Universidade de Aveiro

2014

Departamento de Eletrónica, Telecomunicações e Informática

JOSÉ ADELINO ALMEIDA BRAZETA

TESTES DE SOFTWARE NO SISTEMA DE INFORMAÇÃO DA JUSTIÇA CABO-VERDIANA

Universidade de Aveiro

2014

Departamento de Eletrónica, Telecomunicações e Informática

JOSÉ ADELINO ALMEIDA BRAZETA

TESTES DE SOFTWARE NO SISTEMA DE INFORMAÇÃO DA JUSTIÇA CABO-VERDIANA

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Sistemas de Informação, realizada sob a orientação do Doutor Cláudio Teixeira, Equiparado a Investigador Auxiliar e do Dr. Joaquim Manuel Henriques de Sousa Pinto, Professor Auxiliar do Departamento de departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho à Christine pelo apoio incansável e aos meus pais por todo o esforço despendido que possibilitou a minha formação académica.

O júri

Presidente

Prof. Dra. Ana Maria Perfeito Tomé
Professora Associada do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Professor Doutor Fernando Joaquim Lopes Moreira
Professor Associado do Departamento de Inovação, Ciência e Tecnologia da Universidade Portucalense

Prof. Doutor Cláudio Jorge Vieira Teixeira
Equiparado a Investigador Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

-

-

-

Agradecimentos

Sem o apoio da minha família, Christine, Mãe e Pai, nunca teria conseguido alcançar o que tenho hoje.
Ao Sousa Pinto e ao Cláudio por me terem aberto as portas ao mundo do trabalho, e pela vossa paciência em continuar o processo de ensino!
Aos meus colegas de trabalho, Gonçalo, Pedro, João, Filipe, Ricardo(s), Tiago, pelo incentivo diário a conquistar novas fronteiras.
A todos aqueles que, durante o meu percurso académico, me ajudaram a ultrapassar os mais diversos obstáculos.
A todos o meu obrigado!

palavras-chave

Cabo Verde, Justiça, sistema de informação, testes, software, e-government, e-justice

resumo

Nesta dissertação será apresentada a metodologia de testes desenvolvida para validar o Sistema de Informação da Justiça de Cabo-Verde.
As metodologias de testes apresentadas nesta dissertação serão focadas apenas nos fatores de erro humano, sendo dado enfase apenas às razões internas que podem levar ao aparecimento de falhas num *software*.
As metodologias apresentadas são o resultado de um processo de tentativa e erro para encontrar a melhor solução que possa satisfazer as necessidades da equipa de desenvolvimento do *software* e que permita validar todos os aspetos do mesmo. Neste processo foi possível encontrar uma arquitetura de testes escalável capaz de executar todos os conjuntos de testes numa fração do tempo original bem como realizar uma avaliação de desempenho da aplicação.

keywords

Cape-Verde, justice, information systems, tests, software, e-government, e-justice

abstract

This paper presents a testing methodology to test/validate the Justice Information System of Cape Verde.
The test methods presented in this dissertation will be focused only on human error factors, emphasis being given only to internal reasons that may lead to the presence of faults in this software.
The methodologies presented are the result of a process of trial and error to find the best solution that can meet the needs of the development and testing team in order to validate all aspects of this software.
In this process it was possible to find a scalable test architecture able to execute all test sets in a fraction of the original time and also to conduct a performance evaluation of the application.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

1 Introdução

O desenvolvimento de testes de *software* deve ser encarado como uma tarefa complexa e fundamental que, quando realizada de forma correta, pode poupar tempo a uma equipa de desenvolvimento de *software*. O objetivo principal de um teste de *software* é verificar o correto funcionamento de uma pequena parte de uma aplicação e assegurar o seu correto funcionamento quando futuras alterações sejam realizadas na mesma [1].

Existem duas causas que podem ser apontadas como os principais motivos para o mal funcionamento de um *software* [2]. Estas causas podem estar relacionadas, por um lado, com condições ambientais ou, por outro, com erros humanos. As condições ambientais estão, na sua generalidade, fora do controlo humano, como é o caso de inundações, fogos, tempestades elétricas, *etc*. Este tipo de condições podem provocar falhas no funcionamento de um *software* através de alterações ou danos causados ao nível do *hardware* sobre o qual é executada uma qualquer aplicação.

Na grande generalidade dos casos o erro humano, tanto na ótica do utilizador como na ótica do programador, pode ser apontado como a causa principal para o mau funcionamento do *software*. Este tipo de causa pode ser explicada pela capacidade limitada dos programadores e utilizadores em realizar tarefas de forma absolutamente correta ou sem qualquer tipo de falhas. Este facto pode ser associado a fatores como a complexidade das tarefas em mãos, deadlines para a finalização de módulos muito curtas, *stress* sentido pelos programadores e utilizadores no seu dia-a-dia, *etc*… [3].

Testar uma aplicação deve, portanto, ser considerada como uma atividade fundamental para assegurar não apenas o correto funcionamento da aplicação mas de todos os outros elementos que dependem dela.

1.1 Motivação

O desenvolvimento de software deve ser encarado como um processo que necessita de ser cuidadosamente monitorizado com um máximo de escrutínio, sendo indispensável que este seja minuciosamente testado e validado de modo a assegurar a sua qualidade em produção.

Regra geral as equipas de desenvolvimento de *software* realizam, aquando do desenvolvimento dos seus produtos, os testes ou validações necessárias para assegurarem a qualidade dos produtos que estão a desenvolver [4]. Estes testes ou validações têm como objetivo máximo salvaguardar as alterações ou implementações que uma equipa realiza num determinado *software*. No entanto,

quando se está a lidar com *software* de média ou grande complexidade, um dos aspetos que deve ser tido em linha de conta é o facto de que pequenas alterações realizadas ao nível do código, ou mesmo em estruturas de bases de dados, podem ter enormes impactos no desempenho geral do *software*, podendo ainda, em situações extremas, elevar de forma acentuada os custos de desenvolvimento de um projeto ou mesmo tornar todo um bloco de *software* inutilizável [5].

Para lidar com este tipo de situações é comum e aconselhado o desenvolvimento de testes de *software* que possam ser executados sempre que necessário, tendo a capacidade de validar o correto funcionamento do *software* em toda a sua extensão no menor intervalo temporal possível de modo a validar e verificar a qualidade do *software* em produção [6].

1.2 Problema

Um dos problemas presentes no desenvolvimento de testes a *software* de elevada complexidade é o facto de tal tarefa requerer o desenvolvimento de vários níveis de testes [7]. Quanto mais complexo for um *software* maior será a quantidade de testes necessários para garantir que todas as suas partes constituintes funcionam conforme o esperado.

Apesar de existir a capacidade de automatizar todo o processo de testes num determinado servidor, a execução linear de todos os testes pode tornar a atividade de verificar/validar um *software* contraproducente devido à quantidade de testes existentes. Mesmo considerando o processo de automação de testes através de *builds* noturnas tal tarefa poderá representar uma janela de execução de testes de várias horas (entre 10 a 14 no caso do SIJ), o que, em muitas situações, poderá ser um cenário inaceitável.

Durante o desenvolvimento do Sistema de Informação da Justiça de Cabo-Verde (SIJ) a equipa de testes deparou-se com a situação descrita anteriormente, onde a execução do universo total de testes a este *software* poderia demorar entre 10 a 14 horas se conduzida de forma sequencial. Considerando que o Sistema de Informação da Justiça se encontra atualmente em desenvolvimento e que, em paralelo, estão continuamente a ser desenvolvidos novos testes para validar as novas funcionalidades é expectável que a dimensão do referido problema tenda a escalar.

De um ponto de vista prático é completamente inviável que, para validar qualquer tipo de alteração ao *SIJ*, seja necessário esperar entre 10 a 14 horas. Devido a este facto foi necessário implementar uma estrutura de execução de testes que permitisse diminuir o tempo necessário a verificar/validar a qualidade do *software*.

1.3 Contribuição

Nesta dissertação será apresentada a metodologia de testes desenvolvida para testar o SIJ.

As metodologias de testes apresentadas nos capítulos subsequentes serão focadas apenas no fator de erro humano e não nos fatores relativos a erros provocados por causas ambientais. A área de estudo da presente dissertação é focada apenas nas razões internas que podem levar ao aparecimento de falhas num *software*.

As metodologias apresentadas são o resultado de um processo de tentativa e erro para encontrar a melhor solução que satisfaça as necessidades da equipa de desenvolvimento do *software* e que permita testar todos os aspetos do mesmo. No final deste processo de tentativa e erro foi possível encontrar uma arquitetura de testes escalável capaz de executar todos os conjuntos de testes numa fração do tempo original.

1.4 Organização da dissertação

A presente dissertação está dividida em seis capítulos, sendo este o capítulo 1. No capítulo dois serão descritos os aspetos gerais em termos de objetivos e arquitetura do *SIJ* bem como a sua estrutura de desenvolvimento. No capítulo três será apresentada uma visão global sobre o conceito de testes de *software*, sendo os testes de *software* classificados pelas suas áreas principais. No capítulo quatro serão apresentadas e descritas as diferentes metodologias utilizadas ao testar o *software* referido anteriormente, abordando-se as especificidades de cada metodologia e sumarizando-se as respetivas vantagens e desvantagens. No capítulo cinco serão apresentados alguns problemas encontrados no decorrer do processo de testes ao *software* descrito anteriormente, sendo apresentadas algumas soluções que permitem fazer face a tais problemas. No capítulo sexto, será apresentada a conclusão da presente dissertação. Por último, no capítulo sétimo será apresentada uma proposta de trabalho futuro para a presente dissertação.

2 Sistema de Informação da Justiça

2.1 Visão geral do sistema

A porção do SIJ visível para os utilizadores consiste numa aplicação *web* que pode ser acedida via *web* browser. O objetivo principal desta aplicação consiste em auxiliar os utilizadores nas ações de gestão e manuseamento de processos penais e processos civis da justiça Cabo-Verdiana. O sistema de Informação da Justiça está detalhado de forma exaustiva em [8], [9] e [10]. Nesta aplicação *web* cada utilizador possui credenciais próprias que lhe garantem acesso a uma área própria da aplicação. Nesta área pessoal é possível aceder e interagir com processos penais ou civis de acordo com as permissões de cada utilizador. Cada utilizador tem um *role* associado sendo que *roles* diferentes têm politicas de permissões diferentes dentro da aplicação, garantindo ou negando acesso a determinadas áreas ou funcionalidades. A Figura 1 pretende ilustrar a estrutura de interação com a aplicação de acordo com a ótica do utilizador.

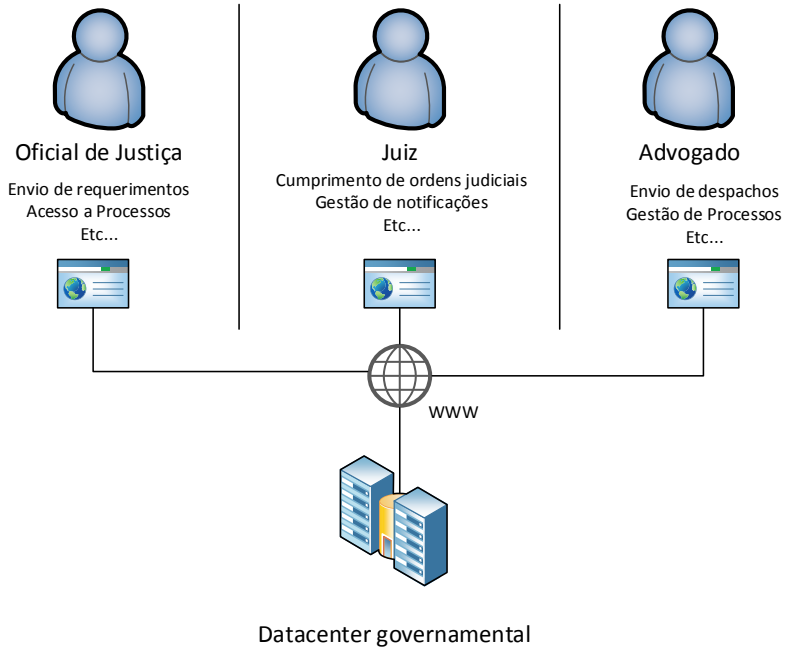

Figura 1 – SIJ – Ótica do utilizador

Como o apresentado em [10], o Sistema de Informação da Justiça é constituído por duas aplicações distintas que se ligam segundo uma arquitetura orientada a serviços (SOA) [11], sendo elas uma aplicação *web* e um serviço de gestão de *workflows*.

A aplicação *web* foi construída segundo um padrão de arquitetura multicamada. Foi desenvolvida com o objetivo de auxiliar os utilizadores finais no processo de gestão de processos judiciais

disponibilizando interfaces *web* para esse efeito. Esta aplicação permite realizar um leque alargado de tarefas como a interação com processos judiciais, gestão de notificações e mensagens internas trocadas entre utilizadores, providenciar acesso a documentação legal, gestão de anotações de utilizadores, calendarização de tarefas, *etc*.

O serviço de gestão de *workflows* tem como objetivo principal a gestão de todos os procedimentos envolvidos nos processos judiciais. Dada a natureza de um processo judicial, onde existe uma noção clara das diferentes fases ou estados que podem compor um processo, é da responsabilidade do serviço de gestão de *workflows* a gestão de todos os estados, transações e interações que podem ocorrer em cada processo judicial. É da responsabilidade deste serviço a definição de *roles* de utilizador que podem interagir com um processo numa determinada fase, os tipo de decisões possíveis por fase processual, *etc*.

A Figura 2 apresenta a estrutura arquitetural de ambas as aplicações.

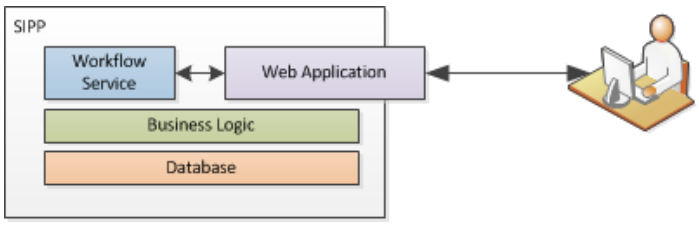

Figura 2 – Estrutura arquitetural [10]

Como o apresentado na imagem anterior, o serviço de gestão de *workflows* está integrado num serviço de comunicação, o qual permite a ligação entre a aplicação *web* (Web Application) com o sistema de *workflows* (*Workflow* System).

Ambas as aplicações (aplicação *web* e serviço de gestão de *workflows*) partilham o mesmo código base. Toda a logica comum (acesso a base de dados, logica de validações, logicas de segurança, *etc*.) foi desenvolvida de forma modular e independente das aplicações que dela dependem. Este código base é designado por “Business Logic Layer”.

A camada de base de dados está implementada recorrendo ao sistema de gestão de base de dados *SQL SERVER 2008 R2* [12], sendo utilizada o serviço de mapeamento disponibilizado pela *Entity Framework* [13] como elemento de ligação entre a camada de base de dados e a camada da lógica de negócio.

2.2 Metodologia de desenvolvimento

Para o presente projeto a equipa de desenvolvimento foi dividida em três grandes áreas. Um grupo de programadores encontra-se encarregue pelo desenvolvimento da camada de apresentação da aplicação (interfaces *web*), outro grupo está encarregue pelo desenvolvimento das camadas base da aplicação (camada de *workflows*, camada da logica de negócio e camada de base de dados) e um terceiro grupo encontra-se encarregue pelo desenvolvimento de testes de *software* com o objetivo de assegurar a qualidade do *software*.

A decisão de separar as equipas de desenvolvimento da equipa de testes foi tomada na fase inicial do projeto de modo a garantir que todos os testes fossem realizados por um grupo de programadores independente.

Apesar do grupo de desenvolvimento ter sido separado pelas suas principais áreas de trabalho todos os seus elementos encontram-se a trabalhar no mesmo espaço físico. Este facto permite melhorar o fluxo de informação entre os elementos do grupo de trabalho independentemente da sua área de desenvolvimento, sendo assim possível promover uma melhor interação entre designers, equipa de desenvolvimento e equipa de testes.

De modo a agilizar o processo de desenvolvimento é crucial que a equipa de testes possua uma compreensão profunda sobre o código que está a testar. Deste modo é possível acelerar o processo de desenvolvimento do *software*, eliminando-se a necessidade de interromper constantemente o processo da equipa de desenvolvimento para que se possa conduzir avaliações sobre potenciais falhas ao nível do código da aplicação. Este tipo de metodologia organizacional é recomentado por autoridades de certificação de testes, como é o caso da ”International Software Testing Qualification Board” [14].

2.2.1 Gestão de tarefas

A equipa de desenvolvimento e testes do SIJ tenta seguir os processos da *framework* de desenvolvimento *Scrum* [15]. Atualmente o *software* é desenvolvido de forma iterativa, sendo colocadas, numa base regular, diferentes versões do *SIJ* em produção. Este processo iterativo é vulgarmente designado como *sprint* de desenvolvimento. Todas as versões do *SIJ* que entram em produção contêm sempre incrementos em relação a versões anteriores. Estes incrementos podem conter elementos como correções de erros (*bugs*) ou adições de novas funcionalidades.

Cada *sprint* de desenvolvimento te m a duração média de duas semanas. O trabalho desenvolvido em cada *sprint* contém o resultado do trabalho da equipa de desenvolvimento e testes do *SIJ*.

A equipa de desenvolvimento e equipa de testes utilizam os mecanismos presentes na plataforma *Team Foundation Server* (TFS) [16] para conduzirem toda a gestão de *backlogs* do *SIJ*. Nesta plataforma os *backlogs* podem ser categorizados em *"User Stories", "Tasks", "Bugs", "Test Cases"* e *"Issues"* [17].

***User S*tories** são utilizadas para descrever, de forma genérica, incrementos ou novas funcionalidades que necessitam de ser implementadas na aplicação. Regra geral uma nova *User Storie* origina a criação de uma ou mais "*Tasks*" associadas.

Tasks são utilizadas para identificar tarefas específicas de desenvolvimento associadas a um elemento da equipa. Regra geral uma nova *Task* origina a criação de um ou mais *Test Cases* associadas.

Bugs são utilizados para identificar eventuais erros no *SIJ*. Regra geral um novo *Bug* origina a criação de um ou mais *Test Cases* associadas.

Test Cases são utilizados como elementos que identificam os testes de *software* que necessitam de ser implementados, sendo o resultado da criação de *Tasks* ou *Bugs*, de modo a validar determinados caraterísticas do *software*.

Issues são utilizados para descrever qualquer questão levantada pelos utilizadores do SIJ. Neste sentido um *issue* poderá originar novas *Tasks* ou *Bugs*.

O esquema de dependência de *backlogs* descrito nos parágrafos anteriores permite organizar o trabalho da equipa de desenvolvimento e testes. Permite ainda identificar e garantir que tanto novas funcionalidades e correções de erros são devidamente registados e testados. Qualquer registo de *backlog* está sempre associado a uma única pessoa sendo da sua responsabilidade a execução dos seus objetivos. Todos os registos de *backlog* são priorizadas e associados a *sprints* de desenvolvimento específicas.

2.2.2 Branching

Cada versão do *SIJ* que entra em produção poderá conter o resultado do trabalho da equipa de testes e desenvolvimento ou apenas da equipa de testes. Este facto está diretamente relacionado

com o feedback dos utilizadores da aplicação, os quais podem detetar eventuais problemas com as versões em produção do *software*. A deteção de problemas pode despoletar apenas a intervenção da equipa de testes ou, dependendo da severidade dos mesmos, da equipa de testes e desenvolvimento. A deteção de problemas numa versão do *software* despoleta sempre operações de correções (*updates*), testes e novo *deploy* do *SIJ*.

É portanto necessário que existam a todo o instante, e por *sprint* de desenvolvimento, duas versões ativas do *software*. Uma das versões contém todas as adições realizadas num determinado *sprint* e entrará em produção no final da mesma. A outra versão encontra-se em produção e está sujeita a eventuais alterações, testes e novos *deploys*.

No ciclo de desenvolvimento do *SIJ* a utilização de *branches* segue um padrão designado por "*Branch for Release*" ou "*Parallel Releasing/Development Lines*" [18], [19]. Neste tipo de padrão de desenvolvimento, e como referido anteriormente, existem a todo o instante duas versões ativas do *software*. Essas versões são controladas através de *branches* de uma versão principal (ou versão *Main*) do *software*. Em cada *sprint* a equipa de desenvolvimento pode focar o seu trabalho num *branch* que ainda não entrou em produção (*branch* de desenvolvimento), onde pode desenvolver implementações que serão colocadas em produção no futuro. Durante este processo a equipa de testes pode focar o seu trabalho no *branch* do *SIJ* que se encontra em produção (*branch* de produção), implementando as alterações originadas por descobertas de bugs ou outro tipo de feedback dos utilizadores. Cada bloco de correções pode originar novas *releases* do *software* para produção, sendo repetido todo o processo de feedback / *update* / testes / nova *release*.

Por cada *release* intermédia o código de um *branch* de produção é fundido com a versão principal (*Main*) do *software* e, de seguida, com o *branch* de desenvolvimento. Este processo permite enviar eventuais correções de problemas para as versões em desenvolvimento da aplicação. A Figura 3 pretende ilustrar o esquema de desenvolvimento por *branches*. Nesta figura as linhas identificadas por "Main" representam a mesma versão principal do código do SIJ. O espaço identificado entre as linhas de "Dia 0" e "Dia n" representa a duração média de uma sprint de desenvolvimento.

A abertura de um novo *branch* de desenvolvimento é apenas iniciada no final de uma *sprint* se todas as tarefas de desenvolvimento estiverem concluídas e todos os testes de *software* validarem o normal funcionamento do mesmo. A abertura de um novo *branch* de desenvolvimento implica que um *branch* em produção seja terminado e fundido com a versão principal do código (*Main*) bem como com o *branch* de desenvolvimento. A abertura de um novo *branch* implica também que o *branch* de desenvolvimento entre em produção.

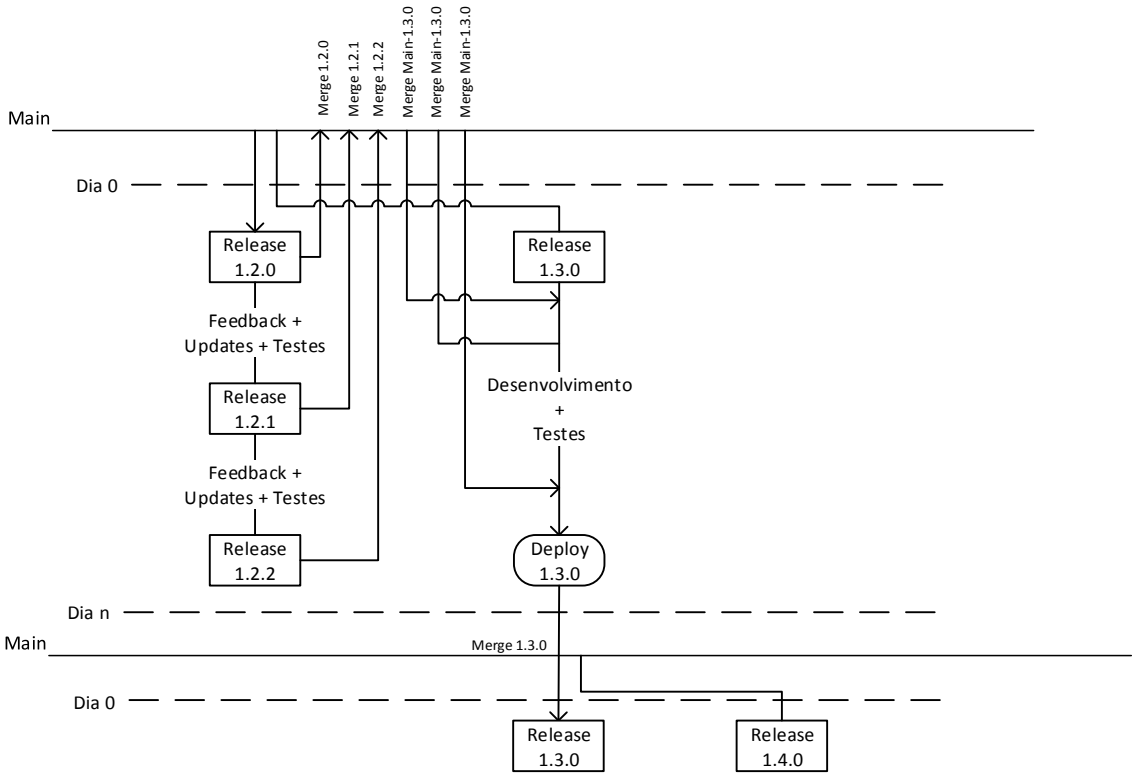

Figura 3 – Esquema de desenvolvimento por branches

Toda a gestão de *branches* no *SIJ* é conduzida através da plataforma de gestão do ciclo de vida do *software* utilizada pela equipa de testes e desenvolvimento [20].

3 Classificação de testes

De acordo com *Jintao Pan* [21] os testes de *software* podem ser classificados em três áreas principais, a área relativa aos objetivos dos testes, o ciclo de vida do *software* e o âmbito do teste.

3.1 Classificação por objetivo

Na classificação por objetivo, segundo o mesmo autor [21], as metodologias de testes poderão ser separadas entre testes de exatidão, testes de performance, testes de fiabilidade e testes de segurança.

3.1.1 Testes de exatidão

Testes de exatidão podem ser definidos como uma metodologia de testes cujo objetivo principal passa por garantir o correto funcionamento de um *software* ao seu nível mais elementar. Para garantir o correto funcionamento é necessário a existência de uma entidade, normalmente designada por oráculo, capaz de distinguir comportamentos corretos de comportamentos incorretos [22]. Os testes de exatidão podem ainda ser divididos em duas categorias, testes de caixa negra (*Black Box Testing*) [23] e testes de caixa transparente (*White Box Testing*) [24], [25]. Um teste pode ser classificado como um teste de caixa negra quando não é possível obter qualquer tipo de informação sobre o modo de desenvolvimento e implementação do *software*. Deste modo o *software* é visto como uma caixa negra onde um conjunto de inputs [2] é passado ao *software* sendo de seguida examinados os outputs devolvidos pelo mesmo de modo a validar o seu funcionamento. Por outro lado nos testes de caixa transparente são conhecidas, à partida, todas as características internas do *software*, desde a linguagem de implementação aos respetivos módulos constituintes, pelo que é possível testar cada aspeto do *software* individualmente [24], [25].

A Figura 4 representa um teste de exatidão onde são validadas duas operações de cifra e decifra de mensagens. Neste contexto este teste de *software* poderá representar tanto um teste de caixa negra como um teste de caixa transparente. A diferença prende-se com o facto de ser ou não conhecido o conteúdo das funções designadas por “GravaDocumentoTemporarioCifrado” e “DevolveDocCifradoPorId”.

```
/// <summary>
///A test for DevolvePorId
///</summary>
[TestMethod()]
public void CifraEDecifra_Test()
{
        byte[] array = null;

        string mensagem = "Mensagem para ser cifrada";

        Guid docID = GravaDocumentoTemporarioCifrado(ref array, mensagem);

        Documento actual = Documentos.DevolveDocCifradoPorId(docID);

        string mensagemOriginal = (string)DesserializarObjecto(actual.Contents);

        Assert.AreEqual(mensagem, mensagemOriginal);
}
```

Figura 4 –Exemplo de um teste exatidão (White Box vs Black Box Testing)

3.1.2 Testes de performance

Na área do desenvolvimento de *software* o conceito de performance pode variar de caso para caso. *Softwares* diferentes podem ter requisitos diferentes em termos de performance. Regra geral são tidos em linha de conta elementos como a memória consumida, tempo de resposta e processamento. Em determinadas situações a análise de apenas um destes indicadores é suficiente enquanto noutros casos poderá ser necessário avaliar vários indicadores. O objetivo principal de um teste de performance passa então por avaliar um ou vários indicadores de performance relevantes para a utilização de um *software* [26]–[28].

Testes de carga podem ser considerados como um exemplo de testes de performance. O seu objetivo passa por testar a capacidade que uma aplicação tem em responder a um aumento súbito da sua carga de utilização, devido a fatores como o aumento de utilizadores em paralelo a interagirem com o *software* [26]. É fundamental testar e compreender os limites de uma aplicação em termos da sua capacidade para responder a pedidos intensivos de utilização por partes dos seus utilizadores. Só assim será possível garantir que uma aplicação cumpre os requisitos para os quais foi construída. É igualmente importante compreender as capacidades de escalabilidade de uma aplicação e avaliar se, de algum modo, estão diretamente relacionadas com atualizações de *hardware* ou se, por outro lado, a capacidade de resposta da aplicação permanece inalterada, sendo assim independentes da capacidade do *hardware* que a sustenta. Uma não correspondência entre um aumento da capacidade de um *hardware* e uma melhoria de performance de um *software* poderá indiciar eventuais falhas na arquitetural do mesmo.

Atualmente existem no mercado várias ferramentas que podem auxiliar equipas de testes de *software* a avaliar a performance de um *software* alvo, como é o caso das ferramentas "BlazeMeter" [29] e "NeoLoad – Load Testing Tool" [27].

3.1.3 Testes de fiabilidade

Testes de fiabilidade não é uma metodologia em si, mas sim uma combinação de várias metodologias que permitem medir o nível de fiabilidade de um *software*. Tal combinação de testes permite medir a probabilidade de um *software* operar num estado live de falhas [21]. Esta medição pode ser obtida combinando o resultado de testes de metodologias diferentes como, por exemplo, testes de robustez [30] e testes de stress [28]. Combinando o resultado de execução dos testes de diferentes metodologias é possível obter uma visão global do nível de confiança que é possível ter num determinado *software*.

Esta metodologia pode ser especialmente importante num processo de tomada de decisão, onde uma equipa de desenvolvimento necessita de decidir se, por exemplo, uma versão de um *software* está ou não pronta para entrar em produção.

3.1.4 Testes de segurança

O objetivo principal de testes de segurança passa por identificar falhas de segurança num sistema [30]. A identificação de tais falhas pode ser alcançada através da simulação de ataques de segurança. Partindo dos resultados de simulações de ataques será possível inferir a capacidade que um *software* tem para reconhecer e bloquear ataques informáticos. A Figura 5 representa um exemplo de um teste de segurança. Neste cenário é validada uma função que permite autenticar um utilizador através do seu *Username* e *Password*. Este teste valida um possível cenário de "SQL Injection", garantindo que não é possível autenticar um utilizador utilizando esquemas de adulteração dos campos referentes ao nome de utilizador e/ou password.

```
/// <summary>
///A test for DevolvePorId
///</summary>
[TestMethod()]
public void Login_Test()
{
       bool sucesso = AutenticarUtilizador("user'OR '1'='1", "password");
       Assert.IsFalse(sucesso);
}
```

Figura 5 – Teste de segurança

3.2 Classificação pelo ciclo de vida do *software*

Os testes de *software* podem ser classificados de acordo com o ciclo de vida do *software*, sendo classificados como testes da fase de requisitos, testes manuais, testes automatizados e testes de aceitação [21].

Diferentes equipas de desenvolvimento podem adotar diferentes metodologias de desenvolvimento durante o ciclo de vida de um *software*. Certas metodologias de desenvolvimento, como a metodologia "*scrum agile*" [31], podem ser indicadas para uma equipa de desenvolvimento mas ser inadequadas ao processo de desenvolvimento de uma outra. No entanto existem alguns aspetos no ciclo de vida do desenvolvimento de um *software* que são independentes da metodologia adotada por uma determinada equipa, como, por exemplo, o facto de depois de libertada uma versão de um *software* ser necessário realizar alterações adicionais. Este tipo de condicionantes leva a que a equipa de desenvolvimento reúna requisitos adicionais sobre as alterações, realize as altere, lance uma nova versão e espera pelo feedback dos *stakeholder* o que poderá levar à repetição de todo o processo mencionado anteriormente [4]. Todo este processo acaba por gerar um conjunto de fases iterativas, comuns no ciclo de desenvolvimento de *software*, como o apresentado na Figura 6.

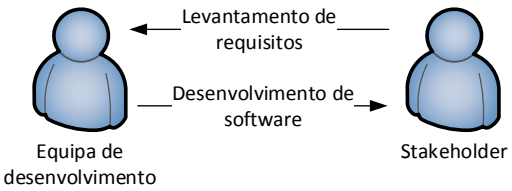

Figura 6 -Ciclo de desenvolvimento de software

Neste contexto iterativo, e de acordo com alguma literatura, os testes de *software* tornam-se uma parte vital no ciclo de desenvolvimento de *software* [4].

