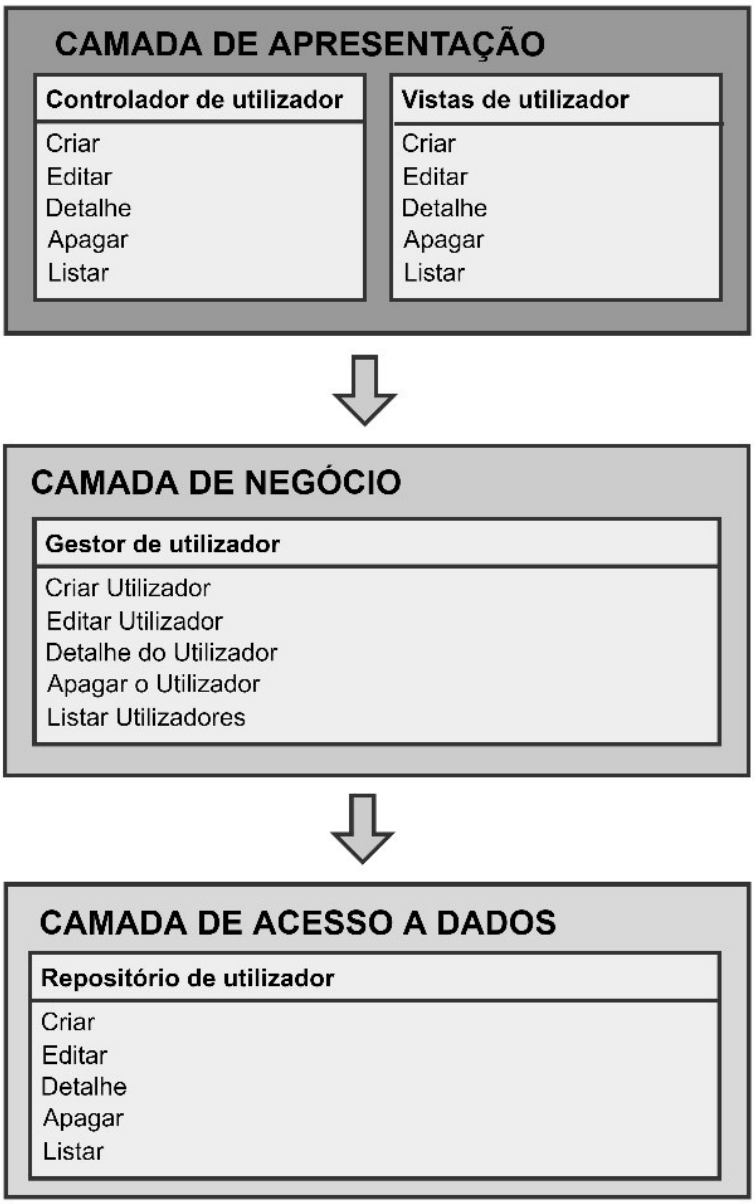


Figura 2.3: Exemplo de CRUD nos três níveis

2.4 Processo de geração

Observou-se que no desenvolvimento de aplicações web normalmente as empresas usam a base de dados como a sua principal fonte de dados. Partindo deste prossuposto, a base de dados pode ser considerada o ponto de partida para o desenvolvimento. No processo de desenvolvimento de uma aplicação existem duas filosofias concorrentes, code first onde o programador cria os modelos de dados e a base de dados é criada a partir destes, e o database first que é o inverso, ou seja, os modelos de dados são criados com base na base de dados previamente criada pelo programador [49]. O code first apresenta desvantagens como:

- É necessário um bom conhecimento da linguagem de programação do ORM e das suas convenções.
- Escrever objetos da base de dados como stored procedures e triggers é complexo.
- Se houver alguma modificação feita na base de dados, esta não será refletida nas entidades da aplicação.

e vantagens como:

- Sincronização entre código e base de dados.
- O controlo absoluto sobre o código pois este sendo gerado pelo programador e não auto gerado como no database first, torna-se fácil de modificar.
- Não existe a necessidade de verificar as alterações das tabelas na base de dados.
- Todas as alterações da base de dados são guardadas (migrações) podendo voltar a qualquer versão, conforme necessário.
- Uma API de código fluente baseada em atributos que permite ao programador seguir uma abordagem de convenção sobre configuração.
- Suporta migrações de bases de dados, o que torna possível manter várias bases de dados em sincronia.

Com base nestas vantagens, no processo de desenvolvimento foi decidido que não só a aplicação gerada deve usar a abordagem de code first, como o próprio gerador de código terá como base os modelos de dados. A partir dos modelos de dados em direção à camada de apresentação serão geradas todas as classes necessárias para implementar os Create Read Update Delete (CRUD) (Fig.2.3) correspondentes nas várias camadas. A criação dos modelos de dados antes de iniciar o projeto oferece muitos benefícios visto que os modelos de dados servem como uma visão geral do sistema [6].

A linguagem de programação C# e a framework ASP.NET Core MVC. foram utilizadas para a demonstração da aplicação gerada.

1. Configurações

   Um ficheiro de configuração dará a informação base para o gerador começar o seu processo. Este ficheiro deverá conter o namespace do projeto, tipo de base de dados, connection string para se ligar, línguas com que a aplicação deverá funcionar e os seus módulos de funcionalidades como por exemplo importar ou exportar os dados de uma tabela.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Config>
  <Project>
    <Namespace>Example.Project</Namespace> <!-- Project Namespace -->
  </Project>
  <Database>
    <Type>SQLServer</Type> <!-- Database Type : SQLServer , PostgreSQL, SQLite  -->
    <ConnectionString>Server=localhost;....</ConnectionString> <!-- Database connection string -->
  </Database>
  <Languages>pt</Languages><!-- Project Extra Languages, default is English -->
  <Modules>
    <Module>SoftDelete</Module> <!-- Create models soft delete "IsDeleteted" insted of real delete -->
    <Module>Validators</Module> <!-- Add a pack of data annotations validations -->
    <Module>AuditTrail</Module> <!-- Audit trail actions -->
    <Module>ImportExport</Module> <!-- Excel Import/Export data from tables -->
  </Modules>
</Config>
```

Exemplo de ficheiro de configuração. (XML)

2. Arquitetura base

   De forma a não ser necessário gerar todos os ficheiros necessários para a arquitetura base e funcionamento de uma aplicação, sugere-se a cópia de uma solução base (Fig. 2.4) acelerando assim o processo. Com base na informação do ficheiro de configuração, durante a cópia deve ser feita a mudança de nomes dos namespaces nos ficheiros. O resultado será uma aplicação que corre, mas sem funcionalidades.

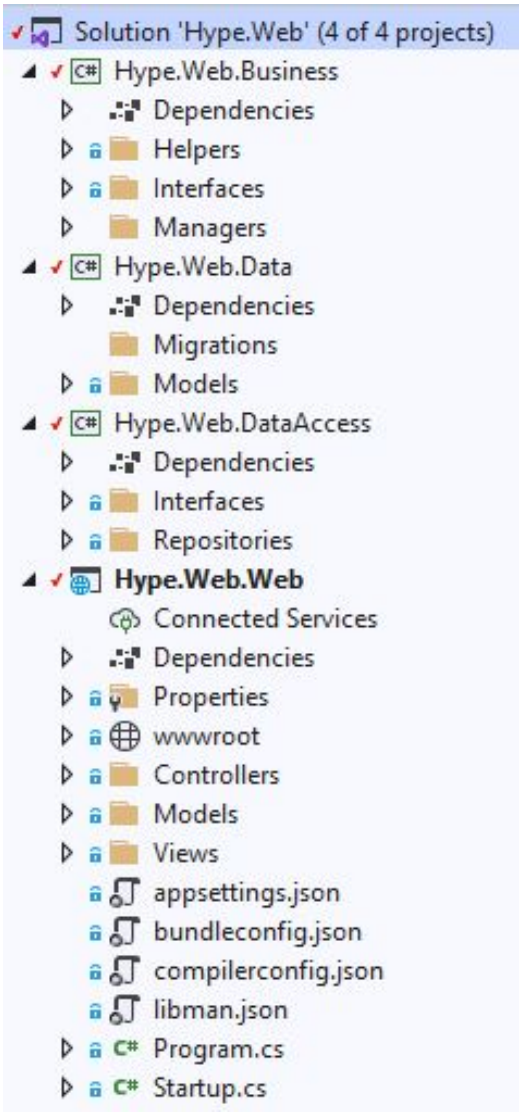


Figura 2.4: Exemplo de um projeto/arquitetura base em C#/ASP.NET Core MVC

3. Geração de classes

   Como mencionado anteriormente o gerador irá basear-se nos modelos criados pelo programador para gerar as várias classes necessárias. Os modelos de dados devem ser criados pelo programador usando a linguagem em que a aplicação vai ser gerada. Criar uma sintaxe própria exigirá uma aprendizagem e tempo desnecessários para quem já trabalha regularmente com a linguagem em causa. Os modelos dados devem primeiramente ser lidos e a sua informação guardada numa estrutura.

```
1 public partial class Role
2 {
3   [key]
4   public int Id {get;set;}
5   public string Name {get;set;}
6 }
```

```
public partial class User
{
  [key]
  public int Id {get;set;}
  public string Name {get;set;}
  public Role Role {get;set;}
}
```

Exemplo de modelos criados pelo programador

```
{
  "tables": [
    {
      "name": "User",
      "properties": [
        {
          "name": "Id",
          "dateType": "Guid",
          "primaryKey": true
        },
        {
          "name": "Name",
          "dateType": "string",
          "default": ""
        },
        {
          "name": "RoleId",
          "dateType": "int",
          "foreignKey": true
        }
      ]
    },
    {
      "name": "Role",
      "properties": [
        {
          "name": "Id",
          "dateType": "int",
          "primaryKey": true
        },
        {
          "name": "Name",
          "dateType": "string",
          "default": ""
        }
      ]
    }
  ]
}
```

Exemplo da estrutura de informação dos modelos de dados. (JSON)

A estrutura irá conter toda a informação necessária para gerar as classes necessárias. Para criar os repositórios o objetivo é criar uma classe para cada tabela que implemente um método de criar, editar, apagar, obter todos e obter apenas um. Para obtermos os nomes das classes basta percorrer a estrutura previamente criada. O seguinte código pode ser melhorado criando uma AST e a partir da AST produzir o código final. Mas para efeitos de demonstração foi simplificado.

```
public void CreateRepository()
{
  //Percorrer todas as tabelas da estrutura
  foreach (var table in structure.Tables)
  {
    // Criar uma string para guardar o conteúdo da classe
    string content = "";
    content += "public class " + table.Name + "Repository";
    content += "{";
    // Chamar os vários métodos com o conteúdo dos métodos a gerar
    content += GetInsertMethod(table.Name);
    content += GetUpdateMethod(table.Name);
    // ...
    content += "}";
    // Escrever o ficheiro
    File.WriteAllText(table.Name + "Repository.cs", content);
  }
}
//Criação do método de inserir
public string GetInsertMethod(string tableName)
{
  //Criar uma string para guardar o conteúdo do método
  string result = "";
  result += "public " + tableName + " Insert(" + tableName + " "  +
    tableName.ToLower() + ")";
  result += "{";
  result += " var result = await DbContext.Add( " + tableName.ToLower() +
     ");";
  result += " await DbContext.SaveChanges();";
  result += " return result.Entity";
  result += "}";
  return result;
}
```

Exemplo de criação dos vários repositórios

```
public class UserRepository
{
  public User Insert(User user)
  {
    var result = await DbContext.Add(user);
    await DbContext.SaveChanges();
    return result.Entity;
  }
}
```

Exemplo de um repositório gerado

A técnica de criação dos gestores e controloradores será igual a dos repositórios. Em relação aos modelos das várias camadas, os modelos de dados podem ser directamente copiados para a pasta de modelos da camada de dados. (Fig 2.4) Hype.Web.Data. Enquanto para a geração dos modelos do negócio ou visualização, não só deve ser percorrida a estrutura para obter os nomes das tabelas como também as suas propriedades.

```
//Percorrer todas as tabelas da estrutura
foreach (var table in structure.Tables)
{
  // Criar uma string para guardar o conteúdo da classe
  string content = "";
  content += "public class " + table.Name + "BusinessModel";
  content += "{";
  // Para cada tabela percorrer todasd as propriedades
  foreach (var property in structure.Properties)
  {
    content += "public " + property.Datatype + " " + property.Name + " {
    get;set;}";
  }
  content += "}";
  // escrever o ficheiro
  File.WriteAllText(table.Name + "BusinessModel.cs", content);
}
```

Exemplo de criação dos vários modelos de negócio

```
public class UserBusinessModel
{
  public int Id {get;set;}
  public string Name {get;set;}
  public Role Role {get;set;}
  public boolean IsMinor {get;set;}
}
```

Exemplo de um modelo de negócio

As vistas têm uma abordagem semelhante aos modelos mas a geração terá de ter lógica em relação ao tipo de dados. Se por exemplo estivermos a gerar uma vista de criação em Hypertext Markup Language (HTML), e tivermos uma propriedade de visibilidade "isVisible" do tipo booleana faz sentido criar uma checkbox, mas se existir uma propriedade observações do tipo string faz sentido criar uma caixa de texto.

```
1  //Percorrer todas as tabelas da estrutura
2  foreach (var table in structure.Tables)
3  {
4    // Criar uma string para guardar o conteúdo da classe
5    string content = "";
6    content += "<Html><Head><Title>Criar "+table.Name+"</Title></Head>";
7    content += "<Body>";
8    // Para cada tabela percorrer todasd as propriedades
9    foreach (var property in structure.Properties)
10   {
11     switch(property.Datatype)
12     {
13       case "boolean" :
14         content += "<input type=\"checkbox\" name=\""+property.Name+"\">"
     ;
15       break;
16       case "string" :
17         content += "<input type=\"text\" name=\""+property.Name+"\">";
18       breaj;
19     }
20   }
21   content += "</body></html>";
22   // escrever o ficheiro
23   File.WriteAllText(table.Name + "\Create.cshtml", content);
24 }
25
26 <Html><Head><Title>Criar User</Title></Head>
27 <Body>
28   <Input type="checkbox" >
29 </Body></Html>
```

Exemplo de criação das várias vistas de criação

```
1 <Html><Head><Title>Criar User</Title></Head>
2 <Body>
3     <Input type="text" name="Name">
4     <Input type="checkbox" name="IsMinor">
5 </Body></Html>
```

Exemplo de uma vista de criação

4. Base de dados

   O ficheiro de contexto da base de dados deve ser criado com base nas configurações previamente lidas sendo que de seguida será possível criar e executar uma migração resultado na criação de uma base de dados baseada nos modelos de dados criados inicialmente. Estas operações são possíveis pois foi decido que a aplicação funcionaria com a abordagem code first.

5. Interface visual

   Quanto se copia o projeto base no início do processo, por exemplo, se a interface gráfica for HTML, poderá copiar-se também folhas de estilos (CSS) ou frameworks visuais como o Boostrap de maneira a tornar a interface com o utilizador mais agradável.

6. Modulos

   Existem funcionalidades que não fazem sentido para todas as aplicações geradas, por exemplo um cliente de lojas de retalho pode querer exportar todos os registos da tabela faturas para Excel, mas para outro cliente essa tabela pode nem existir. Funcionalidades que não sejam 90% recorrentes devem ser tratadas como módulos e não deverão ser incluídas no projeto base. Os módulos poderão funcionar como plugins, ou seja, depois de todo o projeto estar gerado e a funcionar poderá ser aplicado um módulo. Se quisermos adicionar o módulo de exportação nas faturas, este deverá criar métodos de exportação nos controladores e nos gestores. Para isso, os respetivos ficheiros devem ser editados e acrescentados os respetivos métodos.

Como referido de um modo geral as ferramentas de geração de código para aplicações web criam um projeto base, uma arquitetura, funcionalidades comuns, como logging, segurança e exportação de dados e, a partir de um modelo ou entidade da base de dados, geram todos os ficheiros necessários para o funcionamento do CRUD dessa entidade. Estes ficheiros são gerados em todas as camadas da aplicação, desde a camada de acesso à base de dados até à interface gráfica.
Os geradores de código conseguem realizar estas tarefas de um modo rápido e objetivo pois, as aplicações web seguem uma arquitetura comum e contêm um conjunto de funcionalidades iguais. Contudo, geralmente, a lógica de negócio entre estas é diferente pois é específica deste mesmo negócio, o que faz com que, na maioria dos casos após a geração de código, surja a necessidade de alterar o código gerado. A criação e alteração de código numa camada de negócio leva a que também sejam necessárias alterações nas restantes camadas.

2.5 Exemplos práticos

Os próximos exemplos mostram métodos de criação que foram gerados para uma aplicação de gestão de tickets. Estes métodos fazem parte de um sistema CRUD que é gerado automaticamente com base no modelo correspondente. Ou seja, com base no modelo Ticket que contem as suas propriedades são gerados automaticamente os métodos para o criar, ler, editar e apagar.

Linguagem: Java
Gerador: JHispter
Descrição: Este método recebe um modelo Ticket e através do repositório "TicketRepository" grava-o na base de dados. Um modulo de log foi ativado para a geração. (log.debug)

```
@PostMapping("/tickets")
public ResponseEntity<Ticket> createTicket(@Valid @RequestBody Ticket ticket)
    throws URISyntaxException {
    log.debug("REST request to save Ticket : {}", ticket);
    if (ticket.getId() != null) {
        throw new BadRequestAlertException("A new ticket cannot already have
    an ID", ENTITY_NAME, "idexists");
    }
    Ticket result = ticketRepository.save(ticket);
    return ResponseEntity.created(new URI("/api/tickets/" + result.getId()))
        .headers(HeaderUtil.createEntityCreationAlert(applicationName, true,
    ENTITY_NAME, result.getId().toString()))
        .body(result);
}
```

Método de criação de 1 ticket

Linguagem: PHP
Gerador: YII Framework
Descrição: Este método cria um Ticket que é preenchido com o request de uma post vindo de um formulário e grava-o na base de dados.

```
/**
     * Creates a new Ticket model.
     * If creation is successful, the browser will be redirected to the 'view'
     page.
     * @return mixed
     */
    public function actionCreate()
    {
        $model = new Ticket();

        if ($model->load(Yii::$app->request->post()) && $model->save()) {
            return $this->redirect(['view', 'id' => $model->id]);
        }

```

```
14          return $this->render('create', [
15              'model' => $model,
16          ]);
17      }
```

Método de criação de 1 ticket

Após a necessidade da criação de um registo de auditoria na aplicação e também o redireccionamento para a vista de listagem de tickets em vez da vista de detalhe cada vez que existe a criação de um ticket, são criadas as seguintes alterações:

```
1  @PostMapping("/tickets")
2  public ResponseEntity<Ticket> createTicket(@Valid @RequestBody Ticket ticket)
       throws URISyntaxException {
3      log.debug("REST request to save Ticket : {}", ticket);
4      if (ticket.getId() != null) {
5          throw new BadRequestAlertException("A new ticket cannot already have
       an ID", ENTITY_NAME, "idexists");
6      }
7      Ticket result = ticketRepository.save(ticket);
8      Audit resultAudit = auditRepository.save(new Audit(ENTITY_NAME, ticket.Id,
        new Date());
9      return ResponseEntity.created(new URI( [*"/api/tickets/"*]))
10         .headers(HeaderUtil.createEntityCreationAlert(applicationName, true,
       ENTITY_NAME, result.getId().toString()))
11         .body(result);
12 }
```

Exemplo do método gerado com o JHispter alterado pelo programador

A classe ticket que representa a tabela Ticket na base de dados originalmente foi criada com as propriedades identificador, data e descrição. Se após estas alterações fosse necessário adicionar uma propriedade à classe Ticket, como por exemplo o tipo e ao correr novamente o gerador todo método "CreateTicket" será rescrito como o original e todas as alterações serão perdidas. Neste caso a linha 8 desaparecia e a linha 9 que contêm o redireccionamento para a vista de listagem voltaria a conter o redireccionamento para o detalhe do ticket.
As funcionalidades desenvolvidas pelo programador desaparecem após nova geração. Este problema estrutural é comum em diversas ferramentas de geração de código. Quando é necessário alterar uma entidade que leva a uma segunda geração de código, ou existe uma atualização na ferramenta, esta torna-se destrutiva. Acontece que todo o código é reescrito e as alterações feitas a este são apagadas.

Alterar código ou atualizar ferramentas são situações recorrentes no desenvolvimento de código pois é normal que as aplicações não sejam um produto final e vão evoluindo com o tempo. As ferramentas de geração de código são também atualizadas, seja para adicionar novas funcionalidades, corrigir bugs ou disponibilizar novas versões das frameworks que usam [63]. Idealmente o código gerado por estas ferramentas deveria ser fechado,

não deixando que programador o alterasse, mas na realidade isto não é uma situação prática pois os modelos de negócio funcionam de forma diferente entre as várias empresas. Mesmo uma empresa que seja do mesmo país ou do mesmo setor, tem processos diferentes de uma empresa concorrente. Por vezes, mesmo processos semelhantes são constituídos por atividades ou tarefas diferentes.

Numa empresa com uma loja online, num processo de venda, podem existir atividades como:

- Dar baixa do produto.
- Comunicar com o armazém para começar a distribuição
- Comunicar com transportadora escolhida para gerar a encomenda.
- Comunicar com uma entidade externa para pagamento (Ex: Paypal).
- Enviar email ao cliente com fatura.
- Auditar esta atividade.

As atividades enumeradas são atividades usuais num processo de venda online, onde poderiam ser pré escolhidas num gerador conforme os requisitos mas na maioria das vezes não seriam suficientes. Basta, por exemplo, a aplicação ter de comunicar com uma entidade externa para pagamento, que ainda não exista no gerador, ou que dentro do processo tenha de comunicar com um software de faturação proprietário da empresa, que já não será possível gerar uma aplicação com as funcionalidades a 100%.É impossível existir um gerador que tenha todas estas possibilidades em consideração. Posto isto, após a geração de uma aplicação, por mais simples que seja o negócio da empresa é sempre necessário existirem alterações por parte dos programadores. Pretende-se, então, chegar a uma solução onde, após novas gerações de código, o novo código gerado possa conviver com o código já desenvolvido pelo programador. Para atingir este objetivo propõe-se o estudo e desenvolvimento de várias técnicas de comparação e análise de código de modo a tornar a geração de código não-destrutiva.

2.6 Trabalho relacionado

Atualmente, já existem soluções implementadas nos programas de edição de desenvolvimento que são utilizados pelos programadores, por exemplo, as sugestões automáticas no editor de Java IntelliJ [42] ou o IntelliSense no Visual Studio Code [60]. A indústria tem trabalhado no sentido de fornecer recomendações enquanto o programador escreve o código mas, e se fosse possível fornecer toda a solução apenas olhando para a especificação do software?

A geração automática de código a partir da linguagem natural é um campo importante para prever código explícito ou estrutura de programas a partir de fontes de dados multimodais, tais como código incompleto, programas noutra linguagem de programação,

descrições de linguagem natural ou exemplos de execução [3]. As técnicas Deep learning (DL) para processamento de linguagem natural (LN) têm vindo a evoluir de uma forma notavelmente rápida. Recentemente, os avanços do DL na modelação de línguas, tradução automática e compreensão de parágrafos têm sido tão proeminentes que o potencial do DL na Engenharia de Software não pode ser ignorado.

Em 2017, Pengcheng Yin e Graham Neubig propuseram um modelo de rede neural baseado na sintaxe dos dados e adaptado para a geração de linguagens de programação comuns como o Python. A fim de captar a forte sintaxe subjacente da linguagem de programação, é definido um modelo que transforma uma declaração de língua natural (NL) numa árvore de sintaxe abstrata (AST) construída na linguagem de programação alvo. Uma vez gerada a AST são utilizadas ferramentas de geração determinísticas para converter a AST em código [93].

Também em 2017 Antonio Barone e Rico Sennrich propuseram usar as "docstring" (descrições das funcionalidades dos métodos) que existem nas classes de interface programadas na linguagem Python como input de descrições em linguagem natural e com este input treinaram modelos de tradução automática neural (NMT) para a geração de código [86].

Em 2020 S. Shim, P. Patil, R. R. Yadav, A. Shinde e V. Devale inspirados no conceito de Programação por exemplos [35] criaram uma solução que usa este modelo como base para alcançar a geração do programa final. A mesma consiste em treinar uma rede neural com um conjunto de dados que contém vários exemplos de entradas-saídas juntamente com os seus programas e que devolve uma matriz de probabilidade. Estas probabilidades indicam se uma determinada função aparecerá no programa final. Uma maior probabilidade indica que a função é mais provável que esteja presente no programa gerado. Após a geração da matriz de probabilidade, foi implementado uma estratégia de pesquisa da Depth First Search que pesquisa nas previsões da rede neural de forma a descobrir o código mais adequado [78].

Estes são apenas alguns exemplos de estudos que existem na área de geração de código baseados em linguagem natural. É expectável que esta área de investigação acabe por melhorar a replicação da capacidade humana de desenvolver algoritmos e que fique intrínseca à geração de código automática. No presente momento, problemas como dependências complexas entre funções e classes, dependências de bibliotecas externas ou mesmo o tamanho do código torna complicada a tarefa de construir um gerador de código realmente funcional.

Capítulo 3

Frameworks e geradores de código

Hoje em dia as linguagens de programação web mais comuns são o Javascript, Python, Php, Java e .Net [89]. No mercado existem disponíveis diversas frameworks e geradores de código que implementam funcionalidades semelhantes para estas linguagens de desenvolvimento. De forma a avaliar estas frameworks e geradores de código foram criadas pequenas aplicações onde se avaliaram os seguintes pontos:

- Número de funcionalidades disponíveis;
- Facilidade e rapidez do desenvolvimento (aprendizagem, codificação e depuração);
- Uso de tecnologias atuais;
- Uso de boas práticas e arquitetura;

3.1 Análise de frameworks e geradores de código

Este tópico demonstra as frameworks e geradores de código que, devido às suas vantagens e capacidades mais se tornam aptas para a geração e desenvolvimento de código tendo como avaliação os pontos enumerados na introdução deste capitulo.

1. Php YII [91]

   "Yes, it is" é uma framework de alta performance da linguagem PHP. É rápida, segura, eficiente e flexível, mas pragmática. É baseada em componentes, altamente escalável e permite a máxima reutilização de código. Implementa o princípio de desenvolvimento MVC. Oferece também bastantes ferramentas para automatizar muitas das tarefas repetitivas em projetos. Isto permite que programadores se concentrem nos requisitos centrais do negócio e na lógica empresarial. Yii oferece uma

série de mecanismos de segurança tais como XSS, Cross Site Scripting, também prevenção de Cross Site Request Forgery (CSRF) e permite a fácil integração de cache que melhora o desempenho da aplicação ao reduzir o tempo de resposta e com mais velocidade. Existe um grande apoio da comunidade através de fóruns e grupos de discussão [91] [9] [50].

Esta contém no seu web site uma documentação elaborada que seguindo a secção "Getting Started" permite rapidamente criar uma aplicação básica como exemplo. A linguagem Php tem vindo ao longo das últimas versões a tornar-se cada vez mais orientada a objetos.

O YII tira proveito destas novas features criando uma arquitetura de aplicação baseada no padrão de desenho MVC e no padrão Active record. A aplicação criada assenta sobre uma framework robusta, que disponibiliza um conjunto das mais comuns Application Programming Interface (API) de modo a permitir um fácil desenvolvimento. A geração de código fica a cargo do módulo interno GII que se baseia num esquema de base de dados.

Fora a estrutura MVC, a aplicação pode ainda ser construída usando a sua arquitetura modular que nos possibilita criar "mini-aplicações", usar comportamentos que são injetados nas classes permitindo, assim, melhores funcionalidades. Também nos permite instalar extensões desenvolvidas pela comunidade que se encontram disponíveis no site.

As aplicações do Yii são organizadas de acordo com o padrão de desenho MVC. Os modelos representam dados, lógica e regras de negócio; as vistas são a representação da saída dos modelos e os controladores recebem entradas e convertem-nas em comandos para os modelos e as vistas.

Além do MVC, as aplicações do Yii também possuem as seguintes entidades:

- Scripts de entrada: são scripts PHP que são diretamente acessíveis aos utilizadores finais. São responsáveis por iniciar o ciclo de tratamento de uma requisição.
- Aplicações: são objetos globalmente acessíveis que gerem os componentes da aplicação e coordenam-nos para atender às requisições.
- Componentes da aplicação: são objetos registados com as aplicações e fornecem vários serviços para atender às requisições.
- Módulos: são pacotes auto-contidos que contém um MVC completo por si só. Uma aplicação pode ser organizada em múltiplos módulos.
- Filtros: representam código que precisa ser chamado pelos Controllers antes e depois do tratamento propriamente dito de cada requisição.
- Widgets: são objetos que podem ser embutidos em vistas. Podem conter lógica de controlador e podem ser reutilizados em diferentes vistas.

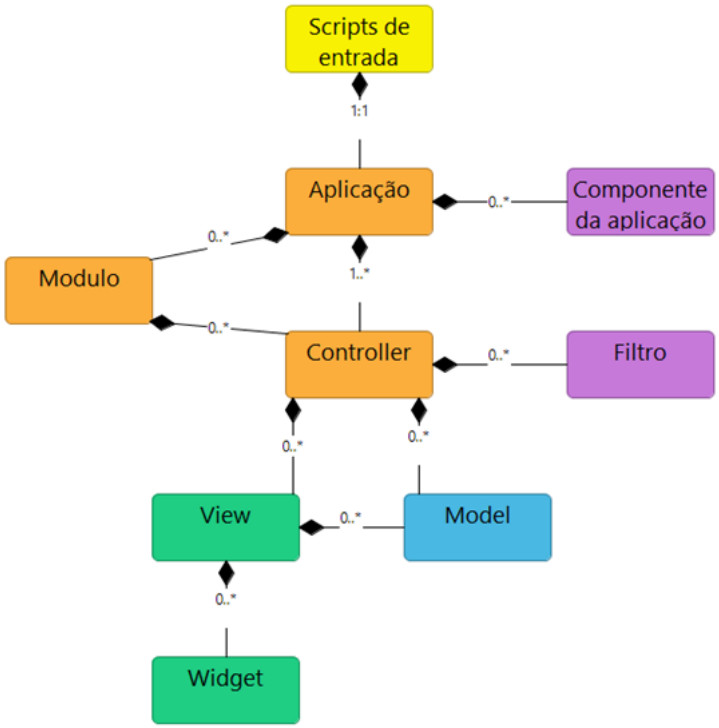


Figura 3.1: Arquitectura de uma aplicação YII

Permite instalar extensões desenvolvidas pela comunidade que se encontram disponíveis no site.

Por exemplo para criar um audit basta recorrer à extensão "yii2-audit-log" que se encontra disponível na página de extensões e seguir as instruções de instalação. Também para criar uma página que mostre os últimos utilizadores criados, e não tendo nenhum conhecimento da framework, basta recorrer às paginas da documentação base (Controllers, Models, Views) e à pagina (ActiveQuery) da API para que rapidamente se desenvolva o código. A framework conta também com vários Helpers para ajudar a criar um front-end limpo e organizado.

Devido à sua estrutura organizada e transparente é possível fazer debug rapidamente desde a página HTML, até à chamada à base de dados, sendo mesmo possível fazer debug ao próprio código da framework, dado este se encontrar disponível. Esta última parte torna-se numa mais valia pois, apesar de existir imensa documentação tanto oficial como proveniente da comunidade, é sempre possível entender o que os métodos disponibilizados pela framework estão internamente a fazer. Isto, de maneira a ganharmos um maior conhecimento desta mesma framekwork e a conseguirmos fazer certas otimizações.

Apesar das linguagens interpretadas serem mais lentas [25], neste caso o PHP acaba por ser uma vantagem no que diz respeito ao debug e ao correr a aplicação também

pois, basta colocar os ficheiros num servidor web como o Apache ou o IIS para rapidamente visualizar a aplicação.

A YII vem também integrada com a framework de tests Codeception e configurada para rapidamente se poder começar a desenvolver e correr testes.

Em suma, a framework é orientada a um desenvolvimento limpo, rápido e simples pois não necessita de grandes conhecimentos prévios e é apoiada por uma boa documentação e uma grande comunidade. É uma das principais frameworks da linguagem PHP estruturando-a ao nível das linguagens de programação mais comuns.

É usado por companhias como a Lenovo, Fujitsu, Deloitte, Descovery [92].

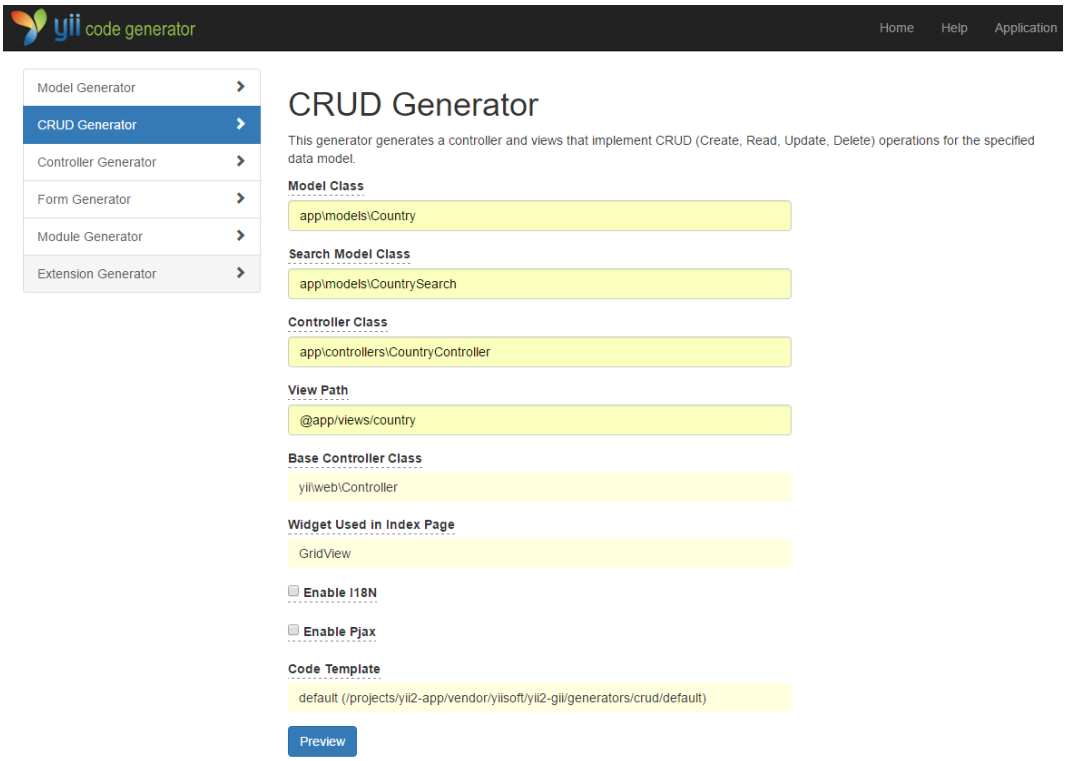


Figura 3.2: YII

2. Django [31]

   Django é uma framework de alto nível web para a linguagem Python que incentiva a um desenvolvimento rápido e uma conceção limpa e pragmática. Construído por programadores experientes, trata de grande parte dos problemas comuns do desenvolvimento web, para que os programadores se possam concentrar em escrever as aplicações sem necessidade de reinventar a roda. É gratuito e de código aberto. Foi concebido para ajudar os programadores a levar as aplicações desde o conceito até à sua conclusão o mais rapidamente possível. Leva a segurança a sério e ajuda os

programadores a evitar muitos erros de segurança comuns. Django inclui dezenas de extras que podem ser utilizados para lidar com tarefas comuns de desenvolvimento Web. Cuida da autenticação do utilizador, administração de conteúdos, mapas do site, feeds RSS, etc. É flexível e rapidamente escalável. Empresas, organizações e governos têm usado Django para construir todo o tipo de coisas, desde sistemas de gestão de conteúdos a redes sociais e plataformas de computação científica [31].

O Django é desenvolvido na linguagem Python que apesar do forte crescimento dos últimos anos, a nível de utilização ainda se encontra atrás das linguagens web mais tradicionais como o PHP ou o .NET. [56] Apesar do Python ter crescido nos ambientes Linux, hoje em dia já é possível instalar facilmente e corrê-lo normalmente num ambiente Windows, tendo apenas problemas com alguns packages que executam instruções específicas de sistemas Linux.

Seguindo a documentação "How to install Django" e "Writing your first Django app" apresentada no site é possível criar uma primeira aplicação básica como exemplo da framework. O django não disponibiliza nenhum gerador de código, apenas cria uma estrutura mínima contendo como por exemplo, o ficheiro de configurações que nos permite selecionar a conexão à base de dados e o ficheiro de rotas do site. Após a criação manual de ficheiros modelo e migração deste para a criação na base de dados é possível ativar um backoffice "out-of-box" para a entidade em questão. Sendo que o código fonte deste backoffice não se encontra disponível para edição, toda a manipulação deste backoffice passa por uma API específica em que é possível configurar as funcionalidades e o aspeto visual.

Esta API disponibiliza ainda métodos para prender "hooks" às ações base das entidades, permitindo-nos assim acrescentar o nosso próprio código. Este mecanismo tem a vantagem do código ser mais limpo pois o código do programador fica completamente separado do código base da função, possibilitando facilmente a recriação de um modelo sem apagar o código desenvolvido pelo programador. O lado negativo destes métodos de implementação usados pelo Django refere-se a ficarmos limitados aos métodos que a framework disponibiliza. Por exemplo, criar uma sub tabela dentro de um registo de uma tabela acaba por ser se tornar uma tarefa complicada e dispendiosa a nível de tempo.

A documentação do Django é adequada no momento em que se pretende implementar funcionalidades básicas como o envio de emails, paginação ou logging pois contempla páginas específicas com todos os passos de implementação e exemplos práticos. Por outro lado, também conta com uma documentação detalhada da sua API.

Apesar da boa qualidade da documentação, o facto da framework assentar sobre um padrão de Model-View-Template (MVT) que não é o mais comum, torna a curva de aprendizagem de utilização mais difícil do que o usual entre outras frameworks.

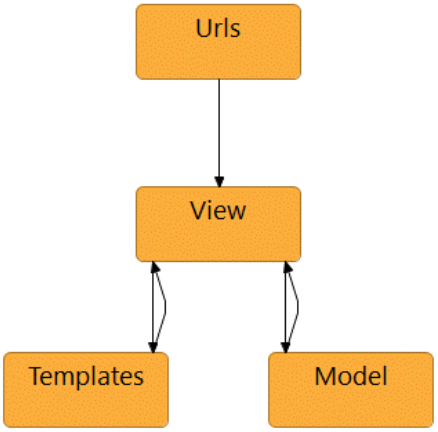


Figura 3.3: Arquitectura de uma aplicação Django

A principal diferença entre o MVC e o MVT é não existir um controlador separado, o controlo é tratado pela framework [45]. A maioria das aplicações web contam com acesso a uma base de dados, no caso do MVT é o modelo que ajuda a manusear a base de dados. É uma camada de acesso aos dados, que contém os campos e comportamentos necessários dos dados que está a armazenar. Um modelo é uma classe Python e não sabe nada sobre outras camadas Django.

Os modelos ajudam os programadores a criar, ler, atualizar e apagar objetos (operações CRUD) na base de dados. Além disso, possuem lógica empresarial, métodos personalizados, propriedades e outras coisas relacionadas com a manipulação de dados.

O template é uma camada de apresentação que trata completamente a parte da Interface de Utilizador. Trata-se de ficheiros com código HTML, que é utilizado para mostrar dados. O conteúdo destes ficheiros pode ser estático ou dinâmico. Um template é utilizado apenas para apresentar dados, uma vez que não há lógica de negócio nele.

A vista é utilizada para executar a lógica de negócio e interagir com um modelo para transportar dados e renderizar um modelo. A vista vai buscar dados a um modelo, processa-os e envia-os para o template. Aceita pedidos HTTP, aplica lógica empresarial fornecida pelas classes e métodos Python, e fornece respostas HTTP aos pedidos dos clientes [10].

De um modo geral o output de erros é abstrato. Mesmo pesquisando sobre o erro em questão, o resultado ou é nulo ou é um conjunto disperso de soluções em que nenhuma é a indicada para o problema. Recorrendo ao debug, este é apenas possível no código criado pelo utilizador não ajudando à resolução de erros. Num exemplo prático onde é criada uma ação em que a sua única tarefa é atualizar uma entidade n2 após a entidade n1 ser atualizada, se tivermos um erro que não seja na linha de update, não nos é possível averiguar a sua origem nem a sua razão.

O Site Django apresenta também milhares de packages desenvolvidos pela comunidade que permitem acelerar o processo de desenvolvimento da aplicação. Um exemplo disso é o package "django-rest-framework" que rapidamente permite criar uma API Representational state transfer (REST) para as entidades do projeto. Foram criadas 4 aplicações semelhantes para testes de performance utilizando a framework YII, o JHispter, código de C# recorrendo a entity framework e o Django. No teste de receber 200 mil registos de uma base de dados SQL Server, o Python (Django) apesar de não ser uma linguagem 100% compilada demonstrou ser o mais rápido.

Em teoria isto não devia ser verdadeiro, pois linguagens compiladas como o C# deveriam ser mais rápidas. Esta situação leva-nos assim à conclusão de que o Django está programado de maneira a ter uma alta performance. O Django conta com um servidor de desenvolvimento em que basta a execução de um simples comando para rapidamente estar a correr e ser possível testar a aplicação.

Em suma, como desvantagem, o grau de aprendizagem é superior e mais moroso que as outras frameworks e é difícil criar automaticamente lógicas mais complexas do que simples CRUD'S. Por outro lado, a sua performance é superior às outras frameworks, o código criado pelo programador fica completamente isolado e apresenta milhares de packages disponíveis com necessidades mais comuns das aplicações web.

É usado por companhias como a Mozilla, Instagram, Pinterest, Open Stack.

3. JHipster [43]

   JHipster é uma plataforma de desenvolvimento para gerar, desenvolver e implementar rapidamente aplicações web modernas e arquiteturas de micros serviços. Gera UI elegantes, modernas e responsive com base em tecnologias de vanguarda como o Angular, React, Vue e Bootstrap. No backend suporta Spring Boot (Java). É possível fazer deployment em Docker, Kubernetes, AWS, Azure, Google Cloud Platform. JHipster é Open Source e todo o desenvolvimento é feito no GitHub [72].

   Habitualmente apresentado como um gerador de aplicações java, a sua instalação é a mais simples possível. Após correr o executável do JHipster , é apresentado um conjunto de perguntas sobre como se pretende que a aplicação seja, que módulos deve conter e que configurações deve ter. O tempo que se demora a responder a este questionário até à aplicação estar gerada é muito breve. As perguntas passam por

saber se se quer uma aplicação monolítica, microservico ou de outro tipo, estilo de base de dados, de cache, suporte multilanguage, inclusão de bibliotecas de testes e se queremos adicionar módulos como elasticsearch, websockets, OpenApi, etc. Por último, mas não menos importante, é também possível escolher se o Frontend é gerado em Angular ou React seguido da escolha do aspeto visual deste. Apesar deste procedimento ser straight-foward no site, existe documentação e até um vídeo tutorial para o acompanhar. Em suma, é logo no início que são incluídos a maioria dos recursos que são comuns nas aplicações web podendo também escolher entre duas das melhores frameworks de front-end usadas atualmente [75].

A criação da aplicação conta também com a configuração do Maven que faz com que, na prática, seja apenas necessário executar um único comando para compilar a aplicação e iniciar o servidor JBoss com esta a correr. Este processo é imensamente moroso mas, devido ao uso do Maven, Webpack e Browersync, não é necessário recomeçar o projeto na maioria das vezes que se faz alterações ao código, tanto a nível de back-end como front-end. Por defeito, a aplicação gerada contém já um mecanismo de autenticação (user/password), um CRUD de utilizadores, ecrã de métrica, ecrã de audit e um swagger com REST API sobre as entidades geradas.

Para gerar os CRUD's o JHipster tem como base ficheiros escritos numa linguagem própria (JHipster Domain Language) mas de fácil compreensão. Para ajudar na criação de ficheiros deste tipo existe uma aplicação online que vai mostrando o diagrama da base de dados ao mesmo tempo que se vai criando as entidades e suas ligações. Após correr o comando de importação deste tipo de ficheiros, são automaticamente criados todos os ficheiros desde da base de dados ao front-end.

A aplicação gerada pelo JHipster segue o padrão de desenho MVC usando as frameworks/bibliotecas mais comuns recorrentes para criação de aplicações web, como por exemplo o Webpack, Bootstrap, Spring, Spring Mvc Rest, Spring Data JPA. Esta também, não só conta com os benefícios destas frameworks/bibliotecas, como segue todo um conjunto de boas práticas de programação.

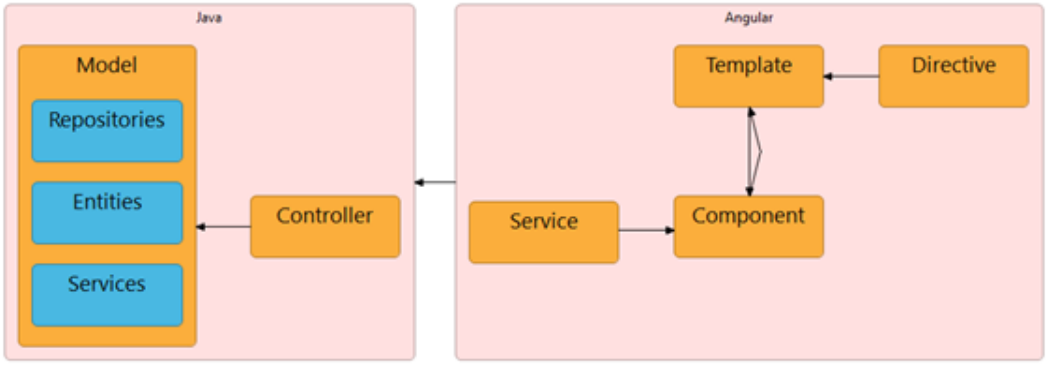


Figura 3.4: Exemplo de uma arquitectura de uma aplicação gerada por o JHipster

Componente de Angular [32]

- Component : Um componente define a lógica, através de uma API interage com a vista e adquire dados do serviço.
- Template : Um template é uma forma de HTML que diz ao Angular como renderizar o componente.
- Directive: Os templates de angulares são dinâmicos. Quando o Angular os renderiza, transforma o DOM de acordo com as instruções dadas pelas directivas.
- Service: O serviço é uma categoria ampla que engloba qualquer valor, função, ou característica que a aplicação necessite. Quase tudo pode ser um serviço. Um serviço é tipicamente uma classe com um objectivo bem definido e específico.

Componente de Java

- Repository : O repositório é utilizado para fornecer operações genéricas CRUD, ou seja, uma classe correspondente a uma tabela de base de dados ou outra estrutura de dados. Em muitos aspetos, os repositórios são análogos ao padrão data access object (DAO).
- Entity: É um modelo que representa uma tabela na base de dados.
- Service: O serviço contem lógica e regras referentes à base de dados.

Após a aplicação estar gerada, tal como as outras frameworks, é possível aceder ao Marketplace e instalar "plugins" como, por exemplo, um "Google Analytics". Porém este mercado contém apenas algumas dezenas de "plugins".

Os benefícios apontados previamente, e a questão do back-end ser todo criado na linguagem Java, uma linguagem orientada a objetos, tornam o desenvolvimento do JHipster simples e rápido. Já em relação ao front-end, tanto a framework "Angular" como a "React" já exigem uma maior atenção e um maior tempo de desenvolvimento. O debug nestas frameworks normalmente é simples, pois basta colocar um breakpoint no browser, mas por vezes de igual forma com o Django os erros são bastante abstratos.

Um dos pontos fortes do JHipster é a geração automática de testes unitários, de integração, de arquitetura, de UI, de performance e de behavior driven, usando as frameworks JUnit, Spring Test, Jest. Archunit, Gatling e Cucumber respetivamente, oferecendo assim de forma chave na mão a maioria dos testes que interessam numa aplicação web.

Em suma, o JHipster não se trata de uma framework em si, mas de uma aplicação que gera aplicações web contendo as frameworks e bibliotecas atuais do mundo web e seguindo boas práticas e técnicas de programação para que, de uma forma simples

e rápida, o programador consiga começar a desenvolver áreas mais complexas da aplicação.

É usado por companhias como a Ericsson, Siemens, HSBC,Hewlett Packard, Ministério da defesa Francês, PWC, HBO.

3.2 Outras frameworks e geradores de código

Este tópico demonstra as frameworks e geradores de código que foram primeiramente excluídos devido às suas desvantagens e falhas críticas perante os pontos avaliados.

- CodeTrigger [14]

  É um gerador de código da linguagem de programação Microsoft.Net que promete aumentar a produtividade do programador, deixando-o concentrar-se nas questões importantes do negócio empresarial enquanto o CodeTrigger gera a maior parte do código. Trabalha com SQL Server, Oracle ou MySQL importando os esquemas de base de dados pré-definidos e modelando as relações numa aplicação WPF/WCF/Winforms/Web multicamada. Visa a simplificar e acelerar os processos de desenvolvimento de software, estabelecendo um modelo não intrusivo para a equipa de desenvolvimento. Ao contrário de outros geradores de código, o CodeTrigger é em grande parte agnóstico de design. Não é obrigado a seguir qualquer filosofia particular de design, nem está ligado a camadas complexas de código impenetrável. Através de uma escolha inicial de opções é possível escolher que partes da aplicação são geradas e as suas configurações [14].

  O CodeTrigger apresenta poucas opções, falta por exemplo a possibilidade de usar o servidor de base dados Postgres e não existe nenhuma forma de criar plugins de maneira a solucionar este problema. A aplicação gerada usa técnicas de programação desatualizadas como por exemplo, um acesso à base de dados por comandos Structured Query Language (SQL) em vez de um usar um Object–Relational mapping (ORM) que lhe poderia permitir conectar-se de forma transparente a diferentes tipos de base dados. A sua arquitetura também não contém um Data Access Layer (DAL) correto pois os modelos e os repositórios estão juntos na mesma camada. Faltam opções como a multilinguagem, filtros e procura nas tabelas geradas. Não oferece plugins ou módulos extra como por exemplo, um logging.

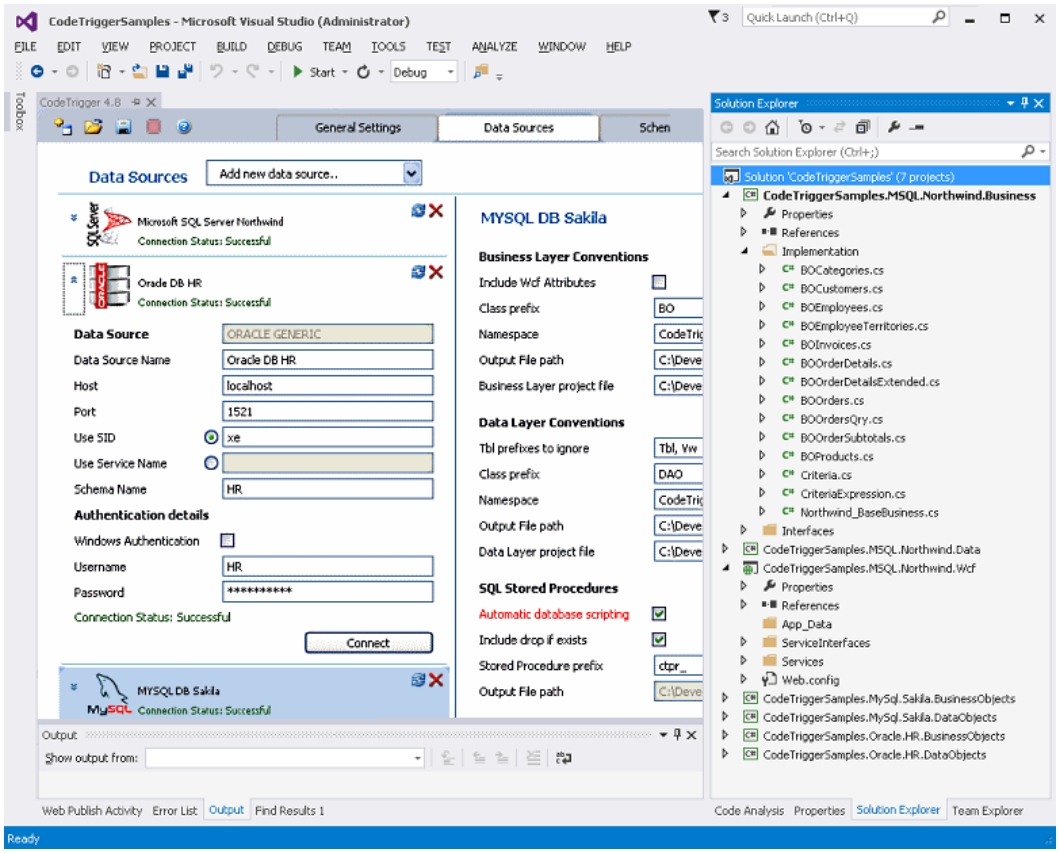


Figura 3.5: `CodeTrigger`

- Asp.net runner [90]

  ASPRunner.NET cria aplicações WEB ASP.NET com ligações à maioria das bases de dados conhecidas (Oracle, SQL Server, MS Access, MySQL, Postgres). Atráves de uma interface gráfica cria-se as configurações necessárias para a construção de páginas. Através destas páginas geradas, os utilizadores podem pesquisar, ordenar, editar, apagar e adicionar dados a uma base de dados. O código gerado e fácil de modificar [90].

  O "Asp.net runner" gera uma aplicação com uma linguagem desatualizada (.NET MVC3) onde de um modo geral não implementa regras de boa programação na sua arquitetura. Faltam-lhe designers patterns como Dependecy Injection e tem excesso de código dinâmico, o que lhe confere uma baixa performance.

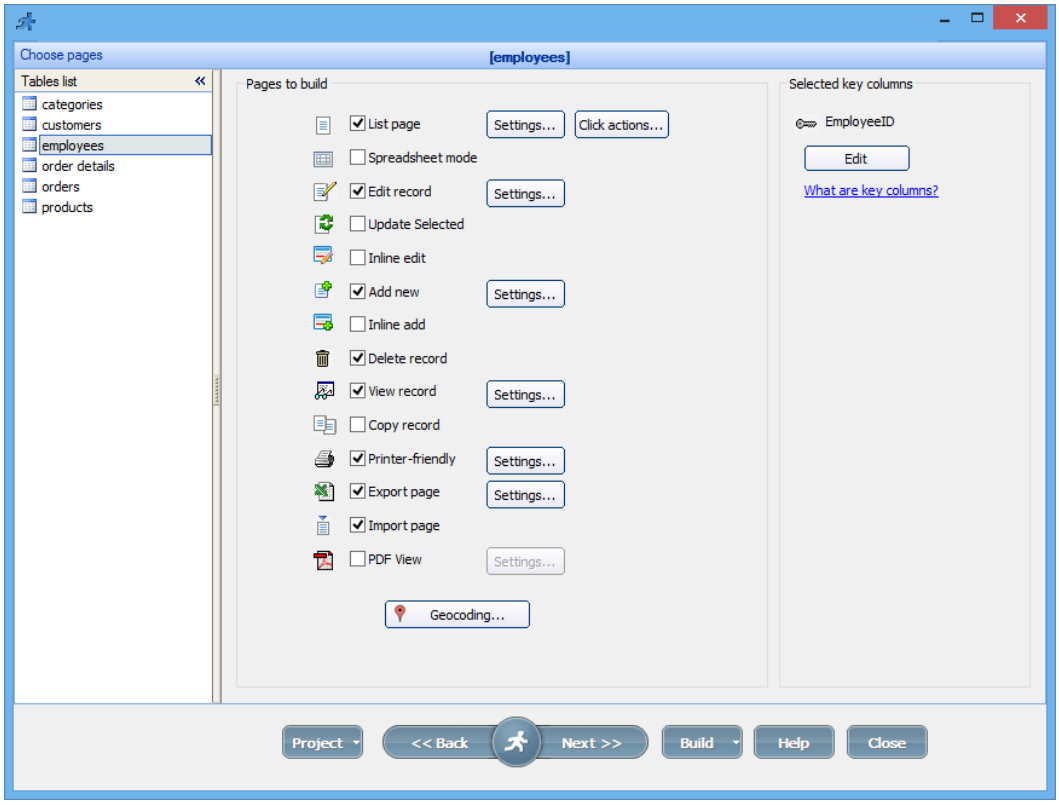


Figura 3.6: ASPRunner.NET

- Spring Roo [22]

  Spring Roo é uma ferramenta de Desenvolvimento Rápido de Aplicações (RAD) que visa fornecer resultados rápidos e instantâneos, centrados em aplicações web Spring com o uso das mais recentes tecnologias Spring, como a Spring IO platform ou Spring Data. A framework Spring é uma plataforma Java que fornece um abrangente suporte de infra-estrutura para o desenvolvimento de aplicações Java e tem como principal objetivo tratar da infra-estrutura da aplicação para que o programador se possa concentrar no negócio [22] [21].

  Com o Spring Roo programa-se tudo em Java. A sua abordagem funciona inteiramente em tempo de compilação e é completamente compatível com o IDE. A adoção é simplificada e de baixo risco. Através do uso de simples comandos é possível gerar uma aplicação em poucos minutos. Porém, é apenas possível gerar uma aplicação até à camada de negócio ficando a faltar a camada visual (Font-End).

- Vaadin [55]

  Vaadin é uma plataforma de desenvolvimento de aplicações web para Java e simplifica o processo com uma plataforma integrada de desenvolvimento de aplicações web para Backends em Java. Vaadin vem com todos os componentes, frameworks e ferramentas necessárias para construir uma aplicação fiável, segura e com um bom

UX. Conta com um motor colaborativo que pode ajudar os programadores a trabalharem em conjunto e em tempo real, independentemente do local onde trabalhem. Inclui componentes e padrões UX que o ajudam a construir boas experiências de utilizador sem ser necessário programar o HTML ou JavaScript, permitindo também construir um sistema de design personalizado [55].

Porém, gera uma aplicação completamente feita em Java, não sendo possível gerar um front-end noutra linguagem. Isto significa que tudo o que é necessário para construir o UI é feito no servidor e transmitido de volta para a aplicação criando, assim, um tráfego da rede muito maior. Significa também que o estado é gerido no servidor, o que aumenta muita a memória deste, tornando-se mais difícil escalar a aplicação [15].

É usado por companhias como a Disney, Wells Fargo, Bank of America, GlaxoSmith-Kline, Volkswagen America.

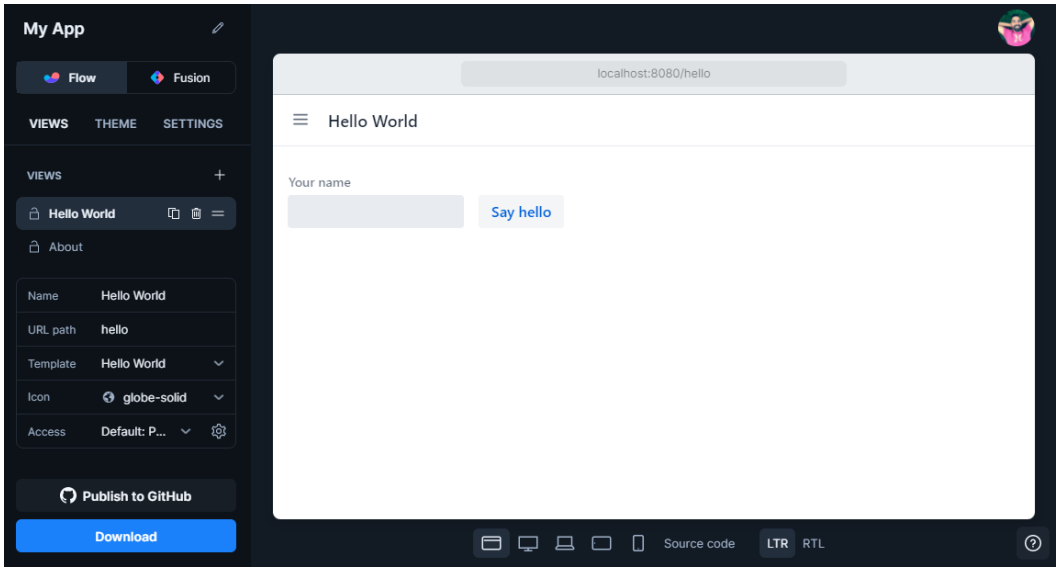


Figura 3.7: Vaadin

- Generjee [18]

  É uma ferramenta online gratuita para criar aplicações completas em Jacarta EE sem ser necessário qualquer programação. Pode ser utilizada para iniciar projetos de desenvolvimento a partir de uma base de código preparada e adaptada de acordo com as configurações específicas do projeto e com as do modelo de dados. O resultado produzido é tecnologicamente baseado em Java EE, JSF, PrimeFaces, Apache Shiro e JPA (por exemplo Hibernate). O código produzido pode ser executado em qualquer servidor de aplicações Jacarta EE e é independente do Generjee. [19].

  O Generjee tem apenas opções muito básicas que permitem apenas criar uma aplicação com um CRUD simplista e uma interface muito desatualizada.

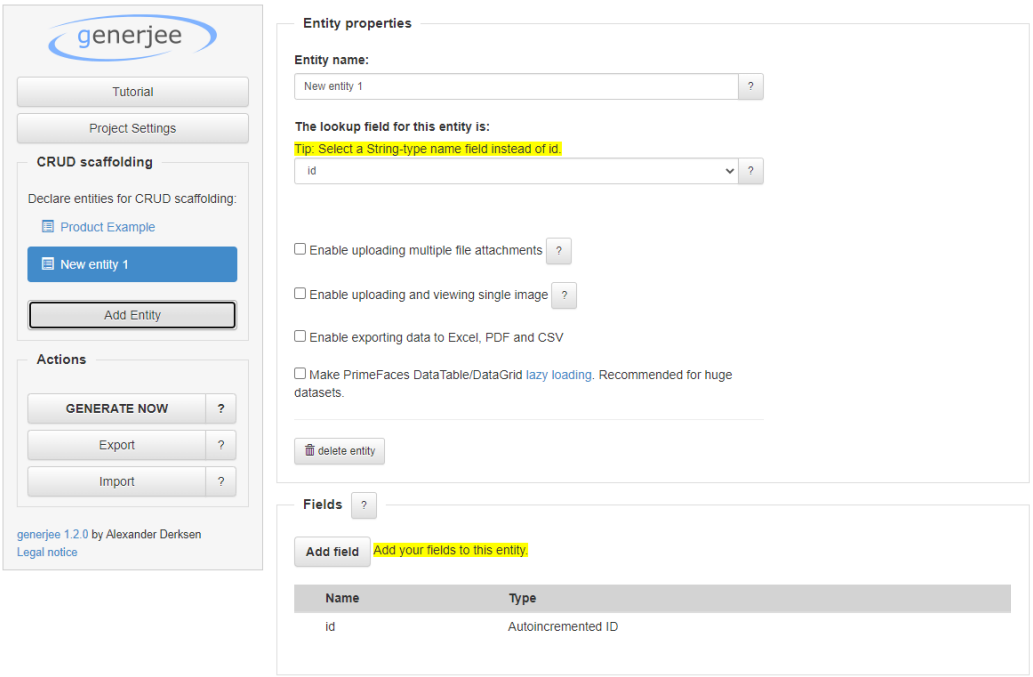


Figura 3.8: Generjee

- Metawidget [47]

  O Metawidget é um widget inteligente que, sem introduzir novas tecnologias, automaticamente preenche-se com componentes UI para corresponder às propriedades dos objetos de domínio, quer estaticamente quer em tempo de execução. Inspeciona a arquitetura do back-end existente (JSON, REST, anotações, ficheiros de configuração) e cria widgets nativos na estrutura de front-end existente (JavaScript, Java Server Faces, Android, etc). A última versão lançada remonta a 2015 gerando código com versões muito desatualizadas [47].

- Grails [73]

  É uma framework Web baseada em Groovy para JVM e assenta em Spring Boot. Implementa o padrão de desenho MVC. Permite aos programadores concentrarem-se mais nas necessidades reais da aplicação e gastar menos tempo a configurar a estrutura. A framework Grails fornece ferramentas para o desenvolvimento e é construído com base em ferramentas como: Quarts, Hibernate, Spring e Gradle. Esta framework tem um elevado nível de complexidade e uma grande curva de aprendizagem. Devido ao facto de usar Groovy que, apesar de se ter tornado mais popular nos últimos anos [48], ainda não está no top das linguagens Web, fazendo com que existam poucos programadores com experiência na sua utilização no mercado. Esta situação é problemática pois exige o custo extra referente à formação de programadores em Groovy e na manutenção das aplicações [73].

O Grails demonstrou ser apenas uma framework e com elevado nível de complexidade.

É usado por companhias como a Epic Games, Cloudera, Apple, IBM , Netflix, PayPal, Visa [94].

- Jmix [36]

  Jmix é uma framework de alto nível para aplicações web empresariais. Vem com ferramentas avançadas e um grande conjunto de módulos funcionais opcionais que apenas acrescentam as dependências necessárias. É uma framework completa no sentido em que fornece suporte para a criação tanto do backend, como da interface de utilizador da sua aplicação. É baseada no Spring Boot e cria aplicações web Java, significando que é possível utilizar muitas bibliotecas e frameworks com configurações mínimas, para além das funcionalidades fornecidas pelo Jmix. Fornece também um plugin (Jmix Studio) para o editor IntelliJ IDEA muito útil em todas as fases do desenvolvimento da aplicação: criação e configuração do projeto, definição de modelo de dados, geração de scripts de migração de bases de dados e desenvolvimento de ecrãs UI num editor visual. O plugin também disponibiliza uma navegação avançada e code completion [36].

  Jmix fornece uma forma conveniente de trabalhar com uma ou várias bases de dados relacionais. Com algum esforço adicional, pode ligar-se a qualquer fonte de dados como uma base de dados no-SQL ou uma API externa. O desenvolvimento com Jmix começa normalmente a partir da definição do modelo de dados. O esquema da base de dados é criado a partir do modelo de dados automaticamente, e quando existe uma evolução do modelo de dados, o Studio sincroniza o esquema da base de dados com o modelo através de migrações. Os ecrãs UI para operações de CRUD são criados através dos modelos da base de dados e a partir dai é possível personalizá-los, como por exemplo alterar o layout ou adicionar e remover componentes visuais.

  Porém, a camada visual usa a framework Vaadin carecendo das mesmas desvantagens desta.

- Phpgrid [40]

  Através de uma rápida configuração é possível poupar tempo de desenvolvimento e esforço ao criar CRUD's que contemplem pesquisa, paginação e importação e exportação. Funciona com a linguagem PHP, permite conectar-se com as principais bases de dados e possibilita a exibição de dados de tabelas cruzadas [40].

  O Grid4Php oferece apenas a geração dos CRUD's, faltando várias funcionalidades essenciais como por exemplo, Autenticação.

- PhpCrudGenerator [68]

Utilizando uma interface intuitiva e a partir de uma análise de uma base de dados cria uma aplicação completa na linguagem PHP e base de dados MySQL. Esta aplicação conta com CRUD's completos e funcionalidades como gestão de utilizadores, modulo de administração e modulo de autenticação [68].

Só funciona com base de dados MySQL, o que é um problema, pois pode não se enquadrar nos requisitos do cliente. Conta com um custo monetário associado ao contrário da maioria das frameworks/geradores de código que são opensource.

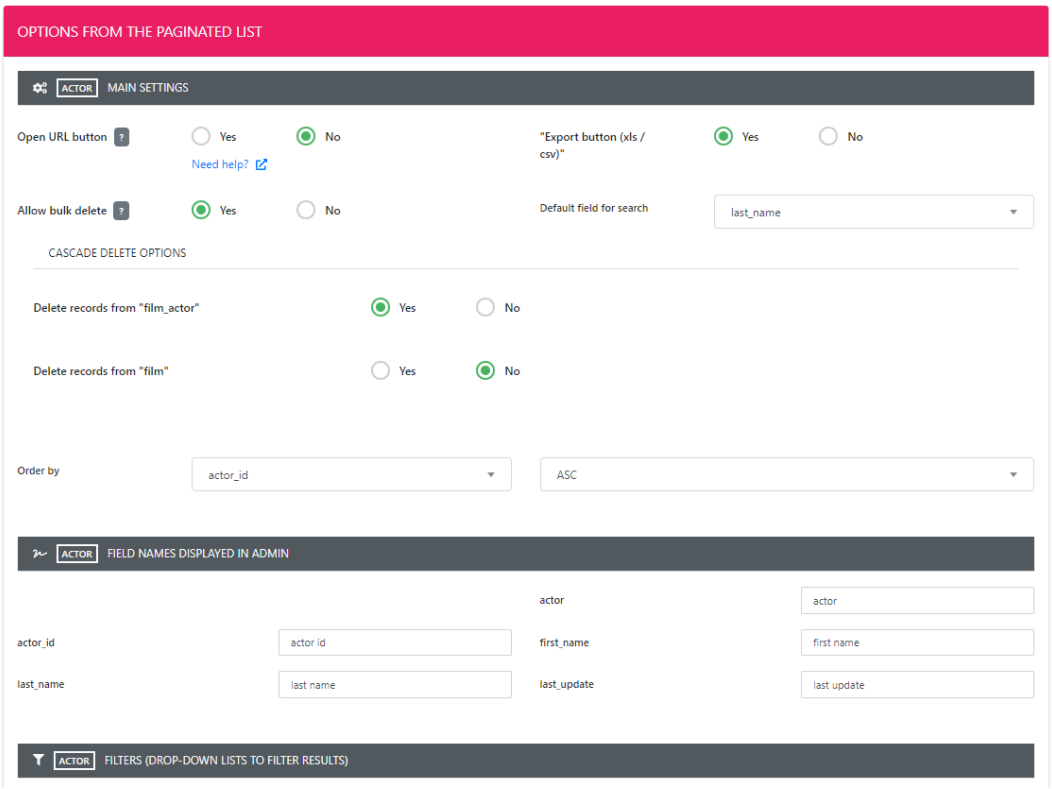


Figura 3.9: Php CRUD Generator

- Laravel [65]

  Uma das frameworks mais potentes do PHP é o Laravel [34] mas não gera qualquer tipo de código. É usado por companhias como a 9GAG, BBC, Pfizer [74]. A geração de código só é possível com a execução de plugins sendo que os seguintes plugins se mostram ineficientes:

  O plugin gerador CRUD Getcraftable [12] não apresenta escolha de alguns servidores de base de dados como por exemplo o SQL Server.

  No Infyom [41] é necessário correr imensos comandos para gerar interfaces gráficos bastante simplistas, fazendo com que o esforço versus o resultado seja negativo.

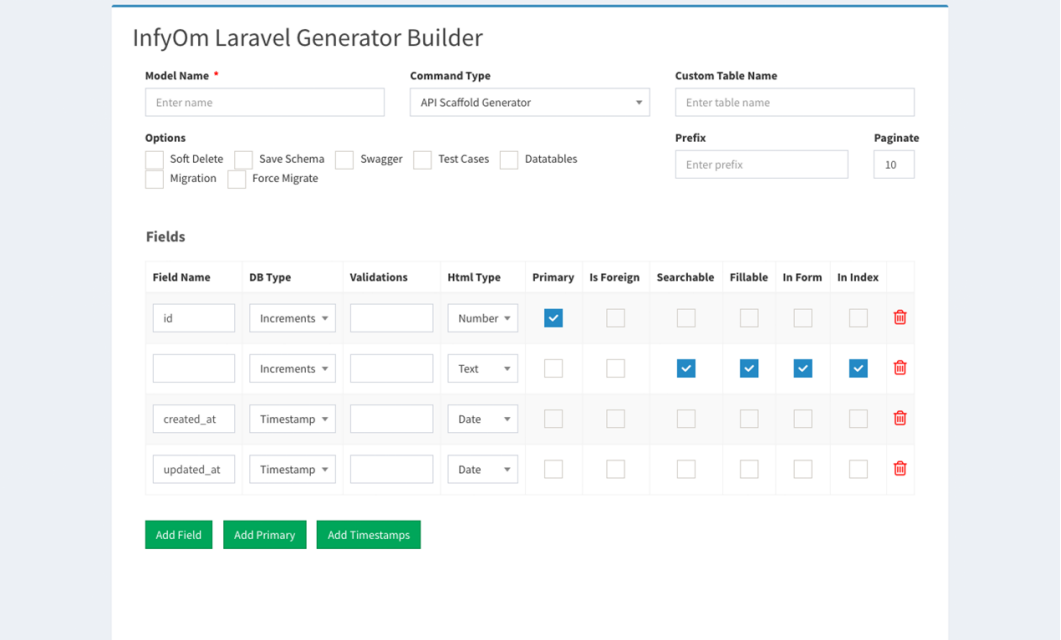


Figura 3.10: Infyom

O Phreeze [39] estava a seguir um bom caminho vendendo-se como "Um ORM, MVC framework que inclui um gerador de código que cria um site limpo e funcional baseado numa base de dados" [39]. Deixou de ser atualizado em 2013 fazendo com que a aplicação gerada esteja com versões desatualizadas de todas as bibliotecas que usa.

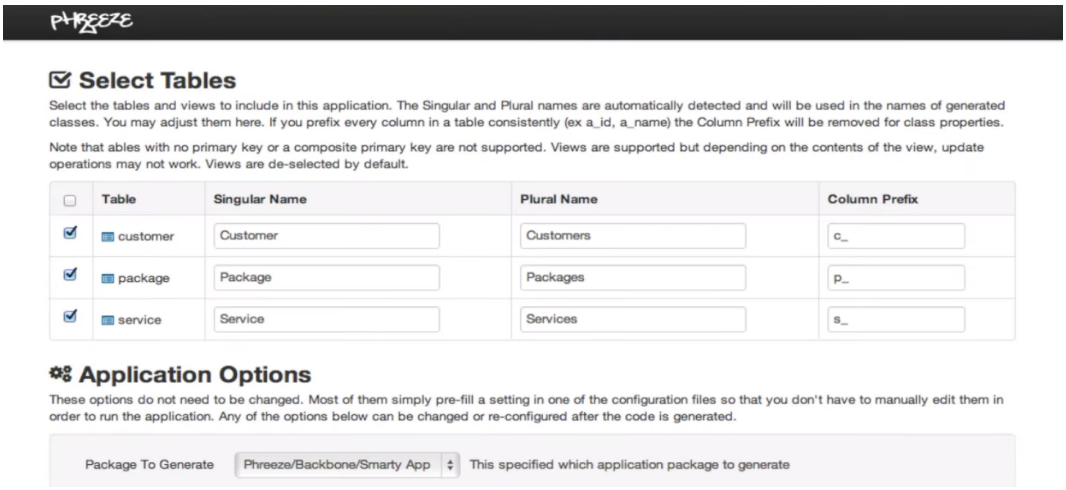


Figura 3.11: Phreeze

- Telosys.org [84]

  É uma ferramenta que impulsiona o arranque do projeto gerando todo o código repetitivo e poupando muitos dias de desenvolvimento. O Telosys studio é fornecido como um plugin para o editor Eclipse e permite configurar e editar modelos de maneira a gerar o código. O Telosys.org tenta ir mais longe e faz por gerar código para qualquer linguagem (Java, Python, PHP, JavaScript, C#, HTML, Scala, Go, etc) ou framwork (AngularJS, JPA, Spring MVC, etc). Este assenta sobre templates previamente criados por programadores [84].

  O gerador é complicado de usar, no caso de C# MVC só existe um template já muito desatualizado sendo que o código gerado por este template não assenta em nenhuma framework e não respeita praticamente nenhuma arquitetura. O gerador acaba por ter poucas opções para a criação de aplicações mais complexas.

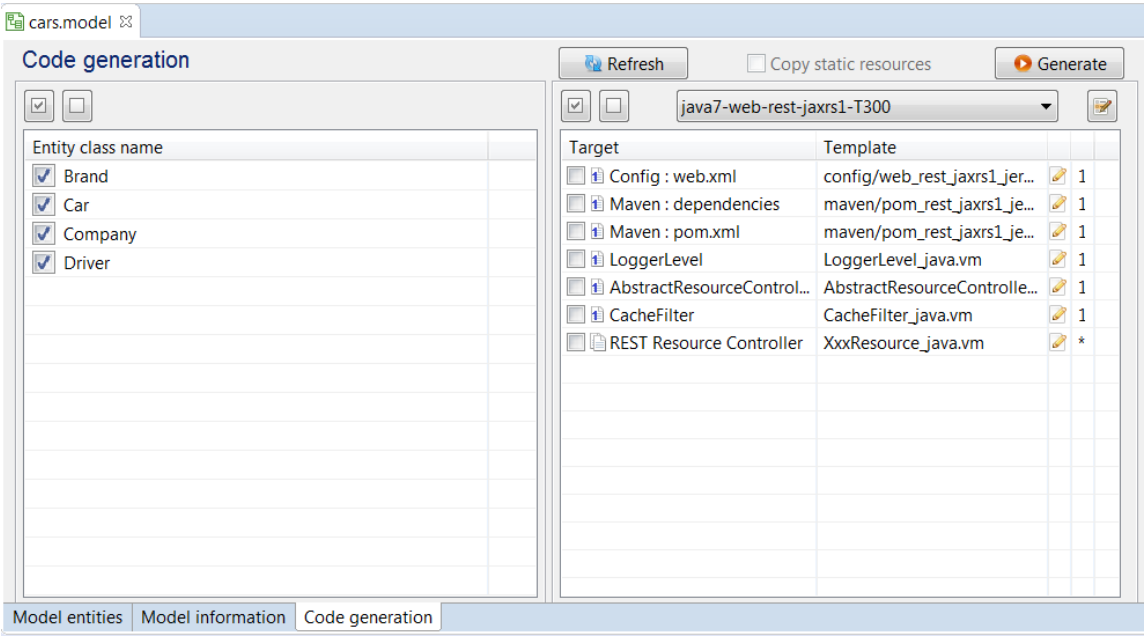


Figura 3.12: Telosys.org

3.3 Listagem de frameworks e geradores de código

A seguinte tabela apresenta um resumo das frameworks e geradores de código enumerados anteriormente à data de 27-07-2021. Nesta tabela é possível observar a empresa/-pessoa que a desenvolve (Marca) e em que linguagem de programação, quando foi o seu primeiro (1º Lan.) e último lançamento (Últ. lan.), a sua versão corrente (Ver.) e o tipo de licença.

Produto	Marca	Linguagem	1º Lan.	Ver. / Últ. lan.	Licença
CodeTrigger	Exotechnic Corporation	C# .Net		6.3.0.3 2020	Sem custo ou Profissional
ASPRunner.NET	XLineSoft	C# .Net	2004	10.6 2021	Profissional
Spring Roo	DISID, Pivotal Software	Java	2009	2.0.0 2019	Open source
Vaadin	Vaadin Ltd.	Java	2002	21.0.3 2021	Open source ou Profissional
Generjee	Alexander Derksen	Java	2015	1.2.0 2020	Open source
Metawidget	Richard Kennard	Java	2008	4.2 2015	Open source
Grails	Graeme Rocher	Groovy	2005	4.0.10 2021	Open source
Jmix	Haulmont	Java	2009	1.0.3 2021	Open source ou Profissional
Php Grid	Kayson Group Inc	PHP	2008	7.5 2021	Sem custo ou Profissional
PHP CRUD Generator	PHPCG	PHP	2018	1.22 2021	Profissional
Laravel	Taylor Otwell	PHP	2011	5.8 2020	Open source
Getcraftable	Brackets	PHP	2017	7.0.0 2020	Open source
Infyom	InfyOmLabs	PHP	2021	3.1.1 2021	Open source
Phreeze	Jason Hinkle	PHP	2012	3.3.7 2015	Open source
Telosys.org	Telosys	Várias	2011	3.3.0 2021	Open source
Php YII	Qiang Xue	PHP	2008	2.0.41 2021	Open source
Django	Django Software	Phyton	2005	3.2.7 2021	Open source
JHipster	Julien Duboi	Java	2013	7.0.1 2021	Open source

3.4 Geração destrutiva de código

A maioria das frameworks / geradores de código geram as classes nas várias camadas que fazem parte da sua arquitetura para o projeto gerado. Observando o JHispter, ao ser definida uma nova entidade "Ticket" os principais ficheiros que serão gerados são:

- "Ticket.Java" classe que representa a tabelada base de dados "Ticket"
- "TicketRepository.java" classe que extende o JpaRepository (repositório genérico) [69] e contém os métodos de acesso à base de dados para entidade "Ticket" como por exemplo os métodos findAll(), getById(), etc.
- "TicketResource.java" classe que contém as ações de Http sobre a entidade "Ticket" como por exemplo os métodos "getTicket", "deleteTicket"," updateTicket", etc.
- "ticket.component.html" ficheiro de html onde se encontra a listagem UI de "Tickets" na camada visual.
- "ticket-update.component.html" ficheiro de html que contém o formulário UI de criação e edição do "Ticket".
- "ticket-detail.component.html" ficheiro de html onde é mostrado visualmente o conteúdo do "Ticket".

No caso do YII os principais ficheiros a serem gerados são os seguintes:

- "Ticket.php" classe que representa a tabela da base de dados "Ticket" e extende a classe active record [53] contendo os métodos de acesso à base de dados como por exemplo o findAll() e findOne().
- "TicketController.php" classe que contém as ações de Http sobre a entidade "Ticket" como por exemplo os métodos "actionView","actionDelete", "actionUpdate".
- "Index.php" ficheiro de PHP onde se encontra a listagem UI de "Tickets" na camada visual.
- "create.php" ficheiro de PHP que contém o formulário UI de criação do "Ticket".
- "edit.php" ficheiro de PHP que contém o formulário UI de edição do "Ticket".
- "view.php" ficheiro de PHP onde é mostrado visualmente o conteúdo do "Ticket".

Comparando as várias frameworks / geradores de código observa-se que existem algumas diferenças entre a intenção dos ficheiros que são gerados, o que é normal pois implementam padrões de desenho diferentes. Porém, como referido anteriormente, é possível observar que são gerados ficheiros nas diferentes camadas da arquitetura (acesso a dados / negócio / interface visual).

No decorrer do estudo foi encontrado um problema comum a todas as frameworks / geradores de código. Entendeu-se que, após a geração da aplicação web, se for necessário acrescentar uma nova entidade, não existe nenhum problema. Por outro lado, quando é necessário alterar algo numa entidade já existente, como por exemplo acrescentar um novo campo ou criar uma nova ligação para outra tabela, todos ficheiros gerados relativos a essa entidade são gerados novamente apagando todo o código já desenvolvido em cima do projeto gerado. No caso da YII, ao tentar novamente gerar a entidade, apresenta apenas a indicação que o ficheiro vai ser rescrito [54]. Já na JHipster aparece a mensagem: "this will replace the existing files for this entity, all the your custom code will be overwritten" [83].

Ao aceitar as alterações, todos os ficheiros gerados mencionados anteriormente são reescritos perdendo todas as alterações efetuadas pelos programadores. Por exemplo, se no ficheiro de acesso à base de dados (TicketRepository.java) for criado um método para devolver apenas tickets ativos e ordenados por ordem de criação, ou se for alterada ação a de apagar no ficheiro "TicketResource.java" para incluir um log, estas alterações vão desaparecer após uma nova geração. A forma rudimentar que existe para contornar este problema é gerar novamente o projeto numa diretoria paralela e manualmente comparar as alterações feitas nos ficheiros gerados. Esta forma é morosa e cansativa, o que acaba por induzir em erros humanos.

Capítulo 4

Geração de código não-destrutivo

Após a geração de aplicações Java com o software JHipster e de aplicações PHP com a framework YII foi identificado um problema estrutural: quando é necessário alterar uma entidade que leva a uma nova geração de código, esta torna-se destrutiva. Acontece que todo o código é reescrito e as alterações feitas a este são apagadas.Esta situação foi apenas testada com as frameworks e geradores de código da secção incluídos mas acredita-se que o problema seja geral para todos.

Ao contrário destes, pretende-se que, quando exista uma nova geração de código, haja previamente uma avaliação do código de maneira a entender-se se falamos do mesmo código ou se é diferente. No caso de ser diferente, pretende-se que exista uma junção entre o código existente com o código que está a ser gerado. Esta junção deve seguir um conjunto de regras de maneira a que o resultado final continue com as funcionalidades para que está a ser gerado e com as funcionalidades alteradas.

As seguintes abordagens que vão ser apresentadas serão divididas em duas fases. A primeira fase consiste em comparar as instruções do método original com o método alterado pelo utilizador e o método novamente gerado (2º geração). Esta comparação entre as instruções pode ser efectuada através de uma comparação textual ou através de comparação com base na AST. A segunda fase, consiste em juntar de uma maneira inteligente as instruções do método alterado com o método novamente gerado (2º geração). Para esta fase serão demonstradas três abordagens diferentes. A abordagem de marcação de início e fim de código, a abordagem do diferencial de instruções e abordagem de Diff And Patch.

4.1 Métodos de comparação

4.1.1 Comparação AST

Uma AST é uma abstração de alto nível do código-fonte representada numa estrutura de árvore. Serve como uma representação intermediária de uma linguagem de programação. Em geral, na sua estrutura em árvore cada nó tem pelo menos um tipo que especifica o que está a representar. Por exemplo, um tipo pode ser um Literal que representa um valor real ou uma "call expression" que representa uma chamada a uma função. O nó Literal pode conter apenas um valor, enquanto o nó "call expression" contém informações adicionais que podem ser relevantes, como o que está a ser chamado ou quais são os argumentos que estão a ser enviados [52] [87] [66].

Existem bibliotecas disponíveis como é o exemplo da "Microsoft.CodeAnalysis" que permitem "navegar" pelas instruções de um método. Estas instruções são organizadas em formato de árvore. Uma hipótese será navegar entre duas árvores diferentes e, através de comparação entre as suas instruções, aferir o que são instruções adicionadas, removidas ou alteradas. Porém, estas árvores são muito detalhadas, o que torna a comparação mais complicada do que uma comparação de caracteres. O seguinte código produz a seguinte árvore de sintaxe :

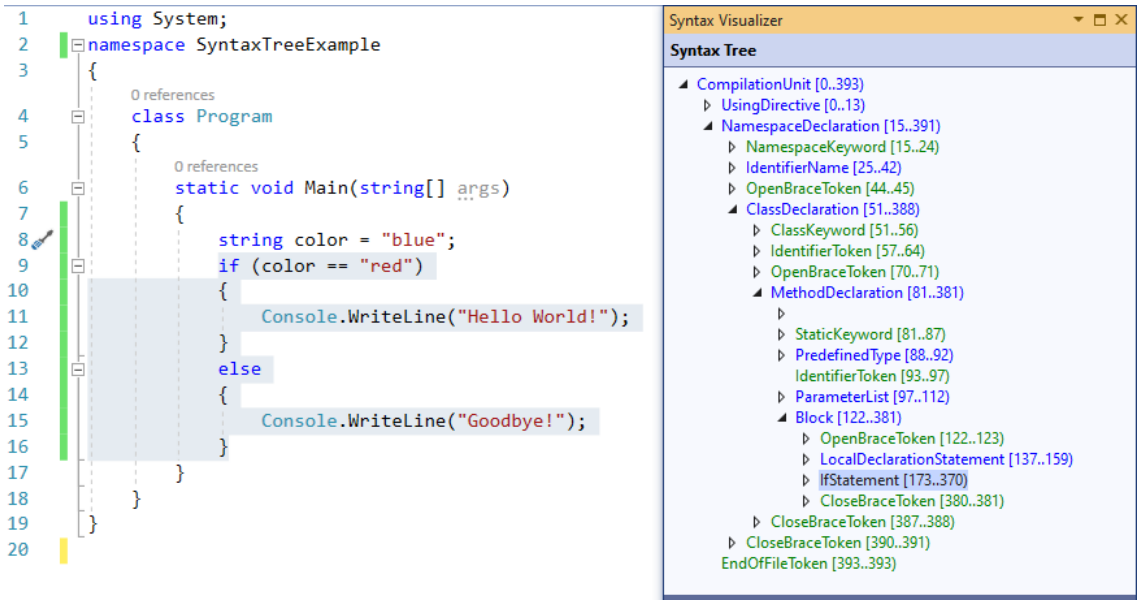


Figura 4.1: Exemplo AST

Cada tipo de instrução: ciclos, chamadas de métodos, afetações, etc. tem a sua própria árvore com o seu próprio conjunto de "componentes" criando assim imensas variantes.

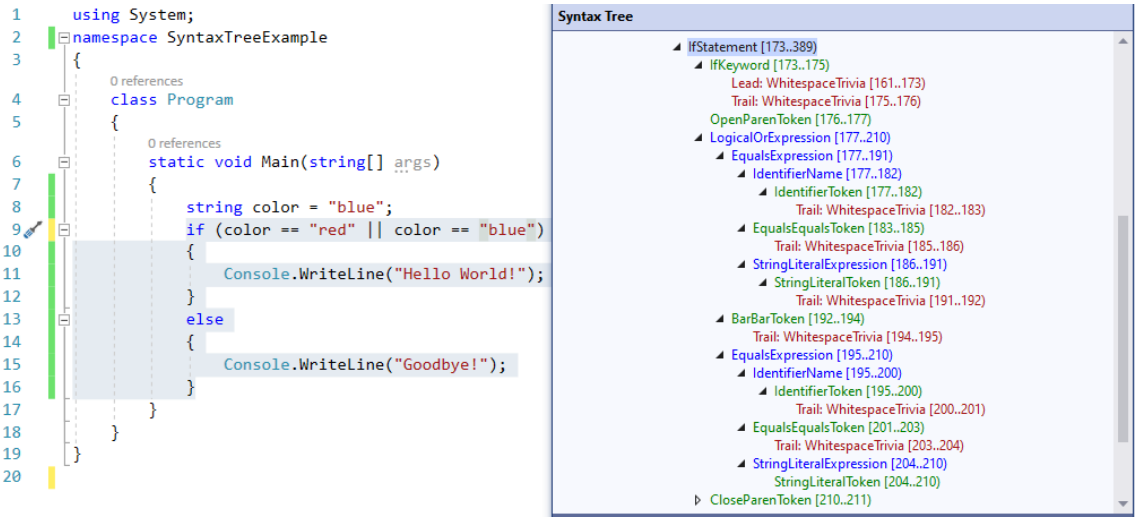


Figura 4.2: Exemplo de árvore para a instrução "If" do código anteriormente apresentado

De forma a ser possível efectuar uma comparação `instrucao1.equals(instrucao2)` numa óptica de programação Orientada a Objectos (OO) será necessário implementar um "equals" específico para cada tipo de instrução que seguirá um conjunto de regras de comparação. Como exemplo, se nas regras de comparação forem ignorados os Lead e Trail com WhitespaceTrivia (espaços em branco) então `(color=="red")` é igual a `(color == "red")`. A ordem das expressões também pode ser ignorada fazendo com que `(color == "red" || color == "blue")` seja igual `(color == "blue" || color == "red")`. Ou é mesmo possível implementar uma avaliação onde `if (x > 0)` `torna-se igual if (0 < x)`.

Hoje em dia o uso de AST é comum tanto em algoritmos de manipulação, ou comparação de código [16]. Como referido acima, por termos a informação estruturada, é possivel fazer uma abstração de pormenores sintáticos e criar um conjunto regras próprio para a comparação. Estes factores tornam este tipo de comparação mais poderoso em relação à comparação textual sendo assim obter melhores resultados. Porém, é necessario ter em conta o alto nível de exigência para criar um conjunto de regras que funcione com todas as instruções e todos os casos.

4.1.2 Comparação Textual

Todas as linguagens de programação oferecem instruções para comparação de strings.