3.2.1 Testes na fase de requisitos

É na fase de requisitos, dentro do contexto do ciclo de desenvolvimento de *software*, que os requisitos de um projeto devem ser reunidos. Dentro da lista de requisitos devem igualmente constar a lista de testes de *software* necessários para validar o correto funcionamento da aplicação em desenvolvimento [32]. Tais requisitos podem ser compostos por documentação que especifica elementos como "user stories", listas de funcionalidades priorizadas do produto, *etc*…, sendo estes elementos de extrema importância tanto para o desenvolvimento do *software* como para criar os requisitos dos testes que o irão validar.

3.2.2 Testes manuais e testes automatizados

Durante a fase de desenvolvimento os programadores desenvolvem o código necessário para cumprir todos os requisitos identificados na fase anterior. Durante esta fase devem ser criados testes de *software* que permitam validar a aplicação em desenvolvimento, podendo estes ser testes manuais ou testes automatizados.

Testes manuais podem ser definidos como a gama de testes de *software* que, para cada sessão de interação com uma aplicação, necessitam de interação humana para as tarefas de submissão de dados e também para as tarefas de análise de resultados. O componente humano é o único que pode ser utilizado para validar o *software* em qualquer ponto temporal [33].

Testes automatizados podem ser definidos como a gama de testes de *software* que não necessitam de interação humana para validar uma parte de um *software* alvo. Nos testes automatizados os processos de entrada de dados, avaliação de resultados e execução calendarizada de testes são, regra geral, realizados forma automática [34].

3.2.3 Testes de aceitação

Numa fase final do ciclo de vida do desenvolvimento de um *software* este apenas estará pronto para entrar em produção quando todos os testes associados a ele validarem o seu correto ou normal funcionamento. Nesta fase o *software* será colocado à disposição dos utilizadores finais para que estes possa conduzir um processo de avaliação sobre o mesmo (ou testes de aceitação) [35].

3.3 Classificação pelo âmbito dos testes

Segundo o âmbito dos testes de *software* estes podem ser classificados como testes unitários, testes de componentes, testes de integração e testes de sistema [21].

3.3.1 Testes unitários

Testes unitários podem ser considerados como o ato de testar um *software* a um nível mais granular. Normalmente implica testar aspetos elementares de uma dada função ou método [36] [37]. A Figura 7 ilustra um teste unitário que valida a função designada por "Levantamento". Neste cenário o foco do teste passa apenas por validar a função referida, independentemente de existirem outras funções constituintes da classe "ContaBancaria".

```
[TestMethod]
public void GestaoDeConta_Test()
{
	double saldo = 10.0;
	double levantamento = 1.0;
	double valorFinal = 9.0;

	var conta = new ContaBancaria("JohnDoe", saldo);
	conta.Levantamento(levantamento);
	double atual = conta.Saldo;

	Assert.AreEqual(valorFinal, atual);
}

class ContaBancaria
{
	public double Saldo { get; set; }
	public string Nome { get; set; }

	public ContaBancaria() { }
	public ContaBancaria(string nome, double saldo)
	{
		Nome = nome;
		Saldo = saldo;
	}
	public void Levantamento(double levantamento)
	{
		this.Saldo = this.Saldo - levantamento;
	}
	public void Deposito(double valor)
	{
		this.Saldo = this.Saldo + valor;
	}
}
```

Figura 7 – Exemplo de um teste unitário

3.3.2 Testes de integração

Testes de integração são testes de *software* semelhantes a testes unitários. A grande diferença reside no facto de o nível de complexidade de um teste de integração ser maior que o nível de complexidade de um teste unitário. O foco de um teste de integração não está em validar aspetos granulares de um *software* mas sim em validar a união de dois ou mais componentes, os quais podem ter sidos testados previamente através de testes unitários. Estes componentes são combinados e a interface que os liga é testada de modo a garantir que, mais do que o facto de funcionarem de forma aparentemente correta individualmente, funcionam de forma correta em conjunto [38].

3.3.3 Testes de componentes

Os testes de componentes têm um nível de integração superior quando comparados com os testes de integração. Ao passo que os testes de integração focam-se em assegurar que a ligação entre métodos ou funções dentro de um módulo funcionam de forma correta, os testes de componentes tentam assegurar o correto funcionamentos de um módulo como um todo [36].

3.3.4 Testes de sistema

Os testes de sistema podem ser definidos como o ato de testar um *software* do ponto de vista dos seus utilizadores, recorrendo-se normalmente a interfaces de utilizador para a realização dos testes [39]. Um exemplo de testes de sistema podem ser os testes designados por “UI Tests” [40]. Neste tipo de testes tenta-se avaliar o correto funcionamento do *software* simulando as ações que os utilizadores podem realizar nas interfaces de uma aplicação.

3.4 Visão geral

A Figura 8 ilustra os diferentes tipos de classificações de testes de *software* apresentados nos capítulos anteriores. Nesta figura são apresentadas as diferentes classificações e respetivas terminologias de testes por tipo de classificação.

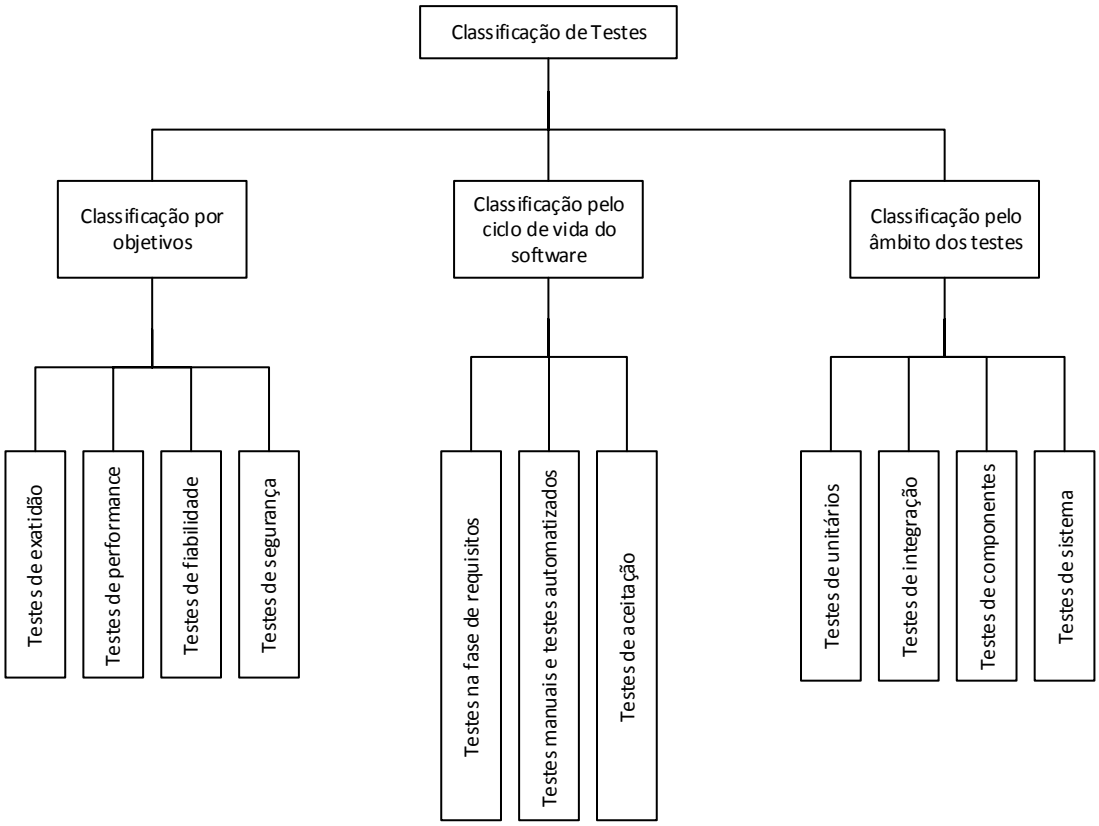

Figura 8 – Classificações de testes

3.5 Considerações finais

Nos próximos capítulos será realizada uma apresentação resumida do SIJ. Será ainda explicada a razão pela qual foi necessário adotar diferentes metodologias de testes de modo a responder a diferentes necessidades do *software*. Todos os testes desenvolvidos foram adaptados para diferentes partes do *software* ou para responder a necessidades específicas do mesmo, resultando disso um conjunto de padrões de testes que serão explicados nos capítulos subsequentes. Serão ainda abordados os problemas encontrados na utilização dos referidos padrões de testes e será ainda explicado o processo pelo qual foi possível ultrapassar tais problemas.

4 Metodologias de testes

Como descrito anteriormente, todas as camadas do *SIJ* têm diferentes funcionalidades/objetivos resultando em diferentes necessidades ao nível dos testes para validar o funcionamento de cada uma delas.

Todos os testes desenvolvidos seguem um padrão comum. Todas as interfaces, métodos, funções, *stored procedures*, *triggers* de base de dados, *etc*…, têm, regra geral, mais do que um teste para validar o seu correto funcionamento. Cada teste valida apenas um aspeto singular ou especifico do elemento que está a ser testado. Deste modo é relativamente fácil isolar e identificar potenciais problemas.

Existe portanto um esforço constante para produzir testes que sejam o mais simples possível, contendo apenas o código estritamente necessário para validar funcionalidades ou comportamentos alvo.

É também importante enfatizar que um teste em falha não corresponde diretamente a um potencial problema com o elemento a ser testada. Na grande generalidade dos casos um teste em falha apenas identifica uma discrepância entre a lógica do teste e a lógica do método, função, *stored procedure*, *etc*. Dessa discrepância resulta uma necessidade de rever tanto o código do teste como do elemento alvo do teste. Não obstante a falha de um teste é um bom indicador de que algo não está correto sendo necessário uma nova análise da discrepância para que seja possível identificar potenciais problemas.

Nas próximas subsecções serão discutidas as possíveis abordagens em termos de metodologias de execução de testes bem como os diferentes padrões de testes desenvolvidos para testar cada camada do SIJ. Tais padrões são o resultado de características semelhantes entre os grupos de testes para uma determinada camada do SIJ. Estas semelhanças centram-se ao nível de scripts de configuração, comportamento dos testes, tempos de execução, *etc*…, dando resultado ao aparecimento de grupos de testes bastante semelhantes, independentemente de analisarem aspetos diferentes dentro da mesma camada do *SIJ*.

4.1 Metodologia de execução de testes

Utilizando a ferramenta de gestão do ciclo de vida de aplicações "*Team Foundation Server"* é possível subdividir as metodologias de execução de testes, nas definições de *builds* da referida

ferramenta [41], em três categorias, sendo elas: execuções manuais, execuções calendarizadas e execuções proactivas [42].

A execução manual de testes pode ser encarada como a opção mais básica para executar um grupo de testes, podendo ser realizada selecionando um grupo de testes, num qualquer ambiente de desenvolvimento integrado, e ordenando a sua execução. No caso do *SIJ* o ambiente de desenvolvimento utilizado é o *Visual Studio 2010* e, dependendo das definições de configuração dos testes de *software* [43], um conjunto de testes poderá ser executado localmente ou em máquinas remotas.

A execução calendarizada de testes é a capacidade de agendar uma compilação de código e execução de um grupo de testes pós compilação, na eventualidade da compilação ocorrer sem erros [44]. Regra geral a execução calendarizada de testes precedem as compilações diárias do código fonte de uma aplicação, o qual, geralmente, se encontra num qualquer sistema centralizado de controlo de versões [45]–[47]. Esta compilação e execução de testes permite validar o código mais recente de uma aplicação, possibilitando ainda a deteção prévia de eventuais problemas.

A execução de testes proactiva pode ser encarada como uma alternativa à execução calendarizada de testes [44]. Neste sentido a execução de testes pode ser ativada não de forma calendarizada mas sim por tentativas de submissão de código para um sistema central de controlo de versões. Esta compilação proactiva precedida por uma execução de testes pode ocorrer em cada submissão de código para um sistema de controlo de versões ou pode ser ativada apenas com o acumular de várias submissões de código. Neste cenário a inclusão do código submetido pode estar sujeito a processos de controlo de qualidade. Este processo de controlo de qualidade é assegurado tanto ao nível da compilação do código como da execução dos testes de software.

4.1.1 Execuções manuais de testes de *software*

Como referido anteriormente o ciclo de desenvolvimento do *SIJ* pode ser separado em dois grandes blocos. Por um lado, e regra geral, a equipa de desenvolvimento encontra-se a trabalhar numa versão (ou *branch*) do código diferente da equipa de testes.

A equipa de desenvolvimento tende a focar o seu esforço no desenvolvimento de versões futuras do *software*, trabalhando assim nas versões em desenvolvimento do *SIJ*.

Por outro lado, depois de estanque o desenvolvimento de novas funcionalidades de um *sprint*, é função da equipa de testes assegurar a qualidade e correto funcionamento desse *branch* do *software* para que este possa entrar em produção.

Existem portanto, a todo o instante, duas versões do *SIJ*, uma em desenvolvimento (ou a versão "*Dev*") e a versão que esta a ser validada para que possa entrar em produção (versão de "*Release*").

A execução manual de testes tende a ser utilizada com mais frequência pela equipa de testes sempre que esta necessita de validar a qualidade de um *branch* de *release.* Durante a fase de validação de uma *release* são, regra geral, encontrados diversos problemas com a aplicação, sendo necessário proceder à correção dos problemas e à validação das respetivas correções. Este processo iterativo de avaliação-correção-avaliação apenas termina se todos os testes validarem o funcionamento do software ou se de entre um grupo de testes em falha existir um nível de confiança elevado na qualidade do *software* e na sua prontidão para entrar em produção.

4.1.2 Execuções calendarizadas de testes de *software*

A execução calendarizada de testes tende a ter maior utilidade para a equipa de desenvolvimento. Existem execuções calendarizadas diárias iniciadas em horários pós laboral que servem como um método de validação do trabalho diário da equipa de desenvolvimento.

Este método permite que a equipa de desenvolvimento tenha a perceção de eventuais problemas presentes nos *branches* de desenvolvimento.

4.1.3 Execução proactiva de testes

A execução proactiva é útil quando existe uma taxa de submissão de código relativamente lenta para um sistema de controlo de versões. Durante o dia de trabalho a equipa de desenvolvimento tende a criar uma grande quantidade de submissões de código para o sistema de controlo de versões o que, eventualmente, conduziria a uma sobrecarga de processamento na máquina onde se encontra alojado o respetivo sistema de controlo. De modo a evitar este processamento desnecessário todo o código desenvolvido diariamente é validado durante períodos noturnos.

4.2 Testes unitários

Todos os testes unitários foram desenvolvidos com o intuito de garantir o correto funcionamento da camada da lógica de negócio e também da camada da base de dados. Estas camadas podem ser

descritas como sendo a base ou o núcleo da aplicação, onde estão concentrados os métodos/funções mais básicos que a compõem. Este facto gerou a necessidade de desenvolver uma tipologia de testes que fosse de fácil e rápida implementação de modo a garantir o correto funcionamento dos elementos constituintes destas camadas.

A camada da base de dados e a camada da lógica do negócio servem propósitos diferentes. A camada da lógica do negócio baseia-se numa enorme quantidade de métodos necessários para realizar operações de *create, read, update* e *delete* (CRUD) sobre a base de dados. Por seu lado a camada da base de dados utiliza centenas de *stored procedures* e *triggers* de tabelas para manter toda a informação presente na base de dados num estado consistente. É portanto necessário testar tanto o correto funcionamento dos métodos e funções da camada da lógica do negócio como o correto comportamento das *stored procedures* e *triggers* de tabelas presentes na camada da base de dados, tentando sempre isolar e testar individualmente cada funcionalidade ou parte de cada algoritmo.

4.2.1 Testes unitários para métodos e funções

Como explicado anteriormente, os testes unitários são utilizado para validar métodos ou funções presentes na camada da lógica de negócios, seguindo sempre um padrão comum:

1. São conjuntos muito simples de código
2. Validam apenas um aspeto de um método/função alvo
3. Regra geral é necessária a utilização de um script de inicialização pré-teste que permita
    - Limpar toda a informação da base de dados
    - Preencher tabelas alvo com informação necessária no decorrer do teste
4. Métodos ou funções alvo são executadas de modo a validar os resultados da execução
5. O processo de validação passa sempre por verificar os resultados devolvidos pela chamadas a um método ou função e/ou verificar alterações a nível da base de dados (inserção, remoção ou alteração de informação)

Na Figura 9 é apresentado um exemplo de um teste unitário a uma função designada por “DevolveTaxaResolucaoProcessosEntradosJuizoAreaCrime”.

```
protected Guid _processoID;

/// <summary>
/// Inicialização do teste. Toda a informação é apagada da base de dados sendo, de
/// seguida, gerado um processo novo
///</summary>

[TestInitialize()]
public void MyTestInitialize()
{
       using (TestesProcessosEntity entidade = new TestesProcessosEntity())
              entidade.CleanAllData();

       using (MJCVModeloProcesso entidade = new MJCVModeloProcesso())
              _processoID =
              TestesFramework.Processo.CriaBusinessProcesso(entidade).ProcessoID;
}


/// <summary>
///A test for DevolveTaxaResolucaoProcessosEntradosJuizoAreaCrime
///</summary>
[TestMethod()]
public void DevolveTaxaResolucaoProcessosEntrados_Test()
{
       #region #### Inserção de informação na base de dados ####
       using (MJCVModeloProcesso entidade = new MJCVModeloProcesso())
       {
              MJCVInterfaces.BusinessObjects.Processo processo =
              entidade.Processo.Single();

              processo.ProcessoActivo = false;
              entidade.SaveChanges();

              TestesFramework.Processo.CriaProcessoFaseTipoProcesso(entidade, 1, 3,
              _processoID);

              TestesFramework.Processo.CriaProcessoJuizo(entidade, _processoID,
              TestesFramework.Utilitarios.AmaPessoaID,
              TestesFramework.Utilitarios.PrimeiroJuizoID);
       }
       #endregion ####

       #region #### Evocação de uma função alvo ####
       DateTime DataInicio = DateTime.Now.AddDays(-1);
       DateTime DataFim = DateTime.Now;
       List<MJCVCommon.PublicObjects.Estatistica> estatistica =
       MJCVInterfaces.BusinessObjects.Estatistica.DevolveTaxaResolucaoProcessosEntra
       dosJuizoAreaCrime(DataInicio, DataFim);
       #endregion ####

       #region #### Avaliação de resultados ####
       Assert.AreEqual(100, estatistica.Single(i => i.Nome == "1.º JCrime
       Praia").NumeroOcorrenciasDecimal);
       #endregion ####
}
```

Figura 9 – Exemplo de teste unitário a uma função

4.2.2 Testes unitários a *triggers* e *stored procedures*

A base de dados utilizada pelo SIJ Cabo-Verdiana depende fortemente de *stored procedures* e *trigger* sobre tabelas, sendo portanto imperativo assegurar o correto funcionamento de ambos os elementos. Este correto funcionamento é assegurado através da utilização de testes unitários, como o explicado anteriormente. A grande diferença para com os testes unitários da camada da lógica de negócio reside nos elementos alvo dos testes. No caso da camada da base de dados o objetivo não é validar o código de funções ou métodos mas sim validar a execução de *stored procedures* e *trigger sobre tabelas*. Formal e conceptualmente estes testes são semelhantes aos testes unitários apresentados na secção anterior. No entanto a diferença contextual de realizar testes ao nível de uma base de dados é a razão principal que levou à separação dos dois conceitos de testes unitários.

Todos os testes unitários a *trigger sobre tabelas* ou *stored procedures* seguem um padrão comum:

1. São blocos de códigos bastante simples
2. Testam apenas aspetos singulares de *stored procedures* ou *trigger sobre tabelas*
3. Regra geral é necessária a utilização de um script de inicialização pré-teste que permita
    - Limpar toda a informação da base de dados
    - Preencher tabelas alvo com informação necessária no decorrer do teste
4. São executadas *stored procedures* ou é provocada a execução de *trigger sobre tabelas* (geralmente através de operações de inserção/alteração/eliminação de dados de tabelas alvo) para validar o seu comportamento
5. O processo de validação pode passar por validar ou os resultados devolvidos por uma *stored procedure* ou verificar alterações a nível da base de dados (inserção, remoção ou alteração de informação) no caso dos testes *trigger sobre tabelas*.

Na Figura 10 é apresentado um exemplo de um teste unitário a uma stored procedure designada por “mjcv_TarefasCountByUser”.

```
/// <summary>
/// Toda a informação é eliminada da base de dados
/// </summary>
[TestInitialize()]
public void MyTestInitialize()
{
	using (MJCVModeloProcessos entidade = new MJCVModeloProcessos ())
		entidade.CleanAllData();
}

/// <summary>
/// Teste à stored procedure mjcv_TarefasCountByUser
/// </summary>
 [TestMethod()]
public void mjcv_TarefasCountByUser_Test()
{
	using (MJCVModeloProcessos entidade = new MJCVModeloProcessos ())
	{
		#region ### Inserção de informação na base de dados ###

		Utilitarios.Tarefas.CriaTarefaPessoa(entidade,
		TestesFramework.Utilitarios.SmpPessoaID,
		MJCVCommon.Enums.TipoTarefa.Misc.ToString(), null, DateTime.Now,
		(int)MJCVCommon.Enums.TipoTarefa.Misc);

		#endregion

		Guid userID = TestesFramework.Utilitarios.SmpUserID

		#region ### Evocação da stored procedure alvo ###

		int? numeroTarefas = entidade.mjcv_TarefasCountByUser(userID,
		Guid.Empty, DateTime.Now.AddYears(-10), DateTime.Now.AddYears(10),
		(int)MJCVCommon.Enums.TipoTarefa.Misc, "7").Single();

		#endregion

		#region ### Validação de resultados ###

		Assert.AreEqual(1, numeroTarefas);

		#endregion

	}
}
```

Figura 10 – Exemplo de teste unitário a uma stored procedure

4.2.3 Vantagens na utilização de testes unitários

Neste momento a camada da lógica de negócios e a camada da base de dados assentam sobre milhares de funções e centenas de *stored procedures* e *trigger sobre tabelas*. É portanto imperativo dispor de um mecanismo que assegure o correto funcionamento de todos estes elementos, pelo que os testes unitários respondem de modo perfeito a estas necessidades.

É relativamente fácil medir o impacto de uma alteração simples numa linha de código da aplicação sendo igualmente fácil encontrar potenciais problemas quando alterações simples são realizadas no *software*.

4.3 Testes a *workflows*

O objetivo do sistema de *workflows* é gerir todos os procedimentos envolvidos em processos judiciais. Este sistema de *workflows* foi implementado recorrendo à tecnologia "*Windows Workflow Foundation*" [48] a qual assenta sobre a *framework* "*.NET*" e uma interface de programação de aplicações (API). Esta *API* possibilita a criação de *workflows* que permitem simular todas as fases e transações possíveis num processo judicial. Estes *workflows* são processos de longa duração que assentam sobre serviços do *Windows* (utilizando a *framework Windows Communication Foundation* [49]) os quais permitem criar instâncias de *workflows* modelo e também a comunicação com instâncias previamente criadas [50]. Cada instância de um *workflow* representa um único processo judicial numa fase específica.

4.3.1 O padrão de testes a *workflows*

A complexidade da camada do sistema de gestão de *workflows* é superior à complexidade das camadas imediatamente anteriores (camada da lógica de negócios e camada da base de dados). Este fator conduziu a uma necessidade para desenvolver testes de *software* mais complexos que permitam garantir o correto funcionamento do sistema de gestão de *workflows*.

É feita a distinção entre os testes do sistema de gestão de *workflows* e testes unitários, referidos nas secções anteriores, pelo facto de o código gerado para os testes a *workflows* ser muito mais complexo quando comparado com o código de um teste unitário.

Os testes aos *workflows* seguem um padrão comum que os permite distinguir dos testes unitários referidos anteriormente. Todos os testes ao sistema de gestão de *workflows*:

1. Testam apenas um pequeno aspecto de um *workflow* alvo
2. Regra geral é necessária a utilização de um script de inicialização pré-teste que permita:
    - Remover toda a informação da base de dados
    - Preencher tabelas alvo com a informação necessário no decorrer dos testes
3. Criam uma instância do *workflow* a ser testado
4. Alteram o estado da instância criada para um estado alvo (se necessário)

5. Executam um evento alvo (dentro de um determinado estado) para o testarem
6. Verificam o resultado da evocação de um evento através de:
    - Possíveis alterações no estado ativo de um *workflow* (se aplicável)
    - Criação/Alteração/eliminação de informação em tabelas alvo na base de dados (se aplicável)
    - Verificação dos objetos devolvidos pela evocação (se aplicável)

Os testes ao sistema de gestão de *workflows* têm levantado algumas dificuldades que tiveram de ser ultrapassadas pela equipa de testes de *software* do SIJ. Como exemplo temos o facto de um dos modelos de *workflow* desenvolvidos, onde foram modelados todos os procedimentos de processos civis de natureza ordinária, ter mais de trinta e dois estados distintos sendo que cada estado poderá conter outros sub-estados com também dezenas de possíveis eventos ou ações.

Numa situação normal, como o decorrer de um processo civil ordinário, para se alcançar uma das fases finais seria necessário percorrer, pelo menos, o caminho mais curto entre o ponto em que a instância do *workflow* é criada e a fase desejada (seguindo-se a lista de trajetos permitidos aquando do desenvolvimento do modelo do *workflow*). Simular tal comportamento, aquando do desenvolvimento dos testes ao sistema de gestão de *workflows*, levantaria um leque alargado de desafios. Elementos como o tempo necessário à execução de cada teste, o seu nível de complexidade e eventuais falhas na linha de execução seriam muito complexos de analisar de modo a compreender o motivo dos eventuais problemas. Devido a estes fatores os testes ao sistema de gestão de *workflows* foram desenvolvidos de modo a ignorar os caminhos criados aquando do desenvolvimento do *workflow*. Para tal é utilizado um mecanismo disponibilizado pela API do sistema de *workflows* (*Windows Workflow Foundation*) que permite, entre outros elementos, alterar o estado de instâncias de *workflows* [51]. Deste modo é possível, depois da instanciação de um *workflow*, definir o estado em que se encontra, ignorando o trajeto mais curto necessário para o alcançar. Assim é possível focar cada teste apenas nos acontecimentos inerentes à execução de um evento dentro de um estado alvo, criando-se testes mais rápidos e fiáveis, evitando-se eventuais problemas associados com transações entre estados. Na Figura 11 é apresentado um exemplo de um teste ao serviço de gestão de *workflows*. Este teste valida o correto funcionamento do evento “DescricaoPeticao”. Esta validação é conduzida tanto ao nível do objeto devolvido pela evocação, como também ao nível de alterações despoletadas na instância próprio workflow.

```
/// <summary>
/// Toda a informação é eliminada da base de dados. É ainda conduzido um
/// pré-preenchimento da base de dados.
/// </summary>
[TestInitialize()]
public void MyTestInitialize()
{
       MJCVInterfaces.BusinessObjects.MJCVModeloProcesso entidade = new
       MJCVInterfaces.BusinessObjects.MJCVModeloProcesso();

       #region ### Eliminação da informação da base de dados ###
       entidade.CleanAllData();
       #endregion ###

       ProcessoCivilOrdinarioService.PeticaoInicialClient client = new
       ProcessoCivilOrdinarioService.PeticaoInicialClient();

       #region ### Instanciação de um novo workflow ###
       MJCVCommon.PublicObjects.BaseReturnValue resultado =
       client.EntradaPeticao(TestesFramework.Processo.CriaPeticaoInicial(TestesFrame
       work.Utilitarios.RafUserID, TestesFramework.Utilitarios.TribunalID));
       #endregion ###

       WorkflowinstanceID = resultado.WorkflowInstanceId;

}

/// <summary>
/// Teste ao evento “DescricaoPeticao”
/// </summary>
[TestMethod]
public void evtDescricaoPeticaoInicialTestMethod()
{
       MJCVModeloProcesso entidade = new MJCVModeloProcesso();

       #region ### Teste ao evento “DescricaoPeticao” ###
       PeticaoInicialClient client = new PeticaoInicialClient();
       TestesFramework.Workflow.SetContextDoCliente(client.InnerChannel,
       WorkflowinstanceID);

       MJCVCommon.PublicObjects.BaseReturnValue brv =
       client.DescricaoPeticao(CriaPeticaoInicialExplicada(entidade,
       WorkflowinstanceID));
       #endregion ###

       #region ### Verificação de resultados ###
       Assert.IsTrue(brv.ReturnValue == TipoReturnValue.Success);

       AlteraEstadoProcessoClient clientFase = new AlteraEstadoProcessoClient();
       string fase = clientFase.FluxoEmEstado(brv.WorkflowInstanceId,
       "DecisaoSobrePeticaoInicial");
       Assert.AreEqual(fase, "DecisaoSobrePeticaoInicial");
       #endregion ###

}
```

Figura 11 – Exemplo de teste ao serviço de gestão de workflows

4.3.2 Vantagens na utilização de *workflows*

Os testes a *workflows* permitem assegurar o correto funcionamento da camada do sistema de gestão de *workflows*. Com esta metodologia de testes é possível garantir que, para cada *workflow* desenvolvido, todas as transações entre estados funcionam conforme o esperado e que todos os eventos em cada estado desempenham a função que lhes foi incumbida.

Esta metodologia permite também avaliar se a camada do sistema de gestão de *workflows*, camada da lógica de negócio e camada da base de dados funcionam em conjunto dentro dos padrões considerados como normais.

Nesta fase é importante salientar a diferença relativamente ao tipo de avaliação conseguida a partir de testes unitários. Nos testes a *workflows* o foco de interesse tem um espetro mais abrangente, não interessando avaliações elementares sobre o comportamento de métodos ou funções específicas. Neste sentido os testes de *workflows* atuam como testes de integração e testes de componentes, validando, por um lado, o funcionamento de vastos conjuntos de elementos dentro do mesmo módulo e a integração entre diferentes módulos por outro.

4.4 Testes a interfaces *web*

As interfaces *web* representam a porção visível do SIJ e é o único elemento que os utilizadores utilizam para interagir com o *SIJ*. Esta é também a camada que, direta ou indiretamente, interage com todas as outras (camada de gestão de *workflows*, camada da lógica do negócio e camada da base de dados).

Os testes desenvolvidos para validar o comportamento desta camada representam a metodologia de testes mais complexa que foi necessário implementar até ao momento da escrita desta dissertação. Esta metodologia requer um elevado nível de coordenação entre a informação que necessita de ser gerada e colocada na base de dados da aplicação, a instanciação de *workflows* e a atribuição de permissões a utilizadores de modo a permitir o acesso às interfaces a serem testadas.

Para desenvolver testes de interface a equipa de testes utilizada um conjunto de ferramentas que permitem registar interações de utilizador (cliques de rato e teclado) no *browser* e gravar essa informação para que possa ser repetida em qualquer ponto futuro [52]. Os testes de interface são conduzidos utilizando um conjunto de utilizadores previamente registados ao nível da base de dados. Estes utilizadores foram criados e associados à aplicação de modo a que os seus logins

pudessem ser utilizados para aceder às interfaces do *SIJ* de modo a conduzir tarefas específicas ao nível das mesmas.

O objetivo base dos testes de interface passa por simular todas as possíveis ações que um utilizador pode realizar nas interfaces *web*, separando essas respetivas ações em grupos de testes distintos, possibilitando a sua reprodução em qualquer ponto temporal. O desenvolvimento de testes de interfaces abrange duas fases distintas, as quais são importantes de salientar, sendo elas a fase de gravação e a fase de execução.

4.4.1 Gravação de testes a interfaces *web*

O conjunto de ações necessárias à gravação de testes de interfaces é bastante semelhante entre todos os testes desta categoria, formando um padrão distinto. Aquando da sua gravação é sempre necessário:

1. A utilização de um script de inicialização pré-teste que permita:
    - Eliminar toda a informação da base de dados
    - Preencher tabelas alvo com a informação necessária no decorrer da gravação do teste (se aplicável)
    - Instanciar os *workflows* que sustentam as respetivas interfaces *web* (se aplicável)[1]
    - Alterar a fase em que o *workflow* se encontra para uma fase alvo
    - Adicionar permissões a um utilizador alvo (se aplicável)[2]
2. Executar o script de inicialização
3. Gravar uma sessão de interação com o *browser* [3]
4. Gravar todas as validações necessárias ao nível das interfaces *web*[4]

[1] As interfaces *web* são os mecanismos utilizados pelos utilizadores para, de entre outras operações, interagir com instâncias de *workflows* do SIJ. Deste modo para testar uma interface específica que permite interagir com um estado de um *workflow* é necessário, numa primeira fase, instanciar o *workflow* respetivo e colocá-lo na fase necessária á gravação do teste.
[2] O mecanismo de permissões do SIJ permite que utilizadores específicos acedam a processos judiciais concretos. Para tal, nos casos dos testes de interface que tentam validar o funcionamento das interfaces que permitem interagir com processos judiciais, é apenas necessário adicionar permissões de visualização aos utilizadores utilizados no decorrer dos testes.
[3] No caso do Sistema de Informa da Justiça são utilizados os mecanismos de testes de *software* disponibilizados pelo *Visual Studio* (versão 2010) que permitem gravar cliques de rato e teclado, designados como "*Coded UI Test*" [84].
[4] São utilizadas as ferramentas referidas no ponto anterior [84]. Estas ferramentas de testes permitem, para além de gravar sessões de interação com o *browser*, validar diretamente a renderização de interfaces *web* no *browser* do cliente/utilizadores da aplicação [85].

5. Adicionar validações adicionais via código (se aplicável)

A ferramenta de testes utilizada permite gerar ficheiros de testes contendo todo o código necessário de modo a recriar todos as ações geradas na fase de gravação do teste de interface. Para tal é apenas necessário assegurar a execução do script de inicialização antes da execução da sessão de interação com o browser [53]. Na Figura 12 é apresentado um exemplo de um script inicial que permite preparar o SIJ para a gravação de um teste de carga.

```
[TestMethod]
public void ConstitAssistFaseAguardJulgTestMethod()
{
        #region ### Eminiação de toda a informação da base de dados ###
        Utils.CleanDB();
        #endregion

        MJCVModeloProcesso entidade = new MJCVModeloProcesso();
        Guid despachoWFID;
        MJCVCommon.PublicObjects.BaseReturnValue brv;

        #region ### cria instância de um workflow para processos penais ###
        NotificacaoCrimeClient client = new NotificacaoCrimeClient();
        brv =
        client.NotificaCrime(TestesFramework.Processo.CriaPublicAutoDenuncia(TestesFr
        amework.Utilitarios.RafUserID, TestesFramework.Utilitarios.SecretariaMPID));

        //altera estado do WF
        AlteraEstadoProcessoClient alteraEstado = new AlteraEstadoProcessoClient();
        alteraEstado.SetState(brv.WorkflowInstanceId, "Instrucao");

        #endregion

        #region ### Inserção de informação na base de dados  ###
        TestesFramework.Processo.CriaProcessoFaseTipoProcesso(entidade, 1, 3,
        brv.IdProcesso);

        Guid processoMPID = entidade.ProcessoMP.Select(i => i.ProcessoMPID).Single();

        TestesFramework.Processo.CriaProcessoMagistradoMP(entidade, processoMPID,
        TestesFramework.Utilitarios.UmfPessoaID);

        TestesFramework.Processo.CriaProcessoJuizoSimples(entidade, brv.IdProcesso,
        TestesFramework.Utilitarios.PrimeiroJuizoID);

        int despachoID = TestesFramework.Processo.CriaDespacho(entidade,
        TestesFramework.Utilitarios.AmaUserID, Guid.Empty, brv.IdProcesso, "03-05-
        2014 | 10:36");

        entidade.SaveChanges();

        #endregion
}
```

Figura 12 – script de preparação do SIJ para um teste de interface

4.4.2 Execução de testes a interfaces *web*

O processo de execução de testes a interfaces *web* é o passo em que todas as ações gravadas ao nível da interface *web* serão executadas. Os scripts de inicialização associados a cada teste de interface são executados de modo a recriar o estado inicial necessário à correta execução do teste,

uma instância do *browser* é iniciada, o conjunto de ações previamente gravadas é executado e, por último, são realizadas as validações finais pós execução. Na Figura 13 é apresentado o resultado final do processo de automação do teste de interface. Ao conjunto de instruções, apresentadas na Figura 12, é acrescentado o restante código que permite executar, de forma automática, os cliques e validações gravados ao nível das interfaces web.

```
[TestMethod]
public void ConstitAssistFaseAguardJulgTestMethod()
{
        #region ### Eminiação de toda a informação da base de dados ###
        ……
        #endregion

        #region ### cria instância de um workflow para processos penais ###
        ……
        #endregion

        #region ### Inserção de informação na base de dados  ###
        ……
        #endregion


        #region ### Execução de todos os cliques de interface gravados

        BrowserWindow browser =
        BrowserWindow.Launch(System.Configuration.ConfigurationManager.AppSettings["U
        RLTestInterface"]);
        try
        {
                this.UIMap.ConstituicaoAssistenteUpdate();
                this.UIMap.VerificacoesFinais();
        }
        catch (Exception ex)
        {
                browser.Close(); if (browser.Process != null) browser.Process.Kill();
                Assert.Fail(ex.ToString());
        }
        browser.Close(); if (browser.Process != null) browser.Process.Kill();

        #endregion
}
```

Figura 13 – Teste de interface totalmente automatizado

4.4.3 Vantagens dos testes a interfaces *web*

Os testes a interfaces *web* permitem assegurar o correto funcionamento da camada de apresentação do SIJ e também garantir que todos os diferentes blocos que constituem o *SIJ* funcionam em conjunto de forma correta. Os testes a interfaces *web* podem ser entendidos como

testes de sistema e de entre todas as metodologias utilizadas eles acabam por ser a metodologia de testes mais valiosa. É fundamental assegurar o correto funcionamento de um método ou função mas é ainda mais importante assegurar o correto funcionamento do sistema como um todo, principalmente quando diferentes métodos funcionam em conjunto como resultado de ações reais de utilização.

É também importante garantir o correto funcionamento do *SIJ* na eventualidade de alterações futuras ao *software*. Neste sentido, os testes a interfaces *web* (juntamente com todas as metodologias descritas anteriormente) desempenham o papel de elementos de validação do *software*, atuando antes dos testes de aceitação, garantindo a deteção de problemas com eventuais alterações no *SIJ* e permitindo avaliar a aptidão do *software* para entrar em produção.

Testar manualmente interfaces *web* é possível no entanto é uma tarefa propicia a erros e que têm enormes implicações a nível temporal. Os testes automatizados a interfaces *web* consomem tempo à equipa de testes apenas na fase de criação. Após a sua criação todo o processo de validar uma interface encontra-se automatizado, libertando a equipa de testes de tarefas futuras de validações manuais de interfaces e permitindo que se foque noutras tarefas criticas associadas ao ciclo de desenvolvimento do *software*.

4.5 Testes de evocação de exceções

Até ao momento todas as metodologias apresentadas permitem garantir que a aplicação, para os cenários testados, funciona de forma correta. É igualmente importante perceber o comportamento da aplicação em situações que estão fora do controlo de um fluxo normal de execução como, por exemplo, o surgimento de exceções/erros na *stack* de execução da aplicação. Aqui o interesse passa por determinar o comportamento da aplicação quando uma situação anormal é despoletada por erros internos. É essencial, para uma qualquer aplicação, garantir que consegue recuperar do surgimento de erros internos sem que ocorram perdas de informação.

De modo a garantir tais requisitos, foi desenvolvida uma metodologia de testes que tem por objetivo validar o comportamento da aplicação em cenários de erros internos e garantir a capacidade da mesma em restabelecer o seu normal funcionamento sem predas de informação. Esta metodologia foi implementada tanto ao nível do sistema de gestão de *workflows* como ao nível da aplicação *web*. Ela tenta validar todo o processo de interceção e tratamento de exceções internas bem como validar a identificação de informação em falta ou inconsistente antes da submissão de formulários *web*.

A Figura 14 descreve o processo dos testes de evocação de exceções tanto ao nível das interfaces *web* como ao nível do sistema de gestão de *workflows*.

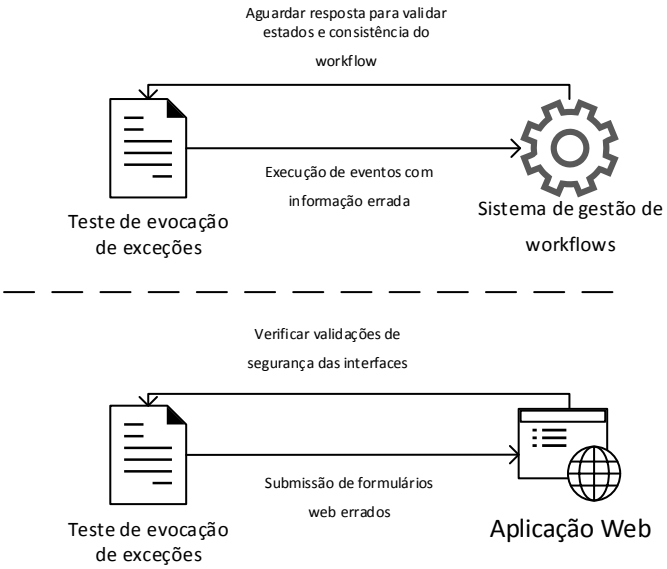

Figura 14 – Testes de evocação de exceções

Os testes de evocação de exceções são pequenas variações dos testes a interfaces *web* e testes de *workflows*. A grande diferença para com estes reside no facto de os testes de evocação de exceções não terem como objetivo assegurar o correto funcionamento dos elementos a serem testados mas sim forçar situações anómalas e avaliar o comportamento da aplicação.

4.5.1 Camada de *workflows*

Devido ao modo de funcionamento da camada de gestão de *workflows* ,caso alguma exceção seja levantada ao nível do código de uma instância e caso essa mesma exceção não seja tratada de forma conveniente então a instância será terminada pelo serviço de gestão, não sendo possível aceder-lhe para operações futuras [54] . Isto significa que toda a informação associada à instância de um *workflow* (em alguns casos referentes a instâncias de longa execução as quais poderão estar ativas durantes vários anos) será perdida na eventualidade de não ser tomada nenhuma medida de contingência. Como uma instância de um *workflow* representa um processo judicial único no sistema então a perda dessa instância representa a perda de toda a informação sobre o estado do processo judicial, possíveis transações e possíveis tomadas de decisão relativamente à instância terminada. De modo a evitar esta situação, todos os eventos dentro de todos os estados, para qualquer *workflow*, estão sempre envoltos num mecanismo interno à API do *Windows Workflow* Foundation designado por "*FaultHandler*" [54], [55]. Este elemento é em todo semelhante ao bloco

de "*Exception Handling*" presente numa grande generalidade de linguagens de programação [56], [57]. Deste modo, se alguma exceção for lançada dentro de uma instância de um *workflow*, então este mecanismo será responsável por capturar a exceção, evitando assim a sua propagação, e também pelo tratamento da mesma. Deste modo é possível evitar a perda de uma instância de um *workflow* devido à propagação de uma exceção. O tratamento de uma exceção, por parte do mecanismo "*FaultHandler*", envolve, no caso do SIJ, o registo/log da mesma ao nível da base de dados e permite ainda evitar o término ou mudanças de estado ao nível da instância do *workflow*.

Ao nível da camada do sistema de gestão de *workflows* foram desenvolvidos testes que permitem, para cada *workflow* modelado, validar o comportamento dos eventos/métodos de todos os sempre que é submetida informação errada ou inconsistente. Através deste método é possível forçar o aparecimento de exceções ao nível de instâncias de *workflows* (previamente criadas para servirem como elementos a serem testados). O objetivo principal desta metodologia passa sempre por assegurar o correto funcionamento do mecanismo "*FaultHandler*", garantir que cada exceção é registada ao nível da base de dados e, finalmente, assegurar que a instância do *workflow* a ser testada nunca é terminada devido a qualquer tipo de bugs.

A possibilidade de perda de informação sobre uma instância de um *workflow* é encarada, tanto pelas equipas de desenvolvimento e testes como pelos restantes *stakeholders*, como uma circunstância completamente inaceitável. A possibilidade de perda de informação devido ao envio de argumentos errados para um qualquer serviço de rede pode ser encarada como, por exemplo, um ataque do tipo "*denial of service*", onde um utilizador malicioso, assumindo que tem permissão de acesso à rede protegida dos servidores, pode causar a perda de informação de valor incalculável.

Ambas as situações (exceções provocadas pelo envio de argumentos errados e por envio deliberado de informação inconsistente) são avaliadas por esta tipologia de testes de injeção de exceções.

4.5.2 Camada de apresentação

A camada de apresentação foi desenvolvida com a capacidade de identificar potenciais problemas e notificar os utilizadores sobre informação inconsistente e/ou em falta antes da submissão de qualquer formulário *web*. Através deste método é possível evitar potenciais problemas dentro das(ou ataques às) camadas subsequentes da aplicação.

Ao nível da camada de apresentação foram desenvolvidos um conjunto de variantes de testes de interface *web* através dos quais se se tenta simular a utilização de interfaces *web* de forma

incorreta. Estes testes podem ainda ser divididos em duas categorias, testes de validação de comportamentos erróneos e testes de validação de comportamento malicioso. A simulação de testes de exceção de comportamento erróneo é conduzida através da submissão de formulários *web* com informação errada ou inconsistente. O objetivo passa por garantir que cada interface é capaz de identificar os erros dos utilizadores antes da submissão de formulários *web*.

Para simular exceções originadas por comportamento malicioso são simuladas adulterações de cabeçalhos de HTTP enviados durante os métodos de POST das páginas *web* e ainda é realizada a simulação de acesso indevido a processos judiciais ou áreas restritas através da adulteração dos parâmetros *QueryStrings* presentes nas *URLs* da aplicação.

4.5.3 Vantagens dos testes de evocação de exceções

Perceber o comportamento de uma aplicação em condições anormais pode permitir a descoberta de comportamentos perigosos por parte da aplicação, bem como ameaças ou quebras de segurança, fatores estes que podem conduzir à perda ou divulgação indevida de informação.

Assegurar o correto feedback visual pode auxiliar no nível de produtividade de um utilizador. Para além disso, o facto de o processo de sanitização de informatização ocorrer antes da submissão real dos dados (tanto para o servidor *web* como do servidor *web* para o serviço de gestão de *workflows*) assegura o confinamento de dados errados. Este fator permite libertar serviços e/ou aplicações, garantido assim que estes lidam com informação menos propicia a erros, o que auxilia a escalabilidade do sistema como um todo.

4.6 Testes de carga

De modo a medir e entender a capacidade de resposta a picos de utilização ao SIJ foi utilizado um conjunto de ferramentas designadas por ferramentas de testes de performance *web* (*Web Performance Test Tool*) [58] e ferramenta de testes de carga (*Load Test Tool*) [59] de forma a conduzir uma avaliação de performances sobre o *SIJ*.

Num teste de performance *web* é possível gravar uma sessão de interação com um *browser*, onde todos os pedidos que viajam entre um cliente (browser) e servidor (máquina onde a aplicação *web* está alojada) são gravados, sendo este um processo semelhante à gravação de um teste de interface *web*. Depois de gravado todo o processo de interação entre o cliente e o servidor todos

os registos poderão ser armazenados e associados a um teste designado por "*Web Performance Test*". Este teste poderá ser executado de modo a recriar a sessão de navegação no *browser*.

Num teste de carga (*load test*), um conjunto de utilizadores virtuais podem ser associados a um ou mais testes de performance *web* gravados previamente. Cada utilizador virtual irá realizar um conjunto de ações gravadas em um ou mais testes de performance *web*. O objetivo passa por ter um conjunto de utilizadores virtuais a executar sessões de navegação (*web performance tests*), previamente gravadas. No caso do SIJ estas sessões têm uma duração específica e permitem medir os tempos de resposta da aplicação alvo à medida que o número de utilizadores virtuais vai sendo incrementado ao longo do tempo [60].

O SIJ foi desenvolvido com o objetivo de ser utilizado por todos os funcionários dos tribunais Cabo-Verdianos. Neste sentido os testes de performance e carga desenvolvidos tentam simular a utilização do *SIJ* por parte dos funcionários da justiça, tanto a um nível comportamental dentro da aplicação como ao nível do número de utilizadores que acedem à aplicação em simultâneo. A simulação do comportamento de utilizadores dentro da aplicação envolve tarefas como a utilização de formulários *web* para o envio de elementos processuais como requerimentos, despachos, autos iniciais, *download* de documentos, navegação pelos diferentes menus da aplicação, entre outros. O conjunto das operações referidas anteriormente têm a capacidade de permitir simular uma utilização intensiva tanto da aplicação *web* como do serviço de gestão de *workflows*, sendo possível obter um cenário de carga de utilização realista.

Atualmente a taxa de simultaneidade de utilização do SIJ, por parte dos seus utilizadores, está estimada em 100 utilizadores em simultâneo.

De modo a testar a escalabilidade do SIJ fori desenvolvido um conjunto de experiências de modo a avaliar o comportamento da aplicação quando instalada em máquinas com diferentes características de *hardware*. O objetivo desta experiência é compreender se eventuais melhorias ao nível do *hardware* podem conduzir a melhorias de desempenho do *SIJ* e, caso se detetem melhorias de performance, o quão significativas estas poderão ser.

A escalabilidade da arquitetura do SIJ foi também um dos elementos avaliados no decorrer desta experiência. Sendo que este sistema permite desacoplar partes dos seus elementos constituintes por diferentes máquinas foi conduzido um estudo de modo a avaliar as eventuais vantagens que podem ser retiradas deste processo de separação dos elementos constituintes do SIJ.

A equipa de testes do *SIJ* configurou um conjunto de testes de carga onde diferentes números de utilizadores virtuais executam, de forma aleatória, testes de performance *web*. Cada utilizador virtual executa apenas um teste de cada vez e todos os utilizadores virtuais executam testes de forma concorrente. Foram utilizados cinco testes de performance web onde foram simulados tipos de tarefas que os utilizadores podem desempenhar nas interfaces da aplicação, sendo elas a submissão de Autos de Denúncia; submissão de Petições Iniciais; submissão de Pedidos de Habeas Corpus por detenção ilegal; submissão Despachos de decisão relativos a processos a decorrer na aplicação e navegar pelos menus da aplicação.

As quatro primeiras tarefas permitem simular carga de utilização, por parte dos utilizadores, sobre os três grandes blocos do SIJ (aplicação web, serviço de gestão de *workflows* e camada de acesso a dados). A quinta tarefa apenas permite simular carga de utilização sobre o bloco relativo à aplicação web e camada de acesso a dados. Cada teste de performance web, por utilizador virtual, era escolhido de forma aleatória, tendo a probabilidade de escolha um valor de vinte porcento. A Tabela 1 contém uma descrição detalhada sobre o conjunto de testes de performance web utilizados na condução do presente estudo.

Tabela 1 – Testes de performance web, descrições e probabilidades de escolha

Teste de performance web	**Descrição**	**Probabilidade de escolha**
Submissão de Autos de Denúncia	Teste permite realizar o *login* na aplicação com um utilizador, aceder ao menu “Secretaria MP”, aceder ao submenu “Auto Inicial”, submeter um Auto de denúncia utilizando o formulário web correspondente. O teste é finalizado com o processo de *logout* do utilizador.	20%
Submissão de Petições Iniciais	Teste permite realizar o *login* na aplicação com um utilizador, aceder ao menu “Secretaria J.”, aceder ao submenu “Petição Inicial”, submeter uma Petição Inicial utilizando o formulário web correspondente. O teste é finalizado com o processo de *logout* do utilizador.	20%
Submissão de Pedidos de Habeas Corpus	Teste permite realizar o *login* na aplicação com um utilizador, aceder ao menu “Secretaria MP”, aceder ao submenu “Pedido de Habeas Corpus”, submeter um Pedido de Habeas Corpus por detenção ilegal utilizando o formulário web correspondente. O teste é finalizado com o processo de *logout* do utilizador.	20%

Submissão Despachos	Teste permite realizar o *login* na aplicação com um utilizador, aceder ao menu "portefólio", aceder a um processo específico a decorrer na aplicação e submeter um despacho de decisão relativo ao processo acedido. O teste é finalizado com o processo de *logout* do utilizador.	20%
Navegação pelos menus	Teste permite realizar o *login* na aplicação com um utilizador e navegar pelo conjunto de menus visíveis para o utilizador. O teste é finalizado com o processo de *logout* do utilizador.	20%

Toda a descrição relativa à experiência de carga realizada sobre o SIJ encontra-se devidamente descrita em anexo, na secção designada como "9.1 Testes de Carga". Esta secção contém toda uma análise de performance, coleção e interpretação de resultados obtidos. O objetivo principal deste estudo foi determinar o impacto que a carga de utilização por parte de utilizadores, arquitetura do SIJ e a capacidade do *hardware* têm sobre o desempenho geral do SIJ.

4.6.1 Conclusões obtidas

Depois da condução dos testes de carga sobre as diferentes configurações de arquitetura e *hardware* e após uma avaliação cuidada dos resultados recolhidos foi possível obter algumas conclusões, sendo enumeradas de seguida:

- Regra geral é possível melhorar a performance do SIJ desacoplando os seus blocos constituintes. Neste sentido as melhores performances em termos de tempos de resposta da aplicação, utilização otimizada de recursos de processamento e memória e capacidade de resposta a carga foram obtidas desacoplando os três blocos constituintes do SIJ.
- A camada de apresentação pode ser considerada como um dos elementos críticos em termos de performance. Esta camada, em situações de utilização intensiva da aplicação, pode limitar a performance geral do SIJ caso não se tire partido das opções de arquitetura. No estudo de carga desenvolvido a solução de equilíbrio em termos de processamento médio e utilização de memória foi obtido quando o número de máquinas que alojavam a aplicação web correspondiam ao dobro do número de máquinas que alojavam o serviço de gestão de *workflows*. Este fator pode ser explicado pelo facto de a aplicação web do SIJ ser o ponto de entrada para a realização de todos os tipos de tarefas. Qualquer ação de utilização que despolete processamento, tanto ao nível do serviço de gestão de workflows

como apenas ao nível da aplicação web, é sempre desencadeada via as interfaces web do SIJ.

- O impacto do SGBD no desempenho geral da aplicação é mínimo em termos das necessidades de processamento e utilização de memória.
- Não foi possível encontrar uma relação direta entre incrementos de memória e melhorias significativas do desempenho geral da aplicação.
- O consumo de memória por parte do serviço de gestão de *workflows* tende a aumentar quando a intensidade de utilização sobre a aplicação aumenta.

4.6.2 Vantagens dos testes de carga

Os testes de carga permitiram à equipa de desenvolvimento do SIJ ter uma noção das reais necessidade de *hardware* por parte da aplicação. Foi ainda possível perceber quais as limitações do *SIJ* em termos de performance tanto ao nível do número de utilizadores concorrentes que é possível suportar bem como relativamente a tarefas limitadoras da performance da aplicação.

Os resultados dos testes de carga, depois de devidamente compilados, podem auxiliar os gestores deste projeto de *software* a planear as melhores opções relativamente às necessidades de *hardware* para ambientes de produção (em termos de número de servidores, espaço de armazenamento, número de *CPUs* por servidor, *etc*…) tanto a medio como a longo prazo.

5 Problemas e resoluções

5.1 O problema dos tempos de execução

As diferentes tipologias de testes apresentadas anteriormente tendem a ter tempos de execução diferentes entre testes de diferentes tipologias.

Tabela 2 – Tempos de testes de software

	Testes Unitários	**Testes a Workflows**	**Testes a interfaces web**
Tempo (segundos)	[1, 2]	[4, 8]	[40, 50]
Nº total de testes	1842	370	1251
Tempo total (segundos)	[1842, 3684]	[1480, 2960]	[50040, 62550]
Tempo total (horas)	[0.51, 1.02]	[0.41, 0.82]	[13.9, 17.4]

Como apresentado na Tabela 2, todos os testes unitários e testes a *workflows* têm tempos de execução total aproximados (entre 24 minutos a 1 hora). Por outro lado, o grupo de testes a interfaces *web* são o grupo de testes com os tempos totais mais longos (entre 14.8 horas e 19.24 horas para executar toda a gama de testes). Estes tempos levantam um problema muito importante. Sendo a execução de testes de *software* uma tarefa sequencial, então para executar todos os testes de *software* implementados seria necessário esperar entre oito a onze horas. Esperar onze horas para obter um relatório do estado da aplicação não é uma opção viável. Era portanto imperativo desenvolver uma estratégia que permitisse reduzir o tempo de execução.

5.2 A primeira solução e respetivos problemas

Inicialmente, foi pensada uma abordagem em que todos os testes de *software* seriam separados por diferentes projetos, de acordo com a natureza a que pertenciam. Todos os testes unitários seriam reunidos num único projeto e o mesmo seria aplicável aos testes a *workflows* e testes de interface. Cada projeto deveria incluir um ficheiro de configuração próprio o qual conteria, entre outros elementos, as definições de acesso a uma versão da base de dados da aplicação. A Figura 15 contém uma representação da organização dos projetos de testes descrita anteriormente.

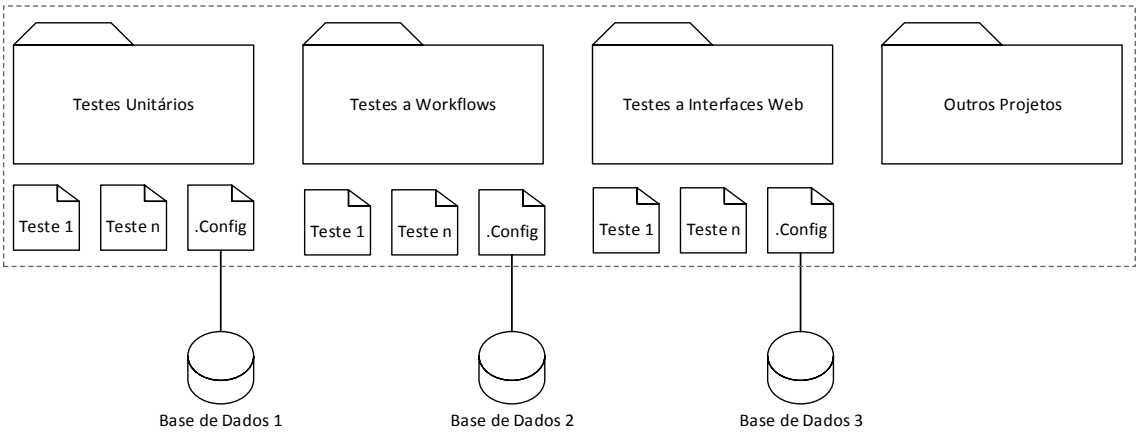

Figura 15 – Estrutura de projetos de testes

Executando todos os testes de cada projeto em máquinas diferentes permitiria paralelizar a execução de testes e reduzir o tempo total de execução do bloco geral de testes da aplicação.

No entanto o processo de paralelização não é dinâmico nem automatizado. Este processo de separação de testes por projeto é realizado manualmente de modo a permitir executar, simultaneamente, testes em máquinas diferentes. Por outro lado o tempo total de execução dos testes de interface rondaria as 9 horas, ao passo que os testes unitários demorariam entre uma a duas horas a serem executados. Poder-se-ia subdividir todos os testes de interface em projetos diferentes no entanto este é um processo moroso e com uma total ausência de dinamismo.

5.3 A presente solução e os respetivos problemas

Considerando que a separação manual dos testes de interface em grupos de teste mais pequenos não é uma solução viável, foi necessário encontrar outras soluções que pudessem responder às necessidades de dinamismo. Portanto, ao invés de se conduzir uma separação manual de testes, foi utilizada uma solução que permite separar os testes de *software* por máquinas diferentes apenas quando existe um pedido de execução (independentemente dos projetos onde os testes estejam alojados).

5.3.1 O controlador e agentes de testes

De modo a conseguir separar dinamicamente a execução de testes foi utilizada uma ferramenta designado por "*Visual Studio Agents*" [61]. Esta ferramenta permite a instalação de um conjunto de componentes designados por "Controlador de Testes" e "Agente de Testes" em cada máquina utilizada para conduzira a execução remota de testes de *software* [62]. Utilizando este *software* é

então possível executar remotamente testes de *software* em máquinas diferentes de forma completamente automatizada e paralela.

O controlador de testes funciona como elemento de ligação entre um ambiente integrado de desenvolvimento (*Visual Studio 2010* no caso do *SIJ*) e um conjunto de agentes de testes. O controlador é o elemento responsável por gerir todos os agentes de testes e, quando é recebido um pedido de execução de um bloco de testes, é da sua responsabilidade gerir o processo de separação/distribuição dos testes pelos agentes de testes disponíveis.

Os agentes de testes são o componente responsável por receber pedidos de execução de testes, executar o bloco de testes recebido, colecionar os resultados da execução e passar os resultados ao controlador.

No caso do *SIJ,* as máquinas onde estão instalados os agentes de testes são as máquinas com a maior carga de processamento devido ao facto de todos os testes serem sempre conduzidos localmente. A Figura 16 pretende demostrar o funcionamento conjunto dos agentes e controlador de testes de *software*.

A utilização de um controlador de testes juntamente com vários agentes, instalados nas diferentes máquinas utilizadas no processo de testes ao *SIJ*, permite que a execução de todo o universo de testes do *SIJ* seja executado de uma forma verdadeiramente paralelizada. Esta arquitetura de testes representa uma grande melhoria no modo como é conduzido o processo de validação da qualidade do *SIJ*. No entanto, mesmo utilizando controladores e agentes de testes, ainda persistem alguns problemas.

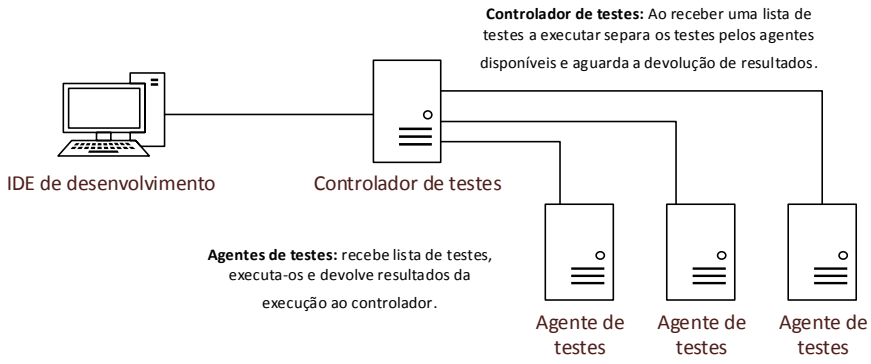

Figura 16 – Controlador e agente de testes

Atualmente o universo de testes do *SIJ* está separado por vários projetos, existindo um ficheiro de configuração único por projeto. Como referido anteriormente, os ficheiros de configuração são utilizados para armazenar, entre outros elementos, os parâmetros de acesso a base de dados utilizadas no processo de execução dos testes de *software*. Mesmo se cada ficheiro de configuração (em cada projeto) possua parâmetros de ligação às bases de dados de testes diferentes de todos os outros projetos, existe sempre a possibilidade de dois ou mais testes de um mesmo projeto serem executados simultaneamente em máquinas diferentes, afetando-se mutuamente, como demonstrado na Figura 17.

De modo a ultrapassar este problema, cada máquina com agentes de testes instalados foi configurada como uma unidade autossuficiente, contendo todos os elementos necessários à execução dos testes do *SIJ*.