```
String instrucao1 = "if (color == \"red\")";
String instrucao2 = "if (color == \"blue\")":
instrucao1.Equals(instrucao2);
```

Exemplo de comparação de strings

Esta forma permite-nos de uma maneira simples comparar strings, basta por exemplo existir um espaço em branco e as strings são consideradas diferentes.

De forma a melhorar a comparação textual para alcançar melhores resultados, o código em questão deverá ser previamente formatado. Para isso existem disponíveis várias bibliotecas de análise de código que seguindo um conjunto de regras configuráveis o fazem, porém estas bibliotecas não conseguem organizar os "new lines" criados pelos programadores.

```
Intrucao1
        .Intrucao2
                .Intrucao3(intrucao3a
        .Intrucao3b)
.Intrucao4
```

Exemplo de código desformatado

Propõem-se a criação de um algoritmo que gere a seguinte formatação:

```
Intrucao1.Intrucao2
        .Intrucao3(intrucao3a.Intrucao3b)
        .Intrucao4
```

Exemplo de código formatado

O resultado da formatação automática derivada de uma biblioteca com junção deste algoritmo vai garantir que todas classes, métodos e instruções estejam formatadas seguindo o mesmo conjunto de regras beneficando todos os algoritmos que utilizem comparação textual.

4.2 Métodos de junção não destrutiva

Após a fase de comparação será executada uma fase de junção inteligente em que, com base na comparação, será gerado um método final que inclui as alterações efetuadas, tanto pelo utilizador como pelo código gerado.

4.2.1 Marcação de início e fim de código

A solução proposta nesta abordagem será criar marcadores que indiquem o início e o fim do código gerado. Desta forma, apenas o bloco de código gerado será substituído após

nova geração, sendo assim possível o programador adicionar código antes ou depois do bloco de código gerado.

Os marcadores poderão ser linhas de texto simples como "// Início de código gerado" e "// Fim de código gerado". Para a geração do novo método, o código entre o marcador de início e o marcador de fim deve ser substituído pelas novas instruções geradas.