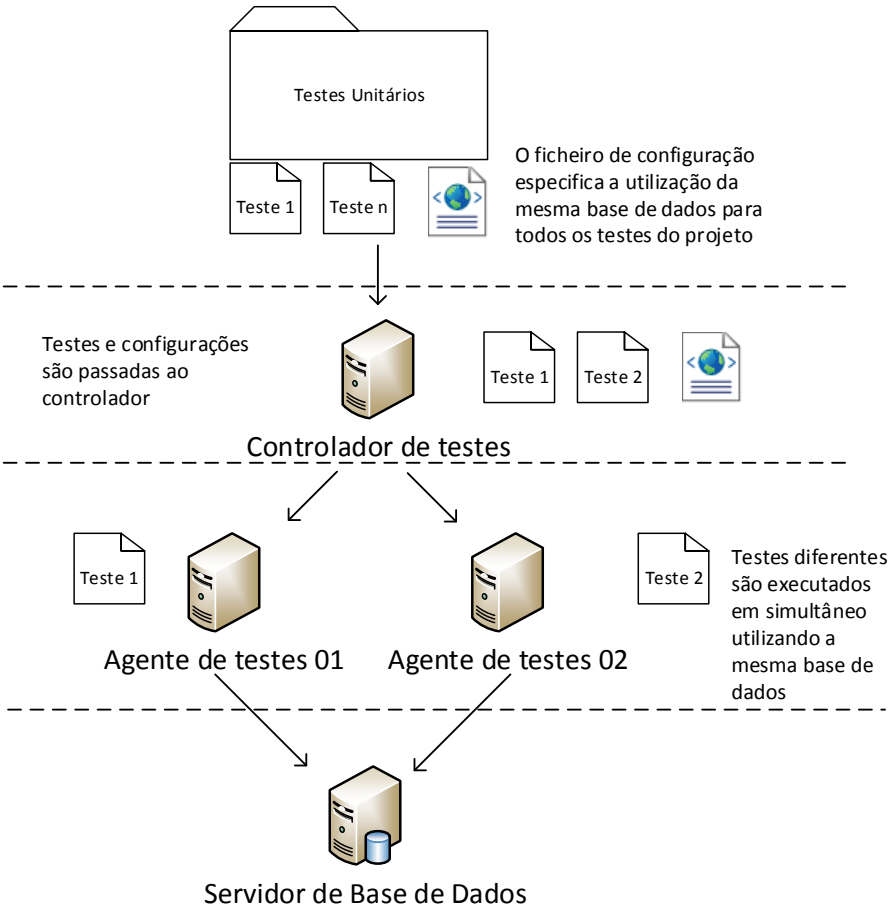

Figura 17 – Execução paralela de testes utilizando o mesmo ficheiro de configuração

5.3.2 *Deploy* local por agente de testes

O *deploy* local, por agente de testes, de todos os elementos necessários ao funcionamento do *SIJ* (servidor web, serviço de *workflows* e base de dados) possibilita que cada agente de testes possa executar todos os testes utilizando apenas recursos locais (base de dados local, *localhost*, *etc*…). Isto significa que se um teste estiver a ser executado numa qualquer máquina "A" então essa máquina necessitará de ter: a aplicação *web* alojada na instância de *IIS* local, o serviço de gestão de *workflows* instalado como um serviço do *Windows* local e uma instância do *SQL Server* com a versão adequada das base de dados da aplicação. Por outro lado, todos os ficheiros de configuração de todos os projetos de testes necessitam de conter as definições de acesso a recursos configuradas para utilizar apenas recursos locais (*localhost* para acesso a aplicação *web*, "(*local*)" ou *".**SQLEXPRESS*" para acesso a base de dados, *etc*…). A Figura 18 representa a estrutura de configuração de testes atual.

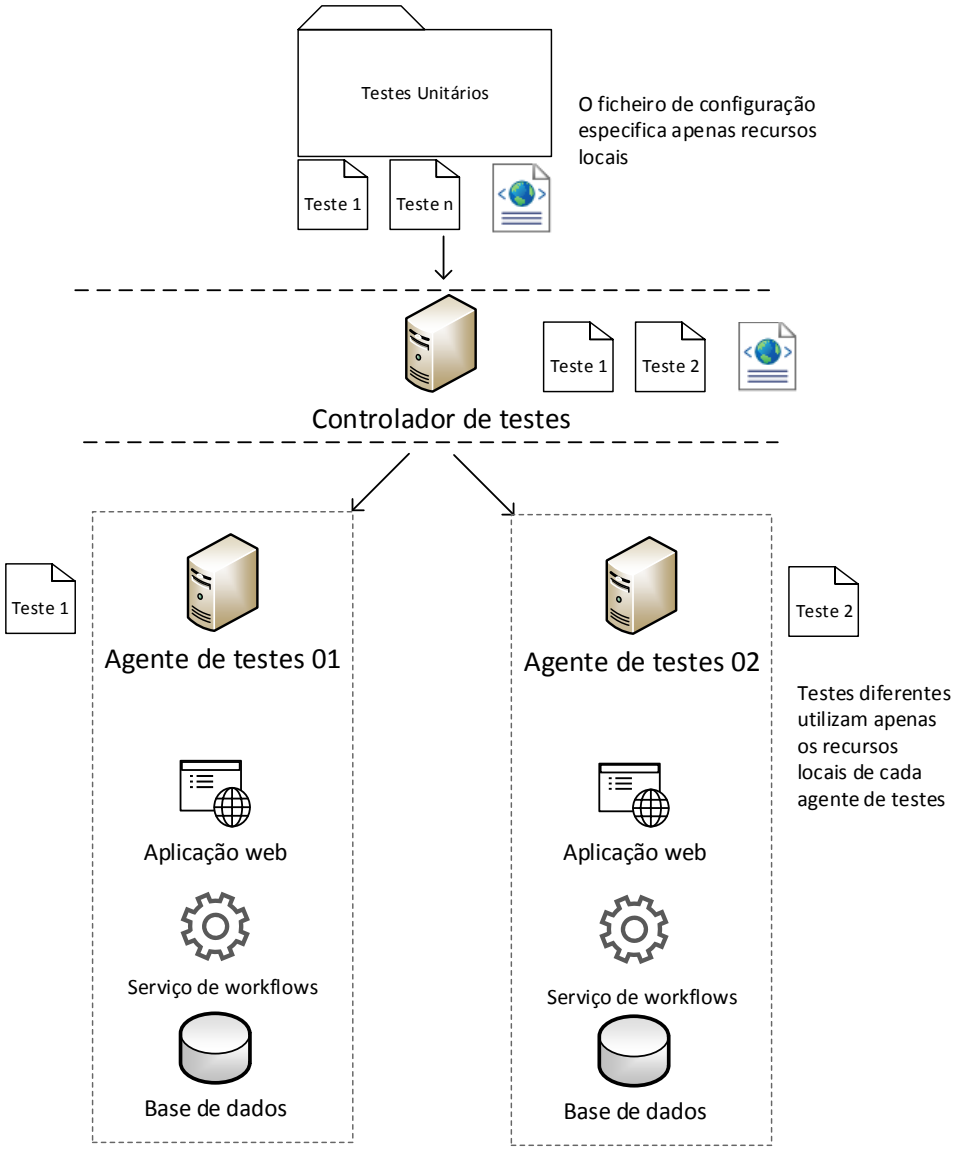

Figura 18 – Configuração de testes atual

Neste momento mesmo que testes diferentes de um mesmo projetos, utilizando as definições do mesmo ficheiro de configuração, estejam a ser executados, simultaneamente, em máquinas diferentes nunca se afetarão visto que apenas irão utilizar os recursos locais da máquina onde estão a ser executados.

5.3.3 A arquitetura de testes e implementação do *software*

Atualmente é utilizado o *toolkit* "*OpenNebula*" como *software* de gestão de máquinas virtuais onde são instalados todos os agentes e controladores de testes, tal como o apresentado na Figura 19.

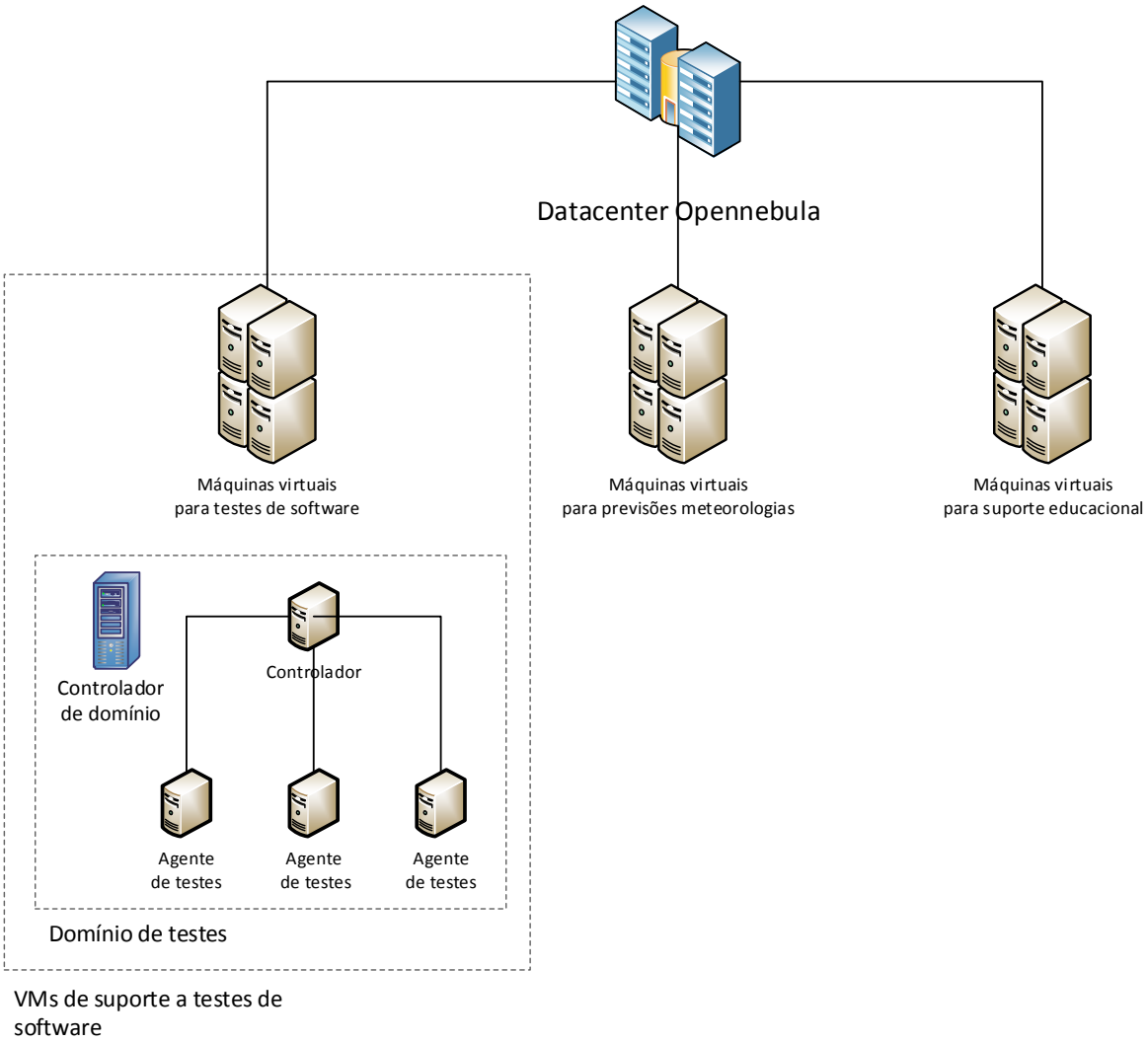

Figura 19 – Opennebula & VMs de suporte a testes de software

Até ao momento da escrita desta dissertação apenas é utilizada uma máquina virtual como controlador de testes e catorze máquinas virtuais como agentes de testes.

A máquina utilizada como controlador de testes não necessita de qualquer tipo de capacidades especiais em termos de *hardware* virtual visto que a sua função principal é apenas gerir agentes de testes. As máquinas utilizadas como agentes de testes são aquelas em que as capacidades de *hardware* virtual podem fazer a diferença em termos de desempenho na execução de testes de *software*.

A equipa de testes do *SIJ* tentou automatizar, tanto quanto possível, o processo de *deploy* de todas as partes do *software* em todos os agentes de testes. Conduzir um *deploy* manual do *software* em todas as máquinas, apesar de possível, seria um processo muito moroso e propício a erros. Um *deploy* completo do *SIJ* numa máquina virtual consiste em publicar a aplicação *web* no *IIS* da respetiva máquina, instalar a versão mais recente do serviço de gestão de *workflows* e realizar ainda um *restore* da uma versão das bases de dados na instância de *SQL Server* do agente de testes.

Todas as máquinas virtuais utilizadas no processo de testes ao *SIJ* estão alojadas dentro de um domínio de rede [63] próprio controlado e administrado pela equipa de testes. Este facto prende-se com a necessidade de simplificar o processo de manutenção e *deploy* do *SIJ* por todas as máquinas do domínio utilizadas como agentes de testes. Deste modo são evitados problemas adicionais relacionados com acesso remoto de serviços, acesso remoto a servidores web (*IIS*), sincronização remota de conteúdos entre servidores web (*IIS*), acesso remoto a instâncias de base de dados, *etc*.

5.3.3.1 *Deploy da aplicação web*

Numa primeira fase, ou utilizando o ambiente integrado de desenvolvimento (*IDE*) de desenvolvimento ou utilizando a plataforma de compilação de código *MSBuild*[5] [64], é conduzida uma compilação e publicação da aplicação *web* para um dos agentes de testes.

De seguida, utilizando a ferramenta *MSDEPLOY* [65], é sincronizado o conteúdo do servidor de IIS entre a primeira máquina, onde foi conduzida a primeira publicação da aplicação *web*, e os restantes agentes de testes. Este processo permite reduzir significativamente o custo de publicação da aplicação *web* em todos os agentes de testes visto evitar publicações individuais da aplicação web por agente de testes.

Este processo de publicação da aplicação *web* está ilustrado na Figura 20

[5] O processo de *deploy* da aplicação web pode ser conduzido tanto através do *IDE* de desenvolvimento como utilizando a plataforma de compilação *MSBuild*. A utilização da plataforma pode ser mais vantajosa pois permite automatizar o processo de *deploy* da aplicação web. Esta ferramenta é apenas executada via linha de comandos e permite conduzir operações de compilação e *deploy* de qualquer projeto do *SIJ*. Neste sentido é possível calendarizar operações sobre qualquer projetos do SIJ.

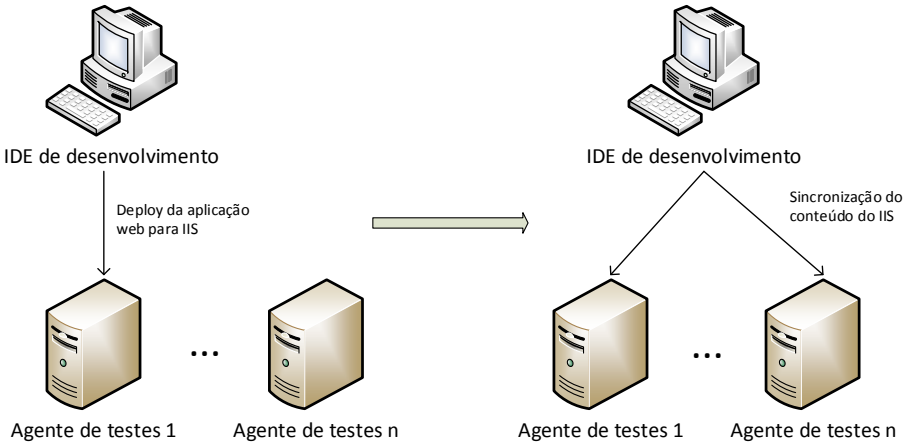

Figura 20 – Deploy da aplicação web

5.3.3.2 *Deploy do serviço de gestão de workflows*

O *update* do sistema de gestão de *workflows* por agente de testes é realizado utilizando *batch scripts que*, sequencialmente e para cada máquina, sinalizar e parar o serviço do *Windows* remoto que aloja o sistema de gestão de ***workflows*** [66].

De seguida são copiadas as versões mais recentes dos ficheiros necessários à execução do sistema de gestão de *workflows* (*dlls*, ficheiros de configuração, *etc*) para o local onde se encontra instalado o respetivo serviço em cada máquina[6] [67].

Numa fase final, utilizando *batch scripts,* é enviada a ordem de reativação do serviço do *Windows* [66]. Este fluxo de evento está representado na Figura 21.

[6] Todos os serviços de gestão de *workflows*, para cada agente de testes, encontram-se instalados e configurados em pasta partilhadas na rede, sendo as respetivas pastas acessíveis pelas contas de administração do domínio dos testes do SIJ. Deste modo é facilitado o processo de atualizações de ficheiros do serviço de gestão de *workflow,* sendo utilizada uma conta de administração para conduzir a manutenção de ficheiros em todas as localizações partilhadas de rede.

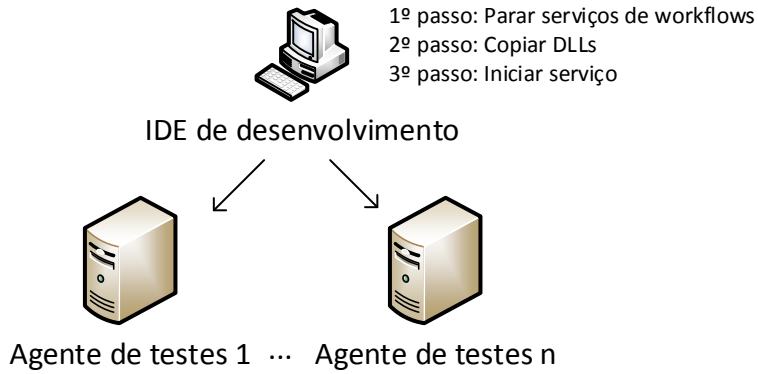

Figura 21 – Deploy do serviço de gestão de workflows

5.3.3.3 *Deploy da base de dados*

O *deploy* da base de dados em cada agente de testes é, numa fase inicial, semelhante ao processo de *deploy* da aplicação *web*. Como pode existir mais do que uma versão da base de dados do *SIJ*, dependendo do *branch* do *software*, então a base de dados é controlada por um projeto de base de dados dentro do *IDE* de desenvolvimento [68].

Sempre que é necessário testar remotamente o *software*, numa primeira fase, é realizado um *deploy* do projeto de base de dados para um dos agentes de testes[7] [68]. Este processo permite apagar e recriar uma base de dados com a informação mínima e necessária à execução dos testes de *software* na máquina em questão.

Numa fase final, recorrendo a *batch scripts*, é realizado um *backup* da base de dados para uma localização partilhada de rede acessível por todos os agentes de testes [69], [70]. A partir do respetivo *backup* é feito o seu *restore* em todas as instâncias de *SQL Server* presente nos agentes de testes [69], [71].

Foi optado por se conduzir apenas um único *deploy* do projeto de base de dados para um dos agentes seguido de um *backup* e *restore* da base de dados para os restantes agentes de testes devido aos ganhos temporais inerentes a este processo. Conduzir *deploys* contínuos do projeto de base de dados para todos os agentes de testes é um processo mais demorado.

[7] O processo de *deploy* do projeto de base de dados pode ser conduzido tanto através do *IDE* de desenvolvimento como utilizando a plataforma de compilação *MSBuild*. À semelhança da aplicação web a utilização da plataforma pode ser mais vantajosa pois permite automatizar o processo de *deploy* da base de dados do *SIJ*.

A Figura 22 pretende ilustrar todo processo de deploy da base de dados pelos diferentes agentes de testes.

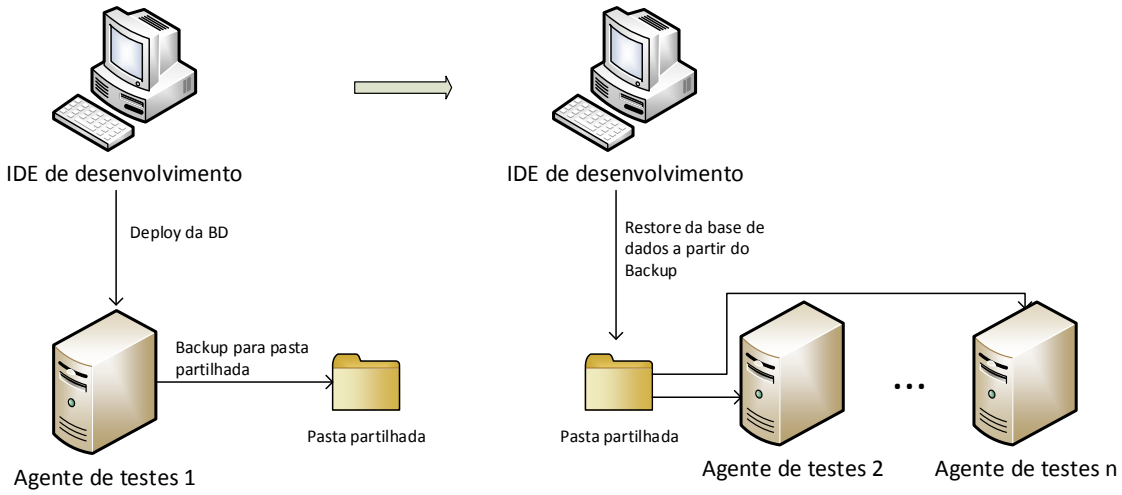

Figura 22 – Processo de deploy da base de dados

5.3.3.4 *O processo de automação*

Atualmente o processo de execução automatizadas de testes é conduzido recorrendo-se à utilização de tarefas calendarizadas [72]. Neste sentido foram agendadas tarefas que, de forma sequencial, permitem executar todos os scripts necessários à compilação e teste das versões mais recentes do SIJ. Todo este processo é realizado apenas numa máquina virtual, a qual se encontra alojada dentro do domínio criado para executar os testes sobre o SIJ.

Toda esta operação, conduzida com recurso à utilização de *batch script* pode ser dividida em quatro fases. Numa primeira fase é devolvido o código mais recente do sistema de controlo de versões [73] referente a um *branch* específico do *software*. Numa segunda fase é compilado todo o código da aplicação sendo, numa terceira fase, conduzido o *deploy* completo da aplicação por todos os agentes de teste. Numa fase final são executados todos os testes associados à aplicação ou apenas um subconjunto dos mesmos [74].

Todos os ficheiros necessários ao processo de automatização de testes do SIJ encontram-se alojados dentro da própria solução da aplicação, sendo a sua gestão realizada à semelhança de qualquer outro ficheiro do *SIJ*. Estão também incluídos todos os ficheiros que contêm a estrutura das tarefas calendarizadas, os quais podem ser consumidos pelo sistema de agendamento de tarefas do sistema operativo.

5.4 A solução atual

Até ao momento a presente solução tem servido os requisitos necessários à validação do *SIJ*, possibilitando diminuir o tempo necessário para executar todo o universo de testes de *software*, tanto pela via da execução paralelizada de testes como pela via dos processos de automatização no *deploy* da aplicação por todos os agentes de testes. Esta diminuição do tempo necessário ao processo de validação do *software* está diretamente relacionada com o número de agentes de testes utilizados. Quanto maior for o número de agentes de testes utilizados maior será o número de testes que poderá ser executado paralelamente, reduzindo-se assim o tempo total de execução.

Nesta fase todos os processos envolvidos no *deploy* do *SIJ* em todos os agentes de testes encontra-se completamente automatizado, utilizando *scheduled scripts*, a plataforma *MSBuild* e o programa *MSTest.exe*.

Na secção em anexo "9.2 Scripts de Automação" é possível consultar detalhes específicos sobre os scripts utilizados no processo de automação descrito anteriormente.

Devido a um conjunto de problemas encontrados durante a fase de automatização, não foi possível utilização das capacidades do servidor de TFS [16] para conduzir todo o processo de automação do *deploy* da aplicação e execução dos testes de *software*. Esse conjunto de problemas foi derivado de fatores internos (ligados a questões de permissão de utilizadores, politicas de *software*, politicas de acesso, *etc*…) da equipa responsável pela gestão do *software* de controlo de versões. Tais fatores acabaram por inviabilizar o uso do respetivo *software* como elemento de automação do processo de testes [44]. As referidas restrições impossibilitam qualquer tipo de comunicação entre o servidor onde está alojado o *software* de controlo de versões e as máquinas utilizadas como agentes de testes[8], não sendo possível conduzir o processo de *deploy* dos elementos necessários para executar localmente o SIJ ou ainda executar remotamente todos os testes de software.

[8] Devido a políticas internas de segurança a comunicação entre o servidor do TFS e todas as máquinas utilizadas no processo de testes não é possível devido ao facto de existirem em domínios de rede diferentes.

6 Conclusão

Testar uma aplicação é um elemento chave que permite assegurar a qualidade de um *software*. Não sendo um processo trivial, testar um *software* requer um grande nível de conhecimento da equipa de testes, tanto ao nível do *software* a ser testado como das metodologias que melhor se adequam ao *software* em causa.

O desenvolvimento de diferentes metodologias de testes podem auxiliar tanto a equipa de testes como a equipa de desenvolvimento a compreender o comportamento da aplicação segundo perspetivas diferentes. É importante compreender como certos elementos da arquitetura do *software* funcionam como unidades individuais e como funcionam em conjunto com outros elementos. É igualmente importante compreender o comportamento do *software* como um todo e ainda como este se comporta sob condições anómalas que podem conduzir a perdas irreparáveis de informação.

Outro aspeto chave passa por perceber a capacidade de resposta de um *software* a pedidos de utilização intensiva por parte dos seus utilizadores. É igualmente importante perceber os elementos limitadores da performance de uma aplicação. Estes elementos podem ainda ser internos, relacionados com a arquitetura da aplicação, ou externos, relacionados com o hardware que sobre o qual é executada a aplicação.

Aquando do desenvolvimento de cenários de testes é importante entender quais os elementos chave de uma aplicação que necessitam de um esforço extra em termos de desenvolvimento de testes. No caso do *SIJ,* a camada designada anteriormente por "Camada de Apresentação" representa um dos elementos de maior importância para os utilizadores finais da aplicação. Esta é a camada que direta ou indiretamente comunica com todas as restantes camadas do *SIJ*, tornando-se vital assegurar a sua qualidade e correto funcionamento.

A estrutura da equipa de desenvolvimento é também um elemento importante quando se desenvolve *software*. Para este projeto, e como o explicado anteriormente, a decisão de separar a equipa de desenvolvimento e a equipa de testes foi conduzida nas fases iniciais do ciclo de desenvolvimento do software. O objetivo principal desta decisão passava por permitir que um grupo programadores imparciais (ao desenvolvimento do código) pudesse analisar e testar a aplicação.

A utilização de *branches* permite conduzir uma separação entre as versões em desenvolvimento e as versões em produção do *SIJ*. Deste modo é possível isolar a identificação e correção de problemas apenas nas versões em produção do *software*. Assim é possível evitar o aparecimento de problemas adicionais, originados pela implementação de novas funcionalidades, sempre que são lançadas novas correções nas versões em produção do *SIJ*. Utilizando *branches* é possível garantir que a implementação de novas funcionalidades afetará apenas as versões em desenvolvimento do *SIJ* e nunca as versões *e*m produção.

À medida que o número e complexidade dos testes aumenta, aumenta também o tempo necessário para executar todo um universo de testes de uma aplicação. É, portanto, vital utilizar uma estratégia de execução automatizada que permita testar continuamente o *software* em cada iteração no ciclo de vida do desenvolvimento do *software*.

A utilização de ferramentas que permitam paralelizar o processo de execução de testes de *software* é, só por si, um dos elementos mais importantes que uma equipa de testes pode ter ao seu dispor. Este tipo de ferramentas permitem reduzir drasticamente o tempo necessário para validar a qualidade de um *software*.

Nesta dissertação foram apresentadas as principais razões que levaram a equipa de testes a adotar as metodologias de testes apresentadas, a configuração geral de testes e o recurso a soluções de *cloud computing* para a instalação de agentes e controladores de testes.

7 Trabalho futuro

Atualmente são utilizadas 16 máquinas virtuais como elementos de suporte à execução automatizada de testes de *software* do *SIJ*. Todas as máquinas virtuais encontram-se ativas 24 horas por dia e 7 dias por semana. Em média são executados testes de *software* a cada 24 horas, sendo que essas execuções remotas têm durações máximas de 2 a 2.5 horas. Neste cenário, a cada dia, existe uma alocação desnecessária de recursos durante cerca de 21 horas. Esta alocação de recursos pode ter especial interesse sempre que existam outros processos, que partilhem a mesma infraestrutura de *cloud,* que necessitem de utilizar as capacidades de processamento da mesma. Neste sentido será necessário desenvolver um mecanismo que permita automatizar os processos de:

1. Determinar os possíveis estados de uma máquina virtual.
2. Iniciar máquinas virtuais sempre que seja necessário executar remotamente testes de *software*.
3. Desligar máquinas virtuais sempre que estas não estejam a ser utilizadas para execução de testes.

Esta gestão automatizada e a pedido, vulgarmente designada por *elastic computing* [75], permitirá otimizar a utilização dos recursos da infraestrutura de *cloud* utilizados no processo de execução remota de testes de *software*.

8 Bibliografia

[1] L. Copeland, “The Testing Process,” in in *A Practitioner’s Guide to Software Test Design*, O’Reilly Media, Inc, 2009, pp. 1–4.

[2] T. Müller, S. Eldh, D. Graham, K. Olsen, M. Pyhäjärvi, G. Thompson, and E. van Veendendal, “Certified Tester Foundation Level Syllabus,” 2005.

[3] A. P. Mathur, “Humans, Errors, and Testing,” in in *Foundations of Software Testing: Fundamental Algorithms and Techniques*, India, Pearson Education, 2007, p. 709.

[4] L. Brader, H. Hilliker, and A. C. Wills, “Testing in the Software Lifecycle,” in in *Testing for Continuous Delivery with Visual Studio 2012*, Microsoft Press, 2012, pp. 171–193.

[5] J. M. Stecklein, J. Dabney, B. Dick, B. Haskins, R. Lovell, and G. Moroney, “Error Cost Escalation Through the Project Life Cycle,” in *Incose 14th Annual International Symposium*, 2004, p. 11.

[6] I. Burnstein, “Practical Software Testing: A Process-Oriented Approach,” in in *Practical Software Testing: A Process-Oriented Approach*, Springer Professional Computing, 2003, p. 709.

[7] P. Ammann and J. Offutt, “Introduction to Software Testing,” in in *Introduction to Software Testing*, Cambridge University Press, 2008, p. 334.

[8] J. Rosa, C. Teixeira, and J. Sousa Pinto, “Risk factors in e-justice information systems,” *Gov. Inf. Q.*, vol. 30, no. 3, pp. 241–256, Jul. 2013.

[9] C. Teixeira, J. Sousa Pinto, and R. Morais, “Specificities of a Nation Wide Information System for the Courts,” in *ICT Law 2013*, 2013.

[10] C. Teixeira and J. Sousa Pinto, “Assisted On-Job Training,” in in *E-Learning - Engineering, On-Job Training and Interactive Teaching*, 1st ed., A. Silva, E. Pontes, A. Guelfi, and S. Kofuji, Eds. Rijeka, Croatia: Intech, 2012.

[11] T. Erl, *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall PTR , 2005.

[12] C. Microsoft, “Microsoft SQL Server.” [Online]. Available: http://msdn.microsoft.com/en-us/library/bb545450.aspx.

[13] C. Microsoft, “Entity Framework (EF) Documentation.” [Online]. Available: https://entityframework.codeplex.com/.

[14] T. Müller, S. Eldh, D. Graham, K. Olsen, M. Pyhäjärvi, G. Thompson, and E. van Veendendal, “Certified Tester Foundation Level Syllabus,” 2005.

[15] K. Schwaber and J. Sutherland, “The Scrum Guide™.” p. 16.

[16] B. Harrys, “Team Foundation Server 2010 Key Concepts,” *Brian Harris’s blog*, 2009. [Online]. Available: http://blogs.msdn.com/b/bharry/archive/2009/04/19/team-foundation-server-2010-key-concepts.aspx.

[17] C. Microsoft, “Getting Started Tracking Work,” 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms181269(v=vs.100).aspx.

[18] C. Microsoft, “Branch Strategically,” 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/ee782536(v=vs.100).aspx.

[19] B. Appleton, S. Berczuk, R. Cabrera, and R. Orenstein, “Streamed Lines: Branching Patterns for Parallel Software Development,” in *Pattern Languages of Programs Conference*, 1998, p. 71.

[20] B. Heys and W.-P. Schaub, “Visual Studio TFS Branching and Merging Guidance,” *MSDN Magazine*, no. February 2011, Feb-2011.

[21] J. Pan, “Software Testing,” *Carnegie Mellon University - Dependable Embedded Systems*, 1999. [Online]. Available: http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/.

[22] P. Le Gall and A. Arnould, “Formal Specifications and Test : Correctness and Oracle.”

[23] C. Kaner, S. Fay, P. Bond, S. Tilley, M. Andrews, and J. Whittaker, “Teaching the Black Box Testing Course,” Melbourne, Florida, 2004.

[24] L. Williams, “White-Box Testing,” *Whit Box Test.*, 2001.

[25] Thomas Ostrand, “White-Box Testing,” *Encycl. Softw. Eng.*, 2002.

[26] D. Menasce, “Load testing of Web sites,” *Internet Comput. IEEE*, vol. 6, no. 4, pp. 70–74, 2002.

[27] NEOLOAD, “Load Testing Software - Performance Testing Tools for all of your Web & Native Mobile Applications,” 2013. [Online]. Available: http://www.neotys.com/performance-load-testing-tools.html.

[28] C. Microsoft, “Testing Application Performance and Stress,” 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/dd293540(v=vs.100).aspx.

[29] BlazeMeter, “What is BlazeMeter’s Load Testing Cloud?,” 2013. [Online]. Available: http://community.blazemeter.com/knowledgebase/articles/42674-what-is-blazemeter-s-load-testing-cloud-.

[30] J. Cohen, D. Plakosh, and K. Keeler, “Robustness Testing of Software-Intensive Systems : Explanation and Guide,” no. April, 2005.

[31] L. Rising, “The Scrum software development process for small teams,” *Software, IEEE*, vol. 17, no. 4, pp. 26–32, 2000.

[32] DOD, “SYSTEMS ENGINEERING FUNDAMENTALS,” FORT BELVOIR, VIRGINIA, 2001.

[33] G. Raman, “Software Testing | Automation Testing | Interview Faqs | Manual Testing Q&A,” 2009. [Online]. Available: http://softwaretestinginterviewfaqs.wordpress.com/tag/manual-testing-basics/.

[34] B. Marick, “When Should a Test Be Automated?,” 2000. [Online]. Available: http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=ART&ObjectId=2010#authorbio.

[35] ISTQB, “What is Acceptance testing?” [Online]. Available: http://istqbexamcertification.com/what-is-acceptance-testing/.

[36] R. Perez, “Unit testing, component level testing and UI testing, what to use and when,” 2010. [Online]. Available: http://blogs.msdn.com/b/raulperez/archive/2010/04/29/unit-testing-component-level-testing-and-ui-testing-what-to-use-and-when.aspx.

[37] R. Osherove, *The Art of Unit Testing: with Examples in .NET*, 1st ed. Manning Publications, 2009, p. 320.

[38] C. Microsoft, “Integration Testing,” 2003. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa292128(v=vs.71).aspx.

[39] C. Erickson, R. Palmer, D. Crosby, M. Marsiglia, and M. Alles, “Make Haste , not Waste : Automated System Testing.” pp. 1–10.

[40] A. M. Memon, “GUI Testing: Pitfalls and Process,” *Aware Comput. IEEE*, no. August 2002, pp. 87–88, 2002.

[41] C. Microsoft, “Create a Basic Build Definition,” 2011. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms181716(v=vs.100).aspx.

[42] C. Microsoft, “Specify Build Triggers and Reasons,” 2011. [Online]. Available: http://msdn.microsoft.com/en-us/library/hh190718(v=vs.100).aspx#reason-individual-ci.

[43] A. Chatterjee’s, “Remote Execution and Data Collection,” 2009. [Online]. Available: http://blogs.msdn.com/b/amit_chatterjee/archive/2009/03/28/remote-execution-and-data-collection.aspx.

[44] C. Microsoft, “How to: Configure and Run Scheduled Tests After Building Your Application,” 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms182465(v=vs.100).aspx.

[45] Apache Software Foundation, “Apache Subversion - ‘Enterprise-class centralized version control for the masses’.”[Online]. Available: http://subversion.apache.org/.

[46] Git, “Git Reference.” [Online]. Available: http://gitref.org/.

[47] C. Microsoft, “Team Foundation Server,” 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/ff637362.aspx.

[48] D. Chappell, “The Workflow Way: Understanding Windows Workflow Foundation,” *April*, 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/dd851337.aspx.

[49] C. Microsoft, “What Is Windows Communication Foundation.” [Online]. Available: http://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx.

[50] M. Milner, “Workflow Services for Local Communication,” *MSDN Mag.*, no. November 2009, 2009.

[51] C. Microsoft, “StateMachineWorkflowInstance.SetState Method,” 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/system.workflow.activities.statemachineworkflowinstance.setstate.aspx.

[52] C. Microsoft, “Introduction to Coded UI Tests with Visual Studio 2010 Ultimate,” 2010. [Online]. Available: http://msdn.microsoft.com/en-us/vs2010trainingcourse_introtocodeduitests.aspx.

[53] C. Microsoft, “How to: Create a Coded UI Test,” 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/dd286681(v=vs.100).aspx.

[54] R. Green, “Windows Workflow Tutorial: Introduction to Fault Handling,” *January*, 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/dd695716.aspx.

[55] M. Milner, “Error Handling In Workflows,” 2009. [Online]. Available: http://msdn.microsoft.com/en-us/magazine/dd419656.aspx.

[56] PHP Group, “Exceptions.” [Online]. Available: http://www.php.net/manual/en/language.exceptions.php.

[57] Oracle, “The try Block.” [Online]. Available: http://docs.oracle.com/javase/tutorial/essential/exceptions/try.html.

[58] C. Microsoft, “How to: Create a New Web Performance Test Using the Web Performance Test Recorder,” 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms182539(v=vs.100).aspx.

[59] C. Microsoft, “Walkthrough: Creating and Running a Load Test Containing Web Performance Tests,” 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms182594(v=vs.100).aspx.

[60] N. Kamkolkar, “Getting Started with Visual Studio 2010 Ultimate - Load and Performance Testing,” 2010. [Online]. Available: http://blogs.msdn.com/b/nkamkolkar/archive/2010/11/02/getting-started-with-visual-studio-2010-ultimate-load-and-performance-testing.aspx.

[61] C. Microsoft, “Installing and Configuring Visual Studio Agents and Test and Build Controllers,” 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/vstudio/dd648127(v=vs.100).aspx.

[62] C. Microsoft, “Managing Test Controllers and Test Agents,” 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/dd695837(v=vs.100).aspx.

[63] C. Microsoft, “Create a Virtual Active Directory Domain Controller.” [Online]. Available: http://technet.microsoft.com/en-us/library/ee256063(WS.10).aspx.

[64] C. Microsoft, “MSBuild,” 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/dd393574(v=vs.100).aspx.

[65] C. Microsoft, “Web Deploy sync Operation,” 2009. [Online]. Available: http://technet.microsoft.com/en-us/library/dd569005(v=ws.10).aspx.

[66] C. Microsoft, “SC.” [Online]. Available: http://technet.microsoft.com/en-us/library/bb490995.aspx.

[67] C. Microsoft, “Robocopy,” 2012. [Online]. Available: http://technet.microsoft.com/en-us/library/cc733145.aspx.

[68] C. Microsoft, “Working with Database Projects,” 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/vstudio/xee70aty(v=vs.100).aspx.

[69] C. Microsoft, “Using the Invoke-Sqlcmd cmdlet,” 2008. [Online]. Available: http://technet.microsoft.com/en-us/library/cc281720(v=sql.105).aspx.

[70] C. Microsoft, “How to: Create a Full Database Backup (Transact-SQL),” 2008. [Online]. Available: http://technet.microsoft.com/en-us/library/ms191304(v=sql.105).aspx.

[71] C. Microsoft, “RESTORE (Transact-SQL),” 2008. [Online]. Available: http://technet.microsoft.com/en-us/library/ms186858(v=sql.105).aspx.

[72] C. Microsoft, “Task Scheduler Overview.” [Online]. Available: http://technet.microsoft.com/en-us/library/cc721871.aspx.

[73] C. Microsoft, “Get Command,” 2010. [Online]. Available: http://msdn.microsoft.com/en-US/library/fx7sdeyf(v=vs.100).aspx.

[74] C. Microsoft, “MSTest.exe Command-Line Options,” 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms182489(v=VS.100).aspx.

[75] N. Roman, Herbst, S. Kounev, and R. Reussner, “Elasticity in Cloud Computing: What It Is, and What It Is Not.” Karlsruhe, p. 5.

[76] OpenNebula.org, “About the OpenNebula.org Project.” [Online]. Available: http://opennebula.org/about/project/.

[77] J. D. Meier, C. Farre, P. Bansode, S. Barber, and D. Rea, “Load-Testing Web Applications,” in in *Performance Testing Guidance for Web Applications*, Microsoft Corporation., 2007, p. 221.

[78] C. Microsoft, “Load Test Analyzer Overview,” 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms404677(v=vs.100).aspx.

[79] OpenNebula.org, “Creating Virtual Machines 4.4.” [Online]. Available: http://archives.opennebula.org/documentation:rel4.4:vm_guide.

[80] C. Microsoft, “Analyzing Load Test Results in the Tables View of the Load Test Analyzer,” 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms404656(v=vs.100).aspx.

[81] C. Hameed, “An Overview of Troubleshooting Memory Issues - Part Two,” 2008. [Online]. Available: http://blogs.technet.com/b/askperf/archive/2008/01/29/an-overview-of-troubleshooting-memory-issues-part-two.aspx.

[82] Micro, “Evaluating Memory and Cache Usage.” [Online]. Available: http://technet.microsoft.com/en-us/library/cc958290.aspx.

[83] SQA, “Monitoring and Optimising System Performance and Reliability,” 2007. [Online]. Available: http://www.sqa.org.uk/e-learning/ClientOS04CD/page_02.htm.

[84] C. Microsoft, “Walkthrough: Creating, Editing and Maintaining a Coded UI Test,” 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/ff977233(v=vs.100).aspx.

[85] C. Microsoft, “How to: Check UIElement Properties Using Coded UI Test.” [Online]. Available: http://msdn.microsoft.com/en-us/library/hh404081.aspx.

9 Anexos

9.1 Testes de Carga

9.1.1 Resultado de testes de carga a configurações de *hardware*

A experiência de carga foi conduzida com a aplicação instalada em máquinas virtuais executadas sobre o virtualizador *OpenNebula* [76]. No decorrer da experiência, para além do incremento no número de utilizadores virtuais a executarem tarefas em simultâneo, foram conduzidos também incrementos nas características de *hardware* das respetivas máquinas.

Tal como referido, foram utilizadas diferentes combinações de utilizadores virtuais de forma a executar pedidos à aplicação de forma paralela, sendo assim possível medir a capacidade de resposta do *SIJ* a pedidos por parte dos seus utilizadores. Toda a gama de testes (para cada configuração de *hardware*) foi iniciada com 10 utilizadores virtuais, os quais executavam pedidos simultâneos (*web performance tests*) à aplicação *web*, durante um total de quatro minutos. Após a execução de cada teste de carga foram recolhidas de métricas (vulgarmente designadas de métricas de performance) [77] como, por exemplo, o tempo médio de resposta por página, tempo médio de execução por *web performance test*, total de *web performance tests* executados, *etc.* [78]. Após a recolha de todas as métricas necessárias o número de utilizadores virtuais era incrementado em 20 unidades sendo repetida a execução do teste de carga com o número de utilizadores incrementado.

Este processo iterativo era terminado apenas quando a aplicação *web* era incapaz de responder, em tempo útil, aos pedidos dos utilizadores virtuais.

Como explicado anteriormente esta experiência foi conduzida em diferentes combinações de *hardware*. A configuração inicial foi baseada num servidor com 1 *gigabytes* de memória (*RAM*) e uma unidade de processamento (*CPU*) contendo todos os elementos necessários ao funcionamento do SIJ (aplicação *web*, serviço de gestão de *workflows* e base de dados).

Sempre que era atingido o ponto de saturação (em termos de pedidos à aplicação *web*) do *SIJ* procedia-se ao redimensionamento das características de *hardware* da máquina virtual (VM), sendo de seguida repetido todo o processo de execução de testes de carga. A máquina virtual utilizada na condução dos testes de carga foi redimensionada até um máximo de 4 *CPUs, 4 VCPUs e* 4 *Gigabytes* de *RAM*, sendo estas as características semelhantes aos nós físicos onde está sediada o ambiente de produção do SIJ.

De seguida é apresentada na Tabela 3 a informação referente às configurações de *hardware*, juntamente com os números mínimos e máximos de utilizadores virtuais, sobre as quais foram conduzidos os testes de carga sobre o SIJ.

Tabela 3 – Configurações de hardware

Configuração de *hardware*	Nº mínimo de utilizadores	Nº máximo de utilizadores
1 CPU; 1 Gigabyte de RAM	10	50
1 CPU; 2 Gigabyte de RAM	10	70
2 CPU; 2 Gigabyte de RAM	10	70
2 CPU; 4 Gigabyte de RAM	10	70
4 CPU; 4 Gigabyte de RAM	10	70
4 CPU; 4 VCPU; 4 Gigabyte de RAM	10	110

No processo de redimensionamento das características de *hardware* das máquinas virtuais apenas se incrementava a capacidade de *CPU* ou *RAM* da máquina virtual, nunca os dois elementos em simultâneo. Este facto prende-se com o objetivo de tentar medir o impacto das alterações de cada um destes elementos isoladamente.

A totalidade dos testes de carga, para todas as configurações de *hardware* e número de utilizadores virtuais, foram sempre executados durante um total de quatro minutos.

Como referido anteriormente foram recolhidas varias métricas de performance aquando da realização dos testes de carga. Estas métricas de performance encontram-se enumeradas e descritas na Tabela 4.

Tabela 4 – Indicadores de Performance

Nome	Descrição
Tests/Sec	Número médio de testes por segundo
Avg. Test Time (sec)	Tempo médio de execução por teste (em segundos)
Pages/Sec	Número médio de paginas (devolvidas pela aplicação) por segundo
Avg. Page Time (sec)	Tempo médio para a receção de uma página (em segundos)
Requests/Sec	Número médio de pedidos enviados à aplicação (em segundos)
Requests Failed	Total de pedidos com falhas
Avg. Response Time (sec)	Tempo médio de resposta da aplicação (em segundos)
Failed Tests (% of total)	Percentagem total de testes falhados

Total Load Response	Total de informação submetida na base de dados diretamente relacionada com a execução do teste de carga. Neste sentido corresponde à contabilização do total de Autos, Petições Iniciais, Pedidos de Habeas Corpus e Despachos submetidos via interfaces web dentro do tempo de execução do teste de carga.

De entre o total de indicadores recolhidos foi dada especial enfase à análise do tempo médio de execução por teste (*Avg. Test Time*), tempo médio para a receção de uma página (*Avg. Page Time*), tempo medido de resposta da aplicação (*Avg. Response Time*) e ao total de informação que foi possível submeter via as interfaces *web* (*Total Load Response*).

Estes indicadores revelam, de forma precisa, o desempenho da aplicação, sendo uma representação da capacidade de resposta da mesma a pedidos intensivos por parte dos seus utilizadores, tanto a nível temporal como ao nível da capacidade de informação que é possível processar por unidade de tempo.

9.1.1.1 *Tempo médio de execução de teste de performance web*

A Tabela 5 contém os resultados relativos aos tempos médios de execução por teste de performance web[9] (em segundos) para as diferentes combinações de *hardware* e número de utilizadores virtuais. Os resultados apresentados são relativos apenas a configurações em que era possível concluir um qualquer número de testes dentro de um intervalo temporal de quatro minutos. Neste sentido, valores mais baixos são indicativos de melhores performance por parte do SIJ.

Tabela 5 – Avg. Test Time (sec)

Nº de utilizadores	1 CPU 1G Ram	1 CPU 2G Ram	2 CPU 2G Ram	2 CPU 4G Ram	4 CPU 4G Ram	4 CPU 4 VCPU 4G Ram
10	59	54,6	30,9	41,3	41,9	24,5
30	151	135	111	129	123	63,5
50	---	---	201	188	127	129
70	---	---	---	---	---	191

[9] Neste sentido um teste de performance web representa todos os passos executados por um utilizador nas interfaces web do SIJ de modo a proceder à submissão de informação (despachos, pedidos de habeas corpus, autos e petições iniciais) ou a aceder aos menus da aplicação web.

Na Figura 23 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 5. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (*x*) e a informação relativa ao tempo médio por teste é apresentada no eixo das ordenadas (*y*).

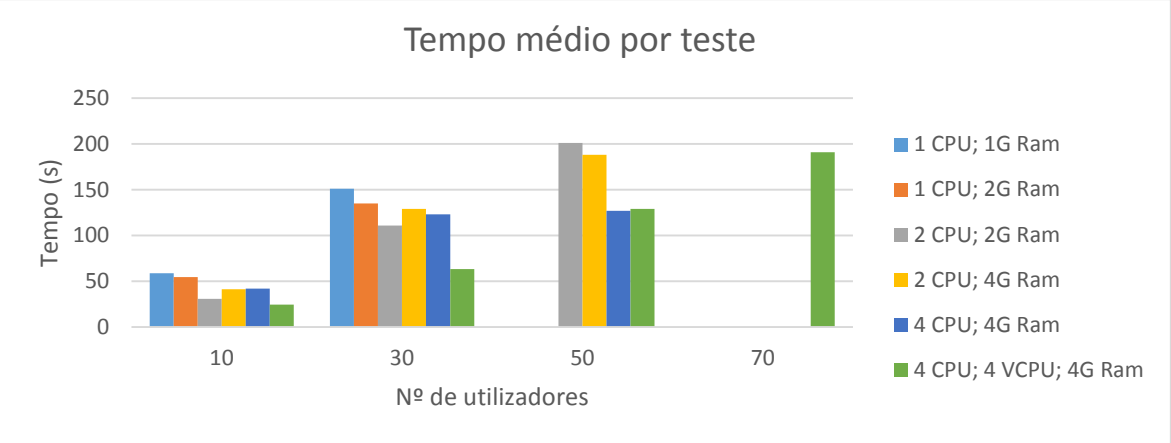

Figura 23 – Tempos médios por teste de carga

A partir da análise do gráfico da Figura 23 é possível concluir que a combinação de *hardware* a partir da qual é possível obter os melhores tempos é, de forma praticamente invariável, relativa à máquina virtual com 4 *CPUs,* 4 *VCPUs e* 4 *Gigabytes de RAM.* Com esta combinação de *hardware* foi não só possível obter os melhores tempos médios de execução de testes como também foi possível testar o SIJ para um máximo de utilizadores virtuais em paralelo. A especificação do número de CPUs apenas restringe o número de processadores que podem ser alocados a uma VM. A utilização de VCPUs permite especificar o número de "cores" visíveis para uma VM. Este facto permite otimizar a performance de uma mesma [79]. Neste sentido é importante salientar o impacto da capacidade de processamento no desempenho geral da aplicação relativamente à rapidez com que é possível completar tarefas utilizando as interfaces web.

Da análise do gráfico da Figura 23 não é possível obter uma relação entre os incrementos de RAM (de 1 *gigabyte* para 2 *gigabytes* e de 2 *gigabytes* para 4 *gigabytes*) e qualquer tipo de melhorias na performance da aplicação.

Por outro lado é possível verificar que incrementos na capacidade de processamento das máquinas virtuais (de 1 *CPU* para 2 *CPU*, de 2 *CPU* para 4 *CPU* e de 4 *CPU* para *4CPU* + *4VCPU*) tendem a levar a melhorias na celeridade com que é possível completar tarefas utilizando as interfaces web.

9.1.1.2 *Tempos médios de obtenção de páginas web*

Na Tabela 6 são apresentados os tempos médios de obtenção de páginas web (em segundos) para diferentes combinações de *hardware* e diferentes combinações de utilizadores[10]. Neste sentido, valores mais baixos são indicativos de melhores performance por parte do SIJ.

Tabela 6 – Tempos médios de obtenção de páginas web

Nº de Utilizadores	**1 CPU 1G Ram**	**1 CPU 2G Ram**	**2 CPU 2G Ram**	**2 CPU 4G Ram**	**4 CPU 4G Ram**	**4 CPU 4 VCPU 4G Ram**
10	2,71	2,63	1,23	2,56	2,59	0,65
30	18	17,2	10,6	12	12,5	5,12
50	32,4	31,8	18,4	27,4	30,6	11,7
70	30,1	28	27,6	23,1	18,3	18,7
90	---	---	---	---	---	29,1
110	---	---	---	---	---	28,2

Na Figura 24 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 6. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa ao tempo médio para a obtenção de páginas web é apresentada no eixo das ordenadas (y).

Através da análise do gráfico da Figura 24 é possível constatar que os tempos de obtenção mais curtos pertencem à máquina virtual com a configuração de *hardware* de 4 *CPUs*, 4 *VCPUs* e 4 *Gigabytes* de *RAM*. Neste sentido é novamente relevante salientar a importância da capacidade de processamento no desempenho do SIJ. Este impacto no processamento é igualmente verificável quando a carga de utilização do SIJ se situa entre os 70 e 110 utilizadores virtuais, tendo as configurações com mais unidades de *CPU* melhores resultados que todas as imediatamente anteriores.

À semelhança do apresentado anteriormente, não é possível associar os incrementos de memória com melhorias nos tempos de obtenção de páginas, sendo que a única melhoria de tempos

[10] Nesta tabela são apresentados resultados para um maior grupo de utilizadores quando comparado com a Tabela 5. Este fator prende-se com o facto de os tempos médios para completar testes de performance web só poderem ser obtidos na eventualidade de ser possível terminar os mesmos num intervalo de 4 minutos. Mesmo não sendo possível terminar qualquer teste nesse período é possível obter tempos de resposta de páginas individuais.

registada está associada à simulação de setenta utilizadores virtuais para as combinações de *hardware* entre 2 *CPUs*, 2 *Gigabytes* de *RAM* e 2 *CPUs*, 4 *Gigabytes* de *RAM*.

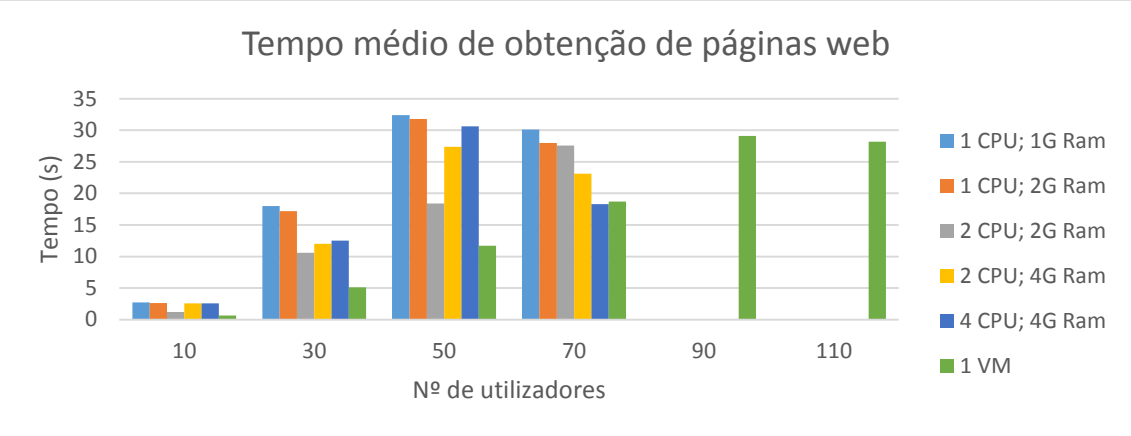

Figura 24 – Tempo médio de obtenção de páginas web

9.1.1.3 *Tempo médio de resposta a* pedidos

Na Tabela 7 são apresentados os tempos médios de obtenção de respostas [80] para diferentes combinações de *hardware* e diferentes combinações de utilizadores. Neste sentido, valores mais baixos são indicativos de melhores performance por parte do SIJ.

Tabela 7 – Tempo médio de resposta a requests

Nº de utilizadores	**1 CPU 1G Ram**	**1 CPU 2G Ram**	**2 CPU 2G Ram**	**2 CPU 4G Ram**	**4 CPU 4G Ram**	**4 CPU 4 VCPU 4G Ram**
10	0,16	0,16	0,1	0,17	0,16	0,1
30	1,51	1,03	0,79	0,96	0,9	0,37
50	3,72	3,39	1,54	2,21	3,1	0,9
70	2,92	2,83	2,64	2,41	2,21	1,41
90	---	---	---	---	---	2,7
110	---	---	---	---	---	2,69

Na Figura 25 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 7. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa ao tempo médio de resposta a pedidos é apresentada no eixo das ordenadas (y).

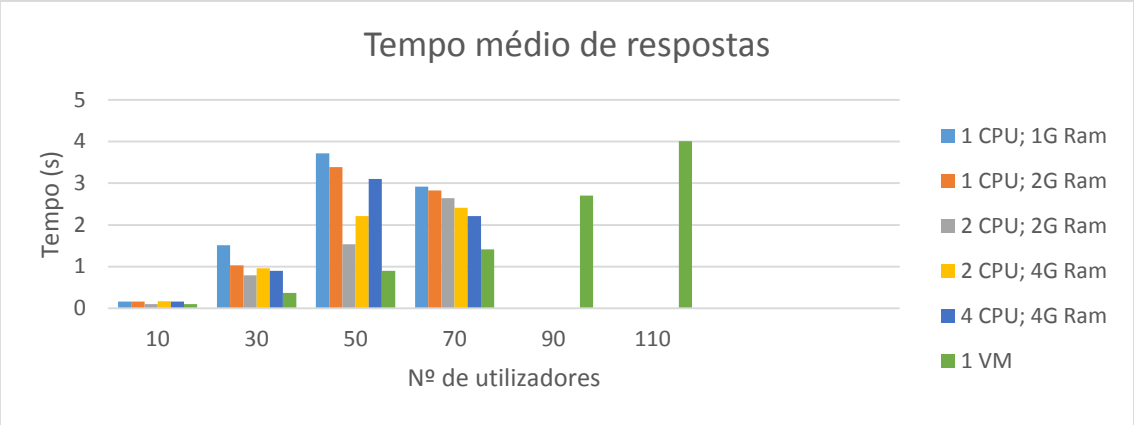

Figura 25 – Tempo médio de obtenção de respostas

Tal como descrito nas análises anteriores, também aqui os menores tempos médios de obtenção de resposta podem ser encontrados para a combinação de *hardware* de 4 *CPUs,* 4 *VCPUs e* 4 *Gigabytes* de *RAM*. Neste sentido, é igualmente possível salientar a importância da capacidade de processamento no desempenho do SIJ.

É também verificável que, na grande generalidade dos casos, incrementos na capacidade de processamento das máquinas virtuais conduz a melhorias de performance nos tempos médios de resposta.

Pelos resultados apresentados não é, novamente, possível encontrar uma relação direta entre incrementos de memória nas máquinas virtuais e melhorias de performance do SIJ em termos de tempos médios de obtenção de respostas.

Neste estudo a média de aumento de performance, em termos de tempos de resposta das duas configurações com os melhores tempos médios (2 CPU; 2G Ram e 4 CPU; 4 VCPU; 4G *Ram*), foi de aproximadamente 35%.

9.1.1.4 *Capacidade de resposta a carga*

Para além dos tempos de resposta da aplicação foram ainda recolhidos dados relativos à quantidade de informação (despachos, petições iniciais, pedidos de habeas corpus e autos iniciais) que é possível submeter na base de dados, num período de quatro minutos, para diferentes combinações de utilizadores. Estes dados foram recolhidos para as duas combinações de *hardware* que apresentaram os melhores tempos de resposta (4 *CPU*; 4 *gigabytes* de *Ram* e 4 *CPU*; 4 *VCPU*; 4 *gigabytes Ram*).

Na Tabela 8 são apresentados os resultados relativos à quantidade de informação que é possível colocar na base de dados do SIJ para duas combinações de *hardware* e utilizadores virtuais. Neste sentido, valores mais elevados são indicativos de melhores performance por parte do SIJ.

Tabela 8 – Capacidade de resposta a carga

Nº de utilizadores	4 CPU; 4G Ram	4 CPU; 4 VCPU; 4G Ram
10	35	50
30	46	78
50	37	71
70	---	65
90	---	63
110	---	72

Na Figura 26 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 8.Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa à totalidade de dados que é possível guardar na base de dados é apresentada no eixo das ordenadas (y).

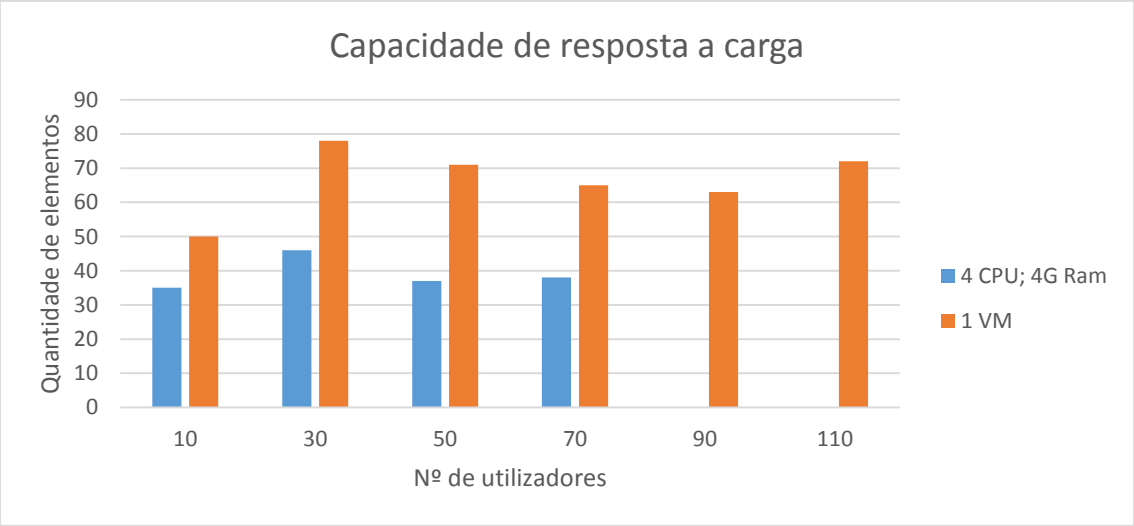

Figura 26 – Capacidade de resposta a carga

Através da análise do gráfico da Figura 26 é possível constatar que um aumento da capacidade de processamento está diretamente relacionado com uma melhoria da capacidade de resposta da aplicação. Um aumento da capacidade de processamento permite aumentar a quantidade de informação que é possível submeter por intervalo de tempo. Neste estudo a média de aumento de

performance, em termos de capacidade de resposta a carga, foi de aproximadamente 69% para as duas configurações de hardware testada

9.1.1.5 *Resultados de utilização média de Processamento*

No decorrer deste estudo, e como referido anteriormente, foram ainda recolhidos indicadores relativos ao processamento e memória das máquinas virtuais onde estava instalado o SIJ.

Os gráficos presentes na Figura 27 e Figura 28 representam a percentagem total de processamento, durante um período de quatro minutos, para as combinações de utilizadores mais baixa e mais alta respetivamente. Estes dados foram recolhidos para uma única máquina virtual com a combinação de *hardware* que apresentou os melhores resultados de performance (4 *CPUs*; *4VCPUs*; 4 *Gigabytes* de *RAM*).

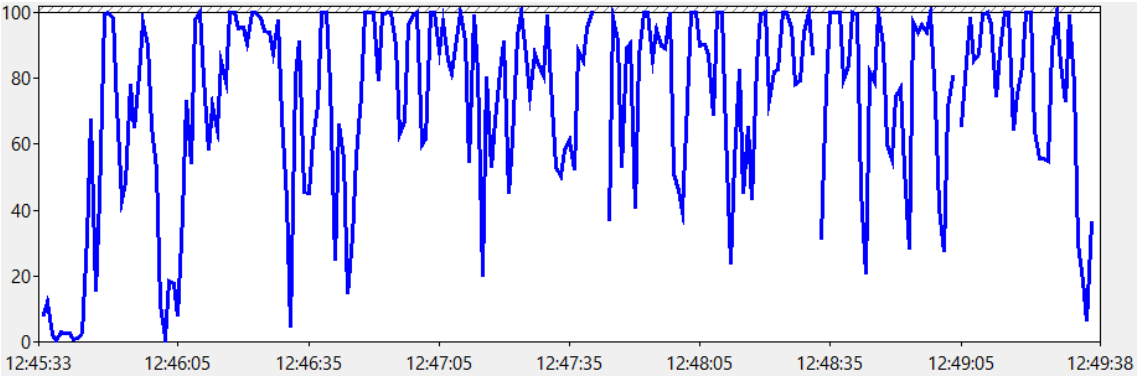

Figura 27 – Percentagem total de processamento para combinação de 10 utilizadores virtuais

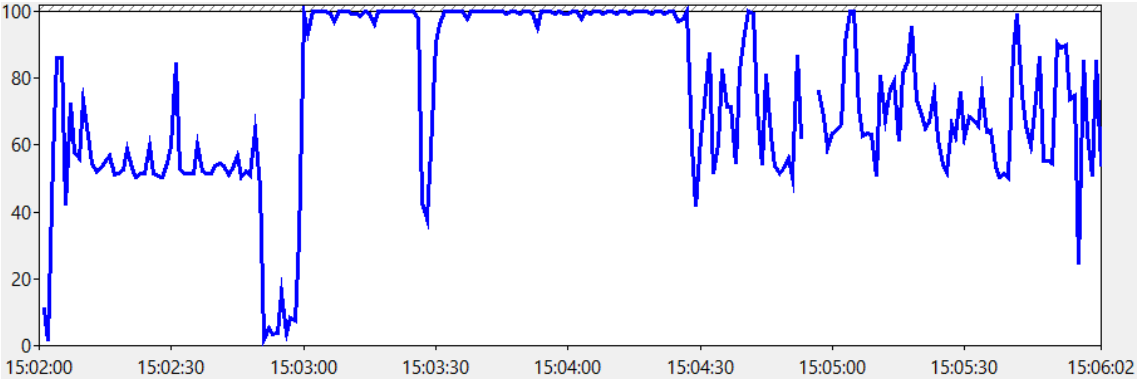

Figura 28 - Percentagem total de processamento para combinação de 110 utilizadores virtuais

Como é possível observar pela análise da Figura 27 e Figura 28 existem períodos temporais onde existe uma alocação total dos recursos de processamento da máquina virtual. Mesmo nos resultados dos testes de performance com o menor número de utilizadores virtuais (10 utilizadores)

é possível constatar os mesmos períodos de processamento máximo mesmo que em intervalos temporais mais reduzidos. Esta alocação máxima de recursos de processamento pode representar uma barreira a eventuais melhorias de performance da aplicação. Nos períodos de utilização máxima de *CPU* não existe margem de melhoria de performance na eventualidade de os utilizadores executarem tarefas sobre a aplicação em simultâneo, visto não poderem ser despendidos mais recursos de processamento. Idealmente a utilização de recursos de processamento nunca deveria atingir valores máximos.