```
1 Inicio do método
2    Código do programador
3    // Inicio de código gerado
4    Código Gerado
5    // Fim de código gerado
6    Código do programador
7 Fim do método
```

Exemplo de abordagem Marcação de início e fim de código

Esta abordagem já é utilizada recorrentemente em alguns softwares que se encontram disponíveis. Um exemplo disso é o "JFormDesigner" que se trata de um designer visual para interfaces em Swing (Java) (Figura 4.3). Através de uma interface gráfica o utilizador pode inserir, apagar ou alterar componentes que faz com que o JFormDesigner gere o respetivo código. Apenas o código compreendido entre os marcadores correspondentes é alterado.

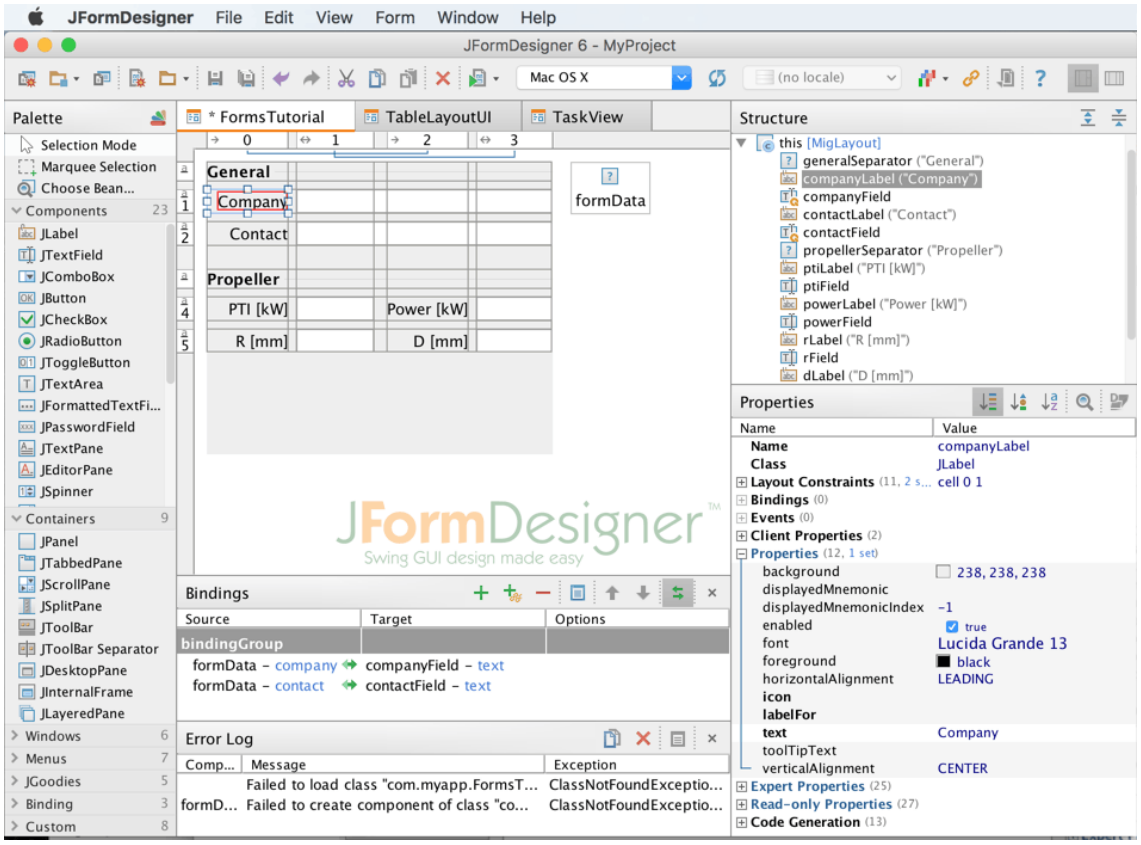


Figura 4.3: JFormDesigner 7.

```
1  private void initComponents()
2  {
3      // JFormDesigner - Component initialization - DO NOT MODIFY  //GEN-BEGIN:
       initComponents
4      label3 = new JLabel();
5      checkBox1 = new JCheckBox();
6      //---- label3 ----
7      label3.setText("Company");
8      add(label3, cc.xy(1, 1));
9      //---- checkBox1 ----
10     checkBox1.setText("Male");
11     add(checkBox1, cc.xy(3, 1));
12     // JFormDesigner - End of component initialization  //GEN-END:
       initComponents
13 }
```

Exemplo de código gerado

Se for criado um "Form" com uma "Label" e uma "Checkbox" é gerado o código anterior. De seguida o utilizador pode criar o seu próprio código antes ou depois dos marcadores "// JFormDesigner". Se o utilizador mudar o texto da label, o código entre os marcadores é novamente gerado.

Esta abordagem garante que o código que o programador introduziu seja mantido, porém, não será possível o programador introduzir instruções no meio do código gerado, ou mesmo apagar ou alterar instruções deste mesmo código. O uso de marcador torna também mais difícil a legibilidade do código.

4.2.2 Diferencial de instruções

Esta abordagem propõe um algoritmo que faz uma comparação textual entre 3 métodos (1º geração, 2ª geração e alterado) removendo ou adicionando as instruções necessárias de forma a refletir tanto as alterações do utilizador como do gerador. Estas alterações devem ser guardadas após cada comparação. Desta forma cria-se um histórico de alterações para serem aplicadas sequencialmente cada vez que existe uma nova geração. Assim é garantido que não se perde as alterações anteriormente feitas.

4.2.2.1 Solução

1. Iterar no método de 2ª geração. Durante a iteração: verificar se a instrução existe no método alterado. Para isto deve iterar sobre as instruções do método alterado e efectuar uma comparação textual com a instrução em causa do método de 2º Geração. Se a instrução existir, deve ser copiada para o método final e retirada do método alterado. Senão, deve verificar-se se existe no método de 1ª geração. Senão deve ser copiada para o método final.

2. Iterar no método alterado. Durante a iteração: verificar se a instrução não existe no método de 2ª geração. Para isto deve iterar sobre as instruções do método de 2ª

geração e efectuar uma comparação textual com a instrução em causa do método alterado. Se não existir, e também não existir no método de 1a geração, deve ser copiada para o método final.

```
for (instGerado2 in [Segunda geração])
{
  if (instGerado2 in [Alterado])
    {
    instGerado2 -> [Final];
    instGerado2 <- [Alterado];  (remover do alterado)
    }
    else
    {
    if (instGerado2 not in [Primeira geração])
      instGerado2 -> [Final];
    }
}
for (instAlt in [Alterado])
{
   if (instAlt not in [Primeira geração] and instAlt not in [Segunda geração])
    instAlt -> [Final];
}
```

Pseudo-código da abordagem Diferencial de instruções

4.2.2.2 Histórico

Esta abordagem de modo a funcionar perante múltiplas gerações e alterações de código por parte do utilizador deve ser complementada com um mecanismo de histórico (Tabela 4.2). As seguintes tabelas demonstram um caso onde a versão do código gerado é sempre igual, mas o utilizador faz alterações a cada geração.

1º Geração	Alterado	2º Geração	Final
Nome = nome; Idade = idade;	Nome = nome; Idade = 39;	Nome = nome; Idade = idade;	Nome = nome; Idade = 39;

Tabela 4.1: 2ª Geração

Guarda em histórico as alterações efectuadas.

Conteúdo	Operação
Idade = idade;	Remove
Idade = 39;	Adiciona

Tabela 4.2: Alterações da 2º geração

Atual	Alterado	3º Geração	Final
Nome = nome; Idade = 39;	Nome = Pedro; Idade = 39;	Nome = nome; Idade = idade;	Idade = idade; Nome = Pedro;

Tabela 4.3: 3ª Geração

O resultado reflecte o pretendido após aplicar o histórico ao resultado final da 3º geração.

Final
Idade = 39; Nome = Pedro;

Tabela 4.4: Resultado final

4.2.3 Diff And Patch

Em suma, a abordagem anteriormente apresentada é baseada em comparação de instruções que no fundo não passam de linhas de texto. Desde 1972 [17] existem sistemas como o SCCS que nos permitem fazer essa comparação de texto. Um desses sistemas, atual e muito usado na área de engenharia informática, é o Git. As técnicas de comparação e fusão de texto deste sistema assentam sobre a biblioteca LibXDiff de Davide Libenzi [20].

Esta, por sua vez, é uma biblioteca que implementa um algoritmo de Diff. Os algoritmos Diff procuram comparar duas sequências de caracteres (Tabela 4.5) e descobrir como é que a primeira pode ser transformada na segunda. De um modo geral, o funcionamento do algoritmo passa por, ao comparar as sequências, criar uma listagem (Tabela 4.6) com várias sequências de caracteres, cada uma destas sequências têm associado uma operação, que pode ser de remoção, inserção ou modificação [76].

Sequencia 1	Sequencia 2
Verde	Verde
Amarelo	Amarelo
Azul	Vermelho

Tabela 4.5: Sequencias

Conteúdo	Operação
Verde, Amarelo	Iguais
Azul	Remover
Vermelho	Inserir

Tabela 4.6: Listagem

Existem disponíveis várias bibliotecas com pequenas variações que implementam o algoritmo de Diff, sendo que os resultados finais são muito similares. A maior parte destas

é desenvolvida por "entusiastas", mas em 2006 a Google criou a sua própria implementação (Diff-Match-Patch) para ser usada e potencializar os Google Docs. Esta está disponível em open source e em várias linguagens de programação. Nesta implementação, não só foi desenvolvido o algoritmo de Diff mas também o de Match e Patch. O algoritmo de Patch aplica uma lista de patches a um texto mesmo quando o texto subjacente não corresponde ao original [33]. Estes patches podem ser obtidos a partir de uma lista resultante de um Diff.

Sequencia 1	**Sequencia 2**	**Sequencia 3**
Verde	Verde	Laranja
Amarelo	Amarelo	Amarelo
Azul	Vermelho	Azul
		Preto

Tabela 4.7: Sequencias

```
@@ -11,8 +11,13 @@
elo%0A
-Azul
+Vermelho%0A
```

Patch gerado com base no Diff entre a sequência 1 e sequência 2.

Como podemos observar, o Patch contém a informação para adicionar a palavra Vermelho e remover a palavra Azul. Se este Patch for aplicado à sequência 3 o resultado será o seguinte.

Resultado
Laranja
Amarelo
Vermelho
Preto

Tabela 4.8: Resultado

O que se pretende com esta abordagem é utilizar esta biblioteca, usufruindo dos seus algoritmos de Diff e Patch para conseguir unir o código gerado com o código alterado pelo programador. Se aplicarmos estes algoritmos ao exemplo utilizado anteriormente na abordagem de "Diferencial de instruções", ou seja:

É gerado um primeiro método com as instruções 1, 2 e 3.

```
Instrução 1
Instrução 2
Instrução 3
```

Método (1º Geração)

O programador irá adicionar a instrução 4 e remover a instrução 3.

```
Instrução 1
Instrução 2
Instrução 4
```

Método alterado pelo programador

O gerador irá remover a instrução 1 e adicionar a instrução 0 e a instrução 5.

```
Instrução 0
Instrução 2
Instrução 3
Instrução 5
```

Método (2º Geração)

```
Instrução 0
Instrução 2
Instrução 4
Instrução 5
```

Método final

O resultado final é o pretendido. E esta ao contrário da abordagem anterior não necessita de um ponto de referência semelhante entre os métodos. Fora solucionar o problema de novas instruções ou remoção destas por ambos (programador ou gerador), permitirá também reter alterações feitas pelo programador.

É gerado um primeiro método com as instruções 1, 2 e 3.

```
Instrução 1
Carro é amarelo
Instrução 3
```

Método (1º Geração)

O programador irá alterar a instrução 2.

```
Instrução 1
Carro é verde
Instrução 3
```

Método alterado pelo programador

O gerador gera novamente as instruções 1, 2 e 3.

```
1 Instrução 1
2 Carro é amarelo
3 Instrução 3
```

Método (2º Geração)

```
1 Instrução 1
2 Carro é verde
3 Instrução 3
```

Método final

Neste exemplo um Diff é criado com as diferenças entre o método (1º geração) e p método alterado.

amarelo	Remover
verde	Inserir

Tabela 4.9: Diff

Ao criar um Patch com base neste Diff e aplicá-lo ao método (2º Geração) as alterações feitas pelo programador permaneceram.

No entanto, prevê-se que, quando houver grandes alterações em instruções, tanto pela parte do programador como pelo gerador, o resultado dos algoritmos Diff e Patch não seja válido. Em relação a este caso propõe-se que seja gerado o método (2ª geração) mas que seja comentado, de forma a que a junção entre os dois fique a cargo do programador após análise deste.

4.2.3.1 Limitações do Diff and Match

A abordagem apresentada anteriormente só por si não chega pois, tanto o gerador como o programador, podem alterar inúmeras vezes os métodos.

São geradas instruções de afetação de Nome e de Idade.

```
1 Nome = nome
2 Idade = idade
```

Método (1º Geração)

O programador altera a afetação da idade para 39.

```
1 Nome = nome
2 Idade = 39
```

Método alterado pelo programador

O gerador volta a escrever exatamente as mesmas instruções.

```
1 Nome = nome
2 Idade = idade
```

Método (2º Geração)

```
1 Nome = nome
2 Idade = 39
```

Método final

O método final fica com alteração da idade, o que está correto pois, pretendemos que a alteração do programador permaneça. Se houver uma segunda interação:

O método de 1º geração é o método final da primeira interação.

```
1 Nome = nome
2 Idade = 39
```

Método (1º Geração)

O programador vai alterar a afetação do nome.

```
1 Nome = Pedro
2 Idade = 39
```

Método alterado pelo programador

O gerador vai acrescentar uma linha de afetação de morada.

```
1 Nome = nome
2 Idade = idade
3 Morada = morada
```

Método (2º Geração)

```
1 Nome = Pedro
2 Idade = idade
3 Morada = morada
```

Método final

É feito um Diff entre o método (1º geração) e o método (2º geração)

Nome = nome	Remover
Nome = Pedro	Inserir

Tabela 4.10: Resultado

Quando é feito o Patch sobre o método, (2ª geração) a afetação do nome é alterada e é acrescentada uma nova linha da afetação de morada, o que está correto, mas a alteração à afetação da idade, que foi alterada pelo programador na 1º interação, é perdida.

Para solucionar este problema, será necessário guardar todos os Patch depois de serem gerados e em cada interação serem aplicados sequencialmente. Para isso, propõe-se agrupar os patches por classe, método numa única estrutura, serializá-la em formato json e guardá-la em ficheiro na raiz do projeto gerado. Este ficheiro deverá também conter informação acerca da própria geração: versão do gerador e data da última geração.

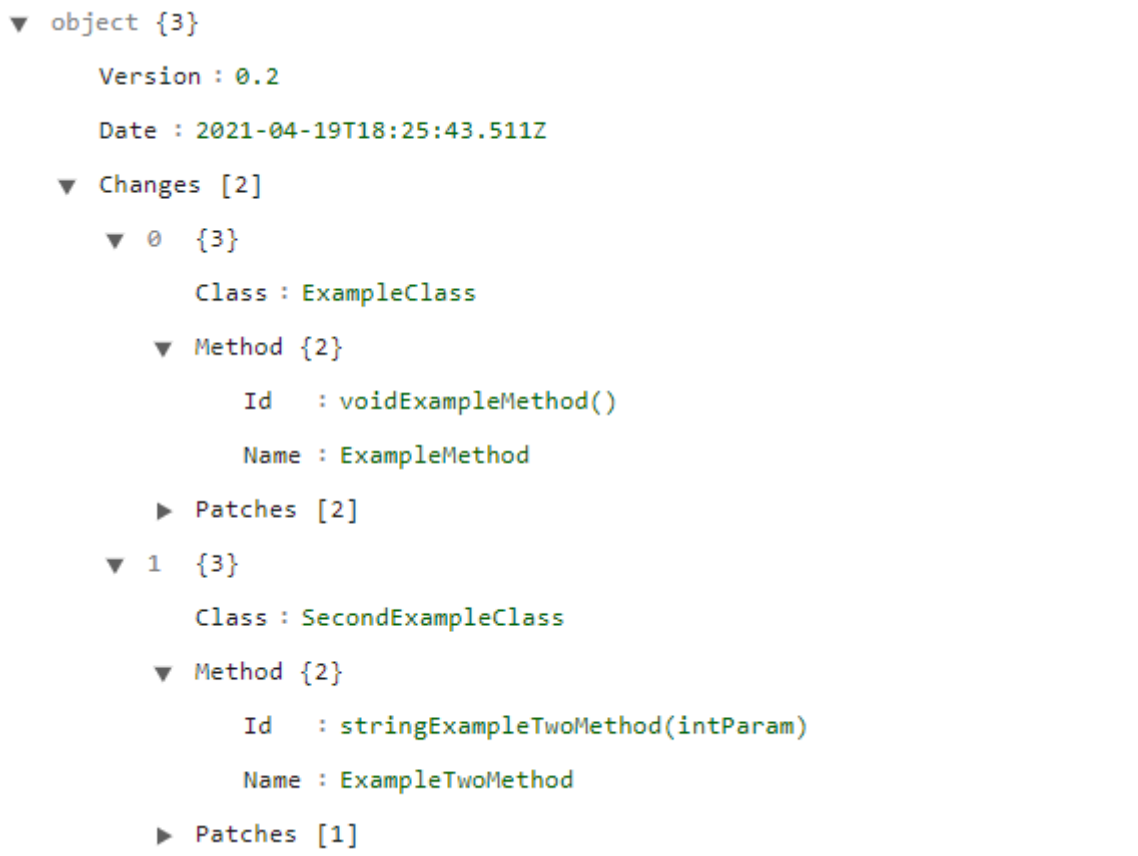


Figura 4.4: Exemplo

Idade = idade	Remover
Idade = 39	Inserir

Tabela 4.11: Patch 1, criado na primeira interação

Nome = nome	Remover
Nome = Pedro Morada = morada	Inserir

Tabela 4.12: Patch 2, criado na segunda interação

```
1 Nome = nome
2 Idade = 39
```

Método (1º Geração)

```
1 Nome = Pedro
2 Idade = 39
```

Método alterado pelo programador

```
1 Nome = nome
2 Idade = idade
3 Morada = morada
```

Método (2º Geração)

```
1 Nome = Pedro
2 Idade = 39
3 Morada = morada
```

Método final

Após aplicar ambos os patches teremos como resultado final a alteração à afetação de idade na primeira interação e a alteração ao nome na segunda interação feitas pelo programador. Como também o adicionar da afetação de morada criada pelo gerador.

4.3 Limitações gerais dos geradores não-destrutivos

Para gerar um projecto múltiplas vezes sem problemas, apenas o é possivel fazer usando uma versão do gerador que não tenha sofrido uma grande atualização desde a primeira vez que foi usado para a geração do projecto em causa. Isto porque podem ser causados bugs principalmente devido a versões de bibliotecas diferentes. Como por exemplo: em 2020 um gerador gerava projectos na framework .net core 2.1, mas em 2021 passou a gerar projetos na framework .net core 3.0. Ao gerar o projeto de 2020 novamente em 2021 iria acontecer um update da framework. Em relação ao código gerado, não era expectável haver problemas, pois já teria sido adaptado para a versão 3.0, mas o código desenvolvido

pelos programadores poderia deixar de funcionar por estar a usar formas ou métodos que, entretanto, deixaram de ser suportados ou foram alterados na framework .net core 3.0.

Capítulo 5

Implementação

De modo a testar as abordagens de junção de código foi alterado o gerador de código descrito no capítulo 7 para implementar as abordagens descritas no capítulo 4. Para a fase de comparação optou-se pela comparação textual em detrimento da comparação AST porque as bibliotecas que usam o algoritmo da abordagem de Diff And Patch estão todas implementadas com a comparação textual e, para o caso da abordagem de Diferencial de instruções, o esforço que é necessário para a implementação de uma comparação de instruções AST é enorme em relação aos benefícios e resultados obtidos. Para melhor compreensão, algumas partes foram simplificadas ou omitidas, como é o caso da geração de comentários do código ou das classes responsáveis por gerar outros componentes da aplicação como por exemplo o gerador de repositorios, neste caso porque a abordagem das classes "Generator" é igual para todas. Foi dado um maior foco nas abordagens de junção.

5.1 Gerador de código

5.1.1 Arquitectura

O seguinte diagrama representa arquitectura do gerador de código.

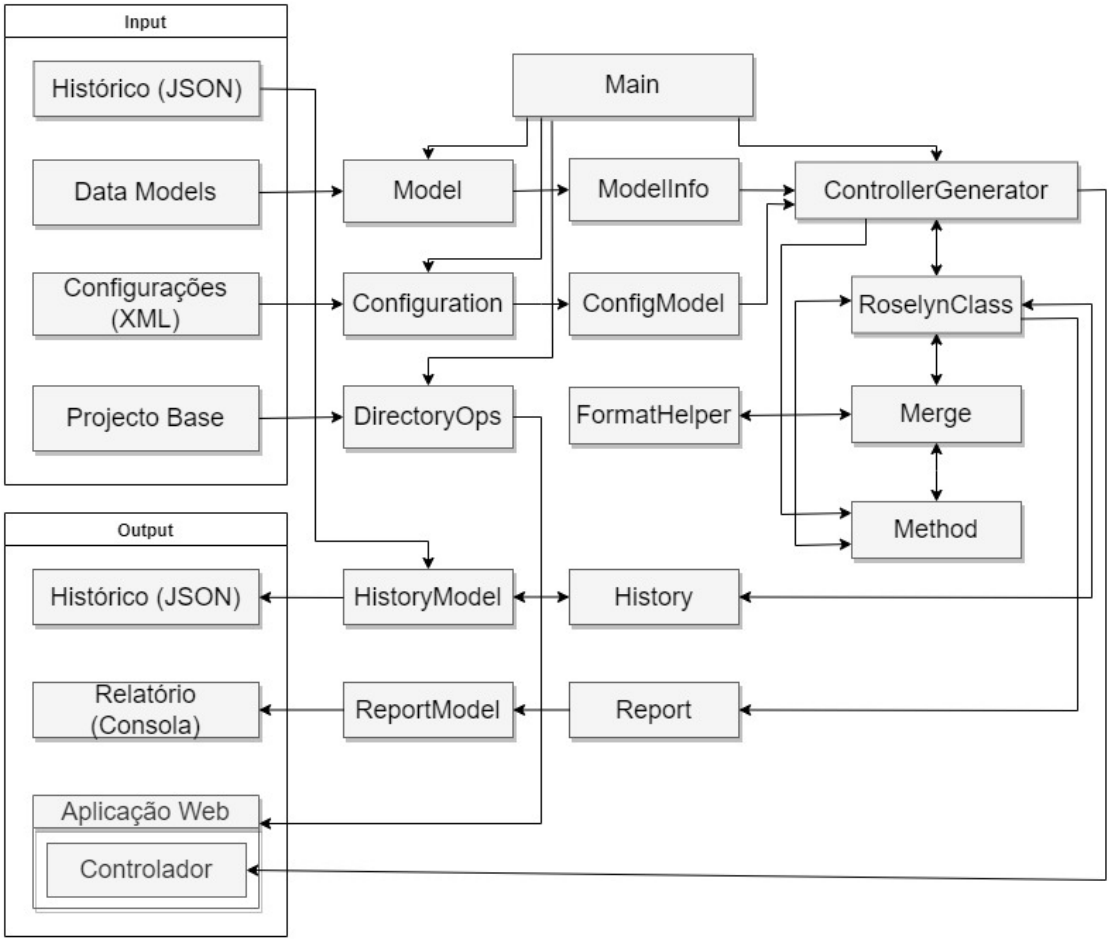


Figura 5.1: Diagrama da arquitectura do gerador de código

5.1.2 Código fonte

A implementação do código encontra-se no apêndice A .

Classe Main

A classe principal da solução é responsável pelas seguintes "tarefas":

1. Ler as configurações que são usadas pelo gerador de código para personalizar a aplicação gerada.

2. Copiar o projecto base para o destino do projecto final.

3. Fazer o parse dos modelos e guardar as suas propriedades e a informação respectiva.
4. Carregar para memória o histórico de gerações
5. Percorrer toda a informação dos modelos, e criar os repositórios, gestores, controladores, modelos, vistas e ficheiros de typescript.
6. Criar os ficheiros de resource em múltiplas línguas.
7. Criar os ficheiros de enum.
8. Criar as configurações da aplicação.
9. Criar e configurar o contexto da base de dados.
10. Criar o menu.
11. Instalar módulos externos.
12. Instalar biblioteca do conector à base de dados.
13. Mostrar o relatório de sucesso na geração de métodos.
14. Fazer build do projecto.
15. Criar a base de dados.

Classe ConfigModel

Esta classe contêm as configurações do gerador de código que vão servir para personalizar a aplicação gerada. As configurações são lidas de um ficheiro Extensible Markup Language (XML).

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Config>
  <ApproachType>DiffPatch</ApproachType>
  <Database>
    <Type>SQLServer</Type><!-- Database Type : SQLServer , PostgreSQL...  -->
    <ConnectionString>Server=localhost;Database=example;user id=sa;password=
    Agap2DEV;Trusted_Connection=False</ConnectionString> <!-- Database
    connection string -->
  </Database>
  <Project>Lusofona.Tese</Project>
  <Languages>en, pt</Languages>
  <Modules>
    <Module>Log</Module> <!-- Add log viewer -->
  </Modules>
</Config>
```

Exemplo de Configurações (XML)

Classe DirectoryOps

Esta classe está encarregue de copiar o projeto base. Usar um projeto base tem a mais valia que, se for necessário alterar alguma função base, não é necessário alterar o gerador, basta alterar o método no respetivo ficheiro que é copiado nesta fase. O gerador pode ser fechado e o output vai variar conforme o input (projeto base) que tiver. É desnecessário estar a gerar ficheiros que são sempre iguais de projeto para projeto e não sofrem alterações. Também desta forma é possível adicionar logo funcionalidades base como por exemplo um login. A maioria das aplicações web tem login. Partindo desta premissa pode ser criado logo todas classes e métodos necessários ao funcionamento do login e adicionadas ao projeto base.

A classe deve não só copiar o projeto base para o a diretoria do projeto final, como ao mesmo tempo que faz esta tarefa, deve modificar todos os nomes dos ficheiros e diretorias para terem o namespace do projeto gerado. O conteúdo dos ficheiros copiados também deve sofrer da mesma alteração.

Exemplo: Se o novo namespace for "Lusofona.Tese" e o projeto base tiver como namespace "Generated.Base", a pasta do sub-projecto de negócios "Assets/Project/Generated.Base.Business" deve passar a "GeneratedProject/Lusofona.Tese.Business" e o conteúdo "namespace Generated.Base.Business/Models" que se encontra na classe "GeneratedProject/Lusofona.Tese.Business/Models/UserBusinessModel.cs" deve passar a "namespace Lusofona.Tese.Business/Models".

Por último, os modelos que servem de base à geração de todo o código são copiados para a pasta dos modelos da camada de acesso à base de dados. É possível fazer esta cópia diretamente pois estes modelos são criados com a sintaxe comum de c#. Os modelos ao serem criados desta forma aceleram o processo de desenvolvimento para quem está familiarizado com a linguagem e dispensa da criação de um parser para uma linguagem própria.

Classe Model

O intuito desta classe é fazer o parse a todos os modelos criados pelo programador que serviram de base para a geração das classes da aplicação. Este parse é feito através da criação de uma árvore de sintaxe de onde é retirada informação do modelo, como por exemplo as suas propriedades. Essa informação é guarda num dicionário, onde a chave é o nome do modelo.

Se os modelos base do gerador tiverem atributos personalizados, essa informação também deve ser recolhida e guardada por esta classe. Atributos personalizados podem por exemplo ser um atributo "[History]" que indicia que a entidade deve ter uma histórico ou "[Crud(Option.Create)]" que indica que para a entidade aquando gerado o CRUD apenas deve ser gerada a funcionalidade de create.

Esta classe deve também ter em atenção se o modelo contém uma classe Abstracta, se for este esse o caso deve também incluir as propriedades.

```
namespace Lusofona.Tese
{
    public partial class User
    {
        [Required]
        [Key]
        public Guid Id { get; set; }
        [MaxLength(100)]
        [DataType(DataType.EmailAddress)]
        [EmailAddress]
        public string Email { get; set; }
        [Required]
        [DataType(DataType.Password)]
        public byte[] Password { get; set; }
    }
}
```

Exemplo de Data Model

Classe ModelInfo

A informação obtida através da classe Model é guardada numa estrutura do tipo ModelInfo. Esta estrutura permite não só guardar a informação base dos modelos, como o seu nome e propriedades mas também acrescentar alguma informação que é útil para as classes encarregues da geração de código. Por exemplo, pode indicar se a propriedade tem uma relação com outro modelo ou com que nome deve ser apresentada a propriedade no interface gráfico.

Classe ControllerGenerator

É apenas demonstrada a classe de geração de controladores pois a abordagem das classes "Generator" é igual em todas. Esta classe está encarregue de criar os métodos de CRUD na classe controladora do modelo correspondente.

```
using Thesis.Example.Business;
using Thesis.Example.Business.Models;
using Microsoft.AspNetCore.Mv;


namespace Lusofona.Tese.Web.Controllers
{
  private readonly IUserManager userManager;
  [Authorize]
  public class UserController
  {
    public UserController(IUserManager userManager, IOptionsSnapshot <
    AppSettings> settings)
    {
      this.userManager = userManager;
```

```
15      }
16
17      public IActionResult Index ()
18      {
19        return View();
20      }
21
22      public async Task <IActionResult> Details(Guid id)
23      {
24        OperationResult <UserBusinessModel> result = await UserManager.GetById(
        id);
25        if (result.Succeeded)
26        {
27          UserViewModel userViewModel = UserViewModel.FromBusinessModel(result.
        Data);
28          return View (userViewModel);
29        }
30        else
31        {
32          ViewBag.ErrorDescription = result.ErrorDescription;
33          return View(new UserViewModel());
34        }
35      }
36    }
37  }
```

Exemplo de ouput gerado pela classe ControllerGenerator

Classe RoselynClass

Esta classe tem a cargo a gestão da junção dos métodos e variáveis de classe geradas. Ao ser inicializa vai fazer parse da classe existente com o intuito de guardar a informação dos métodos e variáveis já existentes. Quando é adicionado um método ou variável é usada a classe Merge para juntar o novo método com o existente.

classe Method

A informação dos métodos obtida através da classe RoselynClass é guardada numa estrutura do tipo Method. Esta estrutura permite guardar a informação base dos métodos.

Classe Merge

É nesta classe que estão aplicadas as abordagens descritas no capítulo 4. Dependendo do tipo de abordagem (approachType) vai executar o algoritmo correspondente com base no ultimo método gerado, o método corrente e o novo método que a ser gerado.

Classe FormatHelper

O objectivo desta classe é ajudar na formatação de modo a facilitar a fase de comparação. Uma má indentação ou mesmo uma grande quantidade de instruções por linha pode

piorar a comparação. Exemplo:

```
public void MetodoDeExemplo()
{
    Instrucao1.Metodo(1).Metodo2()
        .Metodo(3);
}
// Transformar em:
public void MetodoDeExemplo()
{
    Instrucao1.Metodo(1)
              .Metodo2()
              .Metodo(3);
}
```

Classe History

Esta classe é encarregue de gerir o histórico de gerações.

Classe HistoryModel

Esta classe representa a estrutura do Histórico de gerações.

Figura 5.2: Exemplo de Histórico

Classe Report

Esta classe é encarregue de gerir o relatório de sucesso da geração de métodos.

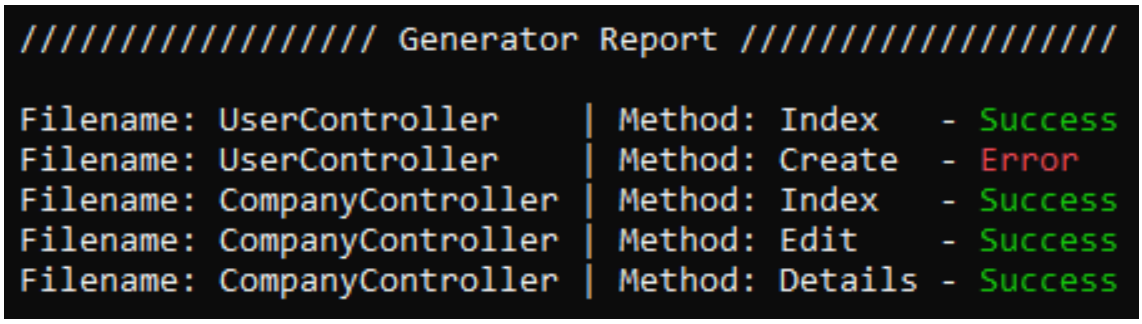


Figura 5.3: Exemplo de relatório

Classe ReportModel

Esta classe representa o item do relatório do sucesso da geração de um método

Capítulo 6

Resultados

Foi criado um conjunto de testes baseado em casos reais comuns. Neste capítulo serão demonstrados os resultados dos testes contra as abordagens estudadas previamente. Os seguintes métodos são um exemplo possível para o 1º teste das tabelas 6.1, 6.3 e 6.4.

O gerador gera o seguinte método:

```
1 public void ExampleMethod
2 {
3   printf ("Método com 1 linha, L1 - Linha 1");
4 }
```

Código de exemplo da coluna 1ª Geração

O programador altera o método e adiciona uma linha (Add. L0) antes da linha existente (L1):

```
1 public void ExampleMethod
2 {
3   printf ("Método com 2 linhas, L0 - Linha 0");
4   printf ("Método com 1 linha, L1 - Linha 1");
5 }
```

Código de exemplo da coluna Alterado

O gerador volta a gerar o método (Fica igual):

```
1 public void ExampleMethod
2 {
3   printf ("Método com 1 linha, L1 - Linha 1");
4 }
```

Código de exemplo da coluna 2ª Geração

Existe uma junção e o método final fica com a alteração do utilizador:

```
public void ExampleMethod
{
  printf ("Método com 2 linhas, L0 - Linha 0");
  printf ("Método com 1 linha, L1 - Linha 1");
}
```

Código de exemplo da coluna Final

6.1 Marcação de início e fim de código

A tabela 6.1 apresenta a aplicação de 35 tipos testes usando a técnica de Diferencial de instruções. As colunas representam uma 1º geração de código, uma alteração pelo programador, uma 2º geração de código e os seus consequentes resultados. A coluna Final é apresentada na cor Verde ou Vermelha consoante o resultado final seja válido ou inválido. Neste contexto, qualquer resultado cujo código tenha um comportamento diferente do expetável é considerado inválido (mesmo que não tenha erros de compilação).

Nº	Cenário Alterado	Cenário 2º Geração	1º Geração	Alterado	2º Geração	Final
1	Add. L0	Fica igual	L1	L0 L1	L1	L0 L1
2	Fica igual	Add. L0	L1	L1	L0 L1	L0 L1
3	Add. L2	Fica igual	L1	L1 L2	L1	L1 L2
4	Fica igual	Add. L2	L1	L1	L1 L2	L1 L2
5	Add. L0	Add. L0	L1	L0 L1	L0 L1	L0 L0 L1
6	Add. L2	Add. L2	L1	L1 L2	L1 L2	L1 L2 L2
7	Add. L0, Add. L2	Fica igual	L1	L0 L1 L2	L1	L0 L1 L2
8	Fica igual	Add. L0, Add. L2	L1	L1	L0 L1 L2	L0 L1 L2
9	Add. L0, Add. L1 Add. L3, Add. L4	Fica igual	L2	L0 L1 L2 L3 L4	L2	L0 L1 L2 L3 L4

Nº	Cenário Alterado	Cenário 2º Geração	1º Geração	Alterado	2º Geração	Final
10	Fica igual	Add. L0, Add. L1 Add. L3, Add. L4	L2	L2	L0 L1 L2 L3 L4	L0 L1 L2 L3 L4
11	Add. L2, Add. L3	Fica igual	L1 L4	L1 L2 L3 L4	L1 L4	L1 L4
12	Fica igual	Add. L2, Add. L3	L1 L4	L1 L4	L1 L2 L3 L4	L1 L2 L3 L4
13	Rem. L1, Add. L3	Fica igual	L1 L2 L1 L4	L2 L3 L4	L1 L2 L1 L4	L1 L2 L1 L4
14	Fica igual	Rem. L1, Add. L3	L1 L2 L1 L4	L1 L2 L1 L4	L1 L2 L3 L4	L1 L2 L3 L4
15	Add. L2, Add. L4	Fica igual	L1 L3 L5	L1 L2 L3 L4 L5	L1 L3 L5	L1 L3 L5
16	Fica igual	Add. L2, Add. L4	L1 L3 L5	L1 L3 L5	L1 L2 L3 L4 L5	L1 L2 L3 L4 L5
17	Add. L2, Add. L4	Add. L2, Add. L4	L1 L3 L5	L1 L2 L3 L4 L5	L1 L2 L3 L4 L5	L1 L2 L3 L4 L5
18	Add. L1	Rem. L2, Add L3	L2	L1 L2	L3	L1 L3
19	Rem. L2, Add L3	Add. L1	L2	L3	L1 L2	L1 L2
20	Fica igual	Add. L0, Rem. L1	L1 L2 L3	L1 L2 L3	L0 L2 L3	L0 L2 L3
21	Add. L0, Rem. L1	Add. L0, Rem. L1	L1 L2 L3	L0 L2 L3	L0 L2 L3	L0 L1 L2 L3
22	Add. L0, Rem. L1	Fica igual	L1 L2 L3	L0 L2 L3	L1 L2 L3	L0 L1 L2 L3

Nº	Cenário Alterado	Cenário 2º Geração	1º Geração	Alterado	2º Geração	Final
23	Add. L2, Rem. L1	Add. L3, Rem. L1	L1	L2	L3	L2 L3
24	Rem. L1	Fica igual	L1 L2	L2	L1 L2	L1 L2
25	Fica igual	Rem. L1	L1 L2	L1 L2	L2	L2
26	Rem. L1	Rem. L1	L1 L2	L2	L2	L2
27	Rem. L2	Fica igual	L1 L2	L1	L1 L2	L1 L2
28	Fica igual	Rem. L2	L1 L2	L1 L2	L1	L1
29	Rem. L2	Rem. L2	L1 L2	L1	L1	L1
30	Add. L4, Add. L3	Add. L1, Add. L3	L2	L2 L4 L3	L2 L1 L3	L2 L1 L3 L4 L3
31	Fica Igual	Rem. L1, Rem. L2 Add. L3	L1 L2	L1 L2	L3	L3
32	Rem. L1	Rem. L2	L1 L2		L1	L1
33	Add. L2 no meio	Fica igual	L1 L3	L1 L2 L3	L1 L3	L1 L3
34	Rem. L2, Add. L3	Rem. L2, Add. L4	L1 L2	L1 L3	L1 L4	L1 L4 L3
35	Rem. L2, Add. L3	Fica Igual	L1 L2 L2	L1 L2 L3	L1 L2 L2	L1 L2 L2 L3

Tabela 6.1: Resultados da Marcação de início e fim de código

Como se pode ver pelos casos indicados a vermelho na tabela 6.1 esta abordagem não resolve todos casos de teste. Um caso de exemplo é o teste 5 onde o gerador gera o seguinte método:

```
1 public void ExampleMethod
2 {
3   // Início de código gerado
4   printf ("Método com 1 linha, L1 - Linha 1");
5   // Fim de código gerado
6 }
```

Código de exemplo da coluna 1ª Geração

O programador descobre um bug e para o corrigir tem de adicionar uma linha (Add. L0). Esta linha é adicionada antes do marcador de início, de forma à linha não ser reescrita numa futura geração:

```
public void ExampleMethod
{
  printf ("Método com 2 linhas, L0 - Linha 0");
  // Início de código gerado
  printf ("Método com 1 linha,  L1 - Linha 1");
  // Fim de código gerado
}
```

Código de exemplo da coluna Alterado

A equipa que programa o gerador descobre o mesmo bug que o programador e para o corrigir gera também uma nova linha (Add. L0). Esta linha é adicionada dentro dos marcadores pois é código gerado automaticamente.

```
public void ExampleMethod
{
  // Início de código gerado
  printf ("Método com 2 linhas, L0 - Linha 0");
  printf ("Método com 1 linhas L1 - Linha 1");
  // Fim de código gerado
}
```

Código de exemplo da coluna 2ª Geração

Existe uma junção e o método final é o seguinte:

```
public void ExampleMethod
{
  printf ("Método com 2 linhas, L0 - Linha 0");
  // Início de código gerado
  printf ("Método com 2 linhas, L0 - Linha 0");
  printf ("Método com 1 linha, L1 - Linha 1");
  // Fim de código gerado
}
```

Código de exemplo da coluna Final

O resultado final do método está errado, pois a linha 0 (L0) aparece repetida 2 vezes.

6.1.1 Marcação de início e fim de código sobre instruções alteradas

A abordagem não permite a junção de instruções alteradas, pois as instruções alteradas encontram-se entre o marcador de início e de fim, sendo rescritas cada vez que existe uma nova geração.

6.2 Diferencial de instruções

A tabela 6.2 apresenta a aplicação de 35 tipos testes usando a técnica de Diferencial de instruções. As colunas representam uma 1º geração de código, uma alteração pelo programador, uma 2º geração de código e os seus consequentes resultados. A coluna Final é apresentada na cor Verde ou Vermelha consoante o resultado final seja válido ou inválido. Neste contexto, qualquer resultado cujo código tenha um comportamento diferente do expetável é considerado inválido (mesmo que não tenha erros de compilação).

Nº	Cenário Alterado	Cenário 2º Geração	1º Geração	Alterado	2º Geração	Final
1	Add. L0	Fica igual	L1	L0 L1	L1	L1 L0
2	Fica igual	Add. L0	L1	L1	L0 L1	L0 L1
3	Add. L2	Fica igual	L1	L1 L2	L1	L1 L2
4	Fica igual	Add. L2	L1	L1	L1 L2	L1 L2
5	Add. L0	Add. L0	L1	L0 L1	L0 L1	L1 L0
6	Add. L2	Add. L2	L1	L1 L2	L1 L2	L1 L2
7	Add. L0, Add. L2	Fica igual	L1	L0 L1 L2	L1	L1 L0 L2
8	Fica igual	Add. L0, Add. L2	L1	L1	L0 L1 L2	L0 L1 L2
9	Add. L0, Add. L1 Add. L3, Add. L4	Fica igual	L2	L0 L1 L2 L3 L4	L2	L2 L0 L1 L3 L4
10	Fica igual	Add. L0, Add. L1 Add. L3, Add. L4	L2	L2	L0 L1 L2 L3 L4	L0 L1 L2 L3 L4
11	Add. L2, Add. L3	Fica igual	L1 L4	L1 L2 L3 L4	L1 L4	L1 L4 L2 L3
12	Fica igual	Add. L2, Add. L3	L1 L4	L1 L4	L1 L2 L3 L4	L1 L2 L3 L4

Nº	Cenário Alterado	Cenário 2º Geração	1º Geração	Alterado	2º Geração	Final
13	Rem. L1, Add. L3	Fica igual	L1 L2 L1 L4	L1 L2 L3 L4	L1 L2 L1 L4	L1 L2 L4 L3
14	Fica igual	Rem. L1, Add. L3	L1 L2 L1 L4	L1 L2 L1 L4	L1 L2 L3 L4	L1 L2 L3 L4
15	Add. L2, Add. L4	Fica igual	L1 L3 L5	L1 L2 L3 L4 L5	L1 L3 L5	L1 L3 L5 L2 L5
16	Fica igual	Add. L2, Add. L4	L1 L3 L5	L1 L3 L5	L1 L2 L3 L4 L5	L1 L2 L3 L4 L5
17	Add. L2, Add. L4	Add. L2, Add. L4	L1 L3 L5	L1 L2 L3 L4 L5	L1 L2 L3 L4 L5	L1 L2 L3 L4 L5
18	Add. L1	Rem. L2, Add L3	L2	L1 L2	L3	L3 L1
19	Rem. L2, Add L3	Add. L1	L2	L3	L1 L2	L1 L3
20	Fica igual	Add. L0, Rem. L1	L1 L2 L3	L1 L2 L3	L0 L2 L3	L0 L2 L3
21	Add. L0, Rem. L1	Add. L0, Rem. L1	L1 L2 L3	L0 L2 L3	L0 L2 L3	L0 L2 L3
22	Add. L0, Rem. L1	Fica igual	L1 L2 L3	L0 L2 L3	L1 L2 L3	L2 L3 L0
23	Add. L2, Rem. L1	Add. L3, Rem. L1	L1	L2	L3	L2
24	Rem. L1	Fica igual	L1 L2	L2	L1 L2	L2
25	Fica igual	Rem. L1	L1 L2	L1 L2	L2	L2
26	Rem. L1	Rem. L1	L1 L2	L2	L2	L2
27	Rem. L2	Fica igual	L1 L2	L1	L1 L2	L1
28	Fica igual	Rem. L2	L1 L2	L1 L2	L1	L1

Nº	Cenário Alterado	Cenário 2º Geração	1º Geração	Alterado	2º Geração	Final
29	Rem. L2	Rem. L2	L1 L2	L1	L1	L1
30	Add. L4, Add. L3	Add. L1, Add. L3	L2	L2 L4 L3	L2 L1 L3	L2 L1 L3 L4
31	Fica Igual	Rem. L1, Rem. L2 Add. L3	L1 L2	L1 L2	L3	L3
32	Rem. L1	Rem. L2	L1 L2	L1	L2	
33	Add. L2 no meio	Fica igual	L1 L3	L1 L2 L3	L1 L3	L1 L2 L3
34	Rem. L2, Add. L3	Rem. L2, Add. L4	L1 L2	L1 L3	L1 L4	L1 L4 L3
35	Rem. L2, Add. L3	Fica Igual	L1 L2 L2	L1 L2 L3	L1 L2 L2	L1 L2 L3

Tabela 6.2: Resultados do Diferencial de instruções

6.2.1 Diferencial de instruções sobre instruções alteradas

A tabela 6.3 apresenta a aplicação de 11 testes usando a abordagem de Diferencial de instruções. As instruções representam uma 1º geração de código, uma alteração pelo programador, uma 2º geração de código, o conseguinte resultado e por ultimo um resultado ideal caso o resultado final não tenha sido o espectável. O cabeçalho do teste é apresentada na cor Verde ou Vermelha consoante o resultado final ser válido. Estes testes consistem principalmente na alteração da sintaxe das instruções. Para a demonstração de resultados será utilizada a linguagem c#.

Nº	1
1º Geração	`repository.GetAllPaged<BatchMovement>(pagedRequest);`
Alterado	`repository.GetAllPaged(pagedRequest);`
2º Geração	`repository.GetAllPaged<BatchMovement>();`
Final	`repository.GetAllPaged<BatchMovement>();` `repository.GetAllPaged(pagedRequest);`
Ideal	`repository.GetAllPaged();`

Nº	2
1º Geração	`myArray.Select(p => p.id == 9);`
Alterado	`myArray.Select(p => p.id == 9 && p.isOk == true);`

2º Geração	`if (1 == 2)`
	`{`
	`  myArray.Select(p => p.id == 9 && p.NewProp == 10);`
	`}`

Final	`if (1 == 2)`
	`{`
	`  myArray.Select(p => p.id == 9 && p.newProp == 10);`
	`}`
	`myArray.Select(p => p.id == 9 && p.isOk == true);`

Ideal	`if (1 == 2)`
	`{`
	`  myArray.Select(p => p.id == 9 && p.isOk == true && p.NewProp == 10);`
	`}`

Nº	**3**

1º Geração	`if (Age == "38")`
	`{`
	`  Console.Writeline("38");`
	`}`

Alterado	`if (Age == "38" && Country=="Portugal")`
	`{`
	`  Console.Writeline("38");`
	`  Console.Writeline("Portugal");`
	`}`

2º Geração	`if (Age == "38" && Sex == "Male")`
	`{`
	`  Console.Writeline("38");`
	`  Console.Writeline("Male");`
	`}`

final	`if (Age == "38" && Sex == "Male")`
	`{`
	`  Console.Writeline("38");`
	`  Console.Writeline("Male");`
	`}`
	`if (Age == "38" && Country == "Portugal")`
	`{`
	`  Console.Writeline("38");`
	`  Console.Writeline("Portugal");`
	`}`

Ideal	`if (Age == "38" && Sex == "Male" && Country == "Portugal")`
	`{`
	`  Console.Writeline("38");`
	`  Console.Writeline("Portugal");`
	`  Console.Writeline("Male");`
	`}`

Nº	**4**
1º Geração	`Method().Include("LINE 1");`
Alterado	`Method().Include("LINE 2").Exclude("LINE 3");`
2º Geração	`Method().Include("LINE 2").Exclude("LINE 4");`
Final	`Method().Include("LINE 2").Exclude("LINE 4");` `Method().Include("LINE 2").Exclude("LINE 3");`
Ideal	`Method().Include("LINE 2").Exclude("LINE 3").Exclude("LINE 4");`

Nº	5

1º Geração	`dbSchedule.Description = scheduleBusinessModel.Description;`
	`dbSchedule.ScheduleStatusTypeId =(int)scheduleBusinessModel.`
	`ScheduleStatusType;`

Alterado	`dbSchedule.Description = scheduleBusinessModel.Description;` `dbSchedule.ScheduleStatusTypeId = (int)ScheduleStatusTypeEnum.Closed;`

2º Geração	`dbSchedule.Description = scheduleBusinessModel.Description;`
	`dbSchedule.ScheduleStatusTypeId = (int)scheduleBusinessModel.`
	`ScheduleStatusType;`
	`dbSchedule.CreatedDate = DateTime.CreatedDate;`

Final	`dbSchedule.Description = scheduleBusinessModel.Description;`
	`dbSchedule.CreatedDate = DateTime.CreatedDate;`
	`dbSchedule.ScheduleStatusTypeId = (int)ScheduleStatusTypeEnum.Closed;`

Nº	6
1º Geração	`rp.Where(p => p.Id == id).Include(p => p.PersonStatusType).FirstAsync();`

Alterado	`rp`
	`.Where(p => p.Id == id)`
	`.Include(p => p.PersonStatusType)`
	`.FirstAsync();`

2º Geração	`rp.Where(p => p.Id == id).Include(p => p.PersonStatusType).FirstAsync();`

Final	`rp`
	`.Where(p => p.Id == id)`
	`.Include(p => p.PersonStatusType)`
	`.FirstAsync();`

Nº	7
1º Geração	`Console.Writeline("LINE 1");`
Alterado	`Console.Writeline("LINE 2");` `Console.Readline("LINE 3");`
2º Geração	`Console.Readline("LINE 4");` `Console.Writeline("LINE 2");`

Final	`Console.Readline("LINE 4");`
	`Console.Writeline("LINE 2");`
	`Console.Writeline("LINE 2");`
	`Console.Readline("LINE 3");`

Ideal	`Console.Readline("LINE 4");`
	`Console.Writeline("LINE 2");`
	`Console.Readline("LINE 3");`

Nº	8
1º Geração	`ds.Include(p => p.TeamResponsibleId).ThenInclude(p => p.Persons);`
Alterado	`ds.Include(p => p.TeamResponsibleId).ThenInclude(p => p.PersonTeams)` `.ThenInclude(pt => pt.Person);`
2º Geração	`ds.Include(p => p.TeamResponsibleId).ThenInclude(p => p.PersonTeams)` `.ThenInclude(pt => pt.Persons).ThenInclude(pt => pt.CreatedBy);`

Final	`ds.Include(p => p.TeamResponsibleId)`
	`.ThenInclude(p => p.PersonTeams)`
	`.ThenInclude(pt => pt.Persons)`
	`.ThenInclude(pt => pt.CreatedBy);`
	`.Include(p => p.TeamResponsibleId)`
	`.ThenInclude(p => p.PersonTeams)`
	`.ThenInclude(pt => pt.Person);`

Ideal	`ds.Include(p => p.TeamResponsibleId).Include(p => p.PersonTeams)` `.ThenInclude(pt => pt.Person).ThenInclude(pt => pt.CreatedBy);`

Nº	9
1º Geração	`ds.Include(p => p.Persons);`
Alterado	`ds.Include(p => p.PersonTeams)` `.ThenInclude(p => p.Person);`
2º Geração	`ds.Include(p => p.PersonTeams)` `.ThenInclude(p => p.CreatedBy);`

Final	`ds.Include(p => p.PersonTeams)`
	`.ThenInclude(p => p.CreatedBy)`
	`ds.Include(p => p.PersonTeams)`
	`.ThenInclude(p => p.Person);`

Ideal	`ds.Include(p => p.PersonTeams)`
	`ds.ThenInclude(p => p.Person)`
	`ds.ThenInclude(p => p.CreatedBy);`

Nº	10
1º Geração	`person.CallMethod(tmp1,var2,param3);`
Alterado	`person.CallMethod(tmp1,param3,teste4);`
2º Geração	`person.CallMethod(tmp1,var2);`
Final	`person.CallMethod(tmp1, var2)` `person.CallMethod(tmp1, param3, teste4);`
Ideal	`person.CallMethod(tmp1,teste4);`
Nº	11
1º Geração	`var ac = lp.Where(p => p.Id == legalProcessId, p => p.CreatedBy)` `return allCollaterals;`
Alterado	`return lp.Where(p => p.Id == legalProcessId)` `.SelectMany(p => p.ContractLegalProcesses);`

2º Geração	`var ac = lp.Where(p => p.Id == legalProcessId, p => p.CreatedBy)`
	`.Include(p=> p.Time);`
	`return allCollaterals;`

Final	`var ac = lp.Where(p => p.Id == legalProcessId, p => p.CreatedBy)`
	`.Include(p => p.Time);`
	`return lp.Where(p => p.Id == legalProcessId)`
	`.SelectMany(p => p.ContractLegalProcesses);`

Ideal	`return lp.Where(p => p.Id == legalProcessId)` `.SelectMany(p => p.ContractLegalProcesses).Include(p=> p.Time);`

Tabela 6.3: Resultados do Diferencial de instruções alteradas

6.3 Diff And Patch

A tabela 6.4 apresenta a aplicação de 35 tipos testes usando a técnica de Diferencial de instruções. As colunas representam uma 1º geração de código, uma alteração pelo programador, uma 2º geração de código e os seus consequentes resultados. A coluna Final é apresentada na cor Verde ou Vermelha consoante o resultado final seja válido ou inválido. Neste contexto, qualquer resultado cujo código tenha um comportamento diferente do expetável é considerado inválido (mesmo que não tenha erros de compilação).

Nº	Cenário Alterado	Cenário 2º Geração	1º Geração	Alterado	2º Geração	Final
1	Add. L0	Fica igual	L1	L0 L1	L1	L0 L1
2	Fica igual	Add. L0	L1	L1	L0 L1	L0 L1
3	Add. L2	Fica igual	L1	L1 L2	L1	L1 L2
4	Fica igual	Add. L2	L1	L1	L1 L2	L1 L2
5	Add. L0	Add. L0	L1	L0 L1	L0 L1	L0 L1
6	Add. L2	Add. L2	L1	L1 L2	L1 L2	L1 L2
7	Add. L0, Add. L2	Fica igual	L1	L0 L1 L2	L1	L0 L1 L2
8	Fica igual	Add. L0, Add. L2	L1	L1	L0 L1 L2	L0 L1 L2
9	Add. L0, Add. L1 Add. L3, Add. L4	Fica igual	L2	L0 L1 L2 L3 L4	L2	L0 L1 L2 L3 L4
10	Fica igual	Add. L0, Add. L1 Add. L3, Add. L4	L2	L2	L0 L1 L2 L3 L4	L0 L1 L2 L3 L4
11	Add. L2, Add. L3	Fica igual	L1 L4	L1 L2 L3 L4	L1 L4	L1 L2 L3 L4
12	Fica igual	Add. L2, Add. L3	L1 L4	L1 L4	L1 L2 L3 L4	L1 L2 L3 L4
13	Rem. L1, Add. L3	Fica igual	L1 L2 L1 L4	L1 L2 L3 L4	L1 L2 L1 L4	L1 L2 L3 L4
14	Fica igual	Rem. L1, Add. L3	L1 L2 L1 L4	L1 L2 L1 L4	L1 L2 L3 L4	L1 L2 L3 L4

Nº	Cenário Alterado	Cenário 2º Geração	1º Geração	Alterado	2º Geração	Final
15	Add. L2, Add. L4	Fica igual	L1 L3 L5	L1 L2 L3 L4 L5	L1 L3 L5	L1 L2 L3 L4 L5
16	Fica igual	Add. L2, Add. L4	L1 L3 L5	L1 L3 L5	L1 L2 L3 L4 L5	L1 L2 L3 L4 L5
17	Add. L2, Add. L4	Add. L2, Add. L4	L1 L3 L5	L1 L2 L3 L4 L5	L1 L2 L3 L4 L5	L1 L2 L3 L4 L5
18	Add. L1	Rem. L2, Add L3	L2	L1 L2	L3	L1 L3
19	Rem. L2, Add L3	Add. L1	L2	L3	L1 L2	L1 L3
20	Fica igual	Add. L0, Rem. L1	L1 L2 L3	L1 L2 L3	L0 L2 L3	L0 L2 L3
21	Add. L0, Rem. L1	Add. L0, Rem. L1	L1 L2 L3	L0 L2 L3	L0 L2 L3	L0 L2 L3
22	Add. L0, Rem. L1	Fica igual	L1 L2 L3	L0 L2 L3	L1 L2 L3	L0 L2 L3
23	Add. L2, Rem. L1	Add. L3, Rem. L1	L1	L2	L3	L2
24	Rem. L1	Fica igual	L1 L2	L2	L1 L2	L2
25	Fica igual	Rem. L1	L1 L2	L1 L2	L2	L2
26	Rem. L1	Rem. L1	L1 L2	L2	L2	L2
27	Rem. L2	Fica igual	L1 L2	L1	L1 L2	L1
28	Fica igual	Rem. L2	L1 L2	L1 L2	L1	L1
29	Rem. L2	Rem. L2	L1 L2	L1	L1	L1
30	Add. L4, Add. L3	Add. L1, Add. L3	L2	L2 L4 L3	L2 L1 L3	L2 L1 L3 L4 L3

Nº	Cenário Alterado	Cenário 2º Geração	1º Geração	Alterado	2º Geração	Final
31	Fica Igual	Rem. L1, Rem. L2 Add. L3	L1 L2	L1 L2	L3	L3
32	Rem. L1	Rem. L2	L1 L2	L1	L2	
33	Add. L2 no meio	Fica igual	L1 L3	L1 L2 L3	L1 L3	L1 L2 L3
34	Rem. L2, Add. L3	Rem. L2, Add. L4	L1 L2	L1 L3	L1 L4	L1 L3
35	Rem. L2, Add. L3	Fica Igual	L1 L2 L2	L1 L2 L3	L1 L2 L2	L1 L2 L3

Tabela 6.4: Resultados do Diff And Patch

Denota-se que a percentagem de casos de teste com o resultado final válido é superior às duas abordagens previamente apresentadas, porém esta abordagem também não garante que todos os resultados de teste sejam válidos.

Um caso de exemplo é o teste 34 onde o gerador gera o seguinte método:

```
public void ExampleMethod
{
  printf ("L1 - Linha 1");
  printf ("L2 - Linha 2");
}
```

Código de exemplo da coluna 1ª Geração

O programador remove a linha linha 2 (Rem. L2) e adiciona uma nova linha (Add. L3).

```
public void ExampleMethod
{
  printf ("L1 - Linha 1");
  printf ("L3 - Linha 3");
}
```

Código de exemplo da coluna Alterado

O código volta a ser gerado, mas existiu uma atualização no gerador e já não é gerada a linha 2 (Rem. L2). Uma nova linha (Add. L4) passa a ser gerada.

```
public void ExampleMethod
{
  printf ("L1 - Linha 1");
  printf ("L4 - Linha 4");
}
```

Código de exemplo da coluna 2ª Geração

Existe uma junção e o método final é o seguinte:

```
public void ExampleMethod
{
  printf ("L1 - Linha 1");
  printf ("L3 - Linha 3");
}
```

Código de exemplo da coluna Final

O resultado final do método está errado, se o utilizador adicionou uma nova linha (Add. L3) e o gerador também gera uma nova linha (Add. L4), o resultado expectável é que o código final contenha as 2 linhas:

```
public void ExampleMethod
{
  printf ("L1 - Linha 1");
  printf ("L4 - Linha 4");
  printf ("L3 - Linha 3");
}
```

Código expectável

A abordagem "Diff and Patch" como demonstrada no capítulo 4 cria um Diff entre o método de 1º geração e o método alterado (Tabela 6.6) e tendo como base esse Diff é gerado um Patch que é aplicado ao método de 2º geração produzindo um método final (Tabela 6.7).

Como o algoritmo se baseia em comparar duas sequências de caracteres e não propriamente comparar instruções de código, o algoritmo assume que existe uma alteração em um carácter e não que existem novas instruções para acrescentar produzindo apenas a alteração do carácter. Situações semelhantes a esta fazem com que esta abordagem não garanta uma taxa de sucesso de 100%.

Método 1º Geração	**Método Alterado**	**Método 2º Geração**
printf("L1")	printf("L1")	printf("L1")
printf("L2")	printf("L3")	printf("L4")

Tabela 6.5: Métodos

printf("L1") printf("L	Igual
2	Remover
3	Inserir
")	Igual

Tabela 6.6: Dif gerado

```
{@@ -19,7 +19,7 @@
 f(%22L
-2
+3
 %22)
}
```

Patch contém a informação para adicionar o carácter 3 na posição 26 (19+7) e remover o carácter 2 na posição 26 (19+7).

Resultado
printf("L1")
printf("L3")

Tabela 6.7: Resultado

6.3.1 Diff And Patch sobre instruções alteradas

A tabela 6.8 apresenta a aplicação de 11 testes usando a abordagem de Diff And Patch. As instruções representam uma 1º geração de código, uma alteração pelo programador, uma 2º geração de código, o conseguinte resultado e por ultimo um resultado ideal caso o resultado final não tenha sido o espectável. O cabeçalho do teste é apresentada na cor Verde ou Vermelha consoante o resultado final ser válido. Estes testes consistem principalmente na alteração da sintaxe das instruções. Para a demonstração de resultados será utilizada a linguagem c#.

Nº	1
1º Geração	`repository.GetAllPaged<BatchMovement>(pagedRequest);`
Alterado	`repository.GetAllPaged(pagedRequest);`
2º Geração	`repository.GetAllPaged<BatchMovement>();`
Final	`repository.GetAllPaged();`
Nº	2
1º Geração	`myArray.Select(p => p.id == 9);`
Alterado	`myArray.Select(p => p.id == 9 && p.isOk == true);`

2º Geração	`if (1 == 2)`
	`{`
	`  myArray.Select(p => p.id == 9 && p.NewProp == 10);`
	`}`

Final	`if (1 == 2)`
	`{`
	`  myArray.Select(p => p.id == 9 && p.isOk == true && p.NewProp == 10);`
	`}`

Nº	3

1º Geração	`if (Age == "38")`
	`{`
	`  Console.Writeline("38");`
	`}`

Alterado	`if (Age == "38" && Country=="Portugal")`
	`{`
	`  Console.Writeline("38");`
	`  Console.Writeline("Portugal");`
	`}`

2º Geração	`if (Age == "38" && Sex == "Male")`
	`{`
	`  Console.Writeline("38");`
	`  Console.Writeline("Male");`
	`}`

Final	`if (Age == "38" && Sex == "Male" && Country == "Portugal")`
	`{`
	`  Console.Writeline("38");`
	`  Console.Writeline("Portugal");`
	`  Console.Writeline("Male");`
	`}`

Nº	**4**
1º Geração	`Method().Include("LINE 1");`
Alterado	`Method().Include("LINE 2").Exclude("LINE 3");`
2º Geração	`Method().Include("LINE 2").Exclude("LINE 4");`
Final	`Method().Include("LINE 2").Exclude("LINE 3").Exclude("LINE 4");`
Nº	**5**

1º Geração	`dbSchedule.Description = scheduleBusinessModel.Description;`
	`dbSchedule.ScheduleStatusTypeId =(int)scheduleBusinessModel.`
	`ScheduleStatusType;`

Alterado	`dbSchedule.Description = scheduleBusinessModel.Description;` `dbSchedule.ScheduleStatusTypeId = (int)ScheduleStatusTypeEnum.Closed;`

2º Geração	`dbSchedule.Description = scheduleBusinessModel.Description;`
	`dbSchedule.ScheduleStatusTypeId = (int)scheduleBusinessModel.`
	`ScheduleStatusType;`
	`dbSchedule.CreatedDate = DateTime.CreatedDate;`

Final	`dbSchedule.Description = scheduleBusinessModel.Description;`
	`dbSchedule.ScheduleStatusTypeId = (int)ScheduleStatusTypeEnum.Closed;`
	`dbSchedule.CreatedDate = DateTime.CreatedDate;`

Nº	**6**
1º Geração	`rp.Where(p => p.Id == id).Include(p => p.PersonStatusType).FirstAsync();`

Alterado	`rp`
	`.Where(p => p.Id == id)`
	`.Include(p => p.PersonStatusType)`
	`.FirstAsync();`

2º Geração	`rp.Where(p => p.Id == id).Include(p => p.PersonStatusType).FirstAsync();`

Final	`rp`
	`.Where(p => p.Id == id)`
	`.Include(p => p.PersonStatusType)`
	`.FirstAsync();`

Nº	7
1º Geração	`var ac = lp.Where(p => p.Id == legalProcessId, p => p.CreatedBy)` `return allCollaterals;`
Alterado	`return lp.Where(p => p.Id == legalProcessId)` `.SelectMany(p => p.ContractLegalProcesses);`

2º Geração	`var ac = lp.Where(p => p.Id == legalProcessId, p => p.CreatedBy)`
	`.Include(p=> p.Time);`
	`return allCollaterals;`

Final	`return lp.Where(p => p.Id == legalProcessId)` `.SelectMany(p => p.CreatedBy).Include(p => p.ontTLgaCProcesses);`
Ideal	`return lp.Where(p => p.Id == legalProcessId)` `.SelectMany(p => p.ContractLegalProcesses).Include(p=> p.Time);`
Nº	8
1º Geração	`ds.Include(p => p.TeamResponsibleId).ThenInclude(p => p.Persons);`
Alterado	`ds.Include(p => p.TeamResponsibleId).ThenInclude(p => p.PersonTeams)` `.ThenInclude(pt => pt.Person);`
2º Geração	`ds.Include(p => p.TeamResponsibleId).ThenInclude(p => p.PersonTeams)` `.ThenInclude(pt => pt.Persons).ThenInclude(pt => pt.CreatedBy);`

Final	`ds.Include(p => p.TeamResponsibleId).ThenInclude(p => p.PersonTeams)`
	`.ThenInclude(pt => pt.PersonTeam).ThenInclude(pt => pt.Persons)`
	`.ThenInclude(pt => pt.CreatedBy);`

Ideal	`ds.Include(p => p.TeamResponsibleId).Include(p => p.PersonTeams)` `.ThenInclude(pt => pt.Person).ThenInclude(pt => pt.CreatedBy);`
Nº	9
1º Geração	`ds.Include(p => p.Persons);`
Alterado	`ds.Include(p => p.PersonTeams)` `.ThenInclude(p => p.Person);`
2º Geração	`ds.Include(p => p.PersonTeams)` `.ThenInclude(p => p.CreatedBy);`

Final	`ds.Include(p => p.PersonTeamT)`
	`.ThenInclude(p => p.Personeams)`
	`.ThenInclude(p => p.CreatedBy);`

Ideal	`ds.Include(p => p.PersonTeams)`
	`.ThenInclude(p => p.Person)`
	`.ThenInclude(p => p.CreatedBy);`

Nº	10
1º Geração	`Console.Writeline("LINE 1");`
Alterado	`Console.Writeline("LINE 2");` `Console.Readline("LINE 3");`
2º Geração	`Console.Readline("LINE 4");` `Console.Writeline("LINE 2");`

Final	`Console.Readline("LINE 2");`
	`Console.Writeline("LINE 3");`
	`Console.Readline("LINE 2");`

Ideal	`Console.Readline("LINE 4");`
	`Console.Writeline("LINE 2");`
	`Console.Readline("LINE 3");`

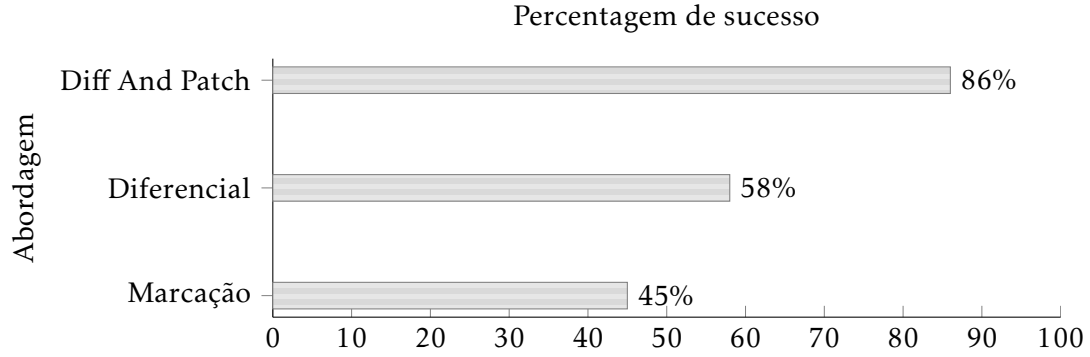
Nº	11
1º Geração	`person.CallMethod(tmp1,var2,param3);`
Alterado	`person.CallMethod(tmp1,param3,teste4);`
2º Geração	`person.CallMethod(tmp1,var2);`
Final	`person.CallMethod(tmp1, ), teste;`
Ideal	`person.CallMethod(tmp1,teste4);`

Tabela 6.8: Resultados do Diff And Patch sobre instruções alteradas

Na secção 6.4 é apresentada uma analise criticas dos resultados demonstrados anteriormente.

6.4 Análise crítica

A tabela 6.9 demonstra o resultado percentual do sucesso do resultado dos testes sobre as abordagens estudadas.



Abordagem	Percentagem de sucesso
Diff And Patch	86%
Diferencial de instruções	58%
Marcação de início e fim de código	45%

Tabela 6.9: Percentagem de sucesso

A primeira abordagem "Marcação de início e fim de código" demonstrada é a mais simplista, tendo como resultado positivo apenas 45% dos testes efetuados. O baixo resultado desta abordagem deve-se principalmente ao problema referido na subsecção 6.1.1. Em suma, o problema reside no facto de que todas as linhas entre os marcadores de início e de fim são substituídas de cada vez que existe uma geração. Isto porque, com esta abordagem, apenas é possível inserir código antes ou depois do bloco principal do método, não sendo possível inserir linhas no meio do bloco ou alterar as linhas geradas. Apenas é recomendada para casos em que queremos garantir que o código gerado não é alterado e o programador pode apenas adicionar novas instruções.

A abordagem "Diferencial de instruções" apresenta melhores resultados do que a abordagem referida anteriormente, cerca de 58% de sucesso. A criação de um algoritmo que se baseia na comparação das instruções entres os três métodos (1º Geração, Alterado, 2º Geração) soluciona o principal problema encontrado na abordagem referida anteriormente. Porém também apresenta resultados negativos principalmente devido à própria comparação. Para remover ou inserir novas instruções este algoritmo compara instruções mas estas instruções podem ter sido alteradas ou removidas, impossibilitando aferir a sua localização. Em relação a instruções alteradas é muito pouco eficiente porque considera uma instrução alterada como uma instrução nova. Uma melhoria nesta abordagem seria realizar a comparação AST em vez da comparação textual de forma a entender que partes dessa instrução foram removidas ou inseridas. Desta forma seria possível não só fazer a junção das instruções mas também a junção dentro da própria instrução, o que se traduz num aumento de casos de sucesso. Mesmo com uma implementação de AST na abordagem "Diferencial de instruções" sobre os testes das tabelas 6.2 e 6.3 é possível concluir que a sua percentagem de sucesso seria inferior a do "Diff and Patch" pois continuaria a não resolver o problema da localização das instruções alteradas.

A abordagem "Diff and Patch" mostrou ter a maior percentagem sucesso. Como referido no capítulo 4 o algoritmo de Diff procura comparar duas sequências de caracteres. É este algoritmo que torna possível não só comparar instruções diferentes, mas também comparar diferenças dentro da mesma instrução resolvendo os principais problemas encontrados nas abordagens previamente referidas. Porém, apesar da elevada percentagem de sucesso (86%) esta abordagem não resolve todos os problemas. Pois a característica principal (comparação de duas sequências de caracteres) que torna este algoritmo vantajoso é também a que causa problemas na junção. Como o algoritmo Diff se baseia em comparar duas sequências de caracteres e não a comparar instruções de código, por vezes temos os seguintes problemas:

- As instruções em vez de serem adicionadas ou removidas são alteradas (Caso de teste 34 da tabela 6.4 e caso de teste 10 da tabela 6.8)
- Instruções com sintaxe errada (Casos de teste 7, 9, 11 da tabela. 6.8)
- Falta de instruções (Casos de teste 8, da tabela. 6.8)

Estes problemas são detetados quando o algoritmo de Patch falha ou quando não é possível compilar o código. Não foi possível chegar a uma solução para os evitar, mas de forma a mitigá-los, no fim da geração é apresentado ao programador um relatório com todos os problemas de junção para que este manualmente os possa corrigir.

Capítulo 7

Utilização do gerador em contexto empresarial

7.1 Análise de caso real

Neste capítulo apresenta-se uma análise da implementação de um gerador de código em contexto empresarial. São descritas as razões que levaram ao desenvolvimento deste, o porquê de se ter optado pelas tecnologias e metodologias escolhidas, os problemas encontrados e como os colmatar, os resultados reais dos testes experimentais sobre uma variedade de cenários e por fim, o resultado prático sobre o impacto do gerador em um projeto real.

Durante o período de 2018-2022 exerci funções na empresa agap2IT. A agap2IT é uma organização europeia fundada em 2005 na área dos Sistemas de Informação, Ciência e Tecnologia. Dentro da empresa estive colocado no departamento DXSpark que consiste no desenvolvimento de soluções informáticas à medida dos clientes. Neste departamento que conta com cerca de 40 elementos tive o cargo de programador sénior e team leader em vários projetos de diferentes áreas de negócio. As minhas principais responsabilidades foram a criação de arquiteturas base, desenvolvimento de funcionalidades complexas e revisão de código.

No desenvolvimento de várias aplicações web observei que independente do tipo de negócio, estas continham características semelhantes e problemas comuns como:

- Cada engenheiro de software criava as aplicações baseadas na sua experiência profissional, escolhendo a arquitetura, tecnologia e metodologias que por vezes não são as que mais se adequam. Por este motivo, não só o processo de integração de outro engenheiro acaba por se verificar complexo, visto que é necessário um estudo exaustivo da arquitetura e metodologias utilizadas para dar seguimento ao projeto,

como também a própria aplicação pode apresentar problemas de qualidade e performance. Em consequência destas dificuldades, o que acontece muitas vezes é que este mesmo engenheiro se vê obrigado a apagar o código existente e a refazê-lo à sua maneira, sendo que mais uma vez pode não ser o código ideal.

Um programador mais júnior, devido à falta de experiência, pode facilmente ignorar uma camada na arquitetura de um projeto como, por exemplo, ligar uma camada visual diretamente à camada de acesso aos dados criando certamente problemas para futuros desenvolvimentos nessa área do código.

- Existia muito desenvolvimento repetitivo, o que acaba por ser maçador para o engenheiro informático, criando por vezes desinteresse.

  Em uma aplicação com vários formulários de inserção, parte do trabalho será criar vários "inputs" para a informação pretendida. Estes "inputs" vão apenas variar no seu nome. Criar este conteúdo acaba por ser algo mecânico que se torna maçador.

- Sendo muitas destas aplicações semelhantes, não só existem módulos que são copiados entre eles e sujeitos a pequenas alterações como, por exemplo, um módulo de autenticação, mas também blocos internos semelhantes como por exemplo um CRUD de uma entidade.

  Se numa aplicação existir todo o mecanismo de manipulação de um fornecedor e se for necessário criar o mesmo mecanismo para um cliente, a maneira mais fácil será copiar todas as classes e fazer algumas alterações como o próprio nome ou as sua propriedades. Esta técnica de "Copy Paste" muitas vezes cria vários pequenos erros.

De um modo geral estes problemas dão origem a uma grande taxa de erros humanos e más implementações que levam à má qualidade do código e atrasos nas entregas das aplicações. Por isso levantei a questão de como melhorar o respetivo processo de desenvolvimento. Numa primeira instância comecei por criar uma forma de automatizar o desenvolvimento repetitivo, algo que me remeteu para a criação de uma ferramenta de geração de código como solução. A primeira grande questão foi em que linguagem deveria ser desenvolvido? A conclusão foi o MVC.Net [58] da Microsoft. Esta escolha foi baseada no potencial da linguagem e, com esta linguagem, é possível desenvolver todo o tipo de tarefas de forma estruturada que resultam em produtos finais estáveis e rápidos. É uma linguagem atual, pois encontra-se em constante desenvolvimento e tem um enorme apoio não só da comunidade, mas como da própria marca (Microsoft). Os projetos finais serem todos desenvolvidos na mesma linguagem, ajuda a que facilmente qualquer programador possa trocar de projeto sem grande dificuldade, resultando logo num ganho de tempo/dinheiro. Ao contrário da linguagem de programação, as bases de dados não podem ser sempre do mesmo tipo, principalmente porque muitas vezes o cliente já tem a sua própria base de dados. Nestes casos, para usar sempre o mesmo tipo de base de dados seria necessário criar migrações entre as bases de dados. Este processo pode ser moroso e

criar alguns problemas, como dados perdidos. Para ser possível o projeto gerado ligar-se a diferentes bases de dados, optou-se por usar um ORM, neste caso a Entity Framework. A Entity Framework [59] é o ORM principal da Microsoft contando com 13 anos de um estável e constante desenvolvimento.

Os geradores de código podem ter vários tipos de input como por exemplo modelos, ficheiros de XML, ficheiros de Unified Modeling Languag (UML), etc. Como com a Entity Framework é possível gerar a base de dados a partir de modelos (code first), porque não usar os mesmos modelos para servirem de base à geração de código? No fundo, estes modelos representam as entidades que queremos manipular na aplicação. Ao usar estes modelos deixa de ser necessário a criação de uma estrutura e sintaxe própria para input e da implementação de um parser. Por outro lado, é mais fácil e rápido para os programadores, pois já estão ambientados com a sintaxe. Após a observação de vários projetos cheguei a acordo sobre uma arquitetura que implementasse os últimos padrões de desenho e fosse suficientemente potente para satisfazer qualquer uma das aplicações web a desenvolver. Usar sempre a mesma arquitetura tem o principal benefício de facilmente fazer com que um programador que mude de projeto, consiga rapidamente ambientar-se e navegar no código. Também deixa de ser necessário ter arquitetos ou elementos mais seniores para desenvolvê-la. Sendo assim, elementos júniores podem começar logo a trabalhar no projeto desde o início deste. Os principais padrões de desenho da arquitetura são o MVC e a injeção de dependências. Esta conta também com um DAL bem elaborado.

Após a criação de classes para gerarem a arquitetura base do projeto, entendi que estas mesmas classes com o código de geração são muito mais extensas do que as classes geradas por este. Ou seja, para criar ficheiros de código quase vazios era necessário um enorme esforço. Visto que a arquitetura era sempre a mesma, ou seja, os ficheiros eram sempre os mesmos, então porque não os copiamos apenas? Daí surgiu a ideia de ter um projeto para servir de base que é copiado no início da geração e apenas são substituídas algumas propriedades como o caso do namespace das classes ou os nomes dos sub-projetos da arquitetura de forma a corresponderam com o projeto a ser gerado.

Como o principal desafio começou pelos écrans de funcionalidades de CRUD foram criados geradores dos vários ficheiros de repositórios, controladores, vistas etc. de forma a garantir um CRUD funcional. Como resultado final ao ser inserido um modelo utilizador que conta com propriedades como nome e idade, é suposto ser gerada uma aplicação onde é possível criar, editar, visualizar, apagar e listar uma lista de utilizadores. Estes geradores contam com alguma inteligência para atingirem os resultados mais próximos do pretendido. Como por exemplo, se o utilizador tiver uma propriedade perfil que é uma relação para outro modelo, o que é interessante é aparecer no écran de criação um select com todos os perfis e não uma caixa de texto para escrever o id do perfil. A partir do pressuposto que os modelos base são criados com as code conventions corretas os geradores implementam um conjunto de regras que resolvem situações como o exemplo anterior.

Para camada de visualização segui a recomendação da Microsoft para o uso de Html5 [62]

e Typescript [85] e usei as bibliotecas de Jquery [46] para facilmente manipular o javascript e o Bootstrap [11] que é um encaixe perfeito para aplicações web com base em CRUDs.

Já com uma versão básica funcional, o procedimento foi tentar entender que funcionalidade deveriam constar das aplicações geradas. Ao identificar funcionalidades transversais a todas as aplicações web, como por exemplo, um login com recuperação de password, um sistema de logging e audit parti para a implementação destas no projeto base. Porém, daí surgiram questões como por exemplo: "com apenas metade dos projetos a implementar a exportação para excel das entidades, será que faz sentido estar a gerar a exportação em todos os projetos? A resposta foi negativa, criando assim um sistema de módulos de funcionalidades que podem ser ou não incluídas no projeto. Estas funcionalidades específicas são acionadas no ficheiro de configurações do gerador. Este ficheiro não só guarda estas preferências como também que tipo de base de dados usa, a connectionstring para se ligar, o nome do projeto etc.

Após o gerador estar a funcionar e a conseguir gerar aplicações funcionais surgiu um problema, quando era necessário alterar algo nos modelos base, todo o código que já tinha sido alterado pelos programadores desaparecia. Originalmente para colmatar este problema os programadores tinham de gerar um projeto paralelo ao que estavam a trabalhar e depois manualmente procederem a junção do código gerado com o código alterado. Este é um trabalho moroso. Para a resolução deste problema surgiu o estudo que é apresentado nesta tese.

A metodologia Agile dita que para haver desenvolvimento tem de existir uma análise prévia [4]. Em termos práticos para os programadores começarem a desenvolver é necessário que já exista um documento de analise funcional. O problema que se levanta é que esta premissa nem sempre é verdade. Um programador parado é um custo e muitas vezes para evitar esta situação o que acontece é que os projetos começam a ser desenvolvidos sem terem uma análise funcional fechada. Na prática vão-se criar modelos sem todas as propriedades e relações entre eles. Por cada novo capítulo do documento de análise funcional aparecem novos modelos, e por cada retificação do cliente existem alterações nos modelos já existentes e possivelmente alterados entrando num ciclo de novas gerações por cada versão do documento de analise funcional.

No capítulo 6, foram apresentados resultados de testes experimentais sobre uma grande variedade de cenários, que poderão ou não ocorrer com frequência numa aplicação real. Apresentam-se agora resultados reais, baseados numa aplicação que foi desenvolvida durante o período 15/10/2020-30/07/2021.

A seguinte tabela 7.1 demonstra alguns exemplos encontrados na aplicação usando a abordagem de Diff And Patch. As instruções representam uma geração de código, uma alteração pelo programador, uma 2º geração de código e o consequente resultado. Nos seguintes exemplos tudo correu bem, ou seja, nem as alterações feitas pelo programador nem pelo gerador foram apagadas, o que aconteceria no caso de um gerador destrutivo.

Nº	1
Descrição	A propriedade "DeathDate" deixa de ser obrigatória. (Programador)

1º Geração	`NIF = personViewModel.NIF,`
	`BirthDate = personViewModel.BirthDate.Value,`
	`DeathDate = personViewModel.DeathDate.Value,`

Alterado	`NIF = personViewModel.NIF,`
	`BirthDate = personViewModel.BirthDate.Value,`
	`DeathDate = personViewModel.DeathDate,`

2º Geração	`NIF = personViewModel.NIF,`
	`BirthDate = personViewModel.BirthDate.Value,`
	`DeathDate = personViewModel.DeathDate.Value,`

Final	`NIF = personViewModel.NIF,`
	`BirthDate = personViewModel.BirthDate.Value,`
	`DeathDate = personViewModel.DeathDate,`

Nº	2
Descrição	As entidades da base de dados relacionadas com as propriedades "Created By" e "Updated By" deixam de ser lidas. (Programador)

1º Geração	`return await ExecuteOperationAsync(async () =>`
	`{`
	`  return Success(await contractRepository.GetAllPaged(p =>`
	`  ContractBusinessModel.FromDataModel(p), pagedRequest,`
	`  p => p.CreatedBy, p => p.UpdatedBy));`
	`});`

Alterado	`return await ExecuteOperationAsync(async () =>`
	`{`
	`  return Success(await contractRepository.GetAllPaged(p =>`
	`  ContractBusinessModel.FromDataModel(p), pagedRequest));`
	`});`

2º Geração	`return await ExecuteOperationAsync(async () =>`
	`{`
	`  return Success(await contractRepository.GetAllPaged(p =>`
	`  ContractBusinessModel.FromDataModel(p), pagedRequest,`
	`  p => p.CreatedBy, p => p.UpdatedBy));`
	`});`

Final	`return await ExecuteOperationAsync(async () =>`
	`{`
	`  return Success(await contractRepository.GetAllPaged(p =>`
	`  ContractBusinessModel.FromDataModel(p), pagedRequest));`
	`});`

Nº	3
Descrição	É acrescentado um novo Manager "ContactsManager" as configurações de Dependecy Injection. (Gerador)
1º Geração	`services.AddTransient<IUserManager, UserManager>();`
Alterado	`services.AddTransient<IUserManager, UserManager>();`
2º Geração	`services.AddTransient<IUserManager, UserManager>();` `services.AddTransient<IContactsManager, ContactsManager>();`
Final	`services.AddTransient<IUserManager, UserManager>();` `services.AddTransient<IContactsManager, ContactsManager>();`

Nº	4
Descrição	É acrescentado um novo parâmetro "ContactsManager" ao método construtor e é atribuído a uma variável de classe. (Gerador)
1º Geração	`public PeopleController(IAccountManager accountManager, IPersonManager personManager, INationalityManager nationalityManager): base() { this.personManager = personManager; this.nationalityManager = nationalityManager;`
Alterado	`public PeopleController(IAccountManager accountManager, IPersonManager personManager, INationalityManager nationalityManager): base() { this.personManager = personManager; this.nationalityManager = nationalityManager;`
2º Geração	`public PeopleController(IAccountManager accountManager, IPersonManager personManager, IContactsManager contactsManager, INationalityManager nationalityManager): base() { this.personManager = personManager; this.contactsManager = contactsManager; this.nationalityManager = nationalityManager;`
Final	`public PeopleController(IAccountManager accountManager, IPersonManager personManager, IContactsManager contactsManager, INationalityManager nationalityManager): base() { this.personManager = personManager; this.contactsManager = contactsManager; this.nationalityManager = nationalityManager;`
Nº	**5**
Descrição	São atribuídos novos valores as variáveis "Information" e "Status" (Programador)
1º Geração	`SubType = GetContactSubType((ContactAddressTypeEnum) contactAddress.ContactAddressTypeId), Information = "", Status = "", IsPreferential = false`
Alterado	`SubType = GetContactSubType((ContactAddressTypeEnum) contactAddress.ContactAddressTypeId), Information = contactAddress.FullAddress, Status = "Confirmado", IsPreferential = false`
2º Geração	`SubType = GetContactSubType((ContactAddressTypeEnum) contactAddress.ContactAddressTypeId), Information = "", Status = "", IsPreferential = false`
Final	`SubType = GetContactSubType((ContactAddressTypeEnum) contactAddress.ContactAddressTypeId), Information = contactAddress.FullAddress, Status = "Confirmado", IsPreferential = false`

Nº	6
Descrição	É acrescentada uma nova variável "ProductAcronym" e a sua respectiva atribuição (Gerador)
1º Geração	`ProductDescription = contractBusinessModel.ProductDescription,` `ProductId = contractBusinessModel.ProductId,`
Alterado	`ProductDescription = contractBusinessModel.ProductDescription,` `ProductId = contractBusinessModel.ProductId,`

2º Geração	`ProductDescription = contractBusinessModel.ProductDescription,`
	`ProductAcronym = contractBusinessModel.ProductAcronym,`
	`ProductId = contractBusinessModel.ProductId,`

Final	`ProductDescription = contractBusinessModel.ProductDescription,`
	`ProductAcronym = contractBusinessModel.ProductAcronym,`
	`ProductId = contractBusinessModel.ProductId,`

Nº	7
Descrição	São comentadas as atribuições das variáveis "PersonaName" e "Nif" (Programador)

1º Geração	`ContractStatusType = contractBusinessModel.ContractStatusType,`
	`PersonName = contractBusinessModel.PersonName,`
	`NIF = contractBusinessModel.NIF,`

Alterado	`ContractStatusType = contractBusinessModel.ContractStatusType,`
	`//PersonName = contractBusinessModel.PersonName,`
	`//NIF = contractBusinessModel.NIF,`

2º Geração	`ContractStatusType = contractBusinessModel.ContractStatusType,`
	`PersonName = contractBusinessModel.PersonName,`
	`NIF = contractBusinessModel.NIF,`

Final	`ContractStatusType = contractBusinessModel.ContractStatusType,`
	`//PersonName = contractBusinessModel.PersonName,`
	`//NIF = contractBusinessModel.NIF,`

Nº	8
Descrição	É removida a atribuição da variável "Contracts" (Programador)

1º Geração	`TotalContracts = personBusinessModel.PersonContracts`
	`== null ? 0 : personBusinessModel.PersonContracts.Count,`
	`Contracts = personBusinessModel.PersonContracts`
	`!= null ? personBusinessModel.PersonContracts.Select(p =>`
	`ContractViewModel.FromBusinessModel(p)).ToList()`
	`: new List<ContractViewModel>(),`
	`UpdatedDate = personBusinessModel.UpdatedDate,`

Alterado	`TotalContracts = personBusinessModel.PersonContracts`
	`== null ? 0 : personBusinessModel.PersonContracts.Count,`
	`UpdatedDate = personBusinessModel.UpdatedDate,`

2º Geração	`TotalContracts = personBusinessModel.PersonContracts`
	`== null ? 0 : personBusinessModel.PersonContracts.Count,`
	`Contracts = personBusinessModel.PersonContracts`
	`!= null ? personBusinessModel.PersonContracts.Select(p =>`
	`ContractViewModel.FromBusinessModel(p)).ToList()`
	`: new List<ContractViewModel>(),`
	`UpdatedDate = personBusinessModel.UpdatedDate,`

Final	`TotalContracts = personBusinessModel.PersonContracts`
	`== null ? 0 : personBusinessModel.PersonContracts.Count,`
	`UpdatedDate = personBusinessModel.UpdatedDate,`

Nº	9
Descrição	A atribuição da variável "contractsList" é colocada numa linha única (Programador)
1º Geração	`public async Task<OperationPagedResult<ContractBusinessModel>> GetByPersonId(Guid personId) { return await ExecuteOperationAsync(async () => { List<ContractBusinessModel> contractsList = await personContractRepository .GetAllWhere(c => c.PersonId == personId) .Select(c => ContractBusinessModel.FromDataModel(c.Contract)) .ToListAsync(); PagedModel<ContractBusinessModel> pagedModel = new PagedModel<ContractBusinessModel>()`
Alterado	`public async Task<OperationPagedResult<ContractsBusinessModel>> GetByPersonId(Guid personId) { return await ExecuteOperationAsync(async () => { List<ContractsBusinessModel> contractsList = await personContractRepository.GetAllWhere(c => c.PersonId == personId).Select(c => ContractsBusinessModel.FromDataModel(c)).ToListAsync(); PagedModel<ContractsBusinessModel> pagedModel = new PagedModel<ContractsBusinessModel>()`
2º Geração	`public async Task<OperationPagedResult<ContractBusinessModel>> GetByPersonId(Guid personId) { return await ExecuteOperationAsync(async () => { List<ContractBusinessModel> contractsList = await personContractRepository .GetAllWhere(c => c.PersonId == personId) .Select(c => ContractBusinessModel.FromDataModel(c.Contract)) .ToListAsync(); PagedModel<ContractBusinessModel> pagedModel = new PagedModel<ContractBusinessModel>()`
Final	`public async Task<OperationPagedResult<ContractsBusinessModel>> GetByPersonId(Guid personId) { return await ExecuteOperationAsync(async () => { List<ContractsBusinessModel> contractsList = await personContractRepository .GetAllWhere(c => c.PersonId == personId) .Select(c => ContractsBusinessModel.FromDataModel(c)) .ToListAsync(); PagedModel<ContractsBusinessModel> pagedModel = new PagedModel<ContractsBusinessModel>()`

Nº	10
Descrição	É adicionado a afectação da propriedade “Relation” e “Status” do objecto “dbContAdd” (Programador)

1º Geração	`dbContAdd.Relation = ContAddBusinessModel.Relation;`
	`dbContAdd.ContactStatusTypeId =`
	`(int)ContAddBusinessModel.ContactStatusType;`

Alterado	`dbContAdd.Type = ContAddBusinessModel.Type;`
	`dbContAdd.Status = ContAddBusinessModel.Status;`
	`dbContAdd.ContactStatusTypeId =`
	`(int)ContAddBusinessModel.ContactStatusType;`

2º Geração	`dbContAdd.Relation = ContAddBusinessModel.Relation;`
	`dbContAdd.IsMainAddress = ContAddBusinessModel.IsMainAddress;`
	`dbContAdd.ContactStatusTypeId =`
	`(int)ContAddBusinessModel.ContactStatusType;`

Final	`dbContAdd.Type = ContAddBusinessModel.Type;`
	`dbContAdd.Status = ContAddBusinessModel.Status;`
	`dbContAdd.Relation = ContAddBusinessModel.Relation;`
	`dbContAdd.IsMainAddress = ContAddBusinessModel.IsMainAddress;`
	`dbContAdd.ContactStatusTypeId =`
	`(int)ContAddBusinessModel.ContactStatusType;`

Tabela 7.1: Resultados reais do Diff And Patch

Apesar de terem sido detetadas algumas limitações na abordagem DiffPatch na fase experimental que se traduziram numa taxa de 86% de sucesso, estas limitações não afetaram o desenvolvimento de um projeto real. A taxa de sucesso deste projecto acabou por ser ser de 100%. Múltiplas gerações não levantarem nenhum problema e desse modo não foi necessario nenhuma intervenção por parte dos programadores. Esta situação de sucesso não seria possível se fosse usado um gerador destrutivo ou um gerador não-destrutivo com abordagem de marcação de código ou diferencial de instruções. Contudo apesar de neste projecto concreto não terem sido detectados erros não é garantido que em outros projectos reais não venham a ser detectados erros, tais como são demonstrados no capitulo 6.

7.2 Entrevista

As seguintes questões foram colocadas a Ricardo Amado correntemente diretor executivo da DXSpark. Estas questões foram respondidas como diretor executivo onde consegue ter uma visão geral sobre o impacto do gerador de código no departamento e como programador, pois teve a experiência de utilizar o gerador para o desenvolvimento de uma aplicação para a gestão dos elementos do departamento.

Tanto a nível pessoal (Como programador) como a nível empresarial, quais são as mais valias na utilização de um gerador de código?
Um gerador de código, seja ele na categoria de low-code ou no-code tem vantagens evidentes. Na vertente no-code permite que um utilizador de negócio tenha capacidade e

autonomia para implementar funcionalidades, não sendo portanto necessário qualquer contratação de recursos especializados de IT/programação. Por outro lado, na vertente low-code, onde se enquadra o gerador aqui em contexto, embora exista a necessidade de envolver recursos especializados de IT, tipicamente existe uma maior flexibilidade de implementação de componentes ou funcionalidades específicas, pois temos acesso ao código gerado permitindo assim a customização do mesmo e adaptações de negócio. Neste contexto específico de um gerador de código, como acelerador inicial, permite a criação da estrutura base de desenvolvimento e arquitetura de código num objetivo near free of bugs. Ou seja, dado que o motor de geração é utilizado diversas vezes vai sendo melhorado e corrigido no seu core, garantindo assim que os projetos gerados a partir de desde têm uma probabilidade reduzida de terem erros. Assim, este tipo de geração permite ganho no tempo de desenvolvimento, padronização da arquitetura e qualidade.

Qual é o nível de conhecimento necessário para a utilização do gerador de código e como é a sua curva de aprendizagem?
A abstração criada por um gerador de código cria dificuldades de Onboarding de um novo consultor na sua utilização, visto que terá que passar por um período de adaptação na estrutura e arquitetura criada pelo gerador. Passado este momento, que é estimado em cerca de 1 mês, o consultor fica autónomo na utilização do mesmo, precisando apenas de apoio de competência de elementos mais séniores em especificidades de integração e desenvolvimento.

Como programador, se for necessário começar um projeto de raiz é preferível usar um gerador de código ou criar um projeto de origem?
Num contexto de programação e assumindo que a base de programação é idêntica de projeto para projeto, o que faz mais sentido em termos de eficácia e eficiência é utilizar um gerador de código. Desta forma garantimos a inicialização de um projeto de base, cumprindo standards com as melhores práticas de organização e arquitetura de desenvolvimento. No contexto de desenvolvimento .NET C, a existência de nuggets sustenta a visão descrita, pois a reutilização de código proporciona um ganho de desenvolvimento assim como a utilização de código já re-utilizado e testado, diminuindo a incidência de bugs. Por estes motivos o início de um projeto é sempre mais benéfico para o programador e para a instituição se for efetuado com um gerador de código.
Qual é a percentagem de tempo ganho num projeto e que benefícios "económicos" existem na utilização de um gerador de código?
No contexto de inicialização de projeto estimado em cerca de 5% a 10% de um projeto desenvolvido de raiz, o gerador de código proporciona esta estrutura out-of-the-box traduzindo esta percentagem em ganho direto com a sua utilização. Ainda no contexto de geração inicial toda a arquitetura e desenvolvimento de Permissões e Autenticação, estima-se um ganho de cerca de 32h em qualquer projeto iniciado com o gerador de código. Se assumirmos uma lógica de desenvolvimento CRUD e no contexto que cada

operação CRUD envolve cerca de 4h para toda a estrutura de Dados, Acesso a Dados, Camada de Negócio e Camada de Apresentação, chegamos a um valor de 16h por cada entidade gerida, valor esse que é gerado automaticamente pelo gerador de código. É certo também que para o gerador de código possa ser utilizado na sua plenitude é recomendada uma análise mais pormenorizada dos requisitos iniciais, por forma a existir informação que permita uma elaboração de entidades ERM, que permita a geração sobre todo o contexto de projeto, o que é um step-back ao desenvolvimento Agile, que visa um incremento de funcionalidades ao longo do tempo, e não um processo waterfall que obrigue ao levantamento inicial exaustivo. Para fazer face a este risco, é necessário que o gerador tenha um comportamento não destrutivo, permitindo a geração à medida da evolução dos sprints.
Existe a necessidade de várias gerações durante o desenvolvimento de um projeto?
Assumindo um cenário de waterfall, onde é feito um levantamento exaustivo das funcionalidades do projeto no início do mesmo, a probabilidade de existir a necessidade de uma nova geração é mais reduzida, mas todos os projetos sofrem alterações (Change Requests) durante a execução do mesmo, altura em que é fundamental que o gerador de código proporcione mecanismos de geração incremental, sem destruir o trabalho feito anteriormente. Já no contexto de uma metodologia de gestão de projeto ágil, é obrigatório que o gerador tenha a funcionalidade de geração não destrutiva, permitindo assim uma utilização contínua no contexto de execução do projeto à medida que o backlog vai sendo definido para cada sprint, visto que as funcionalidades vão sendo adicionadas e detalhadas durante a execução do projeto.
Quais são as principais vantagens na utilização da versão do gerador de código não destrutivo?
Conforme já descrito anteriormente, a geração não destrutiva é fundamental para garantir a sua utilização durante a execução do projeto, evitando assim esforço extra para adicionar novas funcionalidades/entidades. Ou seja, não existindo essa possibilidade, o consultor vê-se obrigado a programar todas as camadas de desenvolvimento para uma nova entidade que não foi prevista no âmbito inicial de geração. Por este motivo, é primordial que o consultor seja capaz de utilizar o gerador de código para funcionalidades/entidades incrementais durante a execução do projeto, e na manutenção de um projeto no âmbito de uma manutenção evolutiva de produto.

Capítulo 8

Conclusão

Nesta capítulo apresentam-se as conclusões deste trabalho e definem-se possíveis direções para trabalhos futuros.

8.1 Conclusão

No presente e no futuro, as tecnologias da informação representam uma das áreas de negócio com maior rápido desenvolvimento tecnológico, acompanhado por um aumento da complexidade das aplicações. Várias empresas de software conseguem responder aos requisitos do cliente com a mesma qualidade. O tempo e o preço (que depende do tempo) tornaram-se fatores cruciais para a obtenção de projetos. Levanta-se a questão de como reduzir estes custos, melhorando a produtividade e superar a concorrência [51]. Os geradores de aplicações estão a ser aplicados como uma solução para o problema. Estes representam software capaz de produzir automaticamente outro software e são principalmente utilizados para diminuir o tempo de desenvolvimento assim como aumentar a qualidade do código [29]. O gerador de código, baseado num conjunto de configurações e em modelos base, assume a missão de escrever de forma repetitiva o código, deixando aos programadores mais tempo para se concentrarem em código específico. Os geradores fornecem mais produtividade e geram grandes volumes de código, o que levaria muito mais tempo se este fosse escrito manualmente. A qualidade consistente do código é preservada ao longo de toda a parte gerada de um projeto. As convenções de código são aplicadas de forma consistente, ao contrário de código manuscrito, onde a qualidade está sujeita a variações. Em caso de serem encontrados erros no código gerado, os erros podem ser corrigidos em pouco tempo através da revisão de modelos e da repetição do processo de geração [38].

No mercado de software existem muitos geradores que contêm imensos parâmetros e personalizações para responder aos requisitos específicos das aplicações que pretendemos. No entanto, como verificado, quando necessitam de gerar novos métodos e é identificado que esse método já existe, limitam-se apenas a perguntar ao programador se quer realmente gerar novo método, apagando tudo o que já tinha sido alterado, como é o caso do JHipster e da framework YII.

Se for necessário que exista uma evolução do código gerado então deve ser considerada a conceção de um gerador próprio que implemente uma das abordagens descritas neste documento. Para casos simples, onde o código gerado deve permanecer igual sendo apenas necessário acrescentar novas instruções, antes ou depois do bloco gerado é recomendada a abordagem de "Marcação de início e fim de código" onde são criados marcadores que indicam o início e o fim do código gerado e apenas o bloco de código entre estes marcadores é substituído após nova geração. Outra solução pode ser o uso de uma abordagem baseada num algoritmo de comparação de instruções como é o caso da abordagem "Diferencial de instruções", que propõe um algoritmo que faz uma comparação textual entre 3 métodos (1º geração, 2ª geração e alterado) removendo ou adicionando as instruções necessárias de forma a refletir tanto as alterações do utilizador, como do gerador. Para casos mais complexos deve ser usada a abordagem "Diff And Patch" que, com a ajuda de bibliotecas que implementam o algoritmo de Diff-Match-Patch (usado por exemplo pelo Git para o merge de commits), procura comparar duas sequências de caracteres e descobrir como é que a primeira pode ser transformada na segunda.

Os testes demonstrados no capítulo 6 com 46 cenários diferentes mostram que, utilizando a abordagem "Diff And Patch", será possível que tanto o programador, como novas versões do gerador alterem ou gerem novo código sem destruir o existente com uma taxa de sucesso de 86% que tem de ser complementada com uma correção manual do código. Sendo que os restantes 14% encontram-se sinalizados pelo relatório final que é criado pelo gerador de forma a auxiliar os programadores na correção. Esta abordagem foi aplicada num gerador de código programado na empresa agap2IT para o desenvolvimento de aplicações web, tendo revelado melhorias relativamente ao modelo de geração destrutivo. Acabou com o trabalho manual e maçador da junção de código que pode induzir à criação de pequenos erros e teve como resultado directo a redução do tempo no desenvolvimento dos projetos. Apesar de experimentalmente se obter uma taxa de 86%, num projeto real essa taxa acabou por ser de 100%. Múltiplas gerações não levantarem nenhum problema e desse modo não dependerem de qualquer intervenção dos programadores.

De qualquer forma como nenhuma das abordagens resulta numa taxa de 100% de sucesso, propõe-se que após a geração seja elaborado um relatório que indique os métodos que não foram unidos com sucesso de forma a serem manualmente corrigidos pelos programadores.

8.2 Trabalho futuro

Como já referido, nenhuma das abordagens apresentadas consegue atingir uma taxa de 100% na junção automática de código. De forma a aumentar a qualidade das abordagens, estas devem ser revistas ou deve ser estudada uma nova abordagem de forma a que não seja necessária a intervenção de programadores e o resultado final das junções seja sempre o expectável.

A abordagem "Diferencial de instruções" tem como base a comparação de instruções e um dos problemas mencionados é a dificuldade de por vezes aferir qual é a instrução correspondente entre 2 métodos. Um algoritmo mais complexo, talvez usando técnicas de Deep learning [95] pode ser elaborado para solucionar este problema. Também em relação à abordagem "Diferencial de instruções" a implementação de AST's para a fase detalhada de comparação irá permitir comparar diferenças dentro da mesma instrução. Estes dois pontos podem ser o caminho para tornar a abordagem de "Diferencial de instruções" 100% eficaz.

Na abordagem "Diff and Patch" foi observado que os principais erros de junção de código são provenientes do algoritmo Diff, que compara duas sequências de caracteres. Por essa razão um estudo sobre como melhorar o algoritmo de Diff é fundamental para a melhoria desta abordagem. Uma possível solução é - também recorrendo à implementação de AST's - o reconhecimento e comparação de instruções em vez de sequências de caracteres.

A maioria dos sistemas de junção utilizam algoritmos baseados em texto. Estas ferramentas consideram apenas as modificações feitas ao texto e não a linguagem de programação em que o código está escrito. Não sendo capaz de agir com base nas estrutura da linguagem de programação significa que as ferramentas de junção dependem fortemente da posição dos textos que estão a ser modificados. Como é demonstrado na tabela de testes da abordagem "Diff and Patch" (Tabela 6.8), depender da posição dos textos pode criar vários problemas. Com vista a melhorar o sistema de junção existem no mercado ferramentas de "consciência de linguagem" como é o caso do SemanticMerge tool [77] que usa a análise de código para implementar os algoritmos de Diff e Merge [79], permitindo uma junção mais precisa. A técnica de "consciência de linguagem" é uma possível solução para resolver os erros reportados na (Tabela 6.8).

Desde os anos 70 existem inúmeras ferramentas para medir a similaridade do código [71], estas ferramentas tem como base a comparação de código. Aferir quais as que tem melhores resultados e que algoritmos implementam, pode ser um caminho a seguir de forma a ter uma melhor fase de comparação em relação às fases de comparação nas abordagens apresentadas.

Bibliografia

[1] P. A. Abdalla e A. Varol. "Advantages to Disadvantages of Cloud Computing for Small-Sized Business". Em: (2019). DOI: 10.1109/ISDFS.2019.8757549 (ver p. 2).

[2] A. Akbulut e S. Toprak. "Code Generator Framework for Smart TV Platforms". Em: (2018). DOI: 10.1049/iet-sen.2018.5157 (ver p. 3).

[3] M. Allamanis et al. "A Survey of Machine Learning for Big Code and Naturalness". Em: *ACM Computing Surveys* 51 (jul. de 2019), pp. 1–37. DOI: 10.1145/3212695 (ver p. 27).

[4] S. W. Ambler. *The object primer : agile model-driven development with UML 2.0*. Cambridge Univ. Press, 2008 (ver p. 98).

[5] J. Arnoldus et al. "Code Generation with Templates". Em: (2012). DOI: 10.2991/978-94-91216-56-5 (ver p. 3).

[6] Y. Asri. "Model Base generation of a 3 tier application". Tese de doutoramento. Universiteit van Amsterdam UvA, 2004 (ver p. 16).

[7] M. Attaran e J. Woods. "Cloud computing technology: improving small business performance using the Internet". Em: *Journal of Small Business Entrepreneurship* (jun. de 2018). DOI: 10.1080/08276331.2018.1466850 (ver p. 1).

[8] R. Balzer. "A 15 Year Perspective on Automatic Programming". Em: (1985). DOI: 10.1109/TSE.1985.231877 (ver p. 7).

[9] B. Basren. "Progressive Web Apps (PWA) for YII Framework Enrichment". Em: *Journal of Telematics and Informatics (JTI)* 6.3 (2018), pp. 193–200 (ver p. 30).

[10] J. Bioco e Á. Rocha. "Web Application for Management of Scientific Conferences". Em: (2019). DOI: 10.1007/978-3-030-16181-1_72 (ver p. 34).

[11] Bootstrap. *Introduction Bootstrap*. URL: https://getbootstrap.com/docs/4.1/getting-started/introduction/. (consultado: 27.07.2021) (ver p. 98).

[12] Brackets. *Getting started | Craftable*. URL: https://getcraftable.com/docs/7.0/overview. (consultado: 27.07.2021) (ver p. 44).

[13] J. Bucanek. *Learn Objective-C for Java Developers*. Apress, 2009. ISBN: 9781430223696 (ver p. 11).

[14] E. Corporation. *CodeTrigger - Code Generation Tool For C*. URL: https://www.codetrigger.com/Default.aspx?vwsess=1081173. (consultado: 27.07.2021) (ver p. 38).

[15] R. S. Couto e I. J. Foschini. "Análise comparativa entre dois Frameworks MVC para a Plataforma Java EE: JSF e VRaptor". Em: *Revista TIS* 4.3 (2016) (ver p. 41).

[16] B. Cui et al. "Code Comparison System based on Abstract Syntax Tree". Em: (2010). DOI: 10.1109/icbnmt.2010.5705174 (ver p. 53).

[17] J. E. S. Davide Libenzi. *LibXDiff by Davide Libenzi (File Differential Library)*. URL: https://github.com/git/git/blob/53f9a3e157dbbc901a02ac2c73346d375e24978c/xdiff/xmerge.c. (consultado: 27.07.2021) (ver p. 58).

[18] A. Derksen. *Free Online Jakarta EE Application Generator*. URL: https://www.generjee.com/. (consultado: 27.07.2021) (ver p. 41).

[19] A. Derksen. *Generjee - Striking New Paths with Java Source Code Generation*. URL: https://www.methodsandtools.com/tools/generjee.php. (consultado: 27.07.2021) (ver p. 41).

[20] *Diff Algorithm*. URL: https://wiki.c2.com/?DiffAlgorithm. (consultado: 27.07.2021) (ver p. 58).

[21] P. S. DISID. *Introduction to Spring Framework*. URL: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/overview.html. (consultado: 27.07.2021) (ver p. 40).

[22] P. S. DISID. *Spring Roo - Reference Documentation*. URL: https://docs.spring.io/spring-roo/docs/2.0.x/reference/html/. (consultado: 27.07.2021) (ver p. 40).

[23] J. F. Dooley. *Software development, design and coding : with patterns, debugging, unit testing, and refactoring*. Apress, New York, Ny, 2017, pp. 253–269 (ver p. 13).

[24] I. C. Education. *What is Three-Tier Architecture*. www.ibm.com, out. de 2020. URL: https://www.ibm.com/cloud/learn/three-tier-architecture (acedido em 04/09/2022) (ver pp. 8, 9).

[25] H. Eichhorn et al. "A comparative study of programming languages for next-generation astrodynamics systems". Em: (2017). DOI: 10.1007/s12567-017-0170-8 (ver p. 31).

[26] T. Elec et al. "A template-based code generator for web applications". Em: *Turkish Journal of Electrical Engineering and Computer Sciences* 28 (mai. de 2020), pp. 1747–1762. DOI: 10.3906/elk-1910-44 (ver p. 2).

[27] J. Fehrman. *Benefits of Code Generators*. URL: https://www.udig.com/digging-in/benefits-of-code-generators/. (consultado: 27.07.2021) (ver p. 3).

[28] E. B. Fernandez et al. "The Secure Three-Tier Architecture Pattern". Em: (2008). DOI: 10.1109/CISIS.2008.51 (ver p. 8).

[29] K. Fertal e M. Brcic. "A Source Code Generator Based on UML Specification". Em: (2008) (ver p. 107).

[30] K. Fertalj e M. Brcic. "A Source Code Generator Based on UML Specification". Em: (2008) (ver p. 3).

[31] D. S. Foundation. *Django documentation*. URL: https://docs.djangoproject.com/en/3.2/. (consultado: 27.07.2021) (ver pp. 32, 33).

[32] Google. *Architecture Overview*. URL: https://v2.angular.io/docs/ts/latest/guide/architecture.html. (consultado: 27.07.2021) (ver p. 37).

[33] Google. *Diff Match and Patch library*. URL: https://github.com/google/diff-match-patch. (consultado: 27.07.2021) (ver p. 59).

[34] J. Griffin. *Introduction to Laravel*. Out. de 2020, p. 99. DOI: 10.1007/978-1-4842-6023-4_4 (ver p. 44).

[35] S. Gulwani. "Programming by Examples: Applications, Algorithms, and Ambiguity Resolution". Em: *Automated Reasoning* (2016), pp. 9–14. DOI: 10.1007/978-3-319-40229-1_2 (ver p. 27).

[36] Haulmont. *Introduction || Jmix Documentation*. URL: https://docs.jmix.io/jmix/intro.html. (consultado: 27.07.2021) (ver p. 43).

[37] N. Heidke, J. Morrison e M. Morrison. "Assessing the effectiveness of the model view controller architecture for creating web applications". Em: *Midwest instruction and computing symposium, Rapid City, SD*. 2008 (ver p. 12).

[38] J. Herrington. *Code generation in action*. Manning, 2003 (ver p. 107).

[39] J. Hinkle. *Phreeze Documentation*. URL: http://phreeze.com/phreeze/documentation/. (consultado: 27.07.2021) (ver p. 45).

[40] K. G. Inc. *phpGrid Documentation API Guide*. URL: https://phpgrid.com/documentation/. (consultado: 27.07.2021) (ver p. 43).

[41] InfyOmLabs. *Introduction | InfyOm Laravel Generator | InfyOmLabs*. URL: https://infyom.com/open-source/laravelgenerator/docs/8.0/introduction. (consultado: 27.07.2021) (ver p. 44).

[42] Jetbrains. *Code completion*. URL: https://www.jetbrains.com/help/go/auto-completing-code.html. (consultado: 27.07.2021) (ver p. 26).

[43] *JHipster - Full Stack Platform for the Modern Developer!* URL: https://www.jhipster.tech/. (consultado: 27.07.2021) (ver p. 35).

[44] S. Jörges. *Construction and Evolution of Code Generators A Model-Driven and Service-Oriented Approach*. Heidelberg Springer, 2013 (ver p. 2).

[45] A. Joy. *Difference Between MVC and MVT Patterns*. URL: https://pythonistaplanet.com/difference-between-mvc-and-mvt/. (consultado: 27.07.2021) (ver p. 34).

[46] jquery. *jQuery API*. URL: https://api.jquery.com/. (consultado: 27.07.2021) (ver p. 98).

[47] R. Kennard. *Documentation - Metawidget*. URL: http://metawidget.sourceforge.net/documentation.php. (consultado: 27.07.2021) (ver p. 42).

[48] P. King. "A History of the Groovy Programming Language". Em: (2020). DOI: 10.1145/3386326 (ver p. 42).

[49] S. Klein. *Pro Entity Framework 4.0 (Expert's Voice in .NET)*. Apress, 2010. ISBN: 9781590599907 (ver p. 16).

[50] H. Landvik. "Web application with Yii Framework". Em: (2013) (ver p. 30).

[51] S. Lazetic et al. "A Generator of MVC-based Web Applications". Em: *World of Computer Science and Information Technology Journal (WCSIT)* 2 (2012), pp. 147–156 (ver pp. 2, 107).

[52] S. Liu et al. "ATOM: Commit Message Generation Based on Abstract Syntax Tree and Hybrid Ranking". Em: (2020). DOI: 10.1109/TSE.2020.3038681 (ver p. 52).

[53] Y. S. LLC. *Class yii*
db
ActiveRecord. URL: https://www.yiiframework.com/doc/api/2.0/yii-db-activerecord. (consultado: 27.07.2021) (ver p. 48).

[54] Y. S. LLC. *Generating an Active Record Class*. URL: https://www.yiiframework.com/doc/guide/2.0/en/start-gii. (consultado: 27.07.2021) (ver p. 49).

[55] V. Ltd. *Vaadin 14 Docs*. URL: https://vaadin.com/docs/v14/. (consultado: 27.07.2021) (ver pp. 40, 41).

[56] *Market position*. URL: https://w3techs.com/technologies/details/pl-python. (consultado: 27.07.2021) (ver p. 33).

[57] T. Mens e M. Wermelinger. "Separation of concerns for software evolution". Em: (2002). DOI: 10.1002/smr.257 (ver p. 11).

[58] Microsoft. *ASP.NET MVC Pattern*. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/mvc. (consultado: 27.07.2021) (ver p. 96).

[59] Microsoft. *Entity Framework documentation*. URL: https://docs.microsoft.com/en-us/ef/. (consultado: 27.07.2021) (ver p. 97).

[60] Microsoft. *IntelliSense*. URL: https://code.visualstudio.com/docs/editor/intellisense. (consultado: 27.07.2021) (ver p. 26).

[61] Microsoft. *Princípios de arquitetura*. URL: https://docs.microsoft.com/pt-br/dotnet/architecture/modern-web-apps-azure/architectural-principles. (consultado: 27.07.2021) (ver p. 11).

[62] Microsoft. *Razor syntax reference for ASP.NET Core*. URL: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-6.0. (consultado: 27.07.2021) (ver p. 97).

[63] T. Nagarajah e G. Poravi. "Automatic Code Generation for Framework Evolutions: Missing Links". Em: (2018). DOI: 10.13140/RG.2.2.33485.69602 (ver p. 25).

[64] E. O'Neil. "Object/relational mapping 2008: hibernate and the entity data model (edm)". Em: (2008). DOI: 10.1145/1376616.1376773 (ver p. 11).

[65] T. Otwell. *Laravel - The PHP Framework For Web Artisans*. URL: https://laravel.com/. (consultado: 27.07.2021) (ver p. 44).

[66] J. L. Overbey e R. E. Johnson. *Generating Rewritable Abstract Syntax Trees*. Ed. por D. Gašević, R. Lämmel e E. Van Wyk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 114–133. ISBN: 978-3-642-00434-6 (ver p. 52).

[67] G. Paolone et al. "Automatic Code Generation of MVC Web Applications". Em: (2020). DOI: 10.3390/computers9030056 (ver p. 11).

[68] PHPCG. *About CRUD Operations - PHP CRUD Generator Documentation*. URL: https://www.phpcrudgenerator.com/documentation/index. (consultado: 27.07.2021) (ver pp. 43, 44).

[69] I. Pivotal Software. *Interface JpaRepository<T,ID>*. URL: https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html. (consultado: 27.07.2021) (ver p. 48).

[70] M. A. Possatto e D. Lucrédio. "Automatically propagating changes from reference implementations to code generation templates". Em: (2015). DOI: 10.1016/j.infsof.2015.06.009 (ver p. 3).

[71] C. Ragkhitwetsagul, J. Krinke e D. Clark. "A Comparison of Code Similarity Analysers". Em: *Empirical Softw. Engg.* 23.4 (ago. de 2018), pp. 2464–2519. ISSN: 1382-3256. DOI: 10.1007/s10664-017-9564-7. URL: https://doi.org/10.1007/s10664-017-9564-7 (ver p. 109).

[72] M. Raible. *The JHipster Mini-Book 5.0*. C4Media, 2019. ISBN: 978-1-329-63814-3 (ver p. 35).

[73] G. Rocher. *Documentation | Grails framework*. URL: https://grails.org/documentation.html. (consultado: 27.07.2021) (ver p. 42).

[74] S. Rose. *Top 10 Laravel Development Companies in 2021*. URL: https://medium.com/@scarlett8285/top-10-laravel-development-companies-in-2021-updated-346291c0473e. (consultado: 27.07.2021) (ver p. 44).

[75] E. Saks. "JavaScript Frameworks: Angular vs React vs Vue." Em: (2019) (ver p. 36).

[76] M. Scott. *The diff-match-patch Reference Manual*. URL: https://quickref.common-lisp.net/diff-match-patch.html. (consultado: 27.07.2021) (ver p. 58).

[77] *SemanticMerge - Intro guide*. www.plasticscm.com. URL: https://www.plasticscm.com/semanticmerge/documentation/intro-guide/semanticmerge-intro-guide. (consultado: 09.10.2022) (ver p. 109).

[78] S. Shim et al. "DeeperCoder: Code Generation Using Machine Learning". Em: *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)* (jan. de 2020). DOI: 10.1109/ccwc47524.2020.9031149 (ver p. 27).

[79] L. D. Silva et al. *Detecting Semantic Conflicts via Automated Behavior Change Detection*. 2020, pp. 174–184. DOI: 10.1109/ICSME46990.2020.00026 (ver p. 109).

[80] F. Solms. "What is Software Architecture". Em: *ACM International Conference Proceeding Series* (out. de 2012). DOI: 10.1145/2389836.2389879 (ver p. 8).

[81] P. Stanley. *Advantages of Web Applications*. URL: https://www.pssuk.com/AdvantagesWebApplicatio aspx. (consultado: 27.07.2021) (ver p. 2).

[82] StephenWalther. *Creating Unit Tests for ASP.NET MVC Applications (C)*. docs.microsoft.com. URL: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/unit-testing/creating-unit-tests-for-asp-net-mvc-applications-cs (acedido em 04/09/2022) (ver p. 13).

[83] H. tariq. *Update Jhipster Entity without overriding existing files and custom code*. URL: https://github.com/jhipster/generator-jhipster/issues/10787. (consultado: 27.07.2021) (ver p. 49).

[84] Telosys. *Telosys documentation*. URL: https://doc.telosys.org/. (consultado: 27.07.2021) (ver p. 46).

[85] Typescript. *TypeScript is JavaScript with syntax for types*. URL: https://www.typescriptlang.org/. (consultado: 27.07.2021) (ver p. 98).

[86] A. Valerio, M. Barone e R. Sennrich. *A parallel corpus of Python functions and documentation strings for automated code documentation and code generation*. 2017 (ver p. 27).

[87] A. Veeramani, K. Venkatesan e N. Kadiresan. "Abstract Syntax Tree based Unified Modeling Language to Object Oriented Code Conversion". Em: (out. de 2014), pp. 1–8. DOI: 10.1145/2660859.2660934 (ver p. 52).

[88] D. P. Voorhees. *Guide to Efficient Software Design - An MVC Approach to Concepts, Structures, and Models*. Springer International Publishing, jan. de 2020 (ver p. 12).

[89] E. Wohlgethan. "Supportingweb development decisions by comparing three major javascript frameworks: Angular, react and vue. js". Tese de doutoramento. Hochschule für Angewandte Wissenschaften Hamburg, 2018 (ver p. 29).

[90] Xlinesoft. *ASPRunnerPro 10.6 Manual*. URL: https://xlinesoft.com/asprunnerpro/docs/asprunnerpro.pdf. (consultado: 27.07.2021) (ver p. 39).

[91] Q. Xue. *The Definitive Guide to Yii 2.0*. URL: https://www.yiiframework.com/doc/guide/2.0/en. (consultado: 27.07.2021) (ver pp. 29, 30).

[92] *Yii - Reviews, Pros Cons | Companies using Yii*. URL: https://stackshare.io/yii. (consultado: 27.07.2021) (ver p. 32).

[93] P. Yin e G. Neubig. *A Syntactic Neural Model for General-Purpose Code Generation*. Jan. de 2017. DOI: 10.18653/v1/P17-1041 (ver p. 27).

[94] P. Yurchuk. *Companies Using Grails*. URL: https://philip.yurchuk.com/software/companies-using-grails/. (consultado: 27.07.2021) (ver p. 43).

[95] G. Zhao e J. Huang. *DeepSim: Deep Learning Code Functional Similarity*. ESEC/FSE 2018. Lake Buena Vista, FL, USA: Association for Computing Machinery, 2018, pp. 141–151. ISBN: 9781450355735. DOI: 10.1145/3236024.3236068 (ver p. 109).

APÊNDICE A

CÓDIGO FONTE

A.1 Classe Main

```
static void Main(string[] args)
{
   // Ler configurações
   ConfigModel config =  Configuration.Read("config.xml");
   // Copiar projecto base
   DirectoryOps.Copy(Config.Project.Namespace,"Assets/BaseProject");
   // Obter informação dos modelos
   Dictionary<string, ModelInfo> modelsInfo = Model.GetAll(config.Project.
    Namespace);
   // Carregar para memória o Histórico de gerações
   History.Load();
   // Percorrer todos os modelos
   foreach (var modelInfo in modelsInfos)
   {
     // Criar o repositório para cada modelo correspondente
     RepositoryGenerator.Create(config.Project.Namespace, config.ApproachType,
     modelsInfos, modelInfo.Name);
     // Criar o gestor da camada de negócio para cada modelo correspondente
     ManagerGenerator.Create(config.Project.Namespace, config.ApproachType,
    modelsInfos, modelInfo.Name);
     // Criar o controlador para cada modelo correspondente
     ControllerGenerator.Create(config.Project.Namespace, config.ApproachType,
     modelsInfos, modelInfo.Name);
     // Criar os modelos de visualização para cada modelo correspondente
     ViewModelGenerator.Create(config.Project.Namespace, config.ApproachType,
    modelsInfos, modelInfo.Name);
     // Criar o controlador de API para cada modelo correspondente
     ApiControllerGenerator.Create(config.Project.Namespace, config.
    ApproachType, modelsInfos, modelInfo.Name);
```

```
24      // Criar os modelos de API para cada modelo correspondente
25      ApiViewModelGenerator.Create(config.Project.Namespace, config.
    ApproachType, modelsInfos, modelInfo.Name);
26      // Criar as vistas para cada modelo correspondente
27      ViewGenerator.Create(config.Project.Namespace, config.ApproachType,
    modelsInfos, modelInfo.Name);
28      // Criar os typescripts para cada modelo correspondente
29      TypeScriptGenerator.Create(config.Project.Namespace, config.ApproachType,
     modelsInfos, modelInfo.Name);
30    }
31    // Criar os resources
32    ResourceGenerator.Create(Config.Project.Namespace, config.Languages);
33    // Criar os enums
34    EnumGenerator.Create(Config.Project.Namespace);
35    // Criar o ficheiro de configurações
36    SetingsGenerator.Create(Config.Project.Namespace);
37    // Criar e configurar o contexto da base de dados.
38    DbContextGenerator.Create(Config.Project.Namespace);
39     // Criar e configurar o contexto da base de dados.
40    MenuGenerator.Create(Config.Project.Namespace);
41    // Instalar modulos
42    Module.Install(Config.Project.Namespace, Config.Modules);
43    // Instalar bibliotecta de base de dados
44    Database.Install(Config.Database.Type);
45    // Mostrar o relatório de sucesso na geração de métodos
46    Report.Print();
47    // Fazer build do projecto
48    Project.Build();
49    // Criar a base de dados
50    Database.Create();
51  }
```

Classe Principal

A.2 ConfigModel

```
1  public class ConfigModel
2  {
3      // Configurações referentes à base de dados
4      public ConfigDatabase Database { get; set; }
5      // Configurações referentes ao projecto
6      public ConfigProject Project { get; set; }
7      // Listagem de línguas disponíveis na aplicação (pt, en, fr ...)
8      public List<string> Languages { get; set; }
9      // Listagem de módulos a instalar na aplicação.
10     public List<string> Modules { get; set; }
11     // Tipo de abordagem a usar (Delimiter , Diferencial, DiffPatch)
12     public ApproachTypeEnum ApproachType { get; set; }
```

```
}
public class ConfigDatabase
{
    // Tipo de base dados (SqlServer, Postgres, MySql, Oracle)
    public DatabaseTypeEnum Type { get; set; }
    // String de conexção à base de dados
    public string ConnectionString { get; set; }
}
public class ConfigProject
{
    // Namespace do projecto
    public string Namespace { get; set; }
}
```

classe ConfigModel

A.3 DirectoryOps

```
public static class DirectoryOps
{
    // Criar a directoria de destino e começa a copia do projecto base
    // Copia os modelos da pasta assets para a pasta de modelos da camada de acesso à
    // base de dados
    public static void Copy(string projectNamespace, string originalPath)
    {
        originalPath = Path.GetFullPath(originalPath);
        destinationPath = Path.GetFullPath("GeneratedProject");
        Directory.CreateDirectory(destinationPath);
        CopyFiles(projectNamespace, originalPath, originalPath, destinationPath);
        DirCopy(projectNamespace, originalPath, originalPath, destinationPath);
        CopyFiles(projectNamespace, "Assets/Models", "Assets/Models", Path.Combine(destinationPath, projectNamespace+".Data/Models"));
    }
    // Copiar recursivamente as directorias
    public static void DirCopy(string projectNamespace, string currentDirectory, string originalDirectory, string destinationDirectory)
    {
        foreach (string d in Directory.GetDirectories(currentDirectory))
        {
            string originDirname = Path.GetFileName(d);
            string originPath = Directory.GetParent(d).FullName;
            string destinationDirname = originDirname.Replace("Generated.Base", projectNamespace);
            string destinationPath = originPath.Replace(originalDirectory, destinationDirectory);
```

```
24              destinationPath = destinationPath.Replace("Generated.Base",
    projectNamespace);
25              Directory.CreateDirectory(Path.Combine(destinationPath,
    destinationDirname));
26              CopyFiles(projectNamespace, d, originalDirectory,
    destinationDirectory);
27              DirCopy(projectNamespace, d, originalDirectory,
    destinationDirectory);
28          }
29      }
30      // Copiar os ficheiros
31      public static void CopyFiles(string projectNamespace, string
    currentDirectory, string originalDirectory, string destinationDirectory)
32      {
33          foreach (string f in Directory.GetFiles(currentDirectory))
34          {
35              string fileContent = File.ReadAllText(f);
36              string originfilename = Path.GetFileName(f);
37              string originPath = Directory.GetParent(f).FullName;
38              string destinationFileName = originfilename.Replace("Generated.
    Base", projectNamespace);
39              string destinationPath = originPath.Replace(originalDirectory,
    destinationDirectory);
40              destinationPath = destinationPath.Replace("Generated.Base",
    projectNamespace);
41              Directory.CreateDirectory(destinationPath);
42              fileContent = fileContent.Replace("Generated.Base",
    projectNamespace);
43              File.WriteAllText(Path.Combine(destinationPath,
    destinationFileName), fileContent);
44          }
45      }
46  }
```

classe DirectoryOps

A.4 Model

```
1  public static class Model
2  {
3      public Dictionary<string, ModelInfo> GetAll(string projectNamespace){
4          Dictionary<string, ModelInfo> modelInfos = new Dictionary<string,
    ModelInfo>();
5          foreach (string f in Directory.GetFiles("GeneratedProject/" +
    projectNamespace + ".Data/Models"))
6          {
7              string fileContent = File.ReadAllText(filePath);
8              string name = Path.GetFileNameWithoutExtension(filePath);
```

```
9             SyntaxTree syntaxTree = SyntaxFactory.ParseSyntaxTree(fileContent)
    ;
10            IEnumerable<SyntaxNode> descendantNodes = syntaxTree.GetRoot().
    DescendantNodes();
11            List<ClassProperty> properties = new List<ClassProperty>();
12            IEnumerable<PropertyDeclarationSyntax>
    propertyDeclarationSyntaxList = descendantNodes.OfType<
    PropertyDeclarationSyntax>().ToList();
13
14            foreach (var propertyDeclarationSyntax in
    propertyDeclarationSyntaxList)
15            {
16                ClassProperty property = new ClassProperty
17                {
18                    Type = propertyDeclarationSyntax.Type.ToString(),
19                    Name = propertyDeclarationSyntax.Identifier.ValueText,
20                    Attributes = new List<PropertyAttribute>(),
21                    Key = propertyDeclarationSyntax.AttributeLists.ToString().
    Contains("Key")
22                };
23                foreach (var attributeListSyntax in propertyDeclarationSyntax.
    AttributeLists.ToList())
24                {
25                    var attributeListSyntaxItem = attributeListSyntax.
    Attributes[0];
26                    var propertyAttribute = new PropertyAttribute
27                    {
28                        Name = attributeListSyntaxItem.Name.ToString(),
29                        Parameters = attributeListSyntaxItem.ArgumentList?.
    Arguments.ToString()
30                    };
31                    property.Attributes.Add(propertyAttribute);
32                }
33                properties.Add(property);
34            }
35            ModelInfo modelInfo = new ModelInfo() { Properties = properties,
    Name = name };
36            modelInfos[name] = modelInfo;
37        }
38    }
39    return modelInfos;
40}
```

classe Model

A.5 ModelInfo

```
1 public class ModelInfo
```

```
{
    // Listagem das propriedades do modelo
    public List<ClassProperty> Properties { get; set; }
    // Nome do modelo
    public string Name { get; set; }
    // Primeira propriedade do modelo do tipo string.
    public ClassProperty FirstPropertyTypeString
    {
        get
        {
            return Properties.Where(p => p.Type.ToUpper() == "STRING").
    FirstOrDefault();
        }
    }
}
public class ClassProperty
{
    // Se a propriedade é chave primaria
    public bool Key { get; set; }
    // Nome da propriedade
    public string Name { get; set; }
    // Tipo da propriedade
    public string Type { get; set; }
    // Atributos das propriedade
    public List<PropertyAttribute> Attributes { get; set; }
    // Se a propriedade é uma relação para outro modelo
    public bool IsRelation(List<ClassProperty> properties)
    {
        return properties.Where(p => p.Name == Name + "Id").Any();
    }
    // Se tiver o atributo de "Display", nome que deve ser apresentado na
    // camada de visualização.
    public string DisplayName
    {
        get
        {
            return new Regex(@"(?<=LabelResource\.).*(?=\))").Match(Attributes
    .Where(p => p.Name == "Display").FirstOrDefault().Parameters).Value;
        }
    }
    // Do tipo password?
    public bool IsPassword
    {
        get
        {
            return Attributes.Any(p => p.Parameters != null && p.Parameters.
    Contains("DataType.Password"));
        }
    }
}
```

```
49 public class PropertyAttribute
50 {
51     // Nome do atributo
52     public string Name { get; set; }
53     // Par metros do atributo
54     public string Parameters { get; set; }
55 }
```

classe ModelInfo

A.6 ControllerGenerator

```
1  public static class ControllerGenerator
2  {
3      public static void Create(string projectNamespace, ApproachTypeEnum
       approachType, string destinationPath, Dictionary<string, ModelInfo>
       modelsInfos, string name){
4          // Nome da classe a ser gerada
5          string className = name.ToPlural() + "Controller.cs";
6          // Obter a localização da classe
7          string classFilename = Path.Combine("GeneratedProject",
       projectNamespace+".Web","Controllers", className);
8          // Obter informação da classe
9          RoselynClass roselynClass = new RoselynClass(classFilename, className,
        approachType);
10         StringBuilder result = new StringBuilder();
11         // Criar as bibliotecas necessárias ao funcionamento da classe
12         result.Append($"using {projNamespace}.Business;\r\n");
13         result.Append($"using {projNamespace}.Business.Models;\r\n");
14         result.Append($"using Microsoft.AspNetCore.Mvc;\r\n");
15         // Criar o namespace
16         result.Append($"namespace {projNamespace}.Web.Controllers\r\n");
17         result.Append($"{{\r\n");
18         // Criar atributo de autorização
19         result.Append($"    [Authorize]\r\n");
20         // Criar classe
21         result.Append($"    public class {name.ToPlural()}Controller :
       BaseController\r\n");
22         result.Append($"    {{\r\n");
23         // Criar variável do gestor correspondente
24         result.Append($"        private readonly I{name}Manager {name.
       ToCamelCase()}Manager;\r\n");
25         result.Append($"\r\n");
26         // Criar método construtor com a injecção de dependência do gestor e
       das configurações.
27         result.Append($"        public {name.ToPlural()}Controller(\r\n");
28         result.Append($"            I{name}Manager {name.ToCamelCase()}Manager
       ,\r\n");
```

```
29          result.Append($"            IOptionsSnapshot<AppSettings> settings)\r\
    n");
30          result.Append($"        {{\r\n");
31          result.Append($"            this.{name.ToCamelCase()}Manager = {name.
    ToCamelCase()}Manager;\r\n");
32          result.Append($"        }}\r\n");
33          // Criar métodos de CRUD
34          result.Append(Index(roselynClass));
35          result.Append(Details(roselynClass, name));
36          result.Append(Create(roselynClass,modelsInfos, name));
37          result.Append(CreatePost(roselynClass,modelsInfos, name));
38          result.Append(Edit(roselynClass,modelsInfos, name));
39          result.Append(EditPost(roselynClass,modelsInfos, name));
40          result.Append(GetAll(roselynClass,modelsInfos, name));
41          result.Append(Delete(roselynClass,modelsInfos, name));
42          result.Append($"    }}\r\n");
43          result.Append($"}}\r\n");
44          //Escrever o ficheiro da classe
45          File.WriteAllText(classFilename, result.ToString());
46      }
47
48      // Criar método Index
49      private static string Index(RoselynClass roselynClass)
50      {
51          // Criar método
52          Method method = new Method("Index");
53          // Adicionar instruções
54          method.AddStatement($"public IActionResult Index();");
55          method.AddStatement($"{{");
56          if (approachType==AproachType.Delimiter) method.AddStartDelimiter();
57          method.AddStatement($"  return View();");
58          if (approachType==AproachType.Delimiter) method.AddEndDelimiter();
59          method.AddStatement($"}}");
60          // retornar o método final no formato string
61          return roselynClass.AddMethod(method).ToString();
62      }
63
64      // Criar método de detalhe
65      private static string Details(RoselynClass roselynClass, string name)
66      {
67          // Criar método
68          Method method = new Method("Details");
69           // Adicionar instruções
70          method.AddStatement($"        public async Task<IActionResult> Details
    (Guid id)\r\n");
71          method.AddStatement($"        {{\r\n");
72          if (approachType==AproachType.Delimiter) method.AddStartDelimiter();
73          method.AddStatement($"            OperationResult<{name}BusinessModel>
     result = await {name.ToCamelCase()}Manager.GetById(id);\r\n");
74          method.AddStatement($"            if (result.Succeeded)\r\n");
```

```
75          method.AddStatement($"            {{\r\n");
76          method.AddStatement($"                {name}ViewModel {name.
    ToCamelCase()}ViewModel = {name}ViewModel.FromBusinessModel(result.Data);\r
    \n");
77          method.AddStatement($"                return View({name.ToCamelCase()}
    ViewModel);\r\n");
78          method.AddStatement($"            }}\r\n");
79          method.AddStatement($"            else\r\n");
80          method.AddStatement($"            {{\r\n");
81          method.AddStatement($"                ViewBag.ErrorDescription =
    result.ErrorDescription;");
82          method.AddStatement($"                return View(new {name.
    ToCamelCase()}ViewModel());\r\n");
83          method.AddStatement($"            }}\r\n");
84          if (approachType==AproachType.Delimiter) method.AddEndDelimiter();
85          method.AddStatement($"        }}\r\n");
86          // retornar o método final no formato string
87          return roselynClass.AddMethod(method,"").ToString();
88      }
89  }
```

classe ControllerGenerator

A.7 RoselynClass

```
1  public class RoselynClass
2  {
3      // Lista de métodos da classe
4      private List<Method> methods;
5       // Lista de variáveis da classe
6      private List<string> variables;
7      // Nome da classe
8      private string name;
9      // Abordagem a usar na junção
10     private AproachType approachType;
11
12     public RoselynClass(string path, string name, AproachType approachType){
13         this.name = name;
14         this.approachType = approachType;
15         // Inicializar a lista de métodos
16         methods = new List<Method>();
17         // verificar se a classe já existe
18         if (File.Exists(path))
19         {
20             // Criar uma árvore de sintaxe da classe
21             SyntaxTree syntaxTree = CSharpSyntaxTree.ParseText(File.
    ReadAllText(path));
22             IEnumerable<SyntaxNode> syntaxNodes = syntaxTree.GetRoot().
    DescendantNodes();
```

```
23            // Obter os métodos da classe
24            List<MethodDeclarationSyntax> methodDeclarationSyntaxs =
    syntaxNodes.OfType<MethodDeclarationSyntax>().ToList();
25            // Percorrer os métodos da classe
26            foreach (MethodDeclarationSyntax methodDeclarationSyntax in
    methodDeclarationSyntaxs)
27            {
28                string name = methodDeclarationSyntax.Identifier.toString();
29                List<string> statements = methodDeclarationSyntax.Body.
    Statements.split("/r/n");
30                string returnType = methodDeclarationSyntax.ReturnType.
    toString();
31                string modifiers = methodDeclarationSyntax.Modifiers.ToString
    ();
32                List<string> parameters = methodDeclarationSyntax.
    ParameterList.ToString().Replace("(","").Replace(")","").Split(",")
33                string attributes = methodDeclarationSyntax.AttributeLists.
    toString();
34                // Criar novo método
35                Method method = new Method(name){Statements = statements,
    ReturnType = returnType, Modifiers = modifiers, Parameters = parameters,
    Attributes = attributes};
36                // Adicionar novo método à lista de métodos
37                methods.Add(method);
38            }
39        }
40    }
41
42
43    // Retorna um método da lista de métodos
44    public Method GetMethod(string name){
45        return methods.Where(p => p.Name == name).SingleOrDefault();
46    }
47
48    // Adiciona um método à lista de métodos
49    public Method AddMethod(Method method){
50        // Ir buscar o método
51        var currentMethod = methods.Where(p => p.Name == method.Name).
    SingleOrDefault();
52        // Se o método não existir, então está a ser gerado pela primeira vez.
53        if (currentMethod==null)
54        {
55            // Adicionar o método à lista de métodos
56            methods.Add(method);
57            // Criar um registo no histórico com o método de 1   geração.
58            History.AddOrUpdate(new HistoryModel() { Class = this.name, Method
     = method, Patches = new List<List<Patch>>()});
59        }
60        else
61        {
```

```
62              // Obter o método do histórico de métodos
63              HistoryModel historyModel = History.GetMethod(method.Name, this.
    name);
64              // Verificar se o método a ser gerado (method) é diferente do mé
    todo corrente (currentMethod)
65              if (currentMethod.Body.Statements.ToString() != method.Body.
    Statements.ToString())
66              {
67                  method = Merge.MergeMethodStatements(approachType,
    historyModel, currentMethod, method);
68              }
69              // Verifica se a geração teve sucesso
70              if (method!=null) {
71                  // Actualiza o método na lista de métodos da classe.
72                  for(var i=0;i<methods.Count;i++)
73                  {
74                      if (methods[i].Name == method.Name)
75                          methods[i] = method;
76                  }
77                  // Actualiza o método no histórico
78                  historyModel.Method = method;
79                  History.AddOrUpdate(historyModel);
80                  // Grava o histórico de métodos
81                  History.Save();
82                  // Adicionar o método ao report de m todos gerados
83                  Report.Add(method, true);
84                  return method;
85              } else {
86              // Adicionar o método ao report de m todos gerados
87                  Report.Add(method, false);
88                  return currentMethod;
89              }
90          }
91      }
92
93  }
```

classe RoselynClass

A.8 Method

```
1  public class Method
2  {
3      // Lista de instruções
4      private List<string> statements { get; set; }
5
6      // Nome do método
7      public string Name { get; set; }
```

```
8
9       // Tipo do retorno
10      public string ReturnType { get; set; }
11
12      // Modifiers
13      public string Modifiers { get; set; }
14
15      // Par metros
16      public string Parameters { get; set; }
17
18      // Attributes
19      public string Attributes { get; set; }
20
21      public Method(string name)
22      {
23          statements = new List<string>();
24          Name = name;
25      }
26
27      // Adicionar uma instrução
28      public void AddStatement(string statement)
29      {
30          statements.Add(statement);
31      }
32
33      // Adicionar marcador de inicio de código gerado
34      public void AddStartDelimitir()
35      {
36          statements.Add("// Início de código gerado");
37      }
38
39      // Adicionar marcador de fim de código gerado
40      public void AddEndelimitir()
41      {
42          statements.Add("// Fim de código gerado");
43      }
44
45      // Adicionar várias instruções
46      public void AddStatements(List<string> statements)
47      {
48          this.statements = statements;
49      }
50
51      // Adicionar uma instrução numa posição específica
52      public void AddStatementAt(string statement, int pos)
53      {
54          statements.Insert(pos, statement);
55      }
56
57      // Devolver a lista de instruções
```

```
58      public List<string> GetStatements()
59      {
60          return statements;
61      }
62
63      // Apaga a lista de instruções
64      public void ClearStatements()
65      {
66          statements = new List<string>();
67      }
68
69      // Devolver a lista de instruções numa única string
70      public string ToString()
71      {
72          return String.Join("\r\n",statements);
73      }
74  }
```

classe Method

A.9 Merge

```
1   public static class Merge
2   {
3       public static Method MergeMethodStatements(ApproachTypeEnum approachType,
        HistoryModel historyModel, Method currentMethod, Method newMethod)
4       {
5           // Dependendo do tipo de abordagem (approachType) vai executar o
        algoritmo correspondente.
6           switch (approachType){
7               case AproachType.Delimiter :
8                   return Delimiter(currentMethod,newMethod);
9               break;
10              case AproachType.Diferencial :
11                  return Diferencial(historyModel,currentMethod,newMethod);
12              break;
13              case AproachType.DiffPatch :
14                  return DiffPatch(historyModel,currentMethod,newMethod);
15              break;
16          }
17      }
18
19      // Implementação da abordagem "Diferencial de instruções"
20      private Method Delimiter(Method currentMethod, Method newMethod)
21      {
22          // Obter as instruções do método actual. (Alterado pelo utilizador)
23          List<string> currentStatements = currentMethod.GetStatements();
24          // Procurar em que posição se encontra o delimitador do início do có
        digo gerado
```

```
25          int beginLine = currentMethod.GetStatements().FindIndex(p => p == "//
    Início de código gerado");
26          // Procurar em que posição se encontra o delimitador do fim do código
    gerado
27          int endLine = currentMethod.GetStatements().FindIndex(p => p == "//
    Fim de código gerado");
28          // Criação de uma lista de instruções
29          List<string> statements = new List<string>();
30          // Adicionar à lista as instruções que existem no método actual desde
    o inicio até ao delimitador de início.
31          statements.AddRange(currentStatements.GetRange(0, beginLine));
32          // Adicionar à lista as instruções do novo método (Segunda geração)
33          statements.AddRange(newMethod.GetStatements());
34          // Adicionar à lista as instruções que existem no método actual desde
    o delimitador de fim até ao fim.
35          statements.AddRange(currentStatements.GetRange(endLine + 1,
    currentStatements.Count-endLine-1));
36           // Apagar as instruções do novo método
37          newMethod.ClearStatements();
38         // Adicionar a lista de instruções ao método de retorno
39          newMethod.AddStatements(statements);
40          // Retornar o método
41          return newMethod;
42      }
43
44      // Implementação da abordagem "Diferencial de instruções"
45      private static Method Diferencial(HistoryModel historyModel, Method
    currentMethod, Method newMethod)
46      {
47          // Obter as instruções formatadas do método original (Primeira geração
    )
48          List<string> originalStatments = FormatHelper.Multiline(historyModel.
    Method.GetStatements()).ToList();
49          // Obter as instruções formatadas do método actual (Alterado pelo
    utilizador)
50          List<string> currentStatments = FormatHelper.Multiline(currentMethod.
    GetStatements()).ToList();
51          // Obter as instruções formatadas do novo método (Segunda geração)
52          List<string> newStatments = FormatHelper.Multiline(newMethod.
    GetStatements()).ToList();
53
54          // Criar de uma lista de instruções
55          List<String> statements = new List<String>();
56          // Percorrer as instruções do novo método
57          foreach (var inst in newStatments)
58          {
59              // Verificar se a instrução existe no método actual
60              var item = currentStatments.Where(p => p == inst).FirstOrDefault()
    ;
61              if (item != null)
```

```
62              {
63                  // Se existir no método actual,
64                  // então adicionar à lista de instruções e remover da lista
    actual.
65                  statements.Add(inst);
66                  currentStatments.Remove(item);
67              }
68              else
69              {
70                  // Se não existir no método actual e se não existir no método
    original,
71                  // então adicionar à lista de instruções
72                  if (!originalStatments.Contains(inst))
73                      statements.Add(inst);
74              }
75          }
76          // Percorrer as instruções do método actual
77          foreach (var inst in currentStatments)
78          {
79              if (!originalStatments.Contains(inst) && !newStatments.Contains(
    inst))
80                  statements.Add(inst);
81          }
82          // Apagar as instruções do novo método
83          newMethod.ClearStatements();
84          foreach (string statement in statements)
85          {
86              // Se não existir no método original e não existir no novo método,
87              // então adicionar à lista de instruções
88              if (!String.IsNullOrEmpty(statement))
89                  newMethod.AddStatement(statement);
90          }
91          // Formatar o método de retorno
92          newMethod = FormatHelper.Ident(newMethod);
93          return newMethod;
94      }
95
96      // Implementação da abordagem "Diff And Patch"
97      private Method DiffPatch(HistoryModel historyModel, Method currentMethod,
    Method newMethod)
98      {
99          // Obter as instruções do método original (Primeira geração)
100         // As instruções ficam guardadas numa string divididas por new lines.
101         string originalMethodText = String.Join("\r\n", historyModel.Method.
    GetStatements()) + "\r\n";
102         // Obter as instruções do método actual (Alterado pelo utilizador)
103         // As instruções ficam guardadas numa string divididas por new lines.
104         string currentMethodText = String.Join("\r\n", currentMethod.
    GetStatements()) + "\r\n";
105         // Obter as instruções do novo método (Segunda geração)
```

```
106         // As instruções ficam guardadas numa string divididas por new lines
107         string newMethodText = String.Join("\r\n", newMethod.GetStatements())
    + "\r\n";
108         // Criação de uma lista de instruções
109         List<string> statements = new List<String>();
110         // Inicialização da classe Diff Match Patch da Google
111         diff_match_patch dmp = new diff_match_patch();
112         // Criar um Diff entre o método original e o método actual
113         List<Diff> diffLists = dmp.diff_main(originalMethodText,
    currentMethodText, false);
114         // Criar um Patch entre o método original e o método actual
115         List<Patch> patchList = dmp.patch_make(originalMethodText,
    currentMethodText, diffLists);
116         // Adicionar o Patch à lista de patches do método
117         historyModel.Patches.Add(patchList);
118         // Aplicar a lista patches ao novo método.
119         object[] results = new object[] { newMethodText };
120         foreach (List<Patch> patch in historyItemModel.Patches)
121         {
122             results = dmp.patch_apply(patch, results[0] as string);
123             // Se não conseguir aplicar um dos patches devolver null
124             if ((results[1] as bool[]).Any(p => !p)) { return null; }
125         }
126         // Apagar as instruções do novo método
127         newMethod.ClearStatements();
128         // Adicionar o resultado ao método de retorno
129         newMethod.AddStatements(results[0].ToString().Split("\r\n").Where(p =>
     !String.IsNullOrEmpty(p)).ToList());
130         // Retornar o método
131         return newMethod;
132     }
133
134 }
```

classe Merge

A.10 FormatHelper

```
1 public static class FormatHelper
2 {
3     // Identa um método
4     private static List<string> Ident(List<string> statements)
5     {
6         AdhocWorkspace workSpace = new AdhocWorkspace();
7         workSpace.AddSolution(SolutionInfo.Create(SolutionId.CreateNewId("
    formatter"), VersionStamp.Default));
8         MethodDeclarationSyntax methodDeclarationSyntax = MethodDeclaration(
    PredefinedType(Token(SyntaxKind.VoidKeyword)), "RoselynMethod");
```

```
9         MemberDeclarationSyntax memberDeclarationSyntax =
    ParseMemberDeclaration(String.Join("\r\n", statements));
10        methodDeclarationSyntax = methodDeclarationSyntax.WithBody((
    memberDeclarationSyntax as MethodDeclarationSyntax).Body);
11        methodDeclarationSyntax = Formatter.Format(memberDeclarationSyntax,
    workSpace) as MethodDeclarationSyntax;
12        return (new List<string>(){ statements.First(),"{"}).Concat(
    methodDeclarationSyntax.Body.Statements.ToString().Split("\r\n")).Concat( (
    new List<string>() { "}" }) ).ToList();
13    }
14    // Colocar cada instrução na sua linha
15    public static void BreakInstructions(List<string> list)
16    {
17        Regex regex = new Regex("\\)\\.(?=(((?!\\)).)*\\()|[^\\(\\)]*$)");
18        string text = list[list.Count - 1];
19        Match resultMatch = regex.Match(text);
20        if (resultMatch.Success)
21        {
22            list[list.Count - 1] = (list.Count > 1) ? new string(' ', list[0].
    Length) + "." + text.Substring(0, resultMatch.Index+1) : text.Substring(0,
    resultMatch.Index+1);
23            list.Add(text.Substring(resultMatch.Index + 2));
24            BreakInstructions(list);
25        }
26        else if (list.Count > 1)
27        {
28            list[list.Count - 1] = new string(' ', list[0].Length) + "." +
    list[list.Count - 1];
29        }
30    }
31    // Colocar cada intrução na sua linha de forma identada
32    public static List<string> Multiline(List<string> statements)
33    {
34        List<String> brokenStatements = new List<String>();
35        // Para cada linha, chama o metodo BreakInstructions de modo a ter as
    instruções divididas por linha
36        foreach (string statement in statements)
37        {
38            List<string> instructions = new List<string>() { statement };
39            BreakInstructions(instructions);
40            brokenStatements.AddRange(instructions);
41        }
42        //Identa todas as linhas que começam por "."
43        List<string> identStatments = new List<string>();
44        for (var i = 0; i < brokenStatements.Count; i++)
45        {
46            string statement = brokenStatements[i];
47            if (statement.ToString().Trim().StartsWith(".") && identStatments.
    Count > 0)
48            {
```

```
49                 int lastDotIndex = identStatments.Last().IndexOf(".");
50                 if (lastDotIndex > -1)
51                 {
52                     identStatments.Add(new string(' ', lastDotIndex) +
    statement.Trim());
53                 }
54             }
55             else
56             {
57                 identStatments.Add(statement.ToString());
58             }
59         }
60         // Retorna o método todo identado
61         return Ident(identStatments);
62     }
63 }
```

classe FormatHelper

A.11 History

```
1  public class History
2  {
3      // Nome do ficheiro JSON onde é guardada a estrutura de Histórico
4      private const string Filename = "history.json";
5      // Lista de histórico
6      private static List<HistoryItemModel> historyModels;
7      // Ler a lista do ficheiro JSON para memória
8      private static void Load()
9      {
10         if (File.Exists(Filename))
11         {
12             // Desserializar o Json para memória
13             historyModels = JsonConvert.DeserializeObject<List<
    HistoryItemModel>>(File.ReadAllText(Filename));
14         }
15         else
16             historyModels = new List<HistoryItemModel>();
17     }
18     // Devolver um método da lista de histórico
19     public static HistoryModel GetMethod(string name, string className)
20     {
21         return historyModels.Where(p => p.ClassName = className and p.Name =
    name).SingleOrDefault();
22     }
23     // Adicionar ou fazer update a uma Método da lista de histórico
24     private static void AddorUpdateMethod(HistoryItemModel historyItemModel)
25     {
```

```
26          for (var i = 0; i <= historyModels.Count; i++)
27          {
28              if (historyItemModel.Name == historyModels[i].Name &&
    historyItemModel.ClassName == historyModels[i].ClassName)
29              {
30                  historyModels[i] = historyItemModel;
31                  return;
32              }
33          }
34          historyModels.Add(historyItemModel);
35      }
36      // Gravar a listar de histórica no ficheiro JSON
37      public static void Save(List<HistoryItemModel> historyItemModels)
38      {
39          File.WriteAllText(Filename, JsonConvert.SerializeObject(historyModels)
    );
40      }
41  }
```

classe History

A.12 HistoryModel

```
1  public class HistoryModel
2  {
3      // Nome da classe que se encontra o método
4      public string ClassName { get; set; }
5      // Lista de patch que foram aplicados no método
6      public List<List<Patch>> Patches { get; set; }
7      // Ultima geração do método com sucesso
8      public Method Method { get; set; }
9  }
```

classe HistoryModel

A.13 Report

```
1  public static class Report
2  {
3      // Lista de items do relatório
4      private static List<ReportModel> reportModels;
5      // Adicionar um item ao relátório
6      private static void Add(Method method, bool success)
7      {
8          if (reportModels==null) reportModels = new List<ReportModel>();
9          reportModels.Add(new ReportModel(){Method = method, Success = true});
10     }
```

```
11      // Mostrar a lista de items do relatório
12      public static void Print(List<HistoryItemModel> historyItemModels)
13      {
14          for (var i = 0; i <= reportModels.Count; i++)
15          {
16              Console.WriteLine(reportModels[i].Method.Name + " - " +
        reportModels[i].Method.Success);
17          }
18      }
19  }
```

classe Report

A.14 ReportModel

```
1  public class ReportModel
2  {
3      // Método
4      public Method Method {get;set;}
5      // Sucesso
6      public bool Success {get;set;}
7  }
```

classe ReportModel