Na Tabela 9 são apresentados os resultados relativos aos tempos médios de processamento para a máquina virtual com a configuração de *hardware* de 4 *CPUs*; *4VCPUs*; 4 *Gigabytes* de *RAM*.

Tabela 9 – Percentagem média de processamento por combinação de utilizadores virtuais

Nº de utilizadores	**Percentagem média de processamento**
10	73
30	93
50	85
70	83
90	82
110	84

Na Figura 29 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 9. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa à percentagem média de processamento é apresentada no eixo das ordenadas (y).

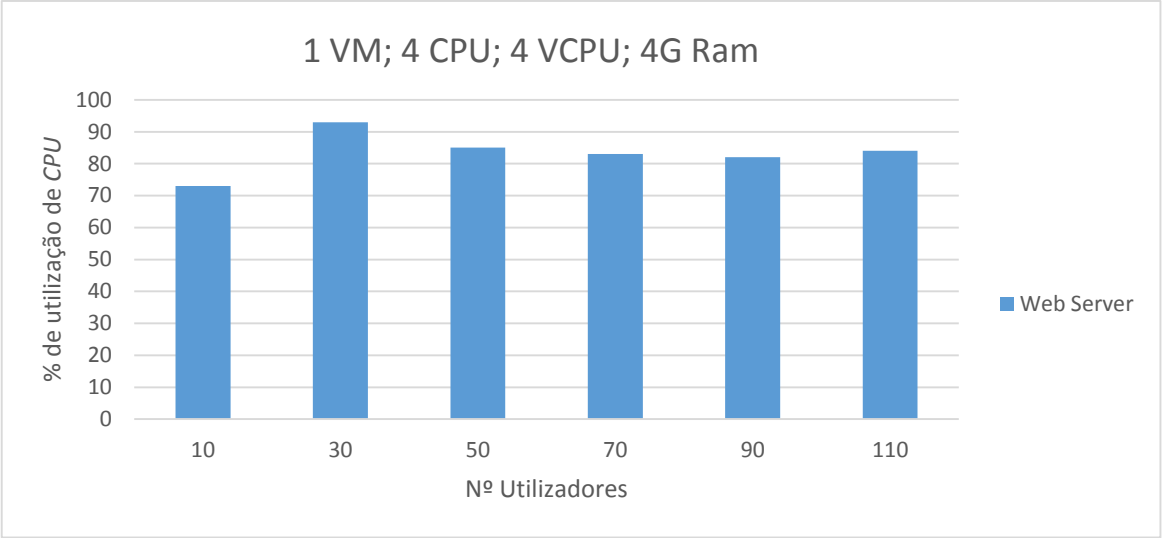

Figura 29 – Percentagem média de processamento por combinação de utilizadores

Como é possível observar pela análise da Figura 29, e para a passagem de 10 para 30 utilizadores, a percentagem média de processamento tende a aumentar à medida que a intensidade de carga de utilização sobre a aplicação aumenta. A percentagem média de processamento tende ainda a estabilizar para cargas de utilização entre 50 a 110 utilizadores. Mesmo para combinações reduzidas de utilizadores a percentagem média de processamento situa-se acima dos 70%.

9.1.1.6 *Resultados de utilização média de memória RAM*

Nos gráficos presentes na Figura 30 e Figura 31 são apresentadas as operações de *Page Fault* que ocorreram, para períodos de quatro minutos, aquando da realização dos testes de carga com o número mínimo e máximo de utilizadores respetivamente.

As operações de *Page Fault* ocorrem quando um processo necessita de aceder a informação presente em memória e o sistema operativo não consegue identificar a localização da informação necessária. Neste sentido as operações de *Page Fault* podem ser subdivididas em duas categorias, *Soft Page Fault* e *Hard Page Fault* [81]. As operações de *Soft Page Fault* ocorrem quando a informação necessária pode ser encontrada noutra posição de memória e não existe a necessidade de acesso ao disco para recuperar a informação. Se a informação necessária não for encontrada e for necessária recorrer a uma operação de leitura do disco então estamos perante uma *Hard Page Fault*. Regra geral uma *Hard Page Fault* é mais penosa em termos de performance pelo facto de as operações de acesso a disco serem, tipicamente, mais lentas.

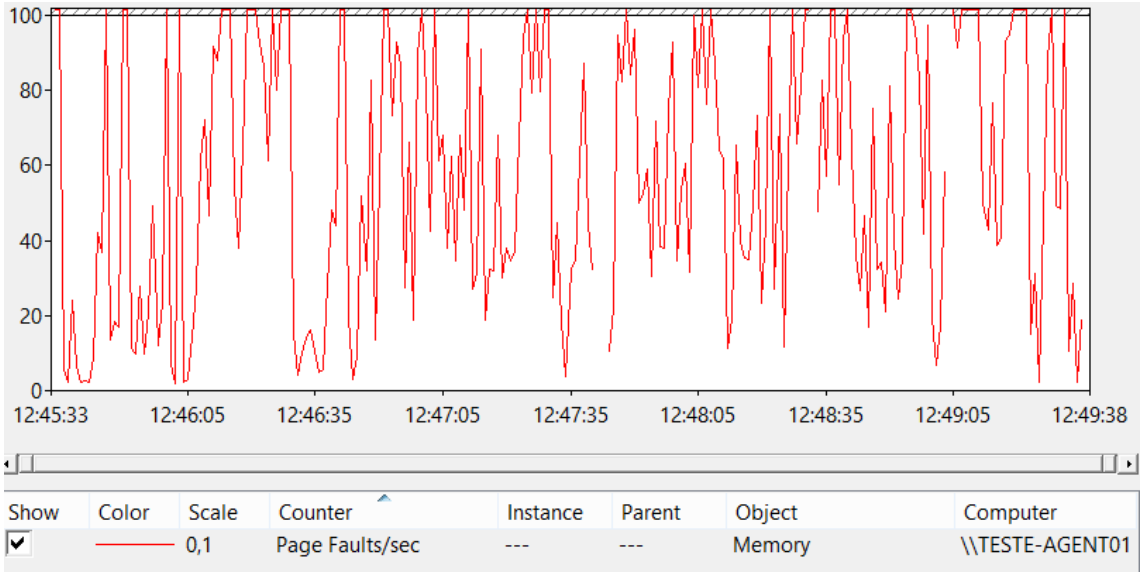

Figura 30 - Page Faults/sec – Teste de carga com 10 utilizadores

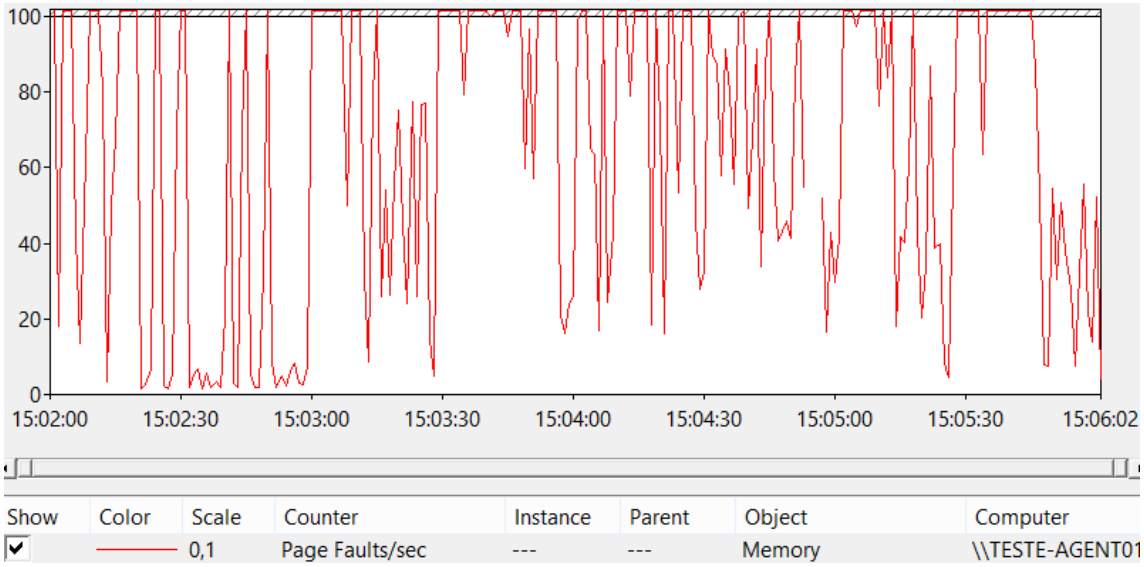

Figura 31 – Page Faults/sec – Teste de carga com 110 utilizadores

Como é possível observar pela análise dos gráficos presentes na Figura 30 e Figura 31 as operações de *Page Fault* ocorrem com alguma frequência durante os períodos de quatro minutos para ambos os testes de carga. Neste sentido estas operações podem, eventualmente, ser responsáveis por quebras de performance da aplicação. No entanto é necessário analisar o tipo de *Page Faults* que estão a ocorrer.

Nos gráficos presentes na Figura 32 e Figura 33 são apresentadas a taxa de escrita para o disco (*Page Writes/sec*) devido a transferências de informação da memória *RAM* para o sistema de armazenamento da máquina virtual; a taxa de leitura de leitura do disco (*Page Reads/sec*) para

recuperação de informação não presente em memória e a taxa geral de transferência de informação entre a memória *RAM* e o sistema de armazenamento da máquina virtual (*Pages/sec*). Tipicamente taxas de transferência, escrita ou leitura elevadas durante longos períodos de tempo podem indicar problemas associados à utilização de memória por parte de uma aplicação [82], [83]. Nestes casos (taxas de transferência, escrita ou leitura elevadas durante longos períodos de tempo) a necessidade de acesso a informação em memória cuja localização não está identificada pelo sistema operativo pode desencadear perdas de performance associadas aos tempos de acesso a informação presente disco.

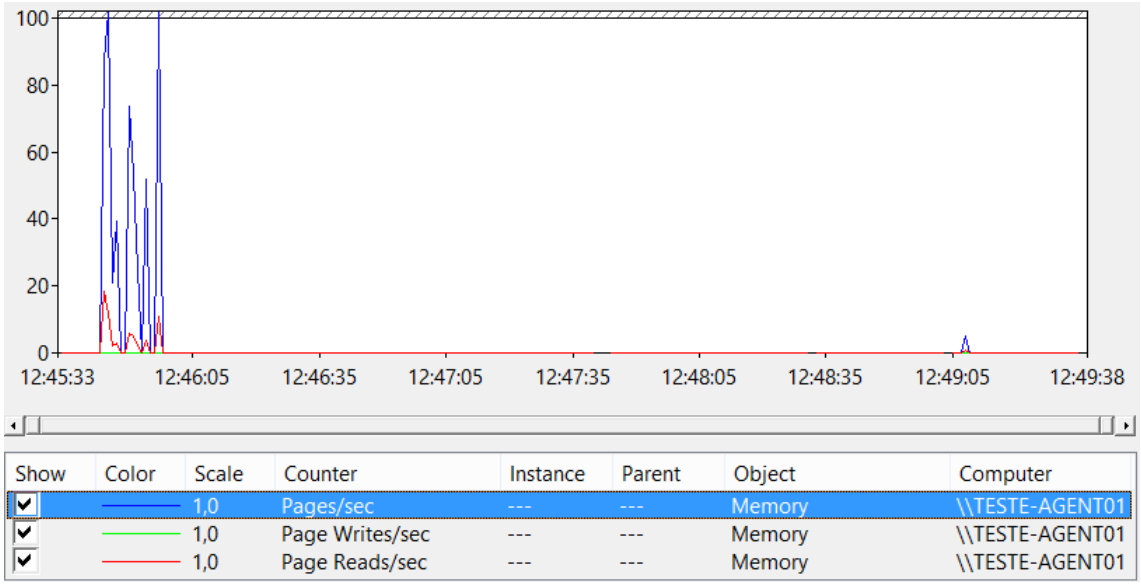

Figura 32 – Operações de leitura e escrita, por segundo, para combinações de 10 utilizadores virtuais

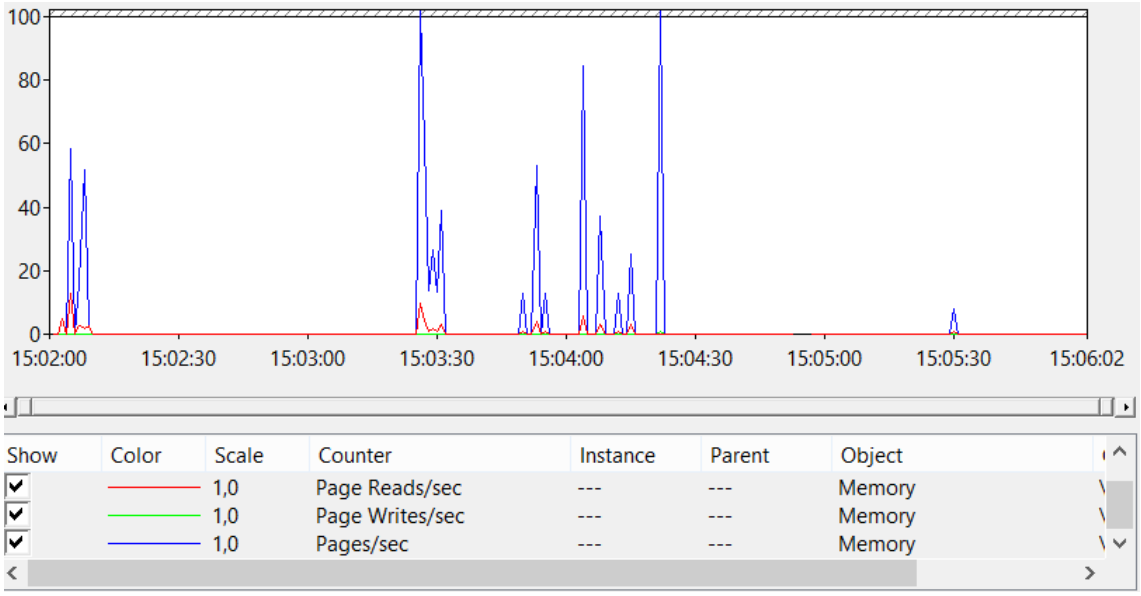

Figura 33 - Operações de leitura e escrita, por segundo, para combinações de 110 utilizadores virtuais

Como é possível observar pela análise dos gráficos presentes na Figura 32 e Figura 33 as taxas de transferência de informação entre o sistema de armazenamento e a memória RAM podem ser consideradas como operações pontuais, mesmo para o teste de carga com o maior número de utilizadores. É possível concluir que as perdas de performance devido à necessidade de acesso a informação em disco é mínima. Este facto sugere uma não relação entre eventuais quebras de performance do SIJ e a quantidade de memória disponível numa qualquer máquina onde a aplicação esteja instalada.

O gráfico presente na Figura 34 representa a taxa média de utilização de memória, para intervalos de quatro minutos, da máquina virtual utilizada no processo de testes de carga para as diferentes combinações de utilizadores virtuais.

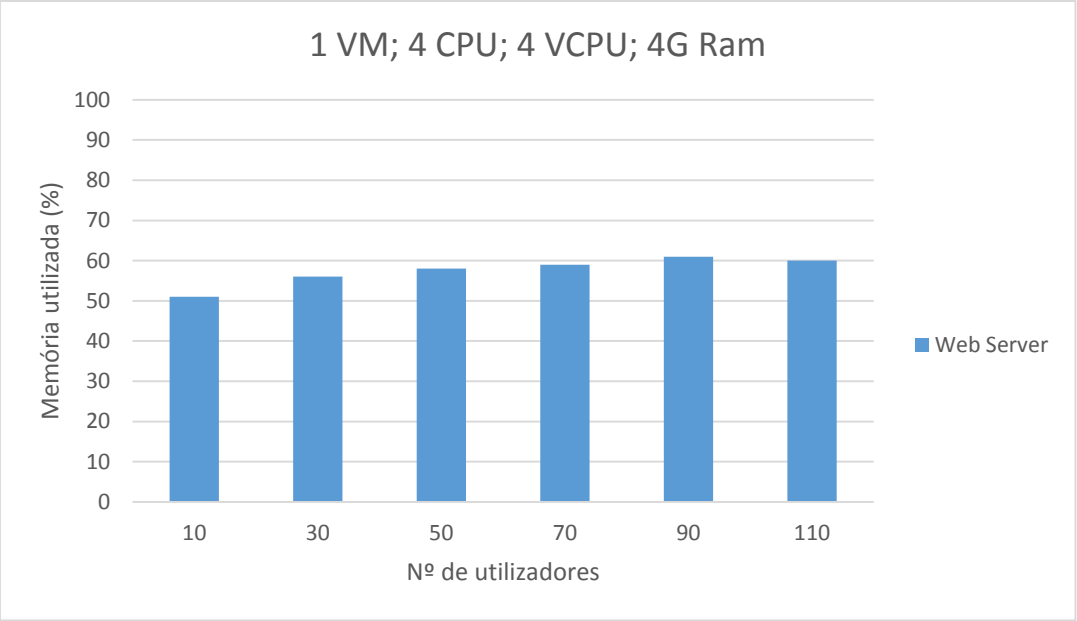

Figura 34 – Taxa média de utilização de memória por testes de carga

Como é possível observar pela análise do gráfico presente na Figura 34 a utilização média de memória apresenta uma ligeira variação, passando de uma utilização média aproximada de 50% para 60% quando a aplicação é testada para a variação mínima e máxima de utilizadores.

Depois de analisados estes fatores (*Page Faults* e utilização média de memória) é possível concluir que o impacto da memória no desempenho geral da aplicação não é substancial, nunca sendo atingidos níveis críticos de utilização de memória nem sendo necessário recorrer a operações de leitura de informação do disco, as quais podem prejudicar a performance do SIJ.

9.1.1.7 *Alocação de CPU*

Na Tabela 10 são apresentados os três processos que contêm a maior percentagem média de processamento (para a configuração de hardware com 4 *CPUs*, 4 *VCPUs* e 4 *Gigabytes de RAM*) durante o período de execução dos testes de carga.

Tabela 10 – Processos com maior percentagem de processamento

Nº utilizadores	**Processo 1**	**Utilização (%)**	**Processo 2**	**Utilização (%)**	**Processo 3**	**Utilização (%)**
10	w3wp	34	MJCVWindowsService	27	sqlservr	5
30	w3wp	41	MJCVWindowsService	39	sqlservr	5

50	MJCVWindowsService	37	w3wp	34	sqlservr	4
70	MJCVWindowsService	36	w3wp	35	sqlservr	3
90	MJCVWindowsService	37	w3wp	31	sqlservr	3
110	w3wp	43	MJCVWindowsService	21	sqlservr	3

O processo designado por "*w3wp"* é o processo do servidor informação de internet (*IIS)* responsável pela gestão de aplicações web. O processo "*MJCVWindowsService*" representa o serviço de gestão de *workflows* utilizado pelo *SIJ*. O processo "*sqlservr*" é o processo utilizado pelo sistema de gestão de base de dados (SGBD).

Invariavelmente, e para todas as combinações de utilizadores, os processos responsáveis pelo maior consumo de recursos de processamento são os processos diretamente relacionados com os três grandes blocos arquiteturais do SIJ (aplicação web, serviço de gestão de *workflows* e base de dados).

9.1.1.8 *Alocação de Memória*

Na Tabela 11 são apresentados os três processos que contêm a maior alocação de memória para o período de execução dos testes de carga.

Tabela 11 – Processos com maior alocação de memória

Nº utilizadores	**Processo 1**	**Alocação (KB)**	**Processo 2**	**Alocação (KB)**	**Processo 3**	**Alocação (KB)**
10	w3wp	556544	MJCVWindowsService	437188	sqlservr	234148
30	w3wp	649928	MJCVWindowsService	563752	sqlservr	252752
50	MJCVWindowsService	617004	w3wp	603024	sqlservr	264976
70	w3wp	655992	MJCVWindowsService	649544	sqlservr	274636
90	MJCVWindowsService	738700	w3wp	650760	sqlservr	278360
110	w3wp	783716	MJCVWindowsService	613556	sqlservr	289332

O processo designado por "w3wp" é o processo do servidor de IIS responsável pela gestão de aplicações web. O processo "MJCVWindowsService" representa o serviço de gestão de workflows utilizado pelo SIJ. O processo "sqlservr" é o processo utilizado pelo SGBD (SQL Server).

Invariavelmente, e à semelhança da alocação de recursos de processamento, os processos responsáveis pelo maior consumo de memória são os processos diretamente relacionados com os três grandes blocos arquiteturais do SIJ.

9.1.2 Resultado de testes de carga a configurações de arquitetura

Para avaliar as vantagens de desacoplar as diferentes partes do *SIJ*, o cenário de testes foi estendido de modo a utilizar várias configurações de máquinas virtuais idênticas (cada uma contendo 4 *CPUs,* 4 *VCPUs* e 4 *Gigabytes* de *RAM*), estando os blocos constituintes do SIJ distribuídos pelas diferentes máquinas virtuais.

Como descrito anteriormente, o SIJ permite desacoplar partes da sua arquitetura por máquinas diferentes. Os três grandes blocos constituintes da sua arquitetura (aplicação web, serviço de gestão de *workflows* e base de dados) podem ser desagrupados e configurados em máquinas independentes. O serviço de gestão de *workflows* pode ainda ser desacoplado em várias unidades ou serviços independentes os quais podem ser instalados e configurados em diferentes máquinas, sendo, deste modo, possível paralelizar as unidades de processamento ao nível deste serviço. A Figura 35 pretende representar as possibilidades de configuração do *SIJ* onde os elementos constituintes se encontram desacoplados por diferentes máquinas. Neste cenário a camada de apresentação encontra-se distribuída por várias máquinas e pode ser acedida utilizando um *load balancer*. O sistema de gestão de *workflows* encontra-se igualmente desacoplado, tendo os seus serviços constituintes distribuídos por máquinas diferentes.

Na Tabela 12 são apresentadas as configurações de arquitetura utilizadas nos testes de carga ao SIJ. Nesta tabela a designação “W*e*b” representa uma máquina virtual utilizada como servidor *web*, a designação “*WF*” designa uma máquina virtual utilizada como serviço de gestão de *workflows* e a designação “BD” designa uma máquina virtual utilizada como servidor de base de dados.

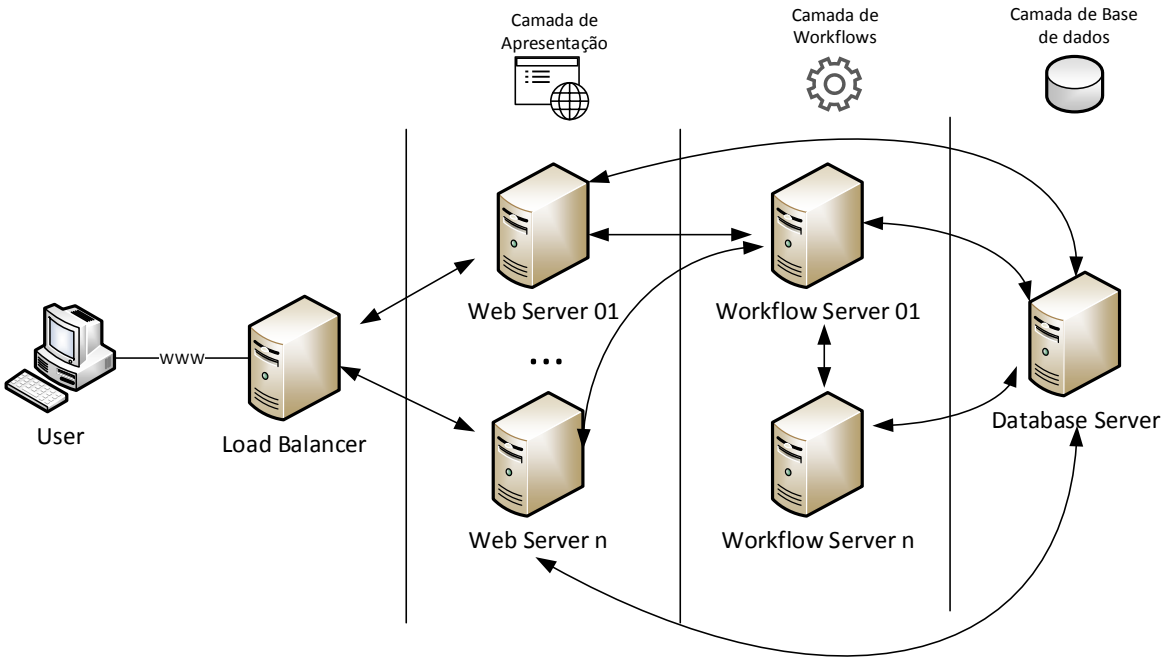

Figura 35 – Configuração do SIJ com os blocos constituintes desacoplados

Tabela 12 – Diferentes configurações do SIJ

Nº *VMs*	VM 1	VM 2	VM 3	VM 4	VM 5	VM 6	VM 7
2	Web	WF + BD	---	---	---	---	---
3	Web	WF	BD	---	---	---	---
4	Web	WF 1	WF 2	BD	---	---	---
5	Web 1	Web 2	WF 1	WF 2	BD	---	---
7	Web 1	Web 2	Web 3	Web 4	WF 1	WF 2	BD

Na primeira sequência de testes, utilizando duas máquinas virtuais, a aplicação web foi configurada numa máquina independente da máquina virtual onde estavam alojados o serviço de gestão de *workflows* e o SGBD.

Na segunda sequência de testes, utilizando três máquinas virtuais, os três blocos principais do *SIJ* (aplicação web, serviço de gestão de workflows e SGBD) foram separados por diferentes máquinas, resultando na utilização de uma máquina responsável pela aplicação web, uma máquina responsável pelo serviço de gestão de *workflows* e uma máquina responsável pelo SGBD.

Na terceira sequência de testes, utilizando quatro máquinas virtuais, o serviço de gestão de *workflows* foi separado por duas máquinas, resultando na utilização de uma máquina responsável

pela aplicação web, duas máquinas responsáveis pelo serviço de gestão de *workflows* e uma máquina responsável pelo SGBD.

Na quarta sequência de testes, utilizando cinco máquinas virtuais, o acesso à aplicação web foi subdividido por duas máquinas virtuais recorrendo-se à utilização de um *Load Balancer* como elemento de distribuição de carga. Esta separação resultou na utilização de duas máquinas responsáveis pela aplicação web, duas máquinas responsáveis pelo serviço de gestão de *workflows* e uma máquina responsável pelo SGBD.

Na quinta sequência de testes, utilizando sete máquinas virtuais, o acesso à aplicação web foi subdividido por quatro máquinas virtuais recorrendo-se à utilização de um *Load Balancer* como elemento de distribuição de carga. Esta separação resultou na utilização de quatro máquinas responsáveis pela aplicação web, duas máquinas responsáveis pelo serviço de gestão de workflows e uma máquina responsável pelo SGBD.

9.1.2.1 *Tempo médio de execução de teste de performance web*

A Tabela 13 contém os resultados relativos aos tempos médios de execução por teste de performance *web* (em segundos) para as diferentes combinações de arquitetura e número de utilizadores virtuais. Nestes resultados são incluídos os resultados para a configuração de *hardware* com os melhores resultados gerais (1 *VM*; 4 *CPUs;* 4 *VCPUs;* 4 *Gigabytes* de *RAM*), servindo como elemento de comparação entre as configurações de *hardware* e arquitetura.

Tabela 13 - Tempo médio por teste para diferentes configurações de hardware

Nº utilizadores	**1 VM**	**2 VMs**	**3 VMs**	**4 VMs**	**5 VMs**	**7 VMs**
10	24,5	23,6	22,8	23,3	22,8	21,9
30	63,5	43,2	41,7	38,5	29	35,8
50	129	104	96,9	79,2	72,1	72,7
70	191	163	141	134	110	100
90	---	229	202	198	156	126
110	---	---	---	248	222	158
130	---	---	---	---	---	182
150	---	---	---	---	---	194

Na Figura 36 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 13. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no

eixo das abscissas (x) e a informação relativa ao tempo médio por teste é apresentada no eixo das ordenadas (y).

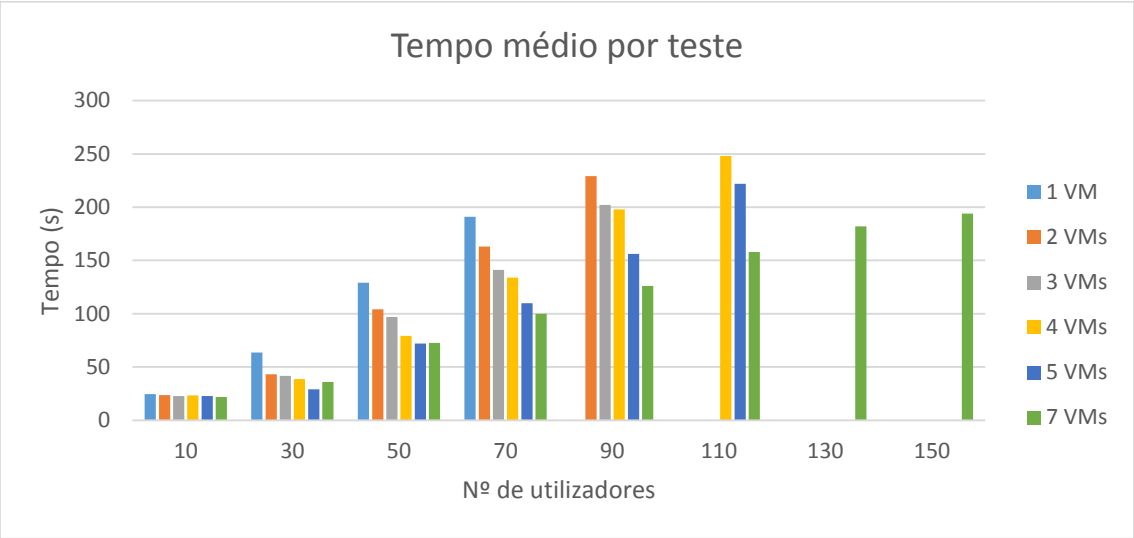

Figura 36 – Tempo médio por teste para diferentes configurações de hardware

Pela análise do gráfico presente na Figura 36 é possível concluir que a separação dos diferentes blocos do SIJ por máquinas diferentes permite aumentar a celeridade com que é possível conduzir tarefas dentro da aplicação.

A distribuição da aplicação web por várias máquinas, utilizando um sistema de *load balancing*, permitiu não só melhorar os tempos necessários à conclusão de tarefas como também permitiu aumentar o número de utilizadores que a aplicação suporta em simultâneo. Neste sentido é possível afirmar que a performance geral da aplicação está intrinsecamente ligada ao desempenho geral dos servidores web.

9.1.2.2 *Tempos médios de obtenção de páginas web*

Na Tabela 14 são apresentados os tempos médios de obtenção de páginas web para diferentes combinações de *hardware* e diferentes combinações de utilizadores.

Tabela 14 – Tempo médio de obtenção de páginas web

Nº utilizadores	**1 VM**	**2 VMs**	**3 VMs**	**4 VMs**	**5 VMs**	**7 VMs**

10	0,65	0,58	0,51	0,56	0,48	0,44
30	5,12	2,73	2,6	2,31	1,25	0,63
50	11,7	9,66	8,22	7,08	6,17	1,99
70	18,7	14,5	12,7	16,4	11,4	5,6
90	29,1	22,6	20,8	21,6	15,1	8,25
110	28,2	27,3	28,9	33,5	23,1	9,77
130	---	30,1	28,1	28,1	22,7	13,7
150	---	---	29,4	27,9	24	21,8
170	---	---	---	---	---	22,9

Na Figura 37 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 14. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa ao tempo médio para a obtenção de páginas web é apresentada no eixo das ordenadas (y).

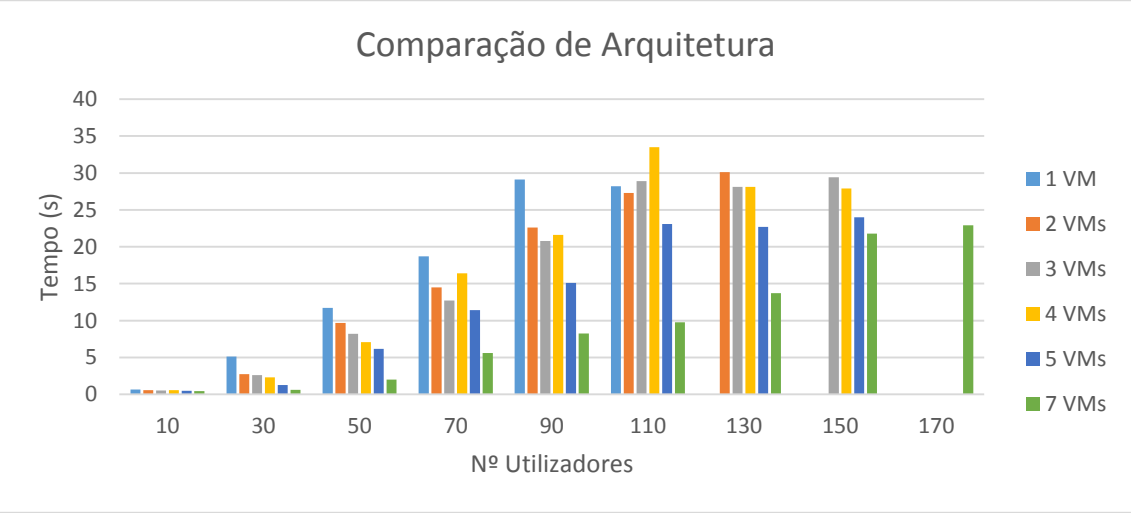

Figura 37 – Tempo médio de obtenção de páginas web

Através da análise do gráfico da Figura 37 é possível constatar que os tempos de obtenção mais curtos pertencem à configuração de arquitetura de onde são utilizados o maior número de máquinas virtuais. Neste sentido é novamente relevante salientar a importância do balanceamento de carga entre as máquinas que alojam a aplicação *web*. Este impacto na capacidade de resposta da aplicação é igualmente verificável quando o balanceamento de carga web é separado de uma para duas máquinas e eventualmente de duas para quatro máquinas virtuais.

9.1.2.3 *Tempo médio de resposta a* pedidos

Na Tabela 15 são apresentados os tempos médios de obtenção de respostas [55] para as diferentes combinações de arquitetura e diferentes combinações de utilizadores.

Tabela 15 – Tempo médio de resposta a requests

Nº utilizadores	1 VM	2 VMs	3 VMs	4 VMs	5 VMs	7 VMs
10	0,1	0,048	0,036	0,042	0,035	0,031
30	0,37	0,22	0,22	0,18	0,12	0,044
50	0,9	0,71	0,67	0,6	0,72	0,16
70	1,41	1,15	1,06	1,4	1,32	0,62
90	2,7	1,98	1,99	2,03	1,58	0,94
110	4,01	2,52	2,86	2,91	1,93	1,15
130	---	3,12	3,22	3,26	3,18	1,34
150	---	---	3,51	3,1	3,42	2,47
170	---	---	---	---	---	2,14

Na Figura 38 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 15. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa ao tempo médio de resposta a pedidos é apresentada no eixo das ordenadas (y).

Tal como descrito nas análises anteriores também aqui os menores tempos médios de resposta podem ser encontrados para a combinação de arquitetura onde são utilizadas sete máquinas virtuais. Neste sentido é igualmente possível salientar a importância do balanceamento de carga entre as máquinas onde a aplicação web está alojada.

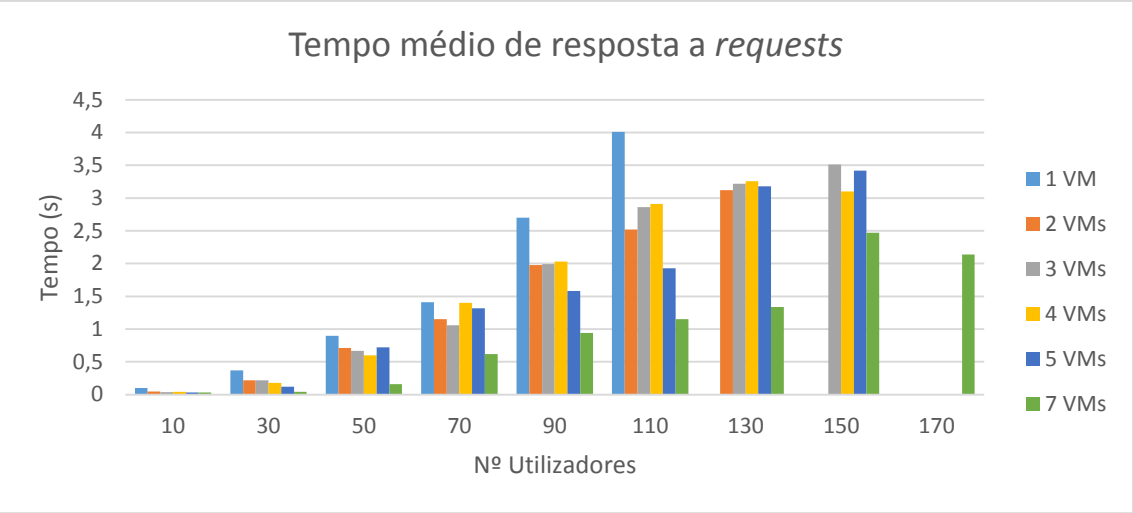

Figura 38 – Tempo médio de resposta a requests

De entre todas as configurações testadas, a configuração que apresentou, de forma consistente, os maiores tempos de resposta foi aquela em que todos os blocos constituintes do *SIJ* se encontravam alojados apenas numa máquina virtual. Deste modo é possível concluir que desacoplar os blocos constituintes do SIJ é uma opção que permite melhorar a performance do mesmo.

Para o limite máximo de utilizadores suportados pela configuração de uma máquina virtual (110 utilizadores) foi ainda possível obter tempos de resposta 3,4 vezes menores com a utilização de um sistema de *load balancing* e quatro servidores web. Neste sentido é possível afirmar que o ato de desacoplar a aplicação web é um dos elementos essenciais na melhoria do desempenho de todo o sistema.

9.1.2.4 *Capacidade de resposta a carga*

À semelhança dos testes de carga a configurações de *hardware* foram igualmente recolhidos dados relativos à quantidade de informação (despachos, petições iniciais, pedidos de habeas corpus e autos iniciais) que é possível inserir no SIJ, num período de quatro minutos, para diferentes combinações de utilizadores. Estes dados foram recolhidos para as configurações de arquitetura apresentadas anteriormente. Nestes resultados são incluídos os valores para a configuração de *hardware* com os melhores resultados gerais (1 *VM*; 4 *CPUs*; 4 *VCPUs*; 4 *Gigabytes* de *RAM*), servindo como elemento de comparação entre as configurações de *hardware* e arquitetura.

Na Tabela 16 são apresentados os resultados relativos à quantidade de informação que é possível colocar na base de dados do SIJ para as diferentes combinações de arquitetura.

Tabela 16 – Capacidade de resposta a carga

Nº utilizadores	1 VM	2 VMs	3 VMs	4 VMs	5 VMs	7 VMs
10	50	53	67	61	53	55
30	78	107	103	109	129	84
50	71	81	86	118	120	125
70	65	109	109	105	105	122
90	63	72	72	74	145	130
110	72	88	83	76	87	129
130	---	102	88	92	101	148
150	---	---	86	105	99	114
170	---	---	---	---	---	126

Na Figura 39 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 16. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa à totalidade de dados que é possível guardar na base de dados é apresentada no eixo das ordenadas (y).

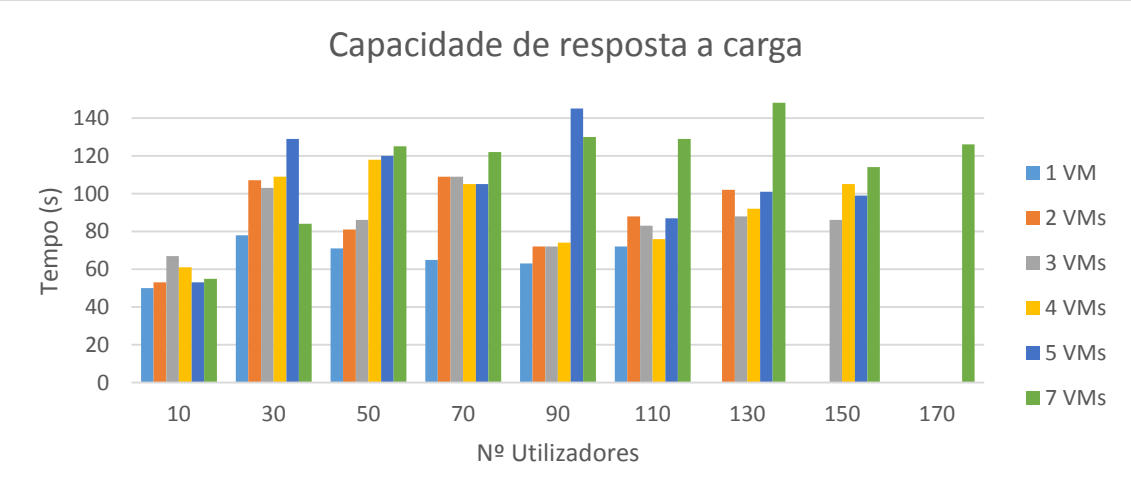

Figura 39 – Capacidade de resposta a carga

Através da análise do gráfico da Figura 39 é possível verificar que as configurações de arquitetura onde foram utilizados *load balancers* obtiveram melhores resultados nos cenários onde foram utilizados 50, 90, 110, 130 e 170 utilizadores virtuais (55.6 % dos casos).

De todas as configurações de arquitetura apresentadas o menor desempenho pertence à arquitetura onde toda a aplicação estava alojada apenas numa única máquina.

O ponto de teste mais estável, de entre todas as configurações, foi encontrado quando foi simulada carga com 70 utilizadores virtuais, existindo apenas uma variação de 13 unidades para as configurações em que foram utilizadas mais do que uma máquina virtual.

9.1.2.5 *Resultados médios de processamento para duas VMs*

Na Tabela 17 são apresentados os resultados relativos aos tempos médios de processamento para as diferentes máquinas virtuais utilizadas no processo de testes de carga. Nesta tabela a designação "*Web*" representa a máquina virtual onde a aplicação web se encontra alojada. A designação "*WF + BD*" representa a máquina virtual onde estão alojados o serviço de gestão de *workflows* e o SGBD.

Na Figura 40 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 17. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa à percentagem média de processamento por máquina virtual é apresentada no eixo das ordenadas (y).

Tabela 17 – Percentagem média de processamento por máquina virtual

Nº Utilizadores	Web	WF + BD
10	51	33
30	74	60
50	59	55
70	55	48
90	48	51
110	50	51
130	55	39

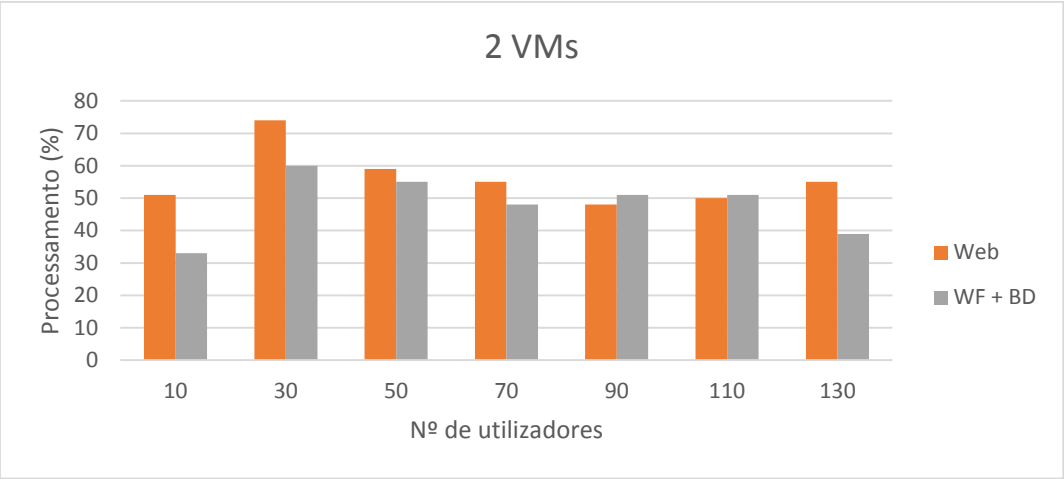

Figura 40 - Percentagem média de processamento por máquina virtual

Pela análise do gráfico presente na Figura 40 é possível concluir que, na generalidade dos testes de carga conduzidos a esta configuração de arquitetura, a máquina que aloja o servidor web contém o maior esforço de processamento por intervalo de tempo (71,4 % correspondendo a 5 dos 7 casos).

O maior esforço de processamento foi atingido, para ambas as máquinas virtuais, quando o SIJ foi sujeite a uma carga de utilização de 30 utilizadores em simultâneo.

A Figura 41 e Figura 42 representam a percentagem total de processamento, durante um período de quatro minutos, para a carga de utilização mais intensa (130 utilizadores). Estes dados foram recolhidos nas duas máquinas virtuais utilizadas (servidor web e servidor de workflows).

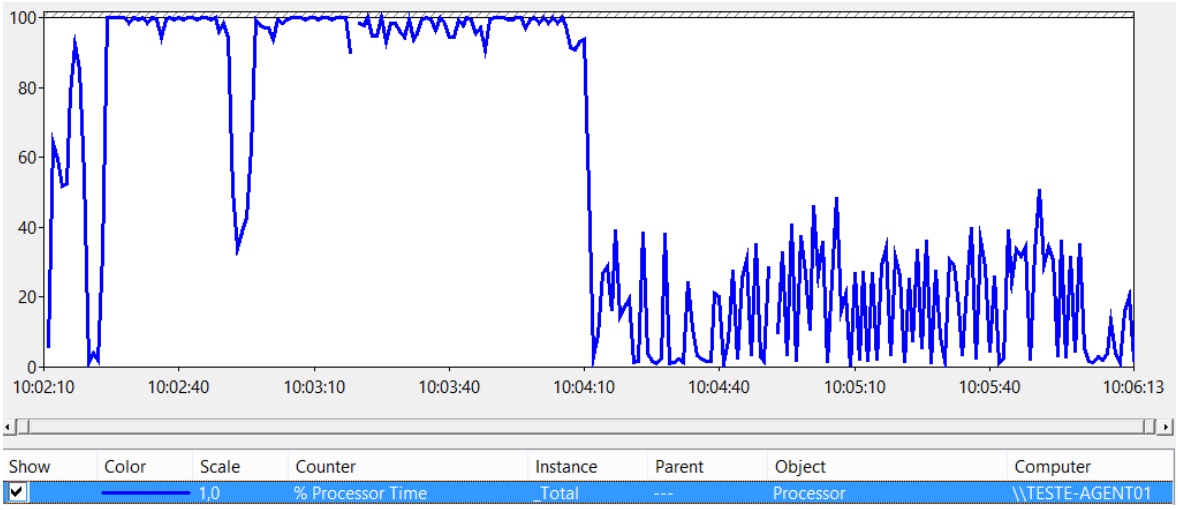

Figura 41 - Percentagem total de processamento para 130 utilizadores virtuais – Servidor Web

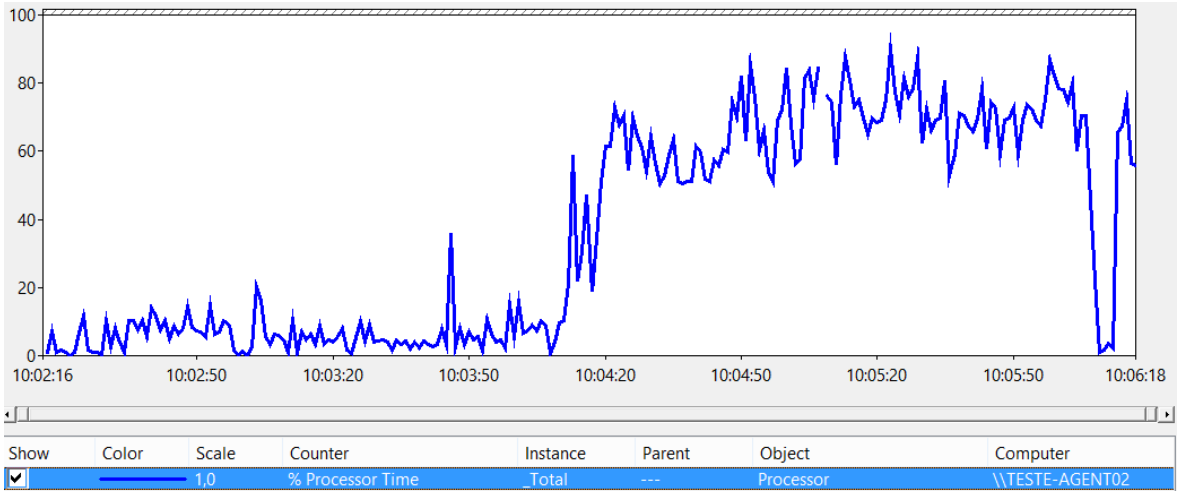

Figura 42 - Percentagem total de processamento para 130 utilizadores virtuais – Servidor de Workflows

Como é possível observar existe uma interdependência entre as duas VMs. Os períodos de maior intensidade de processamento de uma VM são os períodos de menor intensidade da outra. Neste cenário, se uma das VMs atingir o seu ponto de saturação em termos de processamento irá afetar a performance geral da aplicação, independentemente da VM restante estar ou não sobre utilização intensiva.

É ainda possível constatar que a VM onde é sentida, de forma mais severa, a carga de utilização é aquela onde está alojada a aplicação web. Neste cenário esta VM foi a única onde foram registados picos de utilização de 100% durante as fases iniciais da experiência de carga.

Não foram identificados picos de processamento de 100% na VM onde está alojada o serviço de gestão de *workflows*. Este facto sugere que, para a fase inicial da experiência, o elemento limitador da performance da aplicação (apenas para este cenário) é a aplicação web.

9.1.2.6 *Resultados de utilização média de memória RAM para duas VMs*

Na Tabela 18 são apresentados os resultados relativos à utilização média de memória para as diferentes máquinas virtuais utilizadas no processo de testes de carga. Nesta tabela, a designação "*Web*" representa a máquina virtual onde a aplicação web se encontra alojada. A designação "*WF + BD*" representa a máquina virtual onde estão alojados o serviço de gestão de *workflows* e o SGBD.

Tabela 18 - Percentagem média de utilização de memória

Nº Utilizadores	Web	WF + BD

10	63	38
30	65	41
50	46	45
70	50	47
90	48	49
110	47	53
130	49	58

Na Figura 43 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 18. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa à utilização média de memória por máquina virtual é apresentada no eixo das ordenadas (y).

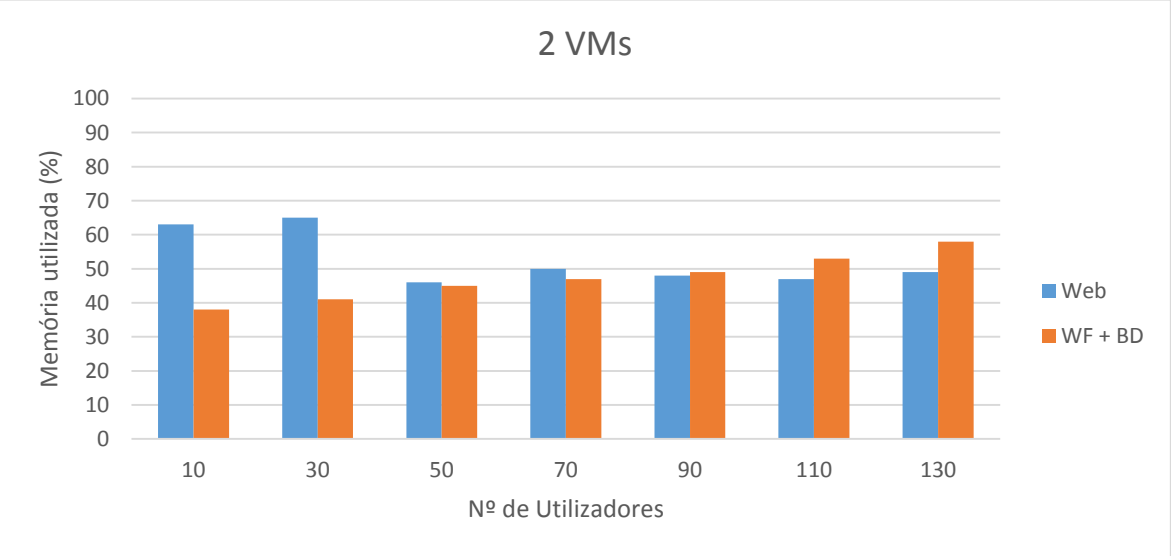

Figura 43 - Percentagem média de utilização de memória

Pela análise do gráfico presente na Figura 43 é possível concluir que para os testes de carga até 70 utilizadores (51.7% dos casos) o servidor web foi aquele que obteve a maior média de utilização de memória. É também observável que a utilização média de memória é relativamente estável (entre os 46% e 50%) para uma carga de utilização entre os 50 e 130 utilizadores virtuais.

A máquina virtual que aloja o serviço de gestão de workflows e o SGBD apresenta um nível médio de utilização de memória com tendência crescente à medida que a carga de utilização sobre a aplicação aumenta, apresentado uma variação de 38% para 58%.

9.1.2.7 *Resultados médios de processamento para três VMs*

Na Tabela 19 são apresentados os resultados relativos aos tempos médios de processamento para as diferentes máquinas virtuais utilizadas no processo de testes de carga. Nesta tabela a designação "*Web*" representa a máquina virtual onde a aplicação web se encontra alojada. A designação "*WF*" representa a máquina virtual onde está alojado o serviço de gestão de *workflows* e a designação "BD" representa a máquina virtual onde está alojado o SGBD.

Tabela 19 - Percentagem média de processamento por máquina virtual

Nº Utilizadores	Web	WF	BD
10	52	28	10
30	73	52	14
50	61	49	14
70	54	44	12
90	50	42	11
110	48	43	11
130	50	45	10
150	52	28	10

Na Figura 44 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 19. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa à percentagem média de processamento por máquina virtual é apresentada no eixo das ordenadas (y).

Pela análise do gráfico presente na Figura 44 é possível concluir que, para 100% dos casos, a máquina que aloja o servidor web contém o maior esforço de processamento por intervalo de tempo.

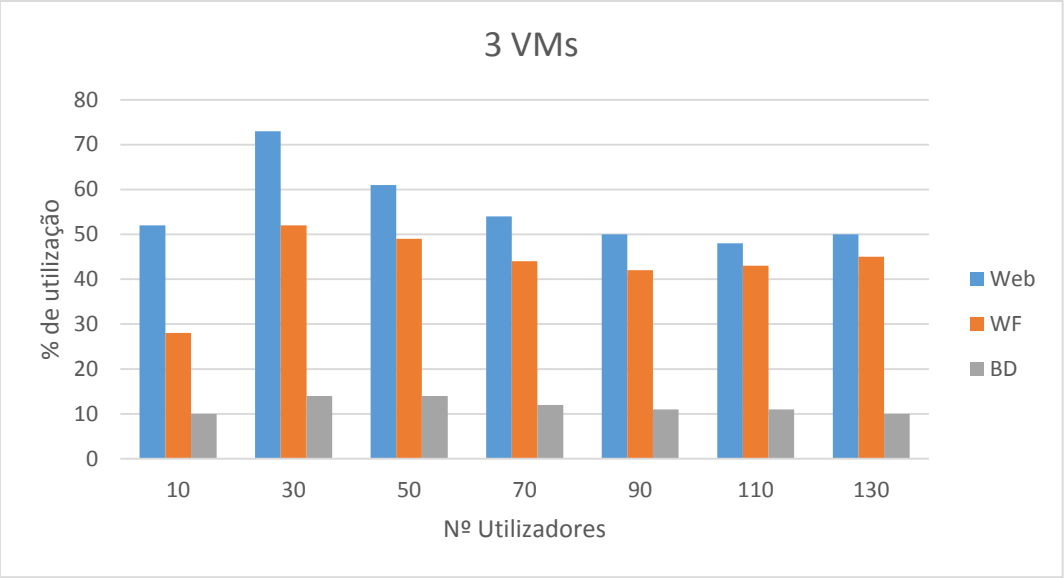

Figura 44 - Percentagem média de processamento por máquina virtual

A máquina virtual que aloja o serviço de gestão de base de dados contém, para 100% dos casos, o menor nível de processamento médio. Este nível de processamento manteve-se ainda constante para os diferentes testes de carga.

A máquina virtual que aloja o serviço de gestão de *workflows* apresentou o nível de processamento intermédio, sendo que este nível médio de processamento tende a aproximar-se da máquina virtual que aloja a aplicação web à medida que o número de utilizadores virtuais aumenta.

O maior esforço de processamento foi atingido quando o SIJ foi sujeito a uma carga de utilização de 30 utilizadores em simultâneo.

A Figura 45, Figura 46 e Figura 47 representam a percentagem total de processamento, durante um período de quatro minutos, para a carga de utilização mais intensa (130 utilizadores). Estes dados foram recolhidos nas três máquinas virtuais utilizadas (servidor web e servidor de workflows e servidor de base de dados).

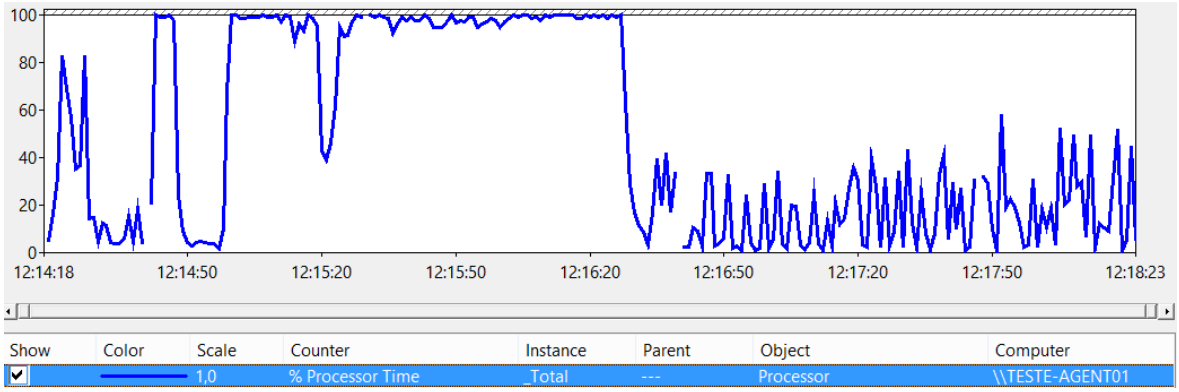

Figura 45 - Percentagem total de processamento para 130 utilizadores virtuais – Servidor Web

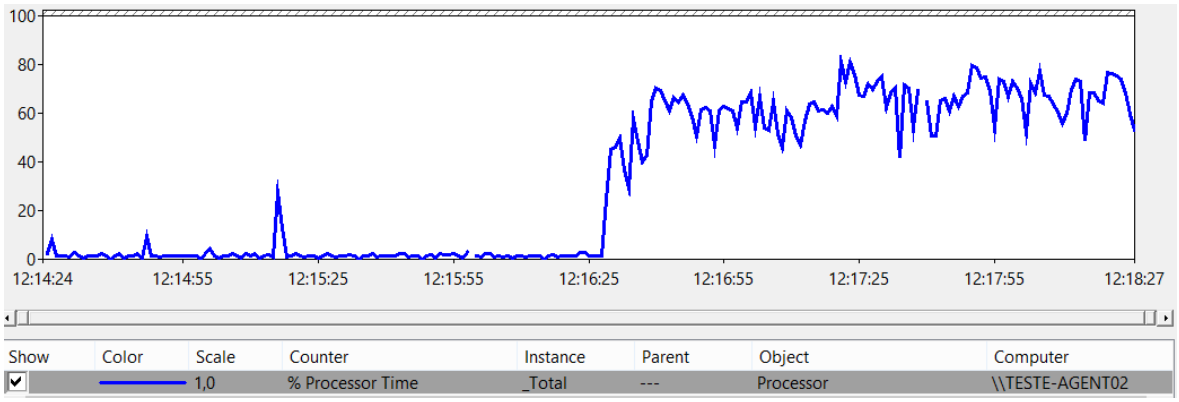

Figura 46 - Percentagem total de processamento para 130 utilizadores virtuais – Servidor de Workflows

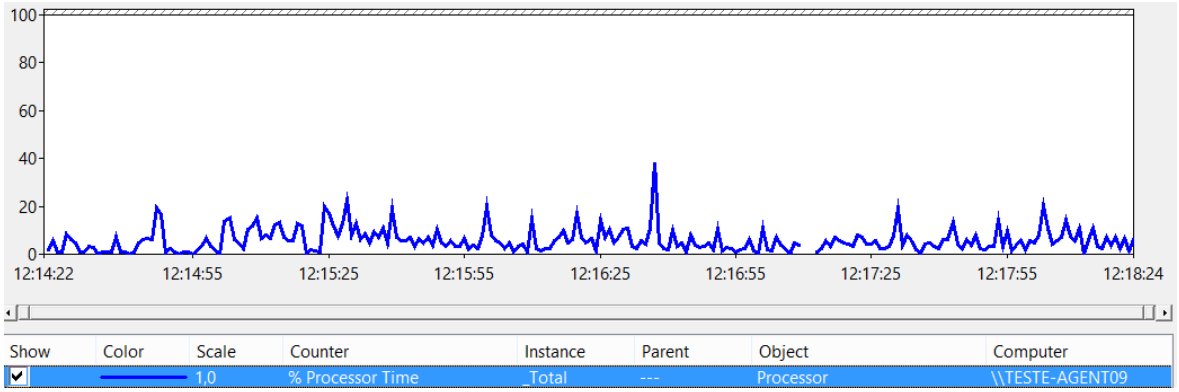

Figura 47 - Percentagem total de processamento para 130 utilizadores virtuais – Servidor de BD

Á semelhança do cenário para duas máquinas virtuais existe uma interdependência entre as duas VMs (servidor web e servidor de BD). Os períodos de maior intensidade de processamento de uma delas corresponde ao período de menor intensidade da outra. Neste cenário, se uma das VMs atingir o seu ponto de saturação em termos de processamento irá afetar a performance geral da aplicação, independentemente da VM restante estar ou não sobre utilização intensiva.

Novamente a VM onde é sentida, de forma mais severa, a carga de utilização é aquela onde está alojada a aplicação web.

É ainda possível constatar que o desacoplamento da base de dados para uma terceira VM têm impactos marginais na performance da VM onde está alojado o serviço de gestão de *workflows*.

9.1.2.8 *Resultados de utilização média de memória RAM para três VMs*

Na Tabela 20 são apresentados os resultados relativos à utilização média de memória para as diferentes máquinas virtuais utilizadas no processo de testes de carga. Nesta tabela, a designação "Web" representa a máquina virtual onde a aplicação web se encontra alojada. A designação "WF" representa a máquina virtual onde está alojado o serviço de gestão de *workflows*. A designação "BD" representa a máquina virtual onde está alojado o SGBD.

Tabela 20 - Percentagem média de utilização de memória

Nº Utilizadores	Web	WF	BD
10	47	44	32
30	50	46	33
50	52	49	34
70	50	52	34
90	51	53	34
110	53	54	35
130	50	56	35

Na Figura 48 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 20. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa à utilização média de memória por máquina virtual é apresentada no eixo das ordenadas (y).

Pela análise do gráfico presente na Figura 48 é possível concluir que a percentagem média de utilização de memória para o servidor web foi relativamente constante para todos os testes de carga, contendo uma variação entre 47% e 53%.

A máquina virtual que aloja o serviço de gestão de workflows apresenta um nível médio de utilização de memória com tendência crescente à medida que a carga de utilização sobre a aplicação aumenta, apresentando uma variação de 44% para 56%.

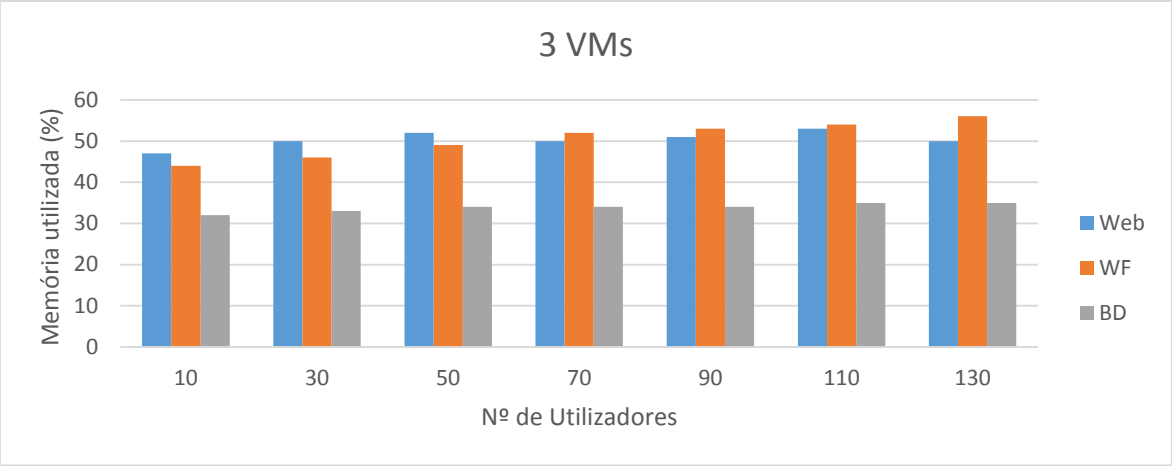

Figura 48 - Percentagem média de utilização de memória

A variação da percentagem média de utilização de memória, para a máquina virtual que aloja o SGBD, é pouco acentuada, mostrando uma tendência crescente de 32% para 35%.

9.1.2.9 *Resultados médios de processamento para quatro VMs*

Na Tabela 21 são apresentados os resultados relativos aos tempos médios de processamento para as diferentes máquinas virtuais utilizadas no processo de testes de carga. Nesta tabela, a designação "Web" representa a máquina virtual onde a aplicação web se encontra alojada. As designações "WF 1" e "WF 2" representam as máquinas virtuais onde estão alojados os serviços de gestão de *workflows* e a designação "BD" representa a máquina virtual onde está alojado o SGBD.

Tabela 21 - Percentagem média de processamento por máquina virtual

Nº Utilizadores	Web	WF 1	WF 2	BD
10	52	17	20	13
30	78	31	35	17
50	65	30	36	16
70	65	22	24	13
90	58	25	28	13
110	60	14	18	12
130	52	23	28	11
150	58	21	29	12

Na Figura 49 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 21. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no

eixo das abscissas (x) e a informação relativa à percentagem média de processamento por máquina virtual é apresentada no eixo das ordenadas (y).

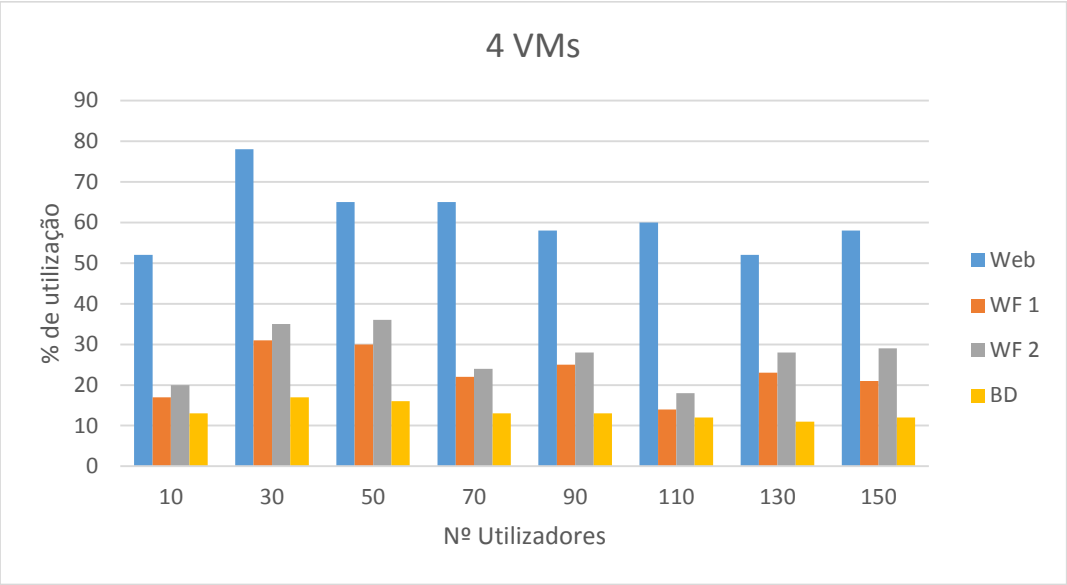

Figura 49 - Percentagem média de processamento por máquina virtual

Pela análise do gráfico presente na Figura 49 é possível concluir que, para 100% dos casos, a máquina que aloja o servidor web contém o maior esforço de processamento por intervalo de tempo.

A máquina virtual que aloja o serviço de gestão de base de dados contém, novamente para 100% dos casos, o menor nível de processamento médio. Este nível de processamento manteve-se ainda relativamente constante para os diferentes testes de carga.

As máquinas virtuais que alojam o serviço de gestão de workflows apresentam o nível de processamento intermédio, tendo ainda, entre elas, níveis médios de processamento bastante semelhantes.

O maior esforço de processamento foi atingido quando o SIJ foi sujeite a uma carga de utilização de 30 utilizadores em simultâneo. Este esforço máximo de processamento (para 30 utilizadores) pode ser explicado pelo facto de não existir a necessidade de recorrer a operações de I/O. Para cargas de utilização superiores é necessário recorrer a operações de leitura e escrita para o sistema de armazenamento, resultando numa diminuição da carga de processamento geral.

9.1.2.10*Resultados de utilização média de memória RAM para quatro VMs*

Na Tabela 22 são apresentados os resultados relativos à utilização média de memória para as diferentes máquinas virtuais utilizadas no processo de testes de carga. Nesta tabela a designação "Web" representa a máquina virtual onde a aplicação web se encontra alojada. As designações "WF 1" e "WF 2" representam as máquinas virtuais onde estão alojados os serviços de gestão de *workflows*. A designação "BD" representa a máquina virtual onde está alojado o SGBD.

Tabela 22 - Percentagem média de utilização de memória

Nº Utilizadores	Web	WF 1	WF 2	BD
10	47	42	35	36
30	51	42	35	37
50	52	44	37	37
70	50	45	39	37
90	50	45	40	37
110	48	47	44	36
130	49	49	44	36
150	51	49	46	37

Na Figura 50 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 22. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa à utilização média de memória por máquina virtual é apresentada no eixo das ordenadas (y).

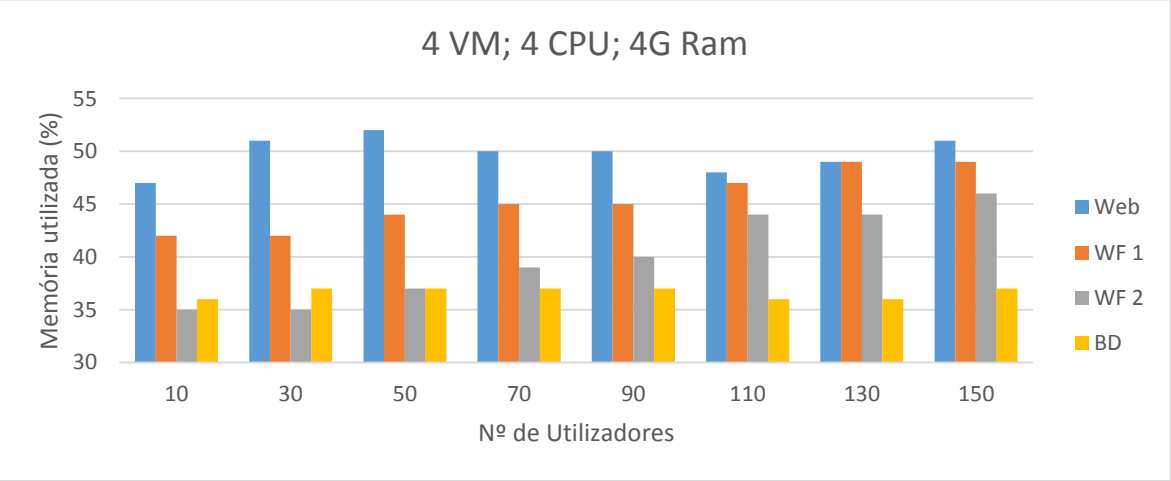

Figura 50 - Percentagem média de utilização de memória

Pela análise do gráfico presente na Figura 50 é possível concluir que a percentagem média de utilização de memória para o servidor web foi relativamente constante para todos os testes de carga, contendo uma variação entre 47% e 51%.

Ambas as máquinas virtuais que alojam o serviço de gestão de workflows apresentam um nível médio de utilização de memória com tendência crescente à medida que a carga de utilização sobre a aplicação aumenta, apresentado uma variação de 42% para 59 e de 35% para 46% respetivamente.

A variação da percentagem média de utilização de memória, para a máquina virtual que aloja o SGBD, é praticamente inexistente, apresentando uma variação de 36% para 37%.

9.1.2.11*Resultados médios de processamento para cinco VMs*

Na Tabela 23 são apresentados os resultados relativos aos tempos médios de processamento para as diferentes máquinas virtuais utilizadas no processo de testes de carga. Nesta tabela as designações "Web 1" e "Web 2" representa as máquinas virtuais onde a aplicação web se encontra alojada. As designações "WF 1" e "WF 2" representa as máquinas virtuais onde estão alojados os serviços de gestão de *workflows* e a designação "BD" representa a máquina virtual onde está alojado o SGBD.

Tabela 23 - Percentagem média de processamento por máquina virtual

Nº Utilizadores	Web 1	Web 2	WF 1	WF 2	BD
10	30	25	17	20	13
30	58	49	38	44	22
50	48	54	30	34	17
70	39	39	31	37	15
90	38	50	30	33	17
110	37	44	30	27	10
130	30	56	38	34	13
150	34	50	34	31	12

Na Figura 51 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 23. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa à percentagem média de processamento por máquina virtual é apresentada no eixo das ordenadas (y).

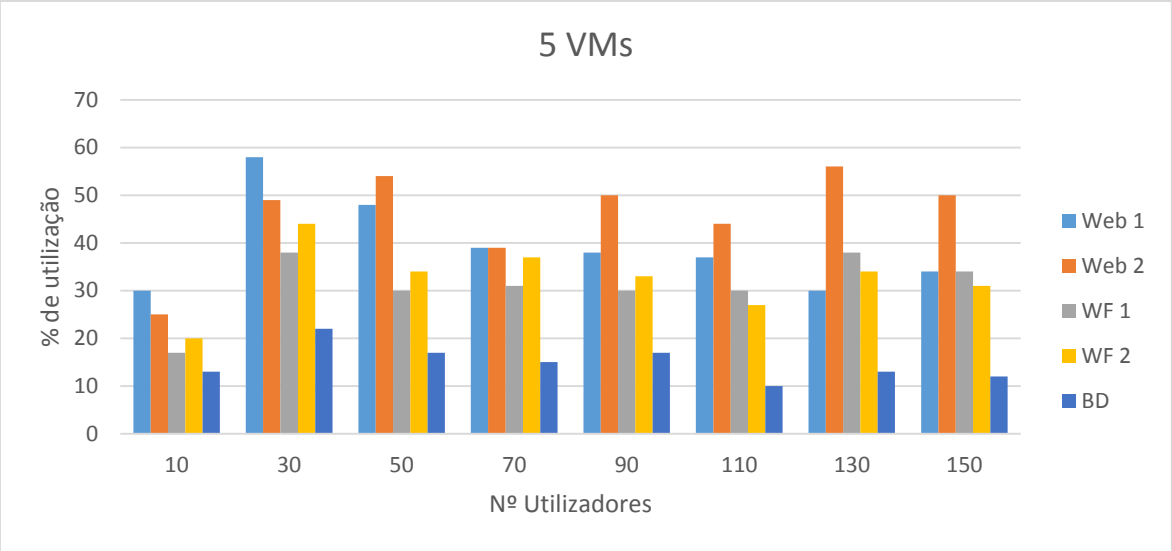

Figura 51 - Percentagem média de processamento por máquina virtual

Pela análise do gráfico presente na Figura 51 é possível concluir que, para 87.5% dos casos, as máquinas que alojam os servidores web contêm o maior esforço de processamento por intervalo de tempo (com a exceção do teste de carga onde foram simulados 130 utilizadores). Utilizando um sistema de *load balancing* como elemento de balanceamento de carga entre as duas máquinas que alojam a aplicação web permitiu reduzir a percentagem média de processamento, a qual na configuração anterior residia entre os 50% a 80%, para valores que se situam entre os 25% a 58% de percentagem média de processamento.

A máquina virtual que aloja o serviço de gestão de base de dados contém, para 100% dos casos, o menor nível de processamento médio. Este nível de processamento manteve-se ainda relativamente constante para os diferentes testes de carga (variando entre 10% e 22%).

As máquinas virtuais que alojam o serviço de gestão de workflows apresentam o nível de processamento intermédio, tendo ainda, entre elas, níveis médios de processamento bastante semelhantes.

O maior esforço de processamento foi atingido quando o SIJ foi sujeite a uma carga de utilização de 30 utilizadores em simultâneo.

9.1.2.12*Resultados de utilização média de memória RAM para cinco VMs*

Na Tabela 24 são apresentados os resultados relativos à utilização média de memória para as diferentes máquinas virtuais utilizadas no processo de testes de carga. Nesta tabela a designação

"Web 1" e "Web 2" representam as máquinas virtuais onde a aplicação web se encontra alojada. A designação "WF 1" e "WF 2" representam as máquinas virtuais onde estão alojados os serviços de gestão de workflows. A designação "BD" representa a máquina virtual onde está alojado o SGBD.

Tabela 24 - Percentagem média de utilização de memória

Nº Utilizadores	Web 1	Web 2	WF 1	WF 2	BD
10	43	43	51	50	37
30	45	43	52	52	37
50	45	41	51	50	37
70	47	42	54	59	38
90	46	44	58	59	38
110	44	38	59	61	39
130	46	43	59	59	39
150	45	41	60	64	39

Na Figura 52 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 24. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa à utilização média de memória por máquina virtual é apresentada no eixo das ordenadas (y).

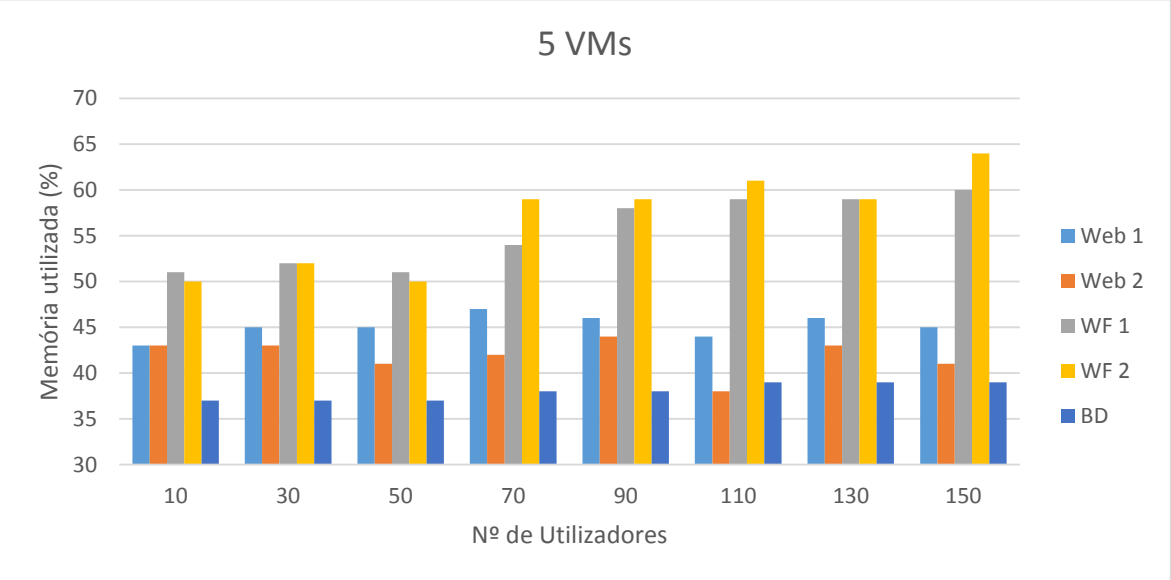

Figura 52 - Percentagem média de utilização de memória

Pela análise do gráfico presente na Figura 52 é possível concluir que as percentagens médias de utilização de memória para os servidores web foram relativamente constantes, apresentando uma variação entre 43% e 47% para a máquina "VM 1" e de 38% para 44% para a máquina "VM 2".

Ambas as máquinas virtuais que alojam o serviço de gestão de workflows apresentam um nível médio de utilização de memória com tendência crescente à medida que a carga de utilização sobre a aplicação aumenta, apresentado uma variação de 51% para 60% e de 50% para 64% respetivamente.

A variação da percentagem média de utilização de memória, para a máquina virtual que aloja o SGBD, é praticamente inexistente, apresentando uma variação de 36% para 39%.

9.1.2.13*Resultados médios de processamento para sete VMs*

Na Tabela 25 são apresentados os resultados relativos aos tempos médios de processamento para as diferentes máquinas virtuais utilizadas no processo de testes de carga. Nesta tabela as designações "Web 1", "Web 2", "Web 3" e "Web 4" representa as máquinas virtuais onde a aplicação web se encontra alojada. As designações "WF 1" e "WF 2" representa as máquinas virtuais onde estão alojados os serviços de gestão de workflows e a designação "BD" representa a máquina virtual onde está alojado o SGBD.

Tabela 25 - Percentagem média de processamento por máquina virtual

Nº Utilizadores	Web 1	Web 2	Web 3	Web 4	WF 1	WF 2	BD
10	12	10	8	7	12	13	8
30	28	22	20	13	25	30	12
50	27	20	18	10	35	42	16
70	28	20	17	11	37	41	15
90	28	23	20	10	33	43	15
110	24	19	23	10	32	46	18
130	28	24	20	10	32	27	17
150	24	15	16	9	30	43	14

Na Figura 53 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 25. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa à percentagem média de processamento por máquina virtual é apresentada no eixo das ordenadas (y).

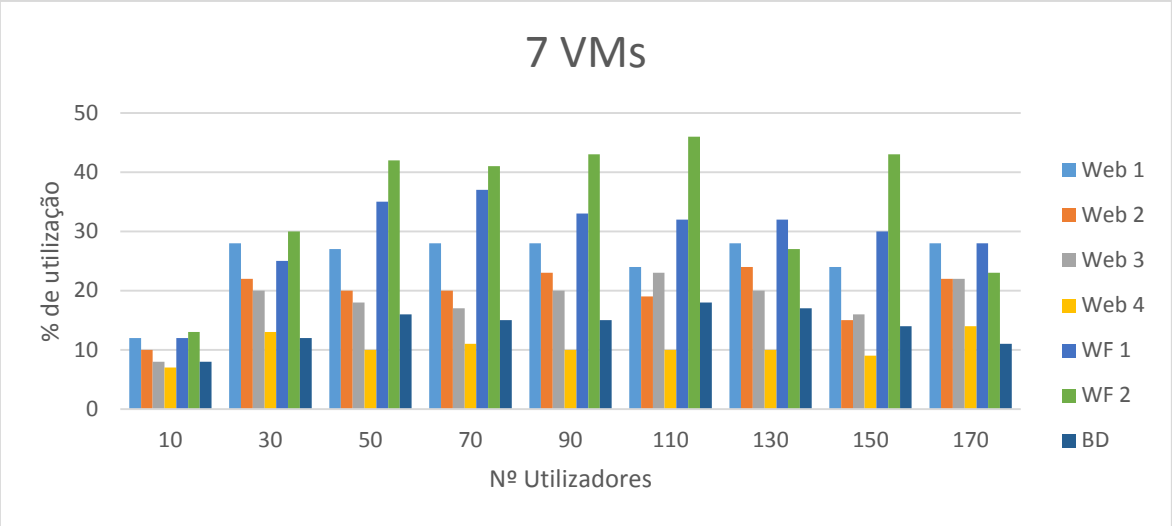

Figura 53 - Percentagem média de processamento por máquina virtual

Pela análise do gráfico presente na Figura 53 é possível concluir que o esforço de processamento para cada uma das máquinas virtuais que alojam a aplicação web foi significativamente reduzido quando comparado com o gráfico da Figura 51. Utilizando um sistema de load balancing como elemento de balanceamento de carga entre as quatro máquinas que alojam a aplicação web permitiu reduzir a percentagem média de CPU, a qual na configuração anterior residia entre os 25% a 58%, para valores que se situam entre os 7% a 37% de percentagem média de processamento.

A máquina virtual que aloja o serviço de gestão de base de dados contém, para 100% dos casos, o menor nível de processamento médio. Este nível de processamento manteve-se ainda relativamente constante para os diferentes testes de carga (variando entre 8% e 18%).

As máquinas virtuais que alojam o serviço de gestão de *workflows* apresentam maior esforço de processamento médio sendo, em 66.7% dos testes, as duas máquinas virtuais em que o esforço de processamento foi mais intenso.

9.1.2.14*Resultados de utilização média de memória RAM para sete VMs*

Na Tabela 26 são apresentados os resultados relativos à utilização média de memória para as diferentes máquinas virtuais utilizadas no processo de testes de carga. Nesta tabela as designações “Web 1”, “Web 2”, “Web 3” e “Web 4” representam as máquinas virtuais onde a aplicação web se encontra alojada. A designação “WF 1” e “WF 2” representam as máquinas virtuais onde estão alojados os serviços de gestão de workflows. A designação “BD” representa a máquina virtual onde está alojado o SGBD.

Tabela 26 - Percentagem média de utilização de memória

Nº Utilizadores	Web 1	Web 2	Web 3	Web 4	WF 1	WF 2	BD
10	45	51	42	42	43	40	40
30	44	41	42	40	40	39	40
50	42	42	40	40	42	34	41
70	43	42	41	40	43	36	41
90	43	42	41	40	45	38	41
110	43	42	42	41	46	40	42
130	42	42	43	41	47	61	42
150	42	41	41	42	48	47	42
170	43	41	42	39	50	49	43

Na Figura 54 é apresentada, sob um formato de gráfico de barras, a informação presente na Tabela 26. Neste gráfico a informação relativa às diferentes combinações de utilizadores é apresentada no eixo das abscissas (x) e a informação relativa à utilização média de memória por máquina virtual é apresentada no eixo das ordenadas (y).

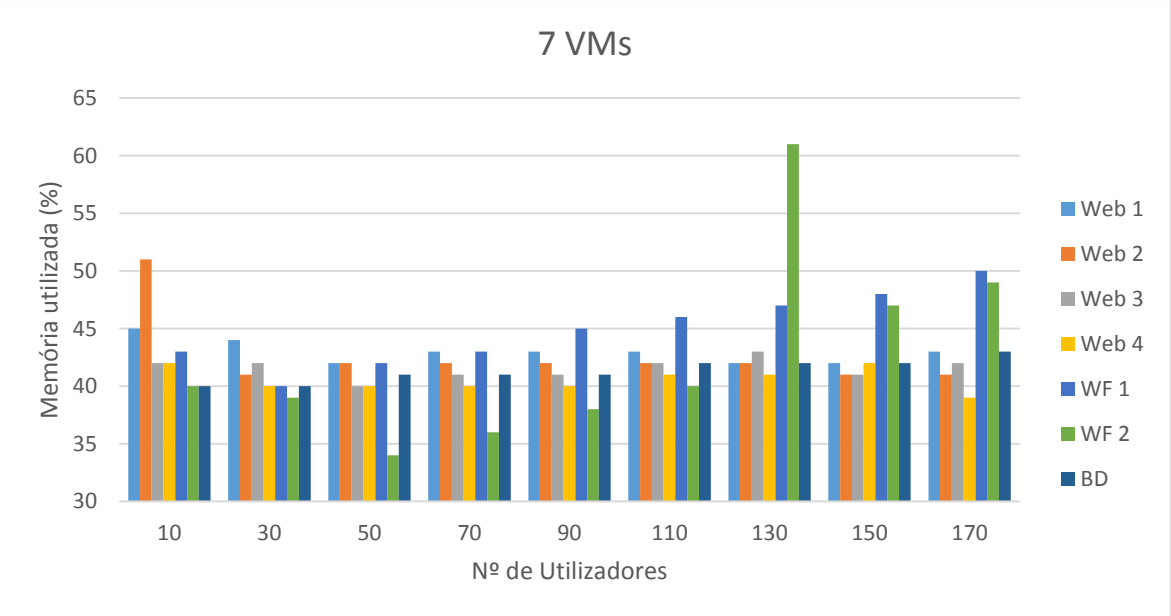

Figura 54 - Percentagem média de utilização de memória

Pela análise do gráfico presente na Figura 54 é possível concluir que as percentagens médias de utilização de memória para os servidores web foram relativamente constantes, apresentando uma variação entre 42% e 45% para a máquina "VM 1", entre 41% e 51% (o valor de 51% foi registado

apenas no teste de carga com 10 utilizadores) para a máquina "VM 2", entre 40% e 43% para a máquina "VM3" e entre 39% e 42% para a máquina "VM 4".

A máquina virtual que aloja o serviço de gestão de workflows "VM 5" apresenta um nível médio de utilização de memória tendencialmente crescente à medida que a carga de utilização sobre a aplicação aumenta, apresentado uma variação entre 40% e 50%. A máquina virtual "VM 6" apresenta algumas inconsistências, sendo que o nível de utilização médio de memória teve uma tendência decrescente para os testes de carga até 50 utilizadores e apresentando uma tendência crescente para os restantes casos.

A variação da percentagem média de utilização de memória, para a máquina virtual que aloja o SGBD, é praticamente inexistente, apresentando uma variação de 40% para 43%.

9.2 Scripts de Automação

9.2.1 Deploy da aplicação web

9.2.1.1 *Compilação e deploy*

Exemplo do comando que permite compilar a aplicação gerar os pacotes de *deploy* desta aplicação. Para tal

```
1. cd c:\MJCV_SIJ\Releases\MJCV_SIJ_1.2\MJCVWebApp msbuild /t:publish
```

9.2.1.2 *Sincronização de conteúdos*

Exemplo do comando de sincronização do conteúdo dos servidores de IIS entre máquinas. Neste cenário a máquina "192.168.160.19" contém uma versão da aplicação web que será sincronizada com a máquina "192.168.160.10".

```
call "C:\Program Files (x86)\IIS\Microsoft Web Deploy\msdeploy.exe" -
verb:sync -source:apphostconfig="Default Web
Site",computername=192.168.160.19 -dest:apphostconfig="Default Web
Site",computername=192.168.160.10"
```

9.2.2 Serviço de Gestão de Workflows

Exemplo dos comandos que permitem conduzir o *deploy* do serviço de gestão de *workflows* em máquinas remotas. Neste exemplo é conduzido o *deploy* na máquina "TESTE-AGENT01.opennebula.ua.pt".

```
call sc "\\TESTE-AGENT01.opennebula.ua.pt" stop MJCVService
call robocopy
C:\MJCV_SIJ\Releases\MJCV_SIJ_1.3\MJCVWindowsService\bin\x86\Test
\\TESTE-AGENT01.opennebula.ua.pt\MJCVWindowsService
call sc "\\TESTE-AGENT01.opennebula.ua.pt" start MJCVService
```

9.2.3 *Deploy* de Base de dados

O processo de *deploy* da base de dados é inicialmente conduzido recorrendo à ferramenta *MSDeploy*. Esta ferramenta permite compilar o projeto de base de dados e definir o *Target* da compilação (sendo neste exemplo o *deploy* do projeto para a base de dados *MJCVProcessos* na máquina "192.168.160.19").

```
msbuild /t:deploy /p:TargetConnectionString="Data
Source=192.168.160.19;Persist Security Info=True;User ID=mjcv;
Password=XXXXXX;Pooling=False" /p:TargetDatabase=MJCVProcessos
```

9.2.3.1 *Comandos base*

Exemplo dos comandos que permitem realizar o backup de uma instância da base de dados, eliminar base de dados alvo e conduzir o restore da bases de dados utilizando ficheiros com instruções de SQL ("BackupDatabase.sql", "DropAllDatabases.sql" e "RestoreDatabaseMJCVProcessos.sql") .

```
if ( (Get-PSSnapin -Name SqlServerCmdletSnapin100  -ErrorAction
SilentlyContinue) -eq $null )
{
    Add-PsSnapin SqlServerCmdletSnapin100
}
if ( (Get-PSSnapin -Name SqlServerProviderSnapin100  -ErrorAction
SilentlyContinue) -eq $null )
{
    Add-PsSnapin SqlServerProviderSnapin100
}

invoke-sqlcmd -inputfile "C:\Powershell Scripts\BackupDatabase.sql" -
serverinstance "192.168.160.19" -querytimeout 65535
"Backup completo"

invoke-sqlcmd -inputfile "C:\Powershell Scripts\DropAllDatabases.sql" -
serverinstance "192.168.160.10" -querytimeout 65535
"Drop BDs da máquina 10 realizado com sucesso...."

invoke-sqlcmd -inputfile "C:\Powershell
Scripts\RestoreDatabaseMJCVProcessos.sql" -serverinstance
"192.168.160.10" -querytimeout 65535
"MJCVProcessos - Restore da máquina 10 realizado com sucesso...."
```

9.2.3.2 *BackupDatabase.sql*

Backup da base de dados "*MJCVProcessos*" para uma localização partilhada na rede ("'C:\DatabaseBackup\MJCVProcessos_backup.bak").

```
USE MASTER
    IF (EXISTS (SELECT name  FROM master.dbo.sysdatabases WHERE ('[' +
name + ']' = 'MJCVProcessos' OR name = 'MJCVProcessos')))
    BEGIN
```

```
        BACKUP DATABASE MJCVProcessos TO
DISK='C:\DatabaseBackup\MJCVProcessos_backup.bak'
        WITH FORMAT;
    END
    ELSE
    BEGIN
        print('MJCVProcessos não encontrada')
    END
```

9.2.3.3 *DropAllDatabases.sql*

Eliminar todas as bases de dados utilizadas no processo de testes.

```
USE MASTER
    IF (EXISTS (SELECT name  FROM master.dbo.sysdatabases WHERE ('[' +
name + ']' = 'MJCVProcessos' OR name = 'MJCVProcessos')))
    BEGIN
        DROP DATABASE MJCVProcessos;
    END
    ELSE
    BEGIN
        print('MJCVProcessos não encontrada')
    END

    IF (EXISTS (SELECT name  FROM master.dbo.sysdatabases WHERE ('[' +
name + ']' = 'MJCVProcessos_TestesBD' OR name =
'MJCVProcessos_TestesBD')))
    BEGIN
            DROP DATABASE MJCVProcessos_TestesBD;
    END
    ELSE
    BEGIN
        print('MJCVProcessos_TestesBD não encontrada')
    END

        IF (EXISTS (SELECT name  FROM master.dbo.sysdatabases WHERE ('['
+ name + ']' = 'MJCVProcessos_TestesProcessoPenal' OR name =
'MJCVProcessos_TestesProcessoPenal')))
    BEGIN
            DROP DATABASE MJCVProcessos_TestesProcessoPenal;
    END
    ELSE
    BEGIN
        print('MJCVProcessos_TestesProcessoPenal não encontrada')
    END
```

9.2.3.4 *RestoreDatabaseMJCVProcessos.sql*

Restore de uma base de dados (“MJCVProcessos”) a partir de uma localização partilhada.

```
USE MASTER
RESTORE DATABASE MJCVProcessos
FROM DISK= '\\TESTE-AGENT01\DatabaseBackup\MJCVProcessos_backup.bak'
WITH
```

```
        MOVE 'MJCVProcessos' TO 'C:\Program Files\Microsoft SQL
Server\MSSQL10_50.MSSQLSERVER\MSSQL\DATA\MJCVProcessos.mdf',
        MOVE 'MJCVProcessos_log' TO 'C:\Program Files\Microsoft SQL
Server\MSSQL10_50.MSSQLSERVER\MSSQL\DATA\MJCVProcessos_1.ldf',
        MOVE 'MJCVProcessosFS_Group' TO 'C:\Program Files\Microsoft SQL
Server\MSSQL10_50.MSSQLSERVER\MSSQL\DATA\MJCVProcessos_2.MJCVProcessosFS'
, REPLACE
GO
